Конструкција и анализа алгоритама

Материјал са предавања

Миодраг Живковић
е-маил: ezivkovm@matf.bg.ac.rs
УРЛ: www.matf.bg.ac.rs/~ezivkovm

Весна Маринковић
е-маил: vesnap@matf.bg.ac.rs
УРЛ: www.matf.bg.ac.rs/~vesnap
Математички факултет, Београд

Аутори:
проф. др Миодраг Живковић, редовни професор Математичког факултета у Београду
др Весна Маринковић, доцент Математичког факултета у Београду

КОНСТРУКЦИЈА И АНАЛИЗА АЛГОРИТАМА

Садржај

Глава 1

Преглед техника доказивања

Сврха доказа је да утврди тачност неког тврђења [1]. Доказ се састоји од низа тврђења који се завршава закључком. Да би доказ био исправан (коректан), закључак мора да следи из премиса (претпоставки) од којих се полази. За извођење нових тврђења од оних која су већ изведена, користе се *правила извођења* — обрасци за конструкцију исправних доказа. Правила извођења представљају основни алат којим се обезбеђује истинитост тврђења.

Размотримо следеће закључивање.

"Ако имаш важећу лозинку, можеш да се пријавиш на мрежу."

"Имаш важећу лозинку."

Дакле:

"Можеш да се пријавиш на мрежу."

Ако са p означимо тврђење "имаш важећу лозинку", а са q "можеш да се пријавиш на мрежу", онда се ово закључивање може записати у облику

$$\frac{p \quad (p \to q)}{q}$$

Изнад хоризонталне линије написане су премисе, а испод закључак. Ако су p и q логичке променљиве, онда је израз $(p \wedge (p \to q)) \to q$ *таутологија*, што се може непосредно проверити помоћу таблице истинитости:

$(p$	$\wedge$	$(p$	$\to$	$q))$	$\to$	q
0	0	0	1	0	1	0
0	0	0	1	1	1	1
1	0	1	0	0	1	0
1	1	1	1	1	1	1

Као што је познато, постоји више врста математичких тврђења,

- теорема (тврђење за које треба доказати да је тачно)
- аксиома (тврђење које се прихвата као тачно без доказа)
- дефиниција (тврђење којим се уводи неки појам)

[1] Основа за овај текст је књига [1].

- лема (тврђење које, као и теорема, подлеже доказивању, међутим мањег је значаја и служи за доказивање теореме)

- последица (теорема чије се тврђење релативно једноставно изводи полазећи од теореме)

- хипотеза (тврђење за које се претпоставља да је тачно, али његов доказ још није познат)

Доказ теореме је низ тврђења који почиње претпоставкама теореме, завршава се тврђењем теореме, при чему свако наредно тврђење следи из претходних на основу правила извођења, или је аксиома, дефиниција, или претходно доказана теорема или лема.

Методе извођења доказа које се разматрају у овом поглављу су значајне не само зато што се користе за доказивање математичких теорема, већ такође и за многе примене у рачунарству. Те примене обухватају проверу исправности рачунарских програма, закључивање у области вештачке интелигенције, доказивање да је нека спецификација система конзистентна и слично.

1.1 Правила извођења

Правила извођења повезују делове доказа тако што обезбеђују логичко извођење закључака на основу скупа претпоставки. Правила извођења су важна не само за доказивање теорема, него и за доказе коректности алгоритама, програма, сигурности оперативног система, конзистентности система спецификација.

Основно правило извођења је *модус поненс*. Према том правилу, ако су тачна тврђења p и $p \to q$, онда је тачно и тврђење q. Ово правило се краће записује на следећи начин:

$$\frac{p \quad (p \to q)}{q}$$

Остала важна правила извођења за исказну логику су приказана у следећој табели:

	Име правила	Правило	Одговарајућа таутологија
1	Модус поненс	$\dfrac{p \quad (p \to q)}{q}$	$(p \wedge (p \to q)) \to q$
2	Додавање	$\dfrac{p}{p \vee q}$	$p \to (p \vee q)$
3	Упрошћавање	$\dfrac{p \wedge q}{p}$	$(p \wedge q) \to p$
4	Конјункција	$\dfrac{p \quad q}{p \wedge q}$	$(p \wedge q) \to (p \wedge q)$
5	Модус толенс	$\dfrac{\neg q \quad (p \to q)}{\neg p}$	$(\neg q \wedge (p \to q)) \to \neg p$
6	Хипотетички силогизам	$\dfrac{(p \to q) \quad (q \to r)}{p \to r}$	$((p \to q) \wedge (q \to r)) \to (p \to r)$
7	Дисјунктивни силогизам	$\dfrac{(p \vee q) \quad \neg p}{q}$	$((p \vee q) \wedge \neg p) \to q$
8	Резолуција	$\dfrac{(p \vee q) \quad (\neg p \vee r)}{q \vee r}$	$((p \vee q) \wedge (\neg p \vee r)) \to (q \vee r)$

Исправан доказ тврђења $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \to q$ је изведен ако се покаже да из тачности свих претпоставки (премиса) $p_1, p_2, \ldots p_n$, следи да је тачно тврђење q.

Ако је закључивање исправно, а нека претпоставка није тачна, онда доказивано тврђење не мора да буде тачно

Пример. У закључивању:

"Ако је $\sqrt{2} > \frac{3}{2}$, онда је $(\sqrt{2})^2 > (\frac{3}{2})^2$. Знамо да је $\sqrt{2} > \frac{3}{2}$. Дакле, $(\sqrt{2})^2 = 2 > (\frac{3}{2})^2 = \frac{9}{4}$"

примењено је исправно правило закључивања (модус поненс), али је тврђење $2 > \frac{9}{4}$ нетачно. То је зато што је коришћена премиса $\sqrt{2} > \frac{3}{2}$ која није тачна.

Пример. Показати да из премиса

"ако ми пошаљеш поруку, завршићу програм",

"ако ми не пошаљеш поруку, идем раније на спавање",

"ако одем раније на спавање, пробудићу се одморан",

следи закључак

"ако не завршим програм, пробудићу се одморан".

Згодно је да најпре уведемо ознаке p, q, r, s редом за реченице "послаћеш ми поруку", "завршићу програм", "идем раније на спавање", "пробудићу се одморан". Дакле, треба показати да из премиса (1) $p \to q$, (2) $\neg p \to r$, (3) $r \to s$, следи закључак $\neg q \to s$.

Доказ:

(1)	$p \to q$	премиса (1)
(2)	$\neg q \to \neg p$	контрапозиција линије (1)
(3)	$\neg p \to r$	премиса (2)
(4)	$\neg q \to r$	хипотетички силогизам од линије (2) и линије (3)
(5)	$r \to s$	премиса (3)
(6)	$\neg q \to s$	хипотетички силогизам од линије (4) и линије (5)

1.1.1 Резолуција

Резолуција се често користи у програмима за аутоматско доказивање теорема. Правило резолуције оправдава таутологија $((p \vee q) \wedge (\neg p \vee r)) \to (q \vee r)$. Овде се $(q \vee r)$ зове *резолвента* премиса $(p \vee q)$ и $(\neg p \vee r)$. Специјално, за $q = r$, добија се таутологија $((p \vee q) \wedge (\neg p \vee q)) \to q$.

Пример. Из премиса "Јасмина иде на факултет или је нерадни дан", "Радни је дан или Марко путује кући" следи закључак "Јасмина иде на факулет или Марко путује кући".

Означимо са p тврђење "Јасмина иде на факултет", са q тврђење "радни је дан", а са r тврђење "Марко путује кући", онда можемо премисе да запишемо као $p \vee \neg q$ и $q \vee r$, те из премиса коришћењем правила резолуције следи тврђење $p \vee r$.

У доказима у којима се примењује резолуција премисе и тврђења морају се приказати у облику конјункције *клауза*, где су клаузе дисјункције променљивих или њихових негација.

- Због $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ премиса $p \vee (q \wedge r)$ замењује са две нове премисе $p \vee q$ и $p \vee r$.
- Због $\neg(p \vee q) \equiv \neg p \wedge \neg p$ премиса $\neg(p \vee q)$ замењује са две нове премисе $\neg p$ и $\neg p$.
- Због $p \to q \equiv \neg p \vee q$ премиса $p \to q$ замењује се (једном) премисом $\neg p \vee q$.

Овде везник $\equiv$ означава еквивалентност два логичка израза, односно чињеницу да су њихове логичке вредности једнаке за било које вредности променљивих.

Пример. Показати да из премиса $(p \wedge q) \vee r$ и $r \to s$ следи закључак $p \vee s$.

Доказ. Премиса $(p \wedge q) \vee r \equiv (p \vee r) \wedge (q \vee r)$ има за последицу клаузу $p \vee r$, а премиса $r \to s \equiv \neg r \vee s$ има за последицу клаузу $\neg r \vee s$. Применом резолуције на ове две клаузе, $p \vee r$ и $\neg r \vee s$, добија се тражени закључак $p \vee s$.

Погрешна закључивања

Чест начин на који се долази до погрешних закључака је закључивање на основу погрешних правила закључивања. На пример, не може се примењивати правило закључивања

$$\frac{p \to q \quad q}{p}$$

јер $((p \to q) \wedge q) \to p$ није таутологија (за $p \equiv 0$, $q \equiv 1$ овај израз је нетачан).

Пример. Из премиса "ако решиш све задатке из књиге, научићеш дискретну математику" и "научио си дискретну математику" не може се извести закључак "решио си све задатке из књиге".

Слично, израз $((p \to q) \wedge \neg p) \to \neg q$ није таутологија, па се не може примењивати правило закључивања

$$\frac{p \to q \quad \neg p}{\neg q}$$

Пример. Из премиса "ако решиш све задатке из књиге, научићеш дискретну математику" и "ниси решио све задатке из књиге" не може се извести закључак "ниси научио дискретну математику".

1.1.2 Правила извођења у логици првог реда

Постоје четири основна правила извођења која се користе за доказивање тврђења у којима се користе квантификатори и она су дата у следећој табели:

Правило	специјализација	генерализација
Универзална	$\dfrac{\forall x P(x)}{P(c)}$	$\dfrac{P(c)}{\forall x P(x)}$
Егзистенцијална	$\dfrac{\exists x P(x)}{P(c)}$	$\dfrac{P(c)}{\exists x P(x)}$

Пример. Нека $I(x)$ означава реченицу "x је на модулу И", и нека $A(x)$ означава реченицу "x слуша АСП". Показати да из премиса $\forall x(I(x) \to A(x))$ и $I(Marija)$ следи закључак $A(Marija)$

Доказ.

(1)	$\forall x(I(x) \to A(x)$	премиса
(2)	$I(Marija) \to A(Marija)$	универзална специјализација од линије 1
(3)	$I(Marija)$	премиса
(4)	$A(Marija)$	модус поненс од линије (2) и линије (3)

Пример. Нека $G(x)$, $K(x)$, $I(x)$, означавају редом реченице "x је у овој групи", "x је прочитао књигу о алгоритмима" и "x је положио испит из алгоритама". Показати да из премиса $\exists x(G(x) \wedge \neg K(x))$ и $\forall x(G(x) \to I(x))$ следи закључак $\exists x(I(x) \wedge \neg P(x))$.

Доказ.

(1)	$\exists x(C(x) \wedge \neg B(x))$	премиса
(2)	$C(a) \wedge \neg B(a)$	егзистенцијална специјализација од линије (1)
(3)	$C(a)$	упрошћавањем од линије (2)
(4)	$\forall x(C(x) \to P(x))$	премиса
(5)	$C(a) \to P(a)$	универзална специјализација од линије (4)
(6)	$P(a)$	модус поненс од линије (3) и линије (5)
(7)	$\neg B(a)$	упрошћавањем од линије (2)
(8)	$P(a) \wedge \neg B(a)$	проширивањем од линије (6) и линије (7)
(9)	$\exists x(P(x) \wedge \neg B(x))$	егзистенцијалном генерализацијом од линије (8)

1.2 Докази

Доказивање теорема може бити тешко. У овом поглављу даћемо примере извођења доказа, као и примере погрешног закључивања који воде погрешним доказима.

1.2.1 Врсте доказа

Докази се најчешће односе на теореме облика $p \to q$. Најједноставнији су директни докази. Полази се од претпоставке да је p тачно и показује се

да је тада q тачно.

Директни доказ

Тврђење $p \to q$ се доказује тако што се докаже да ако је p тачно, онда је q тачно.

Наредни пример је пример директног доказа.

Пример. Доказати да ако је број n непаран, онда је и n^2 непаран број.

Доказ. По дефиницији број n паран ако је $n = 2k$, где је k цели број; број n је непаран ако је $n = 2k + 1$, где је k цели број. Пошто је n непаран, може се записати у облику $n = 2k+1$. Тада је $n^2 = (2k+1)^2 = 4k^2+4k+1 = 2(2k^2 + 2k) + 1$ такође непаран број.

Индиректни доказ

Уместо импликације $p \to q$ доказује се њена контрапозиција, односно импликација $\neg q \to \neg p$ (полазећи од $\neg$).

Пример. Доказати да ако је $3n^2 + 2$ непаран број, онда је и n непаран број.

Доказ. Претпоставимо супротно, да је n паран број, односно да је $n = 2k$, где је k неки цели број. Тада је $3n^2 + 2 = 3(2k)^2 + 2 = 2(6k^2 + 1)$ паран број, чиме је доказана импликација $\neg q \to \neg p$, а тиме и импликација $p \to q$.

Празан доказ

Ако је p нетачно, импликација $p \to q$ је увек тачна.

Пример. Нека је $P(n)$ ознака за тврђење "Ако је $n > 1$, онда је $n^2 > n$". Доказати $P(0)$.

Тривијалан доказ

Ако је q тачно, импликација $p \to q$ је увек тачна.

Пример. Нека је $P(n)$ ознака за тврђење "Ако је $a, b \in N$, $a \geq b$, онда је $a^n \geq b^n$". Доказати $P(0)$.

Доказ извођењем контрадикције

Тврђење p доказује се тако што се претпостави супротно, тј. да је тачно $\neg p$, па се из тога изведе контрадикција.

Пример. Доказати да је $\sqrt{2}$ ирационалан број.

Доказ. Претпоставимо супротно, да је $\sqrt{2}$ рационалан број, односно да се $\sqrt{2}$ може написати у облику несводљивог разломка $\sqrt{2} = \frac{a}{b}$. Квадрирањем се добија $a^2 = 2b^2$, па се закључује да је a^2 паран број. У једном од претходних примера показано је да из ове чињенице следи да је и a паран број, односно да се може написати у облику $a = 2a_1$. Квадрирањем се добија $(2a_1)^2 = 2b^2$, односно $2a_1^2 = b^2$, из чега следи да је b^2, а тиме и b, паран број. Дошли смо до закључка да су оба броја a и b парна, односно да се разломак $\frac{a}{b}$ може скратити, супротно претпоставци да је несводљив. Изведена контрадикција доказује да претпоставка да је $\sqrt{2}$ рационалан број није тачна, односно да је $\sqrt{2}$ ирационалан број.

Пример. Доказати да најмање 4 од 22 произвољних дана у години морају да падну у исти дан у недељи.

Доказ. Претпоставимо супротно, да највише 3 од 22 дана у години могу да падну у исти дан у недељи. Обзиром да постоји седам различитих дана у недељи, ово нам даје да би могли да разматрамо највише 21 дан, а не 22 дана колико је потребно, те добијамо контрадикцију.

Доказ разматрањем случајева

Израз

$$((p_1 \vee p_2 \vee \cdots \vee p_n) \to q) \leftrightarrow ((p_1 \to q) \wedge (p_2 \to q) \wedge \cdots \wedge (p_n \to q))$$

је таутологија. Због тога се тврђење облика $(p_1 \vee p_2 \vee \cdots \vee p_n) \to q$ може доказати тако што се докаже да важи $p_i \to q$ за свако $i = 1, 2, \ldots, n$.

Пример. Доказати да за реалне бројеве x, y, важи једнакост $|xy| = |x||y|$.

Доказ . Сваки од бројева x, y може да буде негативан, или већи или једнак од нуле, па су могућа четири случаја

1. $x \geq 0 \wedge y \geq 0$: $|xy| = xy$, $|x||y| = xy$;
2. $x < 0 \wedge y \geq 0$: $|xy| = -xy$, $|x||y| = (-x)y = -xy$;
3. $x \geq 0 \wedge y < 0$: слично као у претходном случају $|xy| = -xy$, $|x||y| = x(-y) = -xy$;
4. $x < 0 \wedge y < 0$: $|xy| = xy$, $|x||y| = (-x)(-y) = xy$.

Пошто је у сва четири случаја импликација тачна, тврђење је доказано.

Докази еквивалентности

Израз $(p \leftrightarrow q) \leftrightarrow ((p \to q) \wedge (q \to p))$ је таутологија. Због тога се тврђење облика $p \leftrightarrow q$ (p је тачно ако и само ако је тачно q) доказује тако што се докажу обе импликације $p \to q$ и $q \to p$.

Пример. Доказати да је број n непаран ако и само ако је број n^2 непаран.

Тачност овог тврђења последица је две импликације које су доказане у претходним примерима.

Општије, у случају када треба доказати еквиваленцију

$$p_1 \leftrightarrow p_2 \leftrightarrow \cdots \leftrightarrow p_n$$

(односно да је за сваки пар индекса i, j, $1 \leq i < j \leq n$, тачна еквиваленција $p_i \leftrightarrow p_j$) због тачности таутологије

$$(p_1 \leftrightarrow p_2 \leftrightarrow \cdots \leftrightarrow p_n) \leftrightarrow ((p_1 \to p_2) \wedge (p_2 \to p_3) \wedge \cdots \wedge (p_n \to p_1))$$

довољно је доказати n (уместо $\binom{n}{2}$) импликација

$$(p_1 \to p_2) \wedge (p_2 \to p_3) \wedge \cdots \wedge (p_n \to p_1).$$

Пример. Доказати да су следећа три тврђења еквивалентна:

1. Број n је паран.
2. Број $n-1$ је непаран.
3. Број n^2 је паран.

Доказ. Доказ се може извести тако што се докажу три импликације:

1. Ако је n паран (тј. $n=2k$), онда је $n-1$ $(=2(k-1)+1)$ непаран број.
2. Ако је $n-1$ непаран број (тј. $n-1=2k+1$, односно $n=2k+2$), онда је n^2 $(=2\cdot 2(k+1)^2)$ паран број.
3. Ако је n^2 паран број, онда је n паран број (ово тврђење доказано је у једном од претходних примера).

1.2.2 Теореме са квантификаторима

У овом поглављу биће размотрене теореме у облику тврђења са егзистенцијалним квантификатором, односно са универзалним квантификатором.

Тврђења са егзистенцијалним квантификатором

Тврђење $\exists x P(x)$ може се доказати тако што се пронађе конкретно a за које је $P(a)$ тачно; то је *конструктивни доказ*. Ако се пак тачност тог тврђења докаже не пронашавши при томе објекат a за који је $P(a)$ тачно, онда се такав доказ зове *неконструктивни доказ*.

Пример. Доказати да постоји број $n \in N$ који се може на два начина написати у облику збира два куба природних бројева.

Доказ. Непосредно се проверава да је $1729 = 10^3+9^3 = 12^3+1^3$, тј. број 1729 се може на два начина написати у облику збира два куба природних бројева. Ово је пример конструктивног доказа.

Пример. Доказати да постоје ирационални бројеви x, y такви да је x^y рационалан број.

Доказ. У једном од претходних примера доказано је да је $\sqrt{2}$ ирационалан број. Посматрајмо број $a = \sqrt{2}^{\sqrt{2}}$. Број a може да буде или рационалан или ирационалан.

- Ако је број a рационалан, онда је за ирационалне бројеве $x = y = \sqrt{2}$ степен x^y рационалан број.
- Ако је број a ирационалан, онда је за ирационалне бројеве $x = a$ и $y = \sqrt{2}$ степен $x^y = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = \sqrt{2}^2 = 2$ рационалан број.

Тиме је тражено тврђење доказано, иако није пронађен ни један пример ирационалних бројева x, y за које је дефинитивно утврђено да је x^y рационалан број.

Докази јединствености

Понекад је потребно доказати не само $\exists x P(x)$, него и да постоји само један објекат x за који је $P(x)$ тачно, односно $\exists x P(x) \wedge \forall y((y \neq x) \rightarrow \neg P(y))$.

Пример. Сваки цели број има јединствени адитивни инверз.

Доказ. Број x има бар један адитивни инверз $-x$, јер је $x+(-x)=0$. Да бисмо доказали да је адитивни инверз јединствен, претпоставимо супротно, да постоји други адитивни инверз y броја x, тј. да важи $x+y=0$. Ако је то тачно, онда је $y=y+0=y+(x+(-x))=(y+x)+(-x)=0+(-x)=-x$, тј. супротно претпоставци, $y=-x$, и адитивни инверз x је јединствен.

Тврђења са универзалним кватификатором

Да би се доказало да тврђење $\forall x P(x)$ није тачно, довољно је пронаћи један објекат a за који $P(a)$ није тачно. Каже се да је $P(a)$ *контрапример* за тврђење $\forall x P(x)$.

Пример. Сваки природан број n једнак је збиру три квадрата ненегативних целих бројева.

Доказ. За $n=1,2,3,4,5,6$ тврђење да се n може написати као збир три квадрата ненегативних целих бројева је тачно:

$1=0^2+0^2+1^2$,
$2=0^2+1^2+1^2$,
$3=1^2+1^2+1^2$,
$4=0^2+0^2+2^2$,
$5=0^2+1^2+2^2$,
$6=1^2+1^2+2^2$.

Међутим, испоставља се да се $n=7$ не може написати као збир три квадрата ненегативних целих бројева. Заиста, претпоставимо да је $7=x^2+y^2+z^2$, $x\leq y\leq z$. Највећа вредност коју могу да узму бројеви x,y,z је 2, јер је $3^2=9$. Највећи од ова три броја z мора бити једнак 2, јер би у противном било $x^2+y^2+z^2\leq 1^2+1^2+1^2=3$. После замене $z=2$ једначина постаје $3=x^2+y^2$. Међутим, ова једначина нема целобројно решење:

- ако је $y\leq 1$, онда је $x^2+y^2\leq 1^2+1^2=2<3$;
- ако је $y\geq 2$, онда је $x^2+y^2\geq 0^2+2^2=4>3$.

Тиме је доказано да разматрано тврђење са универзалним квантификатором није тачно.

1.2.3 Погрешни докази

Понекад имамо ситуацију да постоји низ корака за које се тврди да је доказ датог тврђења; притом се за то тврђење не зна да ли је тачно, или још горе, за то тврђење се зна да није тачно. У том случају доказ је сигурно погрешан, односно бар један корак у доказу је погрешан (тачност одговарајућег тврђења у ствари није доказана).

Пример. Размотримо следећи доказ очигледно погрешног тврђења $1=2$.

Доказ.

(1)	Нека су $a = b \in N$ два броја	
(2)	$a^2 = ab$	добија се од (1) множењем са a
(3)	$a^2 - b^2 = ab - b^2$	одузимањем b^2 од обе стране у (2)
(4)	$(a-b)(a+b) = b(a-b)$	трансформисањем леве и десне стране (3)
(5)	$a + b = b$	скраћивањем једнакости (4)
(6)	$2b = b$	заменом a са b у (5)
(7)	$2 = 1$	скраћивањем једнакости (6)

Доказано тврђење је очигледно погрешно; који корак у доказу је погрешан? То је корак (5): у једнакости (4) урађено је (погрешно) скраћивање бројем $a - b = 0$.

Пример. Размотримо следећи "доказ" очигледно погрешног тврђења. Ако је $n^2 > 0$, онда је $n > 0$: из "$n > 0 \to n^2 > 0$" и $n^2 > 0$ следи $n > 0$. Доказ је погрешан, јер је примењено погрешно правило закључивања.

1.3 Стратегије доказивања

После прегледа типова доказа, размотрићемо примере стратегија доказивања, односно метода и поступака доказивања, корак по корак. Разматрају се могући рецепти за конструкцију доказа: полазећи уназад од тврђења које треба доказати, преправљањем познатог доказа сличног тврђења, односно разматрањем случајева.

Резултат рада математичара су најпре математичке формулације хипотеза, до којих они долазе уопштавањем појединачних примера. Затим они покушавају да докажу формулисане хипотезе, или да их оповргну, проналажењем контрапримера. Видећемо примере који илуструју процес формулисања хипотеза, њиховог доказивања, односно оповргавања, као и примере познатих хипотеза које су и данас само хипотезе. На крају ће бити изложен доказ важног тврђења: да не постоји алгоритам (процедура) који за дати програм и дати улаз утврђује да ли се дати програм на датом улазу икад завршава.

Конструкција доказа скоро никад није једноставна. Почиње се заменом речи њиховим дефиницијама, анализирањем значења претпоставки и тврђења које треба доказати. Затим се може покушати се директним доказом; ако то не буде успешно, може се покушати са индиректним доказом или свођењем на контрадикцију.

1.3.1 Директно закључивање, закључивање уназад

Директно закључивање бива успешно само кад се доказују најједноставнија тврђења, па се најчешће примењује закључивање уназад. Ако треба доказати тврђење q, онда се покушава са проналажењем неког тврђења p, које је лакше доказати него q, али таквог да је тачна импликација $p \to q$. Тражење тврђења r таквог да је тачна импликација $q \to r$ је бескорисно, јер q не следи из $q \to r$ и r.

У следећем примеру доказује се да је *аритметичка средина* $\frac{a+b}{2}$ два ненегативна броја a и b увек већа или једнака од њихове *геометријске средине* $\sqrt{ab}$.

Пример. Доказати да за произвољна два ненегативна броја a и b важи

неједнакост

$$\frac{a+b}{2} \geq \sqrt{ab}.$$

Доказ. Природно је ову неједнакост квадрирати, да се избегне појава корена. Пошто су обе стране неједнакости позитивне, из нове неједнакости следи стара (у ствари, ове две неједнакости су *еквивалентне*, прва од њих важи ако и само ако важи друга):

$$\left(\frac{a+b}{2}\right)^2 \geq ab.$$

Сређивањем ове неједнакости добија се редом $(a+b)^2 \geq 4ab$, односно $a^2 + 2ab + b^2 \geq 4ab$, односно $a^2 - 2ab + b^2 \geq 0$, односно $(a-b)^2 \geq 0$. Последња неједнакост је очигледно увек тачна. Пошто из ње следе све претходне неједнакости, тиме је у ствари доказана полазна неједнакост.

Пример. На столу је гомила од 15 каменчића. Два играча играју игру у којој наизменично вуку потезе, при чему у једном потезу могу да скину са гомиле један, два или три каменчића. Побеђује играч који скине са гомиле последњи каменчић. Доказати да први играч може да игра тако да увек победи, без обзира на потезе другог играча.

Доказ. Игра се може представити на други начин. На линији су исписани бројеви $0, 1, \ldots, 15$, и на броју 15 налази се жетон. Играчи наизменично померају жетон ка мањим бројевима за једно, два или три места. Побеђује онај ко постави жетон на број 0.

$$0 - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 14 - 15$$

Играч који је на потезу може да игра три различита потеза, па ако се крене од позиције 15, број варијанти је огроман, и не види се како би требало да игра први играч да би сигурно победио. Зато је боље кренути од краја, од завршне позиције, када је жетон на броју 0. Играч који стави жетон на 0, победио је. Према томе, позиција 0 је за играча који је у том тренутку на потезу је изгубљена:

$$\boxed{0} - 1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 14 - 15$$

Пошто се из позиција 1, 2 и 3 може једним потезом доћи у позицију 0 (која је за противника изгубљена), те три позиције су добијене за играча на потезу:

$$\boxed{0} - \boxed{1} - \boxed{2} - \boxed{3} - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 14 - 15$$

Играч који је на потезу у позицији 4 једним потезом може да дође само у једну од позиција које су за противника добијене, па је позиција 4 изгубљена:

$$\boxed{0} - \boxed{1} - \boxed{2} - \boxed{3} - \boxed{4} - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 14 - 15$$

Продужујући даље уназад, закључује се да су позиције 5, 6, 7 добијене, а затим да је позиција 8 изгубљена:

$$\boxed{0} - \boxed{1} - \boxed{2} - \boxed{3} - \boxed{4} - \boxed{5} - \boxed{6} - \boxed{7} - \boxed{8} - 9 - 10 - 11 - 12 - 13 - 14 - 15$$

На крају добијамо за све позиције податак да ли су добијене или изгубљене:

0–1–2–3–4–5–6–7–8–9–10–11–12–13–14–15

Дакле, играч који је први на потезу, скида три каменчића да би дошао у позицију 12, изгубљену за противника. Шта год одиграо противник, он допуњује број скинутих каменчића до 4 и пребацује противника у нову изгубљену позицију 8. После тога противник бива доведен у позицију 4, после чега у наредном потезу први играч може да скине са гомиле све преостале каменчиће и тако победи.

1.3.2 Олакшавање доказа свођењем на случајеве

Када се не види како извести доказ, допунска информација у вези са различитим случајевима може да олакша доказ. На пример, ако се доказује тврђење о целим бројевима, могу се раздвојити случајеви са парним, односно непарним бројевима, или са негативним, односно ненегативним бројевима. При томе је важно не прескочити ни један од могућих случајева.

Пример. Ако $2 \nmid n$, $3 \nmid n$, онда $24 \mid n^2 - 1$ ($n^2 - 1$ је дељиво са 24).

Доказ. Природно је искористити разлагање $n^2 - 1 = (n-1)(n+1)$, али се не види какве то има везе са дељивошћу са 24. С обзиром на услове $2 \nmid n$, $3 \nmid n$, који су могући остаци при дељењу n са 6? Сваки број n може се представити у облику $6k + j$, $j = 0, 1, 2, 3, 4, 5$. Међутим, бројеви $6k + j$, $j = 0, 2, 4$ су дељиви са 2, а бројеви $6k + j$, $j = 0, 3$ дељиви су са 3. Према томе, остаје да се размотре само два случаја, да је n облика $6k + j$, где је $j = 1$ или $j = 5$.

- У првом случају је $n^2 - 1 = (6k+1)^2 - 1 = 36k^2 + 12k = 12k(3k+1)$; овај број је очигледно дељив са 12. Поред тога, ако је k парно, број је дељив са 24; ако је k непарно, онда је $3k + 1$ парно, па је и у том случају $n^2 - 1$ дељив са 24.

- У другом случају је $n^2 - 1 = (6k+5)^2 - 1 = 36k^2 + 60k + 24 = 12(3k^2 + 5k + 2)$; овај број је очигледно дељив са 12. Поред тога, ако је k парно, број је дељив са 24; ако је k непарно, онда је $3k^2 + 5k + 2$ парно, па је и у том случају $n^2 - 1$ дељив са 24.

Пошто су размотрена оба могућа случаја, доказано је да је $n^2 - 1$ увек дељиво са 24.

Уобичајена грешка је да се погрешан закључак изведе на основу примера (на пример, погрешан закључак да су сви непарни бројеви прости). Колико год добрих примера пронашли, то никад не представља доказ. Слично, без обзира колико тестова програм пролази, то не представља доказ да је програм коректан.

Пример. Доказати да једначина $x^2 + 3y^2 = 8$ нема целобројна решења.

Доказ. Ако је $|x| \geq 3$, онда је лева страна бар $9 > 8$; слично, ако је $|y| \geq 2$, онда је лева страна бар $12 > 8$. Према томе, могуће вредности за $|x|$ су $0, 1, 2$, а могуће вредности за $|y|$ су $0, 1$. За $|y| = 0$ једначина постаје $x^2 = 8$ и нема целобројних решења; за $|y| = 1$ једначина постаје $x^2 = 5$ и такође нема целобројних решења. Тиме је разматрањем оба могућа случаја доказано да ова једначина нема целобројна решења.

1.3.3 Прилагођавање познатог доказа сличног тврђења

Корисно је читати и разумети доказе тврђења, јер то омогућује извођење нових доказа тврђења која су слична претходним.

Пример. Доказати да простих бројева облика $4k+3$ (односно $4k-1$) има бесконачно много.

Доказ. Слично тврђење је да простих бројева има бесконачно много, и доказ тог тврђења је познат. Доказ почиње тако што се претпостави да је број простих бројева коначан и једнак $n \in N$, тј. сви могући прости бројеви су $p_1, p_2, \ldots, p_n$, онда број $P = p_1 p_2 \ldots p_n + 1$ није дељив ни једним од простих бројева p_i, $i = 1, 2, \ldots, n$. Према томе, број P је или прост (а није једнак неком од бројева $p_1, p_2, \ldots, p_n$), или је дељив неким простим бројем који није међу простим бројевима $p_1, p_2, \ldots, p_n$. То је у оба случаја у контрадикцији са претпоставком да сем $p_1, p_2, \ldots, p_n$ нема других простих бројева.

Идеја је преправити овај доказ тако да се докаже да простих бројева облика $4k-1$ има бесконачно много. Претпоставимо да је број простих бројева облика $4k-1$ коначан и једнак $n \in N$, тј. да су сви могући прости бројеви тог облика $q_1, q_2, \ldots, q_n$. Посматрајмо број $Q = 4q_1 q_2 \ldots q_n - 1$. Број Q је облика $4k-1$, па мора да има бар један прост чинилац облика $4k-1$ (у противном, ако би имао само просте чиниоце облика $4k+1$, онда би и сам био облика $4k+1$); како Q није дељив ни једним од бројева $q_1, q_2, \ldots, q_n$, закључујемо да, супротно претпоставци, Q мора бити дељив бар још једним простим бројем облика $4k-1$. Ова контрадикција доказује жељено тврђење.

1.4 Хипотезе, докази, контрапримери

1.4.1 Хипотезе и докази

Како се проналазе нови резултати у математици? У сваком случају не онако како је то испричано у уџбеницима. Најпре се на основу специјалних случајева запажају неке законитости, на основу чега се формулишу хипотезе. До хипотезе се понекад долази променом познатог тврђења, или на основу интуиције. Формулисану хипотезу затим треба доказати или оповргнути. Ако је вероватније да је хипотеза тачна, покушава се са извођењем доказа. Ако се не успева са извођењем доказа, покушава се са проналажењем контрапримера; ако то не успе, поново се покушава са доказом. Многе хипотезе су брзо решене, али су зато неке хипотезе остале отворене стотинама година. Следећи пример је илустрација процеса формулисања и доказивања хипотезе.

Пример. Поставља се питање када су бројеви $a^n - 1$ прости?

Анализа примера: Посматрајући примере $2^3 - 1 = 7$, $2^5 - 1 = 31$, $2^6 - 1 = 63 = 7 \cdot 9$, $2^9 - 1 = 511 = 7 \cdot 73$, $3^4 - 1 = 80 = 5 \cdot 16$, $4^5 - 1 = 1023 = 3 \cdot 341$, може се закључити да је $a^n - 1$ сложен број ако је $a > 2$ или ако је $a = 2$ и n је сложен број.

Доказ. На основу разлагања $a^n - 1 = (a-1)(a^{n-1} + a^{n-2} + \cdots + a + 1)$ закључује се да је $a^n - 1$ сложен број ако је $a > 2$ (јер је тада $a^n - 1$ дељиво са $a - 1 \geq 2$). На сличан начин је и број $2^{ab} - 1 = (2^a)^b - 1$ сложен, јер за $a > 1$ важи $2^a > 2$.

Закључује се да број $2^n - 1$ може бити прост једино ако је експонент n прост број; прости бројеви тог облика су тзв. *Мерсенови прости бројеви*.

Пример. Постоје дугачки низови узастопних сложених бројева, на пример $24, 25, 26, 27, 28$; $90, 91, 92, 93, 94, 95, 96$. Поставља се питање: да ли постоје произвољно дугачки низови узастопних сложених бројева? Испоставља се да је одговор на ово питање - да.

Доказ. Посматрајмо низ од $n-1$ узастопних бројева $n!+2$, $n!+3$, $n!+4$, $\ldots$, $n!+n$. Први од њих је увек дељив са 2; други од њих је увек дељив са 3; трећи је увек дељив са 4 $\ldots$; последњи је увек дељив са n. Према томе, сви бројеви у низу су сложени, а њихов број $(n-1)$ може бити произвољно велики.

1.4.2 Хипотезе и контрапримери

За неке од хипотеза испоставило се да су нетачне, пошто су за њих пронађени контрапримери.

Пример. Један од изазова је проналажење таквог израза $f(n)$ чије вредности за сваки природни број n су прости бројеви. Занимљив пример је функција $f(n) = n^2 - n + 41$, чије су вредности прости бројеви за $n = 1, 2, \ldots, 40$; међутим, очигледно је да $f(41)$ није прост број, јер је дељив са 41.

Пример. Понекад проналажење контрапримера није једноставно. Својевремено је Ојлер поставио хипотезу да за $n \geq 3$ важи да збир $n-1$ n-тих степена не може бити једнак n-том степену природног броја. Међутим, 1966. године Ландер и Паркин су пронашли контрапример за $n = 5$:

$$144^5 = 27^5 + 84^5 + 110^5 + 133^5.$$

Контрапример за $n = 4$ пронашао је Елкиз 1988. године

$$95800^4 + 217519^4 + 414560^4 = 422481^4.$$

За $n \geq 6$ не зна се да ли је хипотеза тачна.

1.4.3 Примери нерешених хипотеза

Неке од отворених хипотеза одиграле су значајну улогу у развоју математике.

Последња Фермаова теорема За $n > 2$ и позитивне целе бројеве x, y и z, једначина $x^n + y^n = z^n$ нема целобројна решења. За $n = 2$ ова једначина има решења — *Питагорине тројке*, на пример $3^2 + 4^2 = 5^2$. За $n = 3$ тврђење је доказао Ојлер, а за $n = 4$ сам Ферма. Било је више погрешних доказа овог тврђења. Покушаји да се тврђење докаже довели су до развоја алгебарске теорије бројева. Ендрју Вајлс је коначно доказао ово тврђење (објављено 1995. године); при томе је морала да буде развијена теорија елиптичких кривих. За доказ овог тврђења добио је 2016. године Абелову награду.

Наводимо још неколико примера још увек отворених хипотеза.

Голдбахова хипотеза: Сваки парни број $n > 2$ може се изразити као збир два проста броја. За првих неколико парних бројева тврђење је тачно: $4 = 2+2$, $6 = 3+3$, $8 = 3+5$, $10 = 5+5$, $12 = 5+7$, $\ldots$. Уз помоћ рачунара тврђење је проверено за $n \leq 4 \cdot 10^{14}$. Рамаре је 1995. године доказао да је

сваки довољно велики природни број једнак збиру највише шест простих бројева.

Мерсенови прости бројеви: Као што смо видели, $2^n - 1$ може да буде прост број само ако је експонент n прост број. Мерсен је у 17. веку поставио хипотезу да простих бројева облика $2^p - 1$, где је p прост број, има бесконачно много. За $p = 2, 3, 5, 7$ то су прости бројеви $3, 7, 31, 127$. За $p = 11$ број $2^p - 1 = 2047 = 23 \cdot 89$ је сложен. Највећи познат Мерсенов прост број је 77232917 (50. по реду).

У вези са простим бројевима је и хипотеза да простих бројева облика $n^2 + 1$ има бесконачно много, као и да парова простих бројева *близанаца* (парова простих бројева који се разликују за два), као што су $5, 7$; $11, 13$; $29, 31$, има бесконачно много. Највећи познат пар простих близанаца је $2996863034895 \cdot 2^{1290000} \pm 1$.

Хипотеза "$3x + 1$" Нека је

$$f(x) = \begin{cases} x/2, & \text{ако је } n \text{ паран} \\ 3x + 1, & \text{ако је } n \text{ непаран} \end{cases}$$

Полазећи од неког броја n формира се низ n, $f(n)$, $f(f(n))$, $f(f(f(n)))$, ... На пример, за $n = 1$ добија се периодични низ $1, 4, 2, 1, 4, 2, 1, \ldots$, а за $n = 13$ низ $13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \ldots$ Хипотеза $3x + 1$ гласи: за било који природни број n овај низ раније или касније постаје периодичан, са периодом $1, 4, 2, 1, \ldots$. Дужина низа до прве јединице за $n = 27$ је 111, а за $n = 63728127$ је 949. Хипотеза је проверена за бројеве мање од $5.8 \cdot 10^{18}$.

1.5 Алгоритамски неразрешиви проблеми

Једна од најпознатијих теорема у математици гласи да постоје проблеми који се не могу решити алгоритмом. Постоји више доказа овог тврђења. Један од њих састоји се у томе да се покаже да не постоји алгоритам који решава *халтинг проблем* (проблем заустављања).

Халтинг проблем гласи: да ли постоји процедура (алгоритам) који за дати програм P и улаз U утврђује да ли се програм P са улазом U икада зауставља (завршава са радом)?

Ако би постојао такав алгоритам, он би се могао користити нпр. за исправљање грешака у програмима. Међутим, Алан Тјуринг је 1936. године доказао да такав алгоритам не може да постоји.

Треба запазити да се одговор на питање да ли се програм P зауставља на улазу U не може добити простим извршавањем програма P на улазу U: може се десити да се програм зауставља, али да ради јако дуго. Лако је направити пример програма чије извршавање траје 1000 година.

Доказ. Претпоставимо супротно, да постоји такав алгоритам H, који за произвољни програм P и улаз U после позива $H(P, U)$ завршава рад и штампа поруку "завршава се" или "заглављује".

Сваки програм, односно улаз, може се кодирати низом нула и јединица, па се они могу поређати у лексикографски уређени низ. Због тога се нпр. може извршити позив $H(P, P)$, где улогу улаза за програм P игра сам текст (бинарни) програма P. Нека је $K(P)$ такав програм (алгоритам) који позива $H(P, P)$, и

- завршава се ако $H(P, P)$ даје на излазу "заглављује"

- заглављује се (нпр. уласком у бесконачну петљу) ако $H(P, P)$ даје на излазу "завршава се"

Поставља се питање: да ли се $K(K)$ завршава или се заглављује?

- ако се $K(K)$ завршава, онда би $H(K, K)$ на излазу давало "заглављује", односно позив $K(K)$ би морао да доведе до заглављивања;
- ако се $K(K)$ заглављује, онда би $H(K, K)$ на излазу давало "завршава се", односно позив $K(K)$ би морао да се завршава.

Ова контрадикција показује да претпоставка да постоји алгоритам H са датим својствима није тачна.

Глава 2

Математичка индукција

2.1 Увод

Математичка индукција игра значајну улогу при конструкцији многих алгоритама. У даљем тексту размотрићемо неколико карактеристичних примера примене индукције.

Нека је $\mathbb{N} = \{1, 2, \ldots\}$ скуп природних бројева. Принцип математичке индукције може се формулисати на следећи начин. Претпоставимо да треба доказати да је тврђење $T(n)$ тачно за сваки природан број $n \in \mathbb{N}$. Уместо да се ово тврђење "напада" директно, довољно је доказати следећа два тврђења:

- $T(n)$ је тачно за $n = 1$, и
- за свако $n > 1$ важи: ако је тачно $T(n-1)$, онда је тачно и $T(n)$.

Ова чињеница је такозвани *принцип математичке индукције*, и последица је дефиниције скупа природних бројева: $1 \in \mathbb{N}$, а ако за неко $n > 1$ важи $n - 1 \in \mathbb{N}$ онда $n \in \mathbb{N}$.

У пракси се обично прво тврђење (база индукције) лако доказује. Доказ другог тврђења олакшан је претпоставком да је тачно $T(n-1)$, такозваном *индуктивном хипотезом*. Другим речима, довољно је тврђење свести на случај кад је n умањено за један. Илустроваћемо то једним једноставним примером.

Теорема 2.1. *За свака два броја $x, n \in \mathbb{N}$ важи $x - 1 | x^n - 1$.*

Доказ. Доказ се изводи индукцијом по n. За $n = 1$ тврђење гласи $x-1|x-1$ и очигледно је тачно. Свођење случаја n на случај $n-1$ заснива се на изразу $x^n - 1 = (x^{n-1} - 1)x + (x - 1)$, јер он показује да из индуктивне хипотезе $x - 1 | x^{n-1} - 1$ следи да је тачно $x - 1 | x^n - 1$.

Принцип математичке индукције често се користи у нешто промењеним облицима, који су непосредна последица основног облика.

Теорема 2.2. *Ако је тачно тврђење $P(1)$ и за свако $n > 1$ из претпоставке да је $P(k)$ тачно за свако $k < n$ следи тачност $P(n)$, онда је $P(n)$ тачно за свако $n \in \mathbb{N}$.*

Ово је тзв. потпуна индукција, а добија се од основне варијанте применом на тврђење $T(n) = \bigwedge_{k=1}^{n} P(k)$.

Теорема 2.3 (Регресивна индукција)**.** *Нека је $a_1, a_2, \ldots$ растући низ природних бројева. Ако је тврђење $P(n)$ тачно за $n = a_k$, $k = 1, 2, \ldots$ и за свако $n \geq 2$ из претпоставке да је тачно $P(n)$ следи да је тачно $P(n-1)$, тада је $P(n)$ тачно за свако $n \in \mathbb{N}$.*

Први део претпоставке, да је $P(a_k)$ тачно за $k \geq 1$ доказује се индукцијом ($P(1)$, и ако $P(a_k)$ онда $P(a_{k+1})$, $k \geq 1$). Дакле, најпре се доказује да постоје произвољно велики бројеви n такви да је $P(n)$ тачно, а онда да је $P(n)$ тачно и за све "изостављене" бројеве. Ово је тзв. регресивна индукција.

2.2 Два једноставна примера

Потребно је израчунати суму $S(n) = 1+2+\ldots+n$. Одговор на ово питање даје следећа теорема.

Теорема 2.4. *Нека је $S(n) = \sum_{i=1}^{n} i$. За свако $n \in \mathbb{N}$ је $S(n) = \frac{1}{2}n(n+1)$.*

Доказ. Како је $1 = \frac{1}{2} \cdot 1 \cdot 2$, тврђење је тачно за $n = 1$. Ако се претпостави да је оно тачно за неко n, онда је

$$S(n+1) = 1+2+\ldots+n+(n+1) = S(n)+n+1 = \frac{1}{2}n(n+1)+(n+1) = \frac{1}{2}(n+1)(n+2),$$

односно закључујемо да је тврђење тачно и за $n+1$.

Теорема 2.5. *Ако је $n \in \mathbb{N}$ и $1 + x > 0$ онда је $(1+x)^n \geq 1 + nx$.*

Доказ. За $n = 1$ тврђење је очигледно тачно. Ако је тврђење теореме тачно за неко n (индуктивна хипотеза), онда се множењем те неједнакости позитивним бројем добија:

$$(1+x)^{n+1} = (1+x)(1+x)^n \geq (1+x)(1+nx) = 1+(n+1)x+nx^2 \geq 1+(n+1)x,$$

односно тврђење теореме је тачно и ако се у њему n замени са $n+1$.

2.3 Бројање области на које је подељена раван

За неколико правих у равни каже се да су у општем положају ако никоје две нису паралелне, а никоје три се не секу у истој тачки. Треба одредити број области на које раван дели $n \geq 1$ правих у општем положају (претпоставка да су праве у општем положају уведена је да поједностави проблем, јер тада нема потребе за анализом разних могућих специјалних случајева).

Непосредно се види да једна, две, односно три праве у општем положају деле раван редом на две, четири, односно седам области. Наслућује се да укључивање n-те праве повећава за n број области на које је раван подељена. Ако се претпостави да је то тачно, онда је број области на које n правих у општем положају дели раван $2+2+3+4+\ldots+n = \frac{1}{2}n(n+1)+1$. Дакле, остаје да се докаже

Хипотеза 2.6. *Додавање n-те праве у равни (при чему су свих n правих у општем положају) повећава број области на које је раван подељена за n.*

Доказ. Доказаћемо ово тврђење индукцијом по n. Тврђење је очигледно тачно за $n = 1, 2, 3$. Претпоставимо да је оно тачно за неко n; нека је дато $n+1$ правих у равни у општем положају, и нека је p једна од тих правих. Због тога што су праве у општем положају, права p са произвољном облашћу, од оних на коју раван деле осталих n правих, нема заједничких тачака, или је сече. Права p сече свих осталих n правих (јер није паралелна ни са једном од њих; сече их у различитим тачкама, по претпоставци), и тачке пресека деле праву p на $n+1$ сегмената, од којих сваки лежи у различитој области. Сваку од тих области права p дели на две, те се додавањем праве p број области повећава за $n + 1$.

Примедба. Овде је индукција примењена два пута: повећање броја области додавањем n-те праве, а затим укупан број области на које раван дели n правих. Овакав поступак, понекад и са више нивоа, често се примењује.

2.4 Проблем са бојењем равни

Произвољних n правих у равни дели раван на области; за праве се не претпоставља да су у општем положају. Кажемо да је раван обојена са две боје (на пример црном и белом) ако је свака област обојена једном од те две боје, а сваке две суседне области (оне које имају део праве или целу праву као границу) су обојене различитим бојама.

Теорема 2.7. *Раван подељена са произвољних n правих може се обојити са две боје.*

Доказ. Тврђење теореме доказује се индукцијом по броју правих n. За $n = 1$ тврђење је очигледно тачно (раван је подељена на две области, једна од њих боји се црно, друга бело). Претпоставимо да се раван, издељена са произвољних $n - 1$ правих, може обојити са две боје. Додавање n-те праве на произвољан начин дели неке области на два суседна дела обојена истом бојом. Међутим, ако се свим областима са једне стране n-те праве боја промени у супротну, добија се исправно бојење равни. Заиста, границе између суседних области могу се поделити на три врсте: границе са једне, односно друге стране n-те праве, и границе на n-тој правој. Границе све три врсте одвајају области обојене различитим бојама. Дакле, полазећи од индуктивне хипотезе да се може обојити раван подељена се $n - 1$ правих, показано је како се добија бојење равни подељене са n произвољних правих.

Примедба. Овај пример је илустрација максималног, еластичног коришћења индуктивне хипотезе: од бојења области у једној полуравни направљено је друго исправно бојење заменом беле и црне боје.

2.5 Пример са сумирањем

Претпоставимо да су сви непарни бројеви исписани у оквиру шеме у облику троугла, тако да у i-тој врсти има i бројева, $i = 1, 2, \ldots$:

i									сума
1				1					1
2			3		5				8
3		7		9		11			27
4	13		15		17		19		64
				...					

Потребно је израчунати суму бројева у i-тој врсти.

Израчунавши суме у првих неколико врста, запажамо да је сума једнака кубу редног броја врсте. Да се ово тврђење докаже индукцијом, довољно је доказати да је разлика збирова у $(i+1)$-ој и i-тој врсти једнака $(i+1)^3 - i^3 = 3i^2 + 3i + 1$. Даље, ако са a_i означимо последњи број у i-тој врсти, разлика збирова у $(i+1)$-ој и i-тој врсти једнака је $i \cdot 2i + a_{i+1}$ јер је сваки од првих i бројева у $(i+1)$-ој врсти већи од одговарајућег у i-тој врсти за $2i$ (пошто у i-тој врсти има i узастопних непарних бројева, разлика првог у $(i+1)$-ој врсти и првог у i-тој врсти је $2i$). Дакле, да би се доказало друго тврђење, довољно је доказати да је $2i^2 + a_{i+1} = 3i^2 + 3i + 1$, односно $a_{i+1} = i^2 + 3i + 1 = (i+1)^2 + (i+1) - 1$, или $a_i = i^2 + i - 1$. Ово последње тврђење ћемо наравно доказати индукцијом. За $i = 1$ је $a_1 = 1 = 1^2 + 1 - 1$, тј. тврђење је тачно. Из претпоставке да је за неко i $a_i = i^2 + i - 1$ следи (јер у $(i+1)$-ој врсти има $i+1$ узастопних непарних бројева) да је $a_{i+1} = a_i + 2(i+1) = i^2 + 3i - 1 = (i+1)^2 + (i+1) - 1$, чиме је доказано последње помоћно тврђење, а самим тим и полазно тврђење.

У следећем примеру видећемо да се у доказу индуктивна хипотеза мора користити еластично, а не онако како би на први поглед изгледало да треба учинити.

Теорема 2.8. *За сваки природан број n је $\frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n} < 1$.*

Доказ. За $n = 1$ је ово очигледно тачно. Неједнакост

$$\left(\frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n}\right) + \frac{1}{2^{n+1}} < 1$$

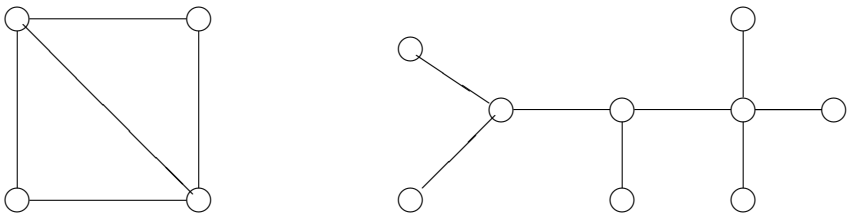
не следи из претходне, јер ако је збир у загради мањи од 1, он после повећавања за $\frac{1}{2^{n+1}}$ не мора да *остане* мањи од 1. Проблеми се избегавају ако се индуктивна хипотеза примени на последњих n сабирака у овој неједнакости:

$$\frac{1}{2} + \left(\frac{1}{4} + \ldots + \frac{1}{2^n} + \frac{1}{2^{n+1}}\right) = \frac{1}{2} + \frac{1}{2}\left(\frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n}\right) < \frac{1}{2} + \frac{1}{2} \cdot 1 = 1.$$

2.6 Ојлерова формула

Сада ћемо доказати тврђење познато као Ојлерова формула (L. Euler, 1707-1783.). Посматрајмо повезану планарну мапу (повезани неусмерени граф

који се може нацртати у равни тако да његове гране као заједничке тачке имају само чворове графа). Бројеве чворова, грана и области на које мапа дели раван (укључујући и спољашњу бесконачну област) означимо редом са V, E, F. Кажемо да је мапа повезана ако за свака два њена чвора постоји низ грана које их повезују. Специјално, повезана планарна мапа са једном облашћу ($F = 1$) је **стабло**. На слици 2.1 приказани су примери мапе (са $V = 4$ чвора, $E = 5$ грана и $F = 3$ области), односно стабла.



Слика 2.1: Пример планарне мапе, односно стабла.

Теорема 2.9 (Ојлерова формула)**.** *За повезану планарну мапу важи једнакост* $V + F = E + 2$.

Доказ. Доказ ће бити изведен двоструком индукцијом: најпре по броју области F, а у оквиру тога по броју чворова V. Посматрајмо најпре случај $F = 1$. Тада мапа (у овом случају стабло) не садржи ни једну затворену област, јер би у противном због постојања спољашње бесконачне области било $F \geq 2$. Треба доказати да тада важи $V = E + 1$. Ово тврђење доказаћемо индукцијом по броју чворова $V = n$. Ако је $n = 1$ онда нема грана и тврђење је тачно. Нека је тврђење тачно за $V = n$. Посматрајмо произвољно стабло са $n+1$ чворова. То стабло мора да има бар један чвор v са само једним суседним чвором, јер би у противном мапа садржала затворену област. Мапа, која се од ове добија уклањањем чвора v и гране e која га повезује са суседним чвором, повезана је и нема затворених области, а број чворова у њој је n. За ту мапу важи индуктивна хипотеза: $(V - 1) = (E - 1) + 1$, па је $V = E + 1$, чиме је завршен доказ теореме за случај $F = 1$.

Претпоставимо да је тврђење теореме тачно за планарне мапе са $F = n$ области, $V + F = E + 2$. Произвољна мапа G са $F' = n + 1$ области има бар једну област која је суседна са спољашњом облашћу. Избацивањем из мапе неке од грана које одвајају ту област од спољашње, добија се мапа G' која је и даље повезана и планарна, при чему она има $F' = F - 1$ области и $E' = E - 1$ грана. На основу индуктивне хипотезе за нову мапу важи $V + (F - 1) = (E - 1) + 2$, из чега непосредно следи да за полазну мапу важи тврђење теореме.

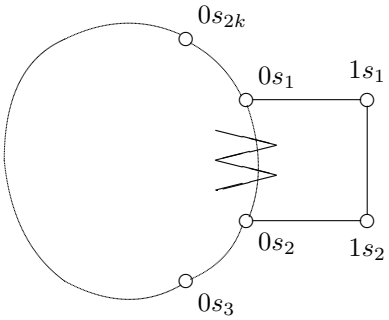
2.7 Грејови кодови

Грејов код (k-битни) дужине $n > 1$ је низ k-торки битова ($k \geq 1$) $s_1, s_2, \ldots, s_n$ таквих да се у том низу свака два узастопна члана, као и s_1, s_n, разликују на тачно једној позицији. На пример, $00, 01, 11, 10$ је Грејов двобитни код

дужине 4. Због ове своје особине Грејови кодови имају широку примену. Лако је видети да не постоје Грејови кодови непарне дужине. Заиста, s_1 и s_i се за $i > 1$ разликују на парном, односно непарном броју позиција ако је i непарно, односно парно. Због тога се s_1 и s_n за непарно n разликују на парном, (дакле различитом од један) броју позиција.

Теорема 2.10. *За сваки паран број n постоји Грејов код дужине n.*

Доказ. Грејов код дужине два је, на пример, низ $0, 1$. Ако постоји Грејов код $s_1, s_2, \ldots, s_n$ дужине $n = 2k$ за неко $k \in \mathbb{N}$ (односно s_i, s_{i+1}, $i = 1, 2, \ldots, n-1$, као и s_n, s_1, разликују се на тачно једној позицији), онда је јасно да је $0s_1, 1s_1, 1s_2, 0s_2, 0s_3, 0s_4, \ldots, 0s_n$ Грејов код дужине $n+2$, видети слику 2.2. Овде су са $0s_i$ односно $1s_i$ означени блокови битова добијени од s_i додавањем нуле, односно јединице са леве стране.



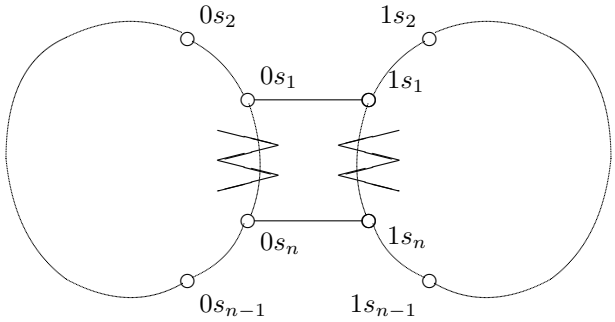
Слика 2.2: Продужавање Грејовог кода за два

На описани начин полазећи од 1-битног Грејовог кода $0, 1$ добијају се двобитни Грејов код $00, 10, 11, 01$ и тробитни Грејов код $000, 100, 110, 010, 011, 001$. Описана конструкција Грејових кодова парне дужине има важан недостатак: за код дужине $2k$ користе се k-торке битова, тј. дужине блокова битова повећавају се за један сваки пут кад се дужина кода повећа за два. С друге стране, ако не би постојао захтев о односу суседних чланова низа, низ дужине $n > 1$ могао би се кодирати k-торкама битова, где је $k = \lceil \log_2 n \rceil$; са $\lceil x \rceil$ означен најмањи цео број већи или једнак од реалног броја x. Нерационалност описане конструкције може се исправити тако што би се повећавањем дужине блокова за један конструисао два пута дужи, а не Грејов код дужи за два:

$$0s_1, 0s_2, \ldots, 0s_n, 1s_n, 1s_{n-1}, \ldots, 1s_1, \tag{2.1}$$

видети слику 2.3. Специјално, овако се полазећи од једнобитног Грјовог кода $0, 1$ добијају тзв. бинарни рефлектовани Грејови кодови $00, 01, 11, 10$; $000, 001, 011, 010, 110, 111, 101, 100$; ...

Назовимо низ блокова битова $s_1, s_2, \ldots, s_n$ — код кога се суседни блокови битова s_i, s_{i+1} за свако $i = 1, 2, \ldots, n-1$ разликују на тачно једној позицији — *затвореним*, односно *отвореним* Грејовим кодом ако јесте, односно није испуњен услов да се блокови s_1, s_n разликују на тачно једној позицији. Доказ следеће теореме даје конструкцију Грејовог кода са минималном дужином блокова битова.

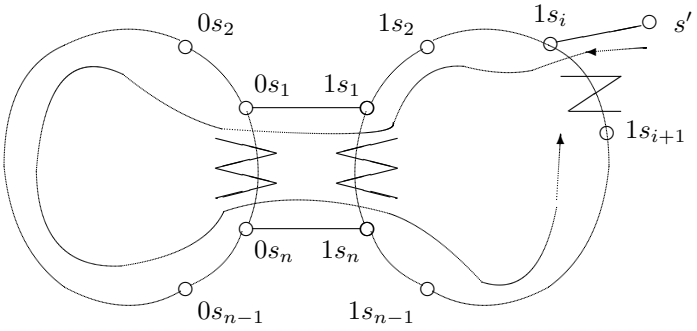


Слика 2.3: Удвостручавање дужине Грејовог кода

Теорема 2.11. *За свако $n > 1$ постоји Грејов код дужине n са $\lceil \log_2 n \rceil$ бита, и то затворен ако је n парно, односно отворен ако је n непарно.*

Доказ. Очигледно је да постоје Грејови кодови дужине један и два са блоковима дужине један. Претпоставимо да постоје отворени, односно затворени Грејови кодови свих дужина $k < n$, са величином блока $\lfloor \log_2 k \rfloor$.

- Ако је n парно, онда се применом конструкције (2.1) (слика 2.3) на код дужине $n/2$ са $\lceil \log_2(n/2) \rceil$ бита (који по индуктивној хипотези постоји), добија код дужине n са $1 + \lceil \log_2(n/2) \rceil = \lceil \log_2 n \rceil$ бита.
- Нека је сад $n = 2m+1$ непарно. Према индуктивној хипотези постоји Грејов код дужине $m+1$ са $\lceil \log_2(m+1) \rceil$ бита. Од њега се на исти начин добија код дужине $2m+2$ са $d = 1 + \lceil \log_2 m \rceil = \lceil \log_2(2m+2) \rceil$ бита. Избацивањем рецимо последњег блока, добија се отворени код дужине $2m+1 = n$. Број бита у блоковима овог кода је
  $\lceil \log_2(2m+2) \rceil = \lceil \log_2(2m+1) \rceil = \lceil \log_2 n \rceil$.



Слика 2.4: Конструкција отвореног Грејовог кода

2.8 Неједнакост између аритметичке и геометријске средине

Као пример тврђења које се може доказати применом регресивне индукције искористићемо неједнакост између аритметичке и геометријске средине.

Теорема 2.12. *Нека су $x_1, x_2, \ldots, x_n$ позитивни реални бројеви. Тада је*

$$\sqrt[n]{x_1 x_2 \ldots x_n} \leq \frac{x_1 + x_2 + \cdots + x_n}{n} \tag{2.2}$$

Доказ. Тврђење ћемо доказати применом регресивне индукције. Најпре ћемо доказати да оно важи за бројеве облика $n = 2^k$, дакле за произвољно велике бројеве, а затим да из претпоставке да оно важи за неко n следи да оно важи за $n - 1$.

У првом делу доказа база индукције ($n = 2^1$) је неједнакост

$$\sqrt{x_1 x_2} \leq \frac{1}{2}(x_1 + x_2),$$

која је последица очигледне неједнакости $(x_1 - x_2)^2 \geq 0$ (одакле се добија $x_1^2 + 2x_1x_2 + x_2^2 \geq 4x_1x_2$, односно $\frac{1}{4}(x_1 + x_2)^2 \geq x_1x_2$). Из претпоставке да (2.2) важи за $n = 2^k$ следи да за $2n = 2^{k+1}$ бројева $x_1, x_2, \ldots, x_{2n}$ важи

$$\sqrt[2n]{\prod_{i=1}^{2n} x_i} = \sqrt{\sqrt[n]{\prod_{i=1}^{n} x_i}\sqrt[n]{\prod_{i=n+1}^{2n} x_i}} \leq \sqrt{\Big(\frac{1}{n}\sum_{i=1}^{n} x_i\Big)\Big(\frac{1}{n}\sum_{i=n+1}^{2n} x_i\Big)} \leq$$
$$\leq \frac{1}{2}\Big(\frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n}\sum_{i=n+1}^{2n} x_i\Big) = \frac{1}{2n}\sum_{i=1}^{2n} x_i,$$

односно тврђење (2.2) важи за двоструко већи број бројева x_i. Тиме је завршен први део доказа.

Сада ћемо показати да ако једнакост (2.2) важи за произвољних n бројева, онда она важи и за произвољних $n - 1$ бројева. Да би се у (2.2) смањио број променљивих, једна од њих може се изразити преко осталих. Нека је дакле $x_n = \frac{1}{n-1}\sum_{i=1}^{n-1} x_i$, тј. x_n бирамо тако да буде једнако аритметичкој средини осталих $n - 1$ променљивих. Тада десна страна неједнакости (2.2) постаје једнака x_n:

$$\frac{1}{n}\sum_{i=1}^{n} x_i = \frac{n-1}{n}\Big(\frac{1}{n-1}\sum_{i=1}^{n-1} x_i\Big) + \frac{1}{n}x_n = \frac{n-1}{n}x_n + \frac{1}{n}x_n = x_n,$$

па (2.2) прелази у $x_n \prod_{i=1}^{n-1} x_i \leq x_n^n$, односно

$$\sqrt[n-1]{\prod_{i=1}^{n-1} x_i} \leq x_n = \frac{1}{n-1}\sum_{i=1}^{n-1} x_i.$$

Према томе, завршетком другог дела доказа доказано је да неједнакост (2.2) између аритметичке и геометријске средине важи за свако $n \in \mathbb{N}$.

2.9 Инваријанта петље — доказ исправности алгоритма

Математичка индукција често се користи за доказивање коректности алгоритма, посебно у случају кад се алгоритам састоји од петље. Идеја је да се покаже да је неко тврђење (*инваријанта петље*) тачно пре уласка

у петљу и после сваког проласка кроз петљу, а да из његове тачности на крају извршења алгоритма следи коректност алгоритма.

Овај метод илустроваћемо доказом коректности алгоритма који за задати природни број n израчунава његове бинарне цифре, компоненте вектора b, тј. врши његову конверзију у бинарни облик. Алгоритам се састоји од петље у којој су три наредбе: повећавање k (индекса за вектор b), израчунавање остатка и целог дела количника при дељењу помоћне променљиве t са два.

$Bin\_cifre(n)$
улаз: n (природан број)
излаз: b (низ бинарних цифара броја n)
1 $t \leftarrow n$ { помоћна променљива t са почетном вредношћу n }
2 $k \leftarrow 0$
3 **while** $t > 0$ **do**
4 $\quad k \leftarrow k + 1$
5 $\quad b[k] \leftarrow t \bmod 2$
6 $\quad t \leftarrow t \operatorname{div} 2$

Теорема 2.13. *По завршетку алгоритма Bin_cifre у вектору b налазе се бинарне цифре броја n.*

Доказ. Доказ исправности алгоритма биће спроведен помоћу инваријанте петље. Размотримо следећу индуктивну хипотезу.

Индуктивна хипотеза. Нека је m број чије су бинарне цифре добијене у k првих пролазака кроз петљу, при чему је $b[1]$ најнижи бит. Тада је

$$t \cdot 2^k + m = n.$$

За $k = 0$ је $t = n$ и $m = 0$, па је ово тврђење тачно. Нека је индуктивна хипотеза тачна за неко $k \geq 0$. У наредном проласку кроз петљу добијају се нове вредности $b[k+1] = t \bmod 2$, $t' = t \operatorname{div} 2$, и $m' = b[k+1]2^k + m = (t \bmod 2)2^k + m$, па је

$$\begin{aligned} t'2^{k+1} + m' &= (t \operatorname{div} 2)2^{k+1} + (t \bmod 2)2^k + m = \\ &= 2^k\big(2 \cdot (t \operatorname{div} 2) + (t \bmod 2)\big) + m = 2^k \cdot t + m = n, \end{aligned}$$

због индуктивне хипотезе. Тиме је доказано да израз $t \cdot 2^k + m$ не мења вредност од проласка до проласка кроз петљу, односно да је он инваријанта петље. Како је после последњег проласка кроз петљу $t = 0$, у том тренутку ће бити $m = n$, чиме је исправност алгоритма доказана.

2.10 Најчешће грешке

Приликом извођења доказа индукцијом честа је грешка да се као очигледна чињеница искористи тврђење које је блиско или еквивалентно доказиваном. Тиме се фактички појачава индуктивна хипотеза, али се не доказује да из тачности појачане хипотезе за n следи њена тачност за $n + 1$.

Размотримо још један пример "доказа" индукцијом.

Теорема 2.14 (нетачна)**.** *Дато је n правих у равни тако да никоје две међу њима нису паралелне. Доказати да свих n правих имају заједничку тачку.*

Доказ. (погрешан) За $n = 1$ тврђење је тачно. Шта више, оно је тачно и за $n = 2$. Нека оно важи за неко n и нека је дато произвољних $n + 1$ правих у равни таквих да никоје две међу њима нису паралелне. Тада према индуктивној хипотези првих n правих имају заједничку тачку. Исто важи и за последњих n правих. Ове две тачке се морају поклапати, што значи да свих $n + 1$ правих имају заједничку тачку.

Грешка у овом доказу прикривена је у чињеници да прелаз са n на $n+1$ не важи за $n = 2$: од три разматране праве a, b и c пресечна тачка првих двеју, a и b, не мора да се поклапа са пресечном тачком последњих двеју, b и c.

2.11 Резиме

Математичка индукција је моћан метод за доказивање теорема. Најпре се формулише индуктивна хипотеза и бира се променљива по којој ће доказ индукцијом бити изведен. Понекад се индуктивна хипотеза може формулисати на више начина — неки воде краћем, а неки дужем доказу; понекад се уводи потпуно нова променљива, потребна само за извођење доказа. Доказ се понекад изводи индукцијом на више нивоа, од којих сваки води све ближе циљу.

Доказ индукцијом састоји се од два дела — базе и корака индукције (редукције, свођења). Базу понекад није лако доказати, али је то ипак ређи случај. Због тога се овај део доказа понекад грешком игнорише. Срж доказа је у редукцији: најчешће се тврђење са параметром n своди на аналогно тврђење са параметром $n-1$. Друга могућа варијанта је потпуна индукција, кад се случај n своди на неколико других за вредности $k < n$. Такође се користи и поступак свођења са $2n$ на n, а затим са n на $n + 1$ – регресивна индукција. У свим варијантама је важно да се о доказиваном тврђењу не претпостави ништа више од индуктивне хипотезе.

Глава 3

Анализа алгоритама

Подсетимо се да је циљ анализе алгоритма је да се предвиди понашање алгоритма, посебно брзина извршавања, не реализујући га на неком конкретном рачунару. Практично је недостижно одредити време извршавања алгоритма за одређени улаз јер је тешко узети у обзир све могуће улазе. Због тога се за сваки могући улаз дефинише његова величина n, после чега се у анализи користи само овај податак. Дакле, анализа алгоритма треба да као резултат дâ израз за утрошено време у зависности од величине улаза n. Који међу улазима величине n изабрати као репрезентативан? Често се за ову сврху бира најгори случај. Најбољи улаз је често тривијалан, а средњи (просечан) улаз само на први поглед изгледа као добар избор. Међутим, анализа просечног случаја је често компликована, а потешкоћу представља и дефинисање расподеле вероватноћа на скупу свих улаза величине n, и то тако да она одговара ситуацијама које се појављују у пракси. Због тога се обично анализа врши за најгори случај. С обзиром на то да је понекад корисно извршити анализу просечног слуцаја, у тачки 3.4 ћемо видети и пример такве анализе алгоритма.

Оцена (временске) сложености алгоритма састоји се у бројању рачунских корака које треба извршити. Међутим, термин рачунска операција може да подразумева различите операције, на пример сабирање и множење, чије извршавање траје различито време. Различите операције се могу посебно бројати, али је то обично компликовано. Поред тога, време извршавања зависи и од конкретног рачунара, изабраног програмског језика, односно преводиоца. Зато се обично у оквиру алгоритма издваја неки *основни корак*, онај који се најчешће понавља. Тако, ако се ради о сортирању, основни корак је упоређивање.

Под просторном сложеношћу алгоритма подразумева се величина меморије потребне за за извршавање алгоритма, при чему се простор за смештање улазних података најчешће не рачуна. То омогућује упоређивање различитих алгоритама за решавање истог проблема. Као и код временске сложености, и за просторну сложеност се најчешће тражи њено асимптотско понашање у најгорем случају, за велике величине улаза. Просторна сложеност $O(n)$ значи да је за извршавање алгоритма потребна меморија пропорционална оној за смештање улазних података. Ако је пак просторна сложеност алгоритма $O(1)$, онда то значи да је потребан меморијски простор

за његово извршавање ограничен константом, без обзира на величину улаза.

3.1 Асимптотске ознаке

Ознака O: Нека су f и g позитивне функције од аргумента n из скупа **N** природних бројева. Каже се да је $g(n) = O(f(n))$ ако постоје позитивне константе c и N, такве да за свако $n > N$ важи $g(n) \leq cf(n)$. Ознака $O(f(n))$ се у ствари односи на *класу функција*, а једнакост $g(n) = O(f(n))$ је уобичајена ознака за *инклузију* $g(n) \in O(f(n))$. Јасно је да је функција f само нека врста **горње границе** за функцију g. На пример, поред једнакости $5n^2+15 = O(n^2)$ (јер је $5n^2+15 \leq 6n^2$ за $n \geq 4$) важи и једнакост $5n^2 + 15 = O(n^3)$ (јер је $5n^2 + 15 \leq 6n^3$ за $n \geq 4$). Ова нотација омогућује игнорисање мултипликативних константи: уместо $O(5n + 4)$ може се писати $O(n)$. Једнакост $O(5n + 4) = O(n)$ је у ствари једнакост две класе функција.

Лако се показује да се O-изрази могу сабирати и множити:

$$\begin{aligned} O(f(n)) + O(g(n)) &= O(f(n) + g(n)), \\ O(f(n))O(g(n)) &= O(f(n)g(n)). \end{aligned}$$

Ознаке Ω и Θ: Други проблем у вези са сложеношћу алгоритма је питање доње границе за потребан број рачунских операција. Док се горња граница односи на *конкретан алгоритам*, доња граница сложености односи се на *произвољан алгоритам из неке одређене класе*. Због тога оцена доње границе захтева посебне поступке анализе. За функцију $g(n)$ каже се да је **асимптотска доња граница** функције $T(n)$ и пише се $T(n) = \Omega(g(n))$, ако постоје позитивне константе c и N, такве да за свако $n > N$ важи $T(n) > cg(n)$. Тако је на пример $n^2 = \Omega(n^2 - 100)$, и $n = \Omega(n^{0.9})$. Види се да симбол Ω одговара релацији $\geq$.

Ако за две функције $f(n)$ и $g(n)$ истовремено важи и $f(n) = O(g(n))$ и $f(n) = \Omega(g(n))$, онда оне имају исте асимптотске брзине раста, што се означава са $f(n) = \Theta(g(n))$. Тако је на пример $5n \log_2 n - 10 = \Theta(n \log n)$, при чему је у последњем изразу основа логаритма небитна.

Ознака o: Кажемо да је $f(n) = o(g(n))$ ако важи $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. Неформално, $f(n)$ је бесконачно мала величина у односу на $g(n)$. На пример, очигледно је $\frac{n}{\log_2 n} = o(n)$, а једнакост $\frac{n}{10} = o(n)$ није тачна.

Ознака ω: Према аналогији, ω се односи према Ω као o према O. Другим речима, $f(n) \in \omega(g(n))$ акко $g(n) \in o(f(n))$. Из релације $f(n) = \omega(g(n))$ следи да је $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$. Неформално, $g(n)$ је бесконачно мала величина у односу на $f(n)$. На пример, важи $n^2/2 = \omega(n)$, али не важи $n^2/2 = \omega(n^2)$.

Подсетимо се и решавања проблема сумирања коришћењем интеграла. Ако се алгоритам састоји од делова који се извршавају један за другим, онда је његова сложеност једнака збиру сложености делова. На пример,

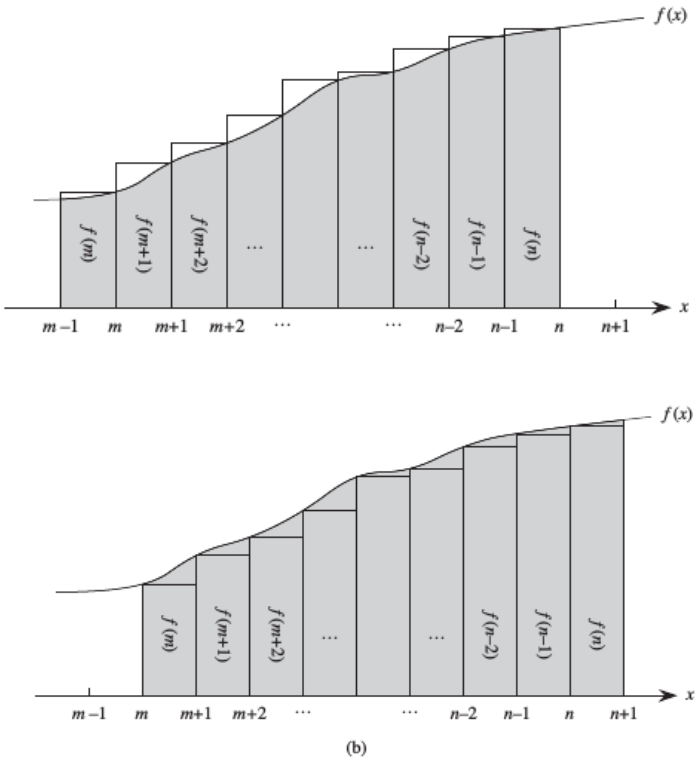
ако је основни део алгоритма петља у којој индекс i узима вредности $1, 2, \ldots, n$, а извршавање i-тог проласка кроз петљу троши $f(i)$ корака, онда је временска сложеност алгоритма $\sum_{i=1}^{n} f(i)$. У случају кад је тешко израчунати суму $\sum_{i=1}^{n} f(i)$, а функција $f(x)$ од реалног аргумента је монотоно неопадајућа непрекидна функција за $x \geq 1$, тада се сумирањем левих страна за $i = 1, 2, \ldots, n$ а десних страна за $i = 1, 2, \ldots, n-1$ неједнакости:

$$f(i) \leq \int_{i}^{i+1} f(x)\,dx \leq f(i+1), \quad 1 \leq i \leq n,$$

добијају границе интервала у коме лежи сума $\sum_{i=1}^{n} f(i)$:

$$\int_{1}^{n} f(x)\,dx + f(1) \leq \sum_{i=1}^{n} f(i) \leq \int_{1}^{n+1} f(x)\,dx.$$

видети слику 3.1.



Слика 3.1: Апроксимација суме $\sum_{k=m}^{n} f(k)$ интегралима. Укупна површина свих правоугаоника представља вредност суме, док је вредност интеграла једанака осенченој површини испод криве. Обратити пажњу да је хоризонтална страница сваког од правоугаоника дужине 1.

Ако је пак функција $f(x)$ монотоно нерастућа, онда је функција $-f(x)$ неопадајућа, па се применом ових неједнакости на $-f(x)$ добија да у овим неједнакостима само треба променити знаке "$\leq$" у "$\geq$".

Пример: Размотримо проблем рачунања суме $S_k(n) = \sum_{i=1}^{n} i^k$ за $k > 0$. У овом случају функција $f(x) = x^k$ је монотоно неопадајућа те можемо користити наредну процену:

$$\int_1^n f(x)\,dx + f(1) \leq \sum_{i=1}^{n} f(i) \leq \int_1^{n+1} f(x)\,dx$$

односно:

$$\int_1^n x^k\,dx + 1 \leq \sum_{i=1}^{n} i^k \leq \int_1^{n+1} x^k\,dx$$

Вредност неодређеног интеграла је:

$$\int x^k\,dx = \frac{x^{k+1}}{k+1} + C$$

с обзиром да је $k \neq -1$. Одавде за $k \neq -1$ добијамо наредну оцену тражене суме:

$$\frac{n^{k+1}-1}{k+1} + 1 \leq \sum_{i=1}^{n} i^k \leq \frac{(n+1)^{k+1}-1}{k+1}$$

односно:

$$\frac{n^{k+1}}{k+1} \leq \frac{n^{k+1}+k}{k+1} \leq \sum_{i=1}^{n} i^k \leq \frac{(n+1)^{k+1}-1}{k+1}$$

тј:

$$S_k(n) = \sum_{i=1}^{n} i^k = \Theta(n^{k+1})$$

Уколико је $k < 0$ функција $f(x) = x^k$ је монотоно нерастућа функција па за $k \neq -1$ важи наредна оцена:

$$\frac{(n+1)^{k+1}-1}{k+1} \leq \sum_{i=1}^{n} i^k \leq \frac{n^{k+1}}{k+1}$$

У специјалном случају када је $k = -1$ вредност неодређеног интеграла је:

$$\int \frac{1}{x}\,dx = \ln x + C$$

те важи наредна оцена:

$$\ln x|_1^{n+1} \leq \sum_{i=1}^{n} \frac{1}{i} \leq \ln x|_1^n + 1,$$

односно:

$$\sum_{i=1}^{n} \frac{1}{i} = \Theta(\ln n)$$

3.2 Решавање рекурентних једначина методом замене

Рекурентне једначине у вези са алгоритмима заснованим на разлагању обично се решавају *методом замене (супституције)*, који се састоји од два корака:

1. претпоставити облик решења
2. применом математичке индукције доказати тачност решења, пошто се претходно на одговарајући начин изаберу вредности константи

Име метода потиче од тога што се претпостављено решење замењује уместо вредности функције за мање аргументе. Метод је моћан, али морамо бити у стању да претпоставимо облик решења. Метод се може искористити за одређивање било горње, било доње границе за решење рекурентне релације.

Пример. Потребно је одредити горњу границу за решење рекурентне релације:

$$T(n) = 2T(\left\lfloor \frac{n}{2} \right\rfloor) + n \tag{3.1}$$

Може се претпоставити да је решење облика $T(n) = O(n \log n)$. Метод замене захтева да докажемо неједнакост $T(n) \leq cn \log_2 n$, ако се на одговарајући начин изабере вредност константе $c > 0$. Полазимо од претпоставке да је неједнакост тачна за све $m < n$ и специјално за $m = \lfloor n/2 \rfloor$, тј. да је $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \log_2(\lfloor n/2 \rfloor)$. Заменом у рекурентну једначину (3.1) добија се:

$$\begin{aligned} T(n) &\leq 2c\lfloor n/2 \rfloor \log_2(\lfloor n/2 \rfloor) + n \\ &\leq cn \log_2(n/2) + n = cn \log n - cn + n \\ &\leq cn \log_2 n \end{aligned}$$

Да би последња од ових неједнакости била тачна, потребно је и довољно да буде $c \geq 1$. За комплетирање овог доказа математичком индукцијом неопходно је доказати да неједнакост важи у базном случају, под претпоставком да је изабрана довољно велика вредност константе c. Међутим, ако је $T(1) = 1$, онда за $n = 1$ неједнакост $T(n) \leq cn \log_2 n$ не важи ни за једно c. Проблем се може решити тако што докажемо да је неједнакост тачна за $n \geq n_0$, где је n_0 константа чију вредност треба да изаберемо. Пре свега, примећујемо да се вредност $T(1)$ не појављује са десне стране неједнакости (3.1) за $n > 3$. Због тога се уместо $T(1)$ за базне случајеве могу узети $T(2)$ и $T(3)$, а за n_0 може се узети $n_0 = 2$. Приметимо да смо тиме раздвојили базу рекурентне релације ($n = 1$) и базу индуктивног доказа ($n = 2$ и $n = 3$). Полазећи од $T(1) = 1$, на основу (3.1) добијамо $T(2) = 4$ и $T(3) = 5$. Индуктивни доказ неједнакости $T(n) \leq cn \log_2 n$ завршава се избором довољно велике константе c тако да буде $T(2) \leq c \cdot 2 \cdot \log_2 2$ и $T(3) \leq c \cdot 3 \cdot \log_2 3$. Испоставља се да су обе ове неједнакости задовољене ако је $c \geq 2$ (јер је $4 \leq 2 \cdot 2$ и $5 \leq 2 \cdot 3 \cdot \log_2 3$). У великом броју случајева индуктивни доказ се лако комплетира тако да покрива и случај малих вредности n, па се у наредним примерима ти детаљи често прескачу.

3.2.1 Избор добре претпоставке

На жалост, не постоји општи метод за погађање тачног облика решења рекурентне релације. До добрих претпоставки може се доћи коришћењем неких хеуристика или применом стабла рекурзије, које ће бити размотрено у наредном одељку.

Ако је рекурентна релација слична претходној, онда се може искористити слична претпоставка. На пример, рекурентна релација $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ изгледа тешко због додавања сабирка "17". Интуитивно, ипак овај сабирак не може суштински да промени облик решења: када је n велико разлика између $\lfloor n/2 \rfloor + 17$ и $\lfloor n/2 \rfloor$ није велика и обе ове вредности су близу $n/2$. Према томе, може се претпоставити да је $T(n) = O(n \log n)$, што се може доказати применом метода замене (видети задатак у наставку).

Задатак: Доказати да је за решење једначине $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ важи $T(n) = O(n \log n)$.

Као прво, ова једначина почиње да има смисла тек за $n \geq 35$. Могући начин да се изведе доказ је да се примени општији облик горње границе, са две константе c и a: $T(n) \leq c(n-a)\log(n-a)$. Дакле претпоставимо да за $k < n$ важи $T(k) \leq c(k-a)\log(k-a)$ и покажимо да онда важи и $T(n) \leq c(n-a)\log(n-a)$.

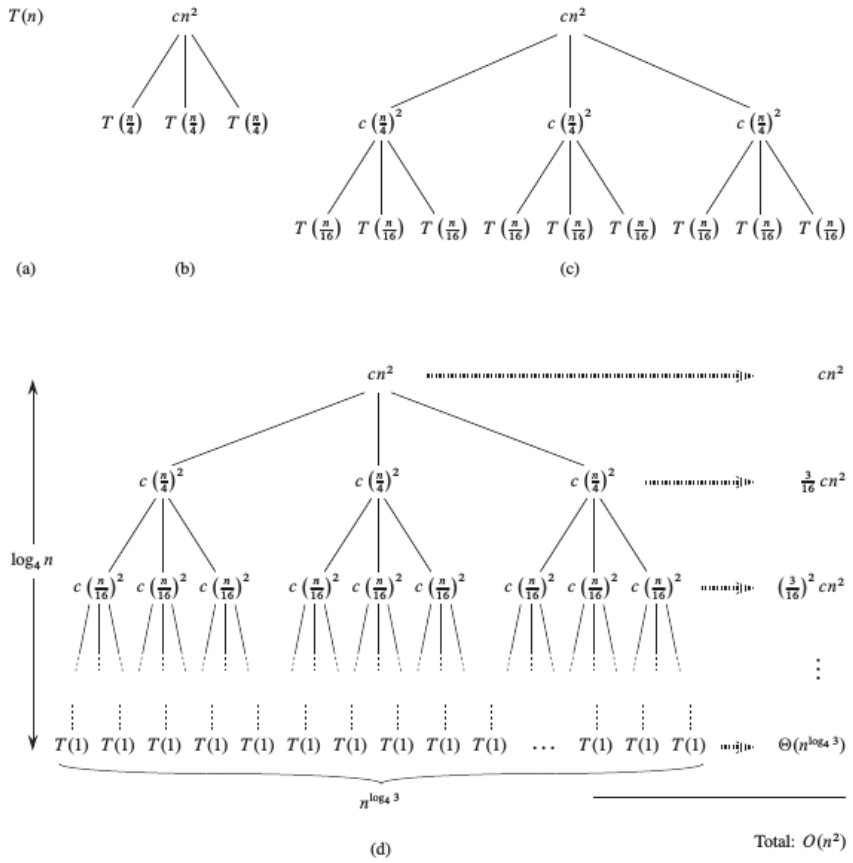
$$\begin{aligned}
T(n) &\leq 2c(\lfloor n/2 \rfloor + 17 - a)\log_2(\lfloor n/2 \rfloor + 17 - a) + n \\
&\leq 2c(n/2 + 17 - a)\log_2(n/2 + 17 - a) + n \\
&\leq c(n + 34 - 2a)\log_2 \frac{n + 34 - 2a}{2} + n \\
&\leq c(n + 34 - 2a)\log(n + 34 - 2a) - c(n + 34 - 2a) + n \\
&\leq c(n + 34 - 2a)\log(n + 34 - 2a) \\
&\leq c(n - a)\log(n - a)
\end{aligned}$$

Последње две неједнакости важе за $c > 1$ и $a \geq 34$.

Други начин да се дође до добре претпоставке је да се докаже да важе "комотна" доња и горња граница, па да се затим смањује разлика између њих. На пример, за рекурентну релацију (3.1) може се поћи од очигледне доње границе $T(n) = \Omega(n)$ (због сабирка n у рекурентној релацији) и може се доказати да важи груба горња граница $T(n) = O(n^2)$. Затим ми можемо да постепено смањујемо горњу границу и постепено повећавамо доњу границу, док не дођемо до решења $T(n) = O(n \log n)$.

3.3 Решавање рекурентних релација методом стабла рекурзије

Иако се метод замене може искористити за доказ да је решење рекурентне релације тачно, до проблема може доћи приликом избора претпоставке о облику решења. Цртање стабла рекурзије, као приликом анализе сортирања обједињавањем (са почетка курса АСП) је начин да се дође до добре претпоставке. У стаблу рекурзије сваки чвор представља цену једног потпроблема унутар скупа рекурзивних позива функције. Сабирањем цена на сваком нивоу добијају се цене за појединачне нивое рекурзије, а затим се њиховим сабирањем добија укупна цена на свим нивоима рекурзије.



Слика 3.2: Конструкција стабла рекурзије за рекурентну релацију $T(n) = 3T(n/4) + cn^2$. Део (a) приказује $T(n)$, што се поступно развија у деловима (b), (c), (d) до комплетног стабла рекурзије. Потпуно развијено стабло рекурзије у делу (d) има висину $\log_4 n$, односно $\log_4 n + 1$ нивоа.

Најбоље је користити стабло рекурзије за одређивање облика решења рекурентне релације, а затим доказати да то јесте решење методом замене. Када се стабло рекурзије користи за погађање облика решења, обично се без последица могу учинити нека занемаривања јер ће претпостављени облик решења ионако касније бити потврђен извођењем доказа. Ако се пак стабло рекурзије исцрта пажљиво, без занемаривања детаља, онда се оно може искористити као доказ решења рекурентне релације.

Размотримо као пример одређивање облика решења рекурентне релације $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$, и то горње границе за $T(n)$. Пошто заокруживање на целе бројеве обично не утиче много на решење рекурентне релације, она се може упростити на облик:

$$T(n) = 3T(n/4) + cn^2 \tag{3.2}$$

при чему је укључена и подразумевана константа $c > 0$.

На слици 3.2 приказано је како се формира стабло рекурзије за рекурентну релацију (3.2). Због једноставности претпоставимо да је n степен броја 4 (то је други пример прихватљивог занемаривања), због чега су величине свих потпроблема цели бројеви. На делу (a) слике је само чвор са $T(n)$, који се у делу (b) развија у еквивалентно стабло на основу (3.2). Члан cn^2 у корену представља цену на највишем нивоу рекурзије, а три подстабла испод корена представљају цене придружене потпроблемима величине $n/4$. Део (c) слике приказује следећи корак у овом процесу, у коме се развијају сви чворови са ценом $T(n/4)$ из дела (b). Цена сваког од три сина корена је $c(n/4)^2$. На исти начин наставља се развијање сваког чвора у три сина, у складу са рекурентном релацијом.

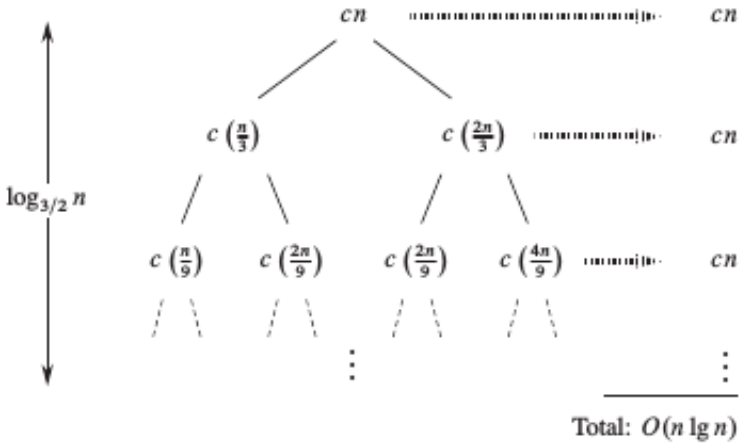
С обзиром на то да се величине потпроблема смањују за фактор 4 са сваким нивоом, величина потпроблема на дубини i једнака је $n/4^i$. Величина поптроблема постаје 1 када је $n/4^i = 1$, односно када је $i = \log_4 n$. Према томе, стабло има $\log_4 n + 1$ нивоа (на дубинама $0, 1, \ldots, \log_4 n$).

Одредимо сада збир цена на сваком нивоу стабла. Сваки ниво има три пута више чворова од претходног, па је број чворова на нивоу i једнак 3^i. Због тога што се величине проблема на сваком следећем нивоу смањују за фактор 4, сваки чвор на нивоу i има цену $c(n/4^i)^2$, $i = 0, 1, \ldots, \log_4 n$. Множењем са 3^i добија се да је укупна цена чворова на нивоу i једнака $3^i \cdot c \cdot (n/4^i)^2 = (3/16)^i \cdot c \cdot n^2, i = 0, 1, \ldots, \log_4 n$. На последњем нивоу $\log_4 n$ има укупно $3^{\log_4 n} = n^{\log_4 3}$ чворова, од којих сваки има цену $T(1)$; укупна њихова цена је $n^{\log_4 3} T(1) = \Theta(n^{\log_4 3})$. Збир цена свих чворова у стаблу је:

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i \cdot c \cdot n^2 + \Theta(n^{\log_4 3}) \\ &< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i \cdot c \cdot n^2 + \Theta(n^{\log_4 3}) \\ &= \frac{1}{1 - (3/16)} cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) = O(n^2) \end{aligned}$$

На тај начин дошли смо до претпоставке $T(n) = O(n^2)$ о облику решења рекурентне релације $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$. У овом примеру коефицијенти уз cn^2 чине опадајући геометријски ред, а сума тих коефицијената ограничена је одозго константом 16/13. Пошто је удео корена у суми једнак cn^2, види се да корен чини значајан удео целе суме.

У ствари, ако је $O(n^2)$ горња граница решења (што ће, иначе, бити доказано у наредном пасусу), онда је то тачна горња граница јер већ први рекурзивни позив има цену $\Theta(n^2)$. Сада још остаје да се методом замене докаже да је претпоставка $T(n) = O(n^2)$ оправдана. Хоћемо да докажемо да је $T(n) \leq dn^2$ за неку константу $d > 0$. Користећи претходно уведену константу $c > 0$, добијамо:



Слика 3.3: Стабло рекурзије за рекурентну релацију $T(n) = T(n/3) + T(2n/3) + O(n)$

$$\begin{aligned} T(n) &= 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= 3/16dn^2 + cn^2 \leq dn^2, \end{aligned}$$

при чему неједнакост у последњем кораку важи ако и само ако је $d \geq (16/13)c$.

Следећи пример је нешто компликованији. На слици 3.3 приказано је стабло рекурзије за рекурентну релацију:

$T(n) = T(n/3) + T(2n/3) + O(n)$

Као и у претходном примеру, нека c означава константни фактор у члану $O(n)$. Кад се саберу вредности на појединачним нивоима стабла рекурзије, испоставља се да је на сваком нивоу збир једнак cn. Најдужи пут од корена до листа је $n \to (2/3)n \to (2/3)^2 n \to 1$. Пошто је $(2/3)^k n = 1$ за $k = \log_{3/2} n$, висина стабла је $\log_{3/2} n$.

Интуитивно, очекујемо да је горња граница за решење једнака производу цене нивоа и броја нивоа, тј. $O(cn \log_{3/2} n) = O(n \log n)$. На слици 3.3 приказано је само првих неколико нивоа стабла; при томе, удео свих нивоа није једнак cn. Размотримо цену листова. Ако би стабло рекурзије било комплетно бинарно стабло висине $\log_{3/2} n$, било би укупно $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$ листова. Пошто је цена сваког листа константна, укупна цена свих листова била би $\Theta(n^{\log_{3/2} 2})$ што би, пошто је $\log_{3/2} 2 > 1$, било $\omega(n \log n)$[1] јер је $n \log n = o(n^{\log_{3/2} 2})$.

Међутим, стабло рекурзије није комплетно бинарно стабло, па има мање од $n^{\log_{3/2} 2}$ листова. Штавише, како се спуштамо од корена, све више и више

[1] ω-нотација служи за означавање доњих граница које нису асимптотски блиске. Важи да је $f(n) = \omega(g(n))$ акко је $g(n) = o(f(n))$, тј. ако је $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

унутрашњих чворова је одсутно из стабла. Због тога нивои који су далеко испод корена имају допринос укупној суми мањи од cn. Могли бисмо да се потрудимо и израчунамо збир свих цена; међутим, да се подсетимо, сврха стабла рекурзије је само да дођемо до закључка о облику решења.

Заиста, методом замене можемо да се уверимо да је $O(n \log n)$ горња граница за решење рекурентне релације. Показаћемо да је $T(n) \leq dn \log_2 n$, где је d одговарајућа позитивна константа. Претпоставимо да тврђење важи за све вредности $k < n$ и покажимо да важи и за n.

$$\begin{aligned} T(n) &\leq T(n/3) + T(2n/3) + cn \\ &\leq d(n/3)\log_2(n/3) + d(2n/3)\log_2(2n/3) + cn \\ &= dn\log_2 n - dn(\log_2 3 - 2/3) + cn \\ &\leq dn \log_2 n \end{aligned}$$

што је сигурно тачно ако је $d \geq c/(\log_2 3 - (2/3))$. Према томе, нема потребе да прецизније рачунамо збир цена чворова у стаблу рекурзије.

3.4 Пробабилистичка анализа алгоритама

3.4.1 Проблем запошљавања помоћника

Претпоставимо да имамо следећи проблем: потребно је да запослимо једног новог помоћника. Агенција нам сваког дана шаље новог кандидата, ми га интервјуишемо и на основу тога одлучујемо да ли да га запослимо или не. За сваког интервјуисаног кандидата агенцији плаћамо накнаду c_I, која није велика. Ако запослимо кандидата, трошак c_H је много већи јер морамо да отпустимо кандидата којег смо претходно запослили, а и накнада агенцији је много већа. Место помоћника нам је важно па желимо да у сваком моменту имамо запосленог најбољег од свих претходних кандидата. Стога, ако установимо да је неки кандидат квалификованији за посао од текућег помоћника, морамо да текућег помоћника отпустимо и да запослимо новог. Претпоставимо да су кандидати нумерисани бројевима од 1 до n. Наша стратегија приликом запошљавања кандидата може се описати следећим кодом:

```
ZaposliPomocnika(n)
1 Najbolji ← 1
2 for i ← 2 to n do
3   интервјуиши кандидата i
4   if кандидат i је бољи од кандидата Najbolji then
5     Najbolji ← i
6     запосли кандидата i
```

Циљ анализе овог алгоритма није процена временске сложености него процена трошкова интервјуисања и запошљавања кандидата. Иако је циљ анализе другачији, технике које се примењују су исте – у оба случаја броје се неке основне операције. Ако је број запослених кандидата m, укупни трошкови су $O(nc_I + mc_H)$, где су, као што је већ речено, c_I и c_H редом трошкови интервјуисања, односно запошљавања кандидата. Без обзира колико се кандидата запошљава, број интервјуисаних кандидата је увек n, што проузрокује трошак nc_I. Према томе, можемо да се ограничимо на разматрање трошкова запошљавања mc_H. Ова величина биће другачија за свако покретање алгоритма. Разматрани сценарио је модел за алгоритамски проблем на који се често наилази: потребно је да одредимо максимум (или минимум) елемената низа разматрајући један по један елемент низа и памтећи тренутног "победника". Од значаја је колико пута се ажурира тренутни "победник".

3.4.2 Анализа најгорег случаја

У најгорем случају запослићемо сваког интервјуисаног кандидата. До овог случаја долази ако је кандидат i бољи од свих претходних за свако $i = 1, 2, \ldots n$. Према томе, трошкови у најгорем случају износе $O(nc_H)$. Наравно, кандидати неће увек долазити овим редоследом. Ми, у ствари, немамо контролу над тиме којим редоследом кандидати долазе. Због тога је природно упитати се шта се може очекивати у типичном просечном случају.

3.4.3 Анализа просечног случаја

Да би се могла извршити анализа просечног случаја, мора се знати расподела вероватноћа могућих улаза за алгоритам. Овде се може претпоставити да се после интервјуа сваком кандидату може приписати оцена (реалан број). Поред тога, претпоставићемо да су оцене свака два кандидата различите и да сваки поредак првих i кандидата према оценама има исту вероватноћу $1/i!$ (у ситуацији када о кандидатима ништа не знамо ово је природна претпоставка). Под овом претпоставком у коду *ZaposliPomocnika* кандидат i бољи је од свих претходних са вероватноћом $1/i$ (његово место на ранг листи првих i кандидата може да буде $1, 2, \ldots, i$ и то са једнаким вероватноћама $1/i$). Према томе, вероватноћа да кандидат буде запослен у линији i кода је $1/i$ и просечна цена која одговара тој линији кода је c_H/i. Пошто нас занима само цена повезана са запошљавањем, укупна просечна цена целог поступка запошљавања је $c_H(1 + \frac{1}{2} + \ldots + \frac{1}{n}) = O(c_H \log n)$ јер је $H(n) = 1 + \frac{1}{2} + \ldots + \frac{1}{n} = O(\log n)$. Видимо да је просечна цена запошљавања најбољег кандидата на описани начин знатно мања од цене $O(c_H \cdot n)$ у најгорем случају.

3.4.4 Рандомизовани алгоритми

Видели смо како се може извршити анализа просечног случаја алгоритма у случају да нам је позната расподела свих могућих улаза. Ипак, у неким ситуацијама, ова информација није позната, што спречава извођење анализе просечног случаја. Тада је могуће користити **рандомизовани алгоритам**.

За проблем као што је проблем запошљавања, у коме је од помоћи претпоставити да су све пермутације на улазу једнако вероватне, анализа просечног случаја може да води развоју рандомизованог алгоритма. Уместо да се претпоставља расподела улаза, расподела се намеће самим алгоритмом. Дакле, пре покретања основног алгоритма прави се случајна пермутација кандидата у намери да се обезбеди својство да је свака пермутација једнако вероватна. Иако смо модификовали алгоритам, и даље очекујемо да и запослимо новог помоћника у просеку $\log n$ пута. Међутим, сада очекујемо да ово буде тачно за произвољан улаз, уместо за улаз који је изведен са претпостављеном расподелом вероватноћа.

Анализирајмо разлике између анализе просечног случаја и рандомизованих алгоритама. Алгоритам разматран у претходном поглављу је детерминистички, дакле за произвољни улаз број пута који се нови помоћник запошљава је увек исти (за свако покретање алгоритма над истим улазом). Додатно, број пута који запошљавамо новог помоћника се (у највећем броју случајева) разликује за различите улазе. С друге стране, рандомизовани алгоритам прво пермутује кандидате и онда одређује најбољег кандидата. У овом случају, рандомизација се врши у самом алгоритму, а не на расподели улаза. За конкретни улаз не можемо рећи колико пута се ажурира вредност максимума, јер се овај број (у највећем броју случајева) разликује при сваком покретању алгоритма. Сваки пут када покренемо алгоритам, извршавање зависи од случајних избора који су направљени. За ову врсту рандомизованих алгоритама ниједан појединачни улаз не гарантује понашање у најгорем случају. Рандомизовани алгоритам се понаша лоше само ако генератор случајних бројева произведе "несрећну" пермутацију.

```
RandomizovaniZaposliPomocnika(n)
1 пермутуј на случајан начин листу кандидата
2 Najbolji ← 1
3 for i ← 2 to n do
4    интервјуиши кандидата i
5    if кандидат i је бољи од кандидата Najbolji then
6       Najbolji ← i
7       запосли кандидата i
```

Са овом једноставном изменом направили смо рандомизовани алгоритам чије се перформансе поклапају са онима када се претпостави да су кандидати дати у случајном поретку.

Пермутовање низа на случајан начин

Многи рандомизовани алгоритми врше рандомизацију улаза пермутовањем датог улазног низа. Размотрићемо два начина на која се ово може постићи. Претпоставимо, без губитка општости, да улаз садржи елементе од 1 до n и да је циљ да произведемо случајну премутацију датог низа. Један стандардан начин да се ово уради јесте да се сваком елементу низа на случајан начин придружи приоритет и да се онда низ елемената сортира у складу са тим приоритетима.

```
PermutujSortiranjem(A, n)
1 нека је P[1..n] нови низ
2 for i ← 1 to n do
3    P[i] ← random(1, n^3)
4 сортирај A коришћењем P као кључа за сортирање
```

У линији 3 генерише се случајан број из опсега од 1 до n^3. Користимо овај опсег да бисмо учинили мање вероватним догађај да два приоритета имају исту вредност. Претпоставимо да су на овај начин сви генерисани приоритети јединствени. Најскупљи део приказаног алгоритма представља корак сортирања у линији 4 алгоритма који је сложености $\Omega(n \log n)$.

Други метод за генерисање случајне пермутације јесте да се дати низ пермутује у месту. У i-тој итерацији бира се елемент $A[i]$ на случајан начин из скупа елемената низа $A[i..n]$. Након i-те итерације $A[i]$ се више не мења.

```
RandomizujUMestu(A, n)
1 for i ← 1 to n do
2    замени A[i] са A[random(i, n)]
```

Приказана процедура *RandomizujUMestu* је временске сложености $O(n)$.

Глава 4

Конструкција алгоритама индукцијом

4.1 Увод

У овом поглављу биће наведени једноставни примери конструкције алгоритама коришћењем индуктивног приступа, који илуструју основне принципе и технике. Идеја је да се искористи принцип домина: све нанизане домине ће попадати ако се поруши прва, и ако се после рушења n-те обавезно поруши $(n+1)$-а домина. Дакле, да би се решио неки проблем, треба решити неки његов мали случај, а затим показати како се решење задатог проблема може конструисати полазећи од (решених) мањих верзија истог проблема. Имајући на уму ово, пажња се може сконцентрисати на налажење начина за свођење проблема на мање проблеме. Примери који ће бити размотрени показују да се различите стратегије конструкције алгоритама у суштини своде на примену математичке индукције.

4.2 Израчунавање вредности полинома

Најпре ћемо размотрити један једноставан проблем: израчунавање вредности задатог полинома у задатој тачки.

Проблем. Дати су реални бројеви a_n, a_{n-1},..., a_1, a_0 и реални број x. Израчунати вредност полинома $P_n(x) = \sum_{i=0}^{n} a_i x^i$.

Улазни подаци за проблем су $n+2$ броја ($n+1$ коефицијената и вредност x). Индуктивни приступ решавању проблема састоји се у свођењу решења на решење мањег проблема. Дакле, проблем покушавамо да сведемо на мањи, који се затим решава рекурзивно. Најједноставније је свођење на упрошћени проблем добијен од полазног уклањањем a_n. Тада имамо проблем израчунавања вредности полинома

$$P_{n-1}(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0.$$

То је исти проблем, али са једним параметром мање.

Индуктивна хипотеза. Претпоставимо да знамо да израчунамо вредност полинома задатог коефицијентима $a_{n-1},\ldots,a_1,a_0$ у тачки x, тј. знамо да израчунамо вредност $P_{n-1}(x)$.

Ова хипотеза је основа за решавање проблема индукцијом. Случај $n=0$ (израчунавање вредности израза a_0) је тривијалан. Затим се мора показати како се полазни проблем (израчунавање $P_n(x)$) може решити коришћењем решења мањег проблема (вредности $P_{n-1}(x)$). У овом случају је то очигледно: треба израчунати x^n, помножити га са a_n и резултат сабрати са $P_{n-1}(x)$: $P_n(x)=a_nx^n+P_{n-1}(x)$. Може се помислити да је коришћење индукције овде непотребно: оно само компликује врло једноставно решење. Описани алгоритам је еквивалентан израчунавању вредности полинома редом члан по члан. Испоставиће се ипак да овај приступ има своју снагу.

Описани алгоритам је тачан, али неефикасан, јер захтева $n+(n-1)+\cdots+1=n(n+1)/2$ множења и n сабирања. Коришћењем индукције на други начин добија се боље решење.

Запажамо да у овом алгоритму има много поновљених израчунавања: степен x^n израчунава се сваки пут "од почетка". Велики број множења може се уштедети ако се при израчунавању x^n искористи x^{n-1}. Побољшање се реализује укључивањем израчунавања x^{n-1} у индуктивну хипотезу.

Појачана индуктивна хипотеза. Знамо да израчунамо вредност полинома $P_{n-1}(x)$ *и* вредност x^{n-1}.

Ова индуктивна хипотеза је јача, јер захтева израчунавање x^{n-1}, али се лакше проширује (јер је сада једноставније израчунати x^n). Да би се израчунало x^n, довољно је извршити једно множење. После тога следи још једно множење са a_n и сабирање са $P_{n-1}(x)$. Укупно је потребно извршити $2n$ множења и n сабирања. Занимљиво је запазити да иако индуктивна хипотеза захтева више израчунавања, она доводи до смањења укупног броја операција. Добијени алгоритам изгледа добро у сваком погледу: ефикасан је, једноставан и једноставно се реализује. Ипак, постоји и бољи алгоритам. До њега се долази коришћењем индукције на нови, трећи начин.

Редукција проблема уклањањем највишег коефицијента a_n је очигледан корак, али то ипак није једина расположива могућност. Уместо тога се може уклонити коефицијент a_0, и свести проблем на израчунавање вредности полинома са коефицијентима $a_n,a_{n-1},\ldots,a_1$, односно полинома

$$\tilde{P}_{n-1}(x)=\sum_{i=1}^{n}a_ix^{i-1}=a_nx^{n-1}+a_{n-1}x^{n-2}+\cdots+a_1.$$

Приметимо да је a_n овде $(n-1)$-и коефицијент, a_{n-1} је $(n-2)$-и коефицијент, и тако даље. Дакле, имамо нову индуктивну хипотезу.

Индуктивна хипотеза (обрнути редослед)**.** Умемо да израчунамо вредност полинома $\tilde{P}_{n-1}(x)$ са коефицијентима $a_n,a_{n-1},\ldots,a_1$ у тачки x.

Оваква индуктивна хипотеза је погоднија јер се лакше проширује. Пошто је $P_n(x)=x\tilde{P}_{n-1}(x)+a_0$, за израчунавање $P_n(x)$ полазећи од $\tilde{P}_{n-1}(x)$ довољно је једно множење и једно сабирање. Израчунавање се може описати следећим изразом:

$$a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0=((\cdots((a_nx+a_{n-1})x+a_{n-2})\cdots)x+a_1)x+a_0,$$

познатим као Хорнерова шема. У наставку је дат псеудокод алгоритма.

$Vrednost\_polinoma(a, x)$
улаз: $a = a_0, a_1 \dots a_n$ (коефицијенти полинома) и x (реалан број)
излаз: P (вредност полинома у тачки x)
1 $P \leftarrow a_n$
2 **for** $i \leftarrow 1$ **to** n **do**
3 $\quad P \leftarrow x \cdot P + a_{n-i}$

Сложеност. Алгоритам обухвата n множења, n сабирања и захтева само једну нову меморијску локацију. Последњи алгоритам је не само ефикаснији од претходних, него је и одговарајући програм једноставнији.

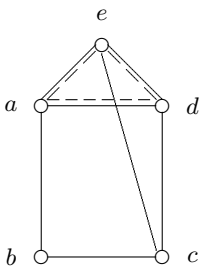
Примедба. Индукција нам омогућује да се сконцентришемо на проширивање решења мањих на решења већих проблема. Претпоставимо да хоћемо да решимо $P(n)$, односно проблем P који зависи од неког параметра n (обично је то величина проблема). Тада полазимо од произвољне инстанце проблема $P(n)$ и покушавамо да је решимо користећи претпоставку да је $P(n-1)$ већ решено. Постоји много начина да се постави индуктивна хипотеза, а исто тако и више начина да се она искористи. Приказаћемо више таквих метода, и демонстрирати њихову моћ при конструкцији алгоритама.

Пример израчунавања вредности полинома илуструје флексибилност која нам је на располагању при коришћењу индукције. Идеја која је довела до Хорнерове шеме састоји се у томе да се улаз обрађује слева удесно, уместо интутивног редоследа здесна улево. Друга уобичајена могућност избора редоследа појављује се при разматрању структуре у облику стабла: она се може пролазити одозго наниже или одоздо навише. Даље, могуће је параметар n повећавати за два (или више), уместо само за један, а постоје и многобројне друге могућности. Штавише, понекад најбољи редослед примене индукције није исти за све улазе. Може се показати корисним конструкција алгоритма који би проналазио најбољи редослед примене индукције. Овакве могућности илустроваћемо неким примерима.

4.3 Максимални индуковани подграф

Посматрајмо следећи проблем. Организујете конференцију за научнике који се баве разнородним дисциплинама и имате списак особа које бисте желели да позовете. Претпоставка је да ће се сваки од њих одазвати позиву ако му се буде указала могућност плодне размене идеја. За сваког научника можете да направите листу оних (са списка) са којима ће он вероватно имати интеракцију. Нека је k задати број. Ви бисте волели да позовете што је могуће више људи, а да при томе гарантујете свакоме од њих могућност да размени мисли са бар k других особа. Ваш проблем није да организујете те интеракције, нити да обезбедите довољно времена за њих. Ви једино желите да обезбедите њихово присуство на конференцији. Како изабрати особе које треба позвати? Описани проблем одговара следећем графовском проблему. Нека је $G = (V, E)$ неусмерени граф. Граф $H = (U, F)$ је *индуковани подграф* графа G ако $U \subseteq V$ и F садржи све гране

из E којима су оба краја у U, видети пример на слици 4.1. Чворови графа одговарају научницима, а грана између два чвора постоји ако одговарајући научници могу да размењују идеје. Индуковани подграф одговара подскупу научника.



$$G = (V, E)$$
$$V = \{a, b, c, d, e\}$$
$$E = \{(a, b), (a, d), (a, e), (b, c), (c, d), (c, e), (d, e)\}$$
$$H = (U, F)$$
$$U = \{a, d, e\}$$
$$F = \{(a, d), (a, e), (d, e)\}$$

Слика 4.1: Пример индукованог подграфа неусмереног графа. Гране су неуређени парови чворова.

Проблем. За задати неусмерени граф $G = (V, E)$ и задати природни број k пронаћи максимални индуковани подграф $H = (U, F)$ графа G (тј. индуковани подграф са максималним бројем чворова) уз услов да сви чворови подграфа H имају степен бар k, или установити да такав индуковани подграф не постоји.

Директни приступ решавању овог проблема састоји се у уклањању свих чворова степена мањег од k. После уклањања ових чворова, заједно са гранама које су им суседне, степени преосталих чворова се у општем случају смањују. Кад степен неког чвора постане мањи од k, тај чвор треба уклонити из графа. Није, међутим, јасан редослед којим чворове треба уклањати. Треба ли најпре уклонити све чворове степена $< k$, а затим обрађивати чворове са умањеним степеновима? Или треба уклонити један чвор степена $< k$, па наставити са чворовима којима су смањени степени? (Ова два приступа одговарају претрази у ширину, односно претрази у дубину). Да ли ће оба приступа дати исти резултат? Да ли ће резултујући граф имати највећи могући број чворова? На ова питања лако је дати одговор коришћењем следећег приступа.

Уместо да алгоритам посматрамо као низ корака које треба извршити да би се дошло до резултата, можемо да себи поставимо за циљ *доказивање теореме* да алгоритам постоји. Потребно је пронаћи максимални индуковани подграф који задовољава задате услове.

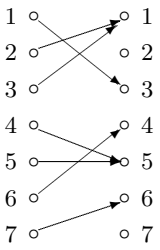
Индуктивна хипотеза. Умемо да пронађемо максимални индуковани подграф са чворовима степена $\geq k$ за граф са мање од n чворова.

Треба да докажемо да је ова "теорема" тачна за базни случај, а затим да из њене тачности за $n-1$ следи њена тачност за n. Први нетривијални случај појављује се за $n = k + 1$, јер ако је $n \leq k$ онда су степени свих чворова мањи од k. Ако је $n = k + 1$ онда само у једном случају постоји индуковани подграф са степенима чворова $\geq k$: то је случај потпуног графа (графа са свим могућим гранама). Претпоставимо даље да је $G = (V, E)$ произвољан граф са $n > k + 1$ чворова. Ако су степени свих чворова у G

већи или једнаки од k, онда граф $H = G$ задовољава услове и доказ је завршен. У противном постоји чвор v степена $< k$. Очигледно је да степен чвора v остаје $< k$ и у сваком индукованом подграфу графа G, па према томе v не припада ни једном подграфу који задовољава услове проблема. Дакле, v се може уклонити из G заједно са свим њему суседним гранама, не мењајући услове теореме. После уклањања v граф има $n-1$ чвор, па према индуктивној хипотези ми умемо да решимо проблем до краја.

4.4 Налажење бијекције

Нека је f функција која пресликава коначан скуп A у самог себе, $f : A \to A$. Без губитка општости може се претпоставити да је $A = \{1, 2, \ldots, n\}$. Претпостављамо да је функција f представљена вектором f дужине n, тако да $f[i]$ садржи вредност $f(i) \in A$, $i = 1, 2, \ldots, n$. Функција f је бијекција, ако за произвољан елемент $j \in A$ постоји елемент $i \in A$ такав да је $f(i) = j$. Функција се може представити дијаграмом као на слици 4.2, тако да сваком елементу A одговара по један леви и један десни чвор, а гране дефинишу пресликавање. Функција приказана на слици 4.2 очигледно није бијекција.



Слика 4.2: Пример пресликавања скупа у самог себе.

Проблем. Нека је задат коначан скуп A и пресликавање $f : A \to A$. Одредити подскуп $S \subseteq A$ са највећим могућим бројем елемената тако да буде $f(S) \subseteq S$ и да рестрикција f на S буде бијекција.

Ако је f бијекција, онда комплетан скуп A задовољава услове проблема, па је максимални подскуп $S = A$. Ако је пак $f(a) = f(b)$ за неке $a \neq b$, онда S не може да садржи и a и b. На пример, скуп S у проблему са слике 4.2 не може да садржи оба елемента 2 и 3, јер је $f(2) = f(3) = 1$. Међутим, није свеједно који ће елемент од ова два бити елиминисан. Ако на пример елиминишемо 3, онда се због $f(1) = 3$ мора елиминисати и 1. Затим се на исти начин због $f(2) = 1$ мора елиминисати и 2. Добијени подскуп сигурно није максималан, јер је могуће елиминисати само елемент 2 уместо 1, 2 и 3. Решење проблема на слици 4.2 је подскуп $\{1, 3, 5\}$, што се може установити нпр. провером свих могућих подскупова $S \subset A$. Поставља се питање налажења општег метода за одлучивање које елементе треба укључити у S, ефикаснијег од потпуне претраге.

На срећу, имамо известан маневарски простор приликом одлучивања како проблем свести на мањи. Величина проблема може се смањити било налажењем елемента који припада скупу S, било налажењем елемента који не припада скупу S. Испоставља се да је друга могућност боља. Користићемо следећу једноставну индуктивну хипотезу.

Индуктивна хипотеза. Знамо да решимо проблем за скупове са $n-1$ елемената.

База индукције је тривијална: ако скуп A има један елемент, он се мора пресликавати у себе, па је f на A бијекција. Претпоставимо даље да имамо произвољан скуп A од n елемената и да тражимо подскуп S који задовољава услове проблема. Тврдимо да ако елемент a не припада скупу $f(A)$, онда a не може припадати S; другим речима, елемент a за који у одговарајући чвор на десној страни дијаграма не улази ни једна грана, не може бити у S. Заиста, из услова проблема следи да је $f(S) = S$, а то је, ако се претпостави да $a \in S$ — немогуће, јер по претпоставци $a \notin f(S)$. Дакле, ако постоји такав елемент a, ми га елиминишемо из скупа. Сада имамо скуп $A' = A \setminus \{a\}$ са $n-1$ елементом, који f пресликава у самог себе; према индуктивној хипотези ми знамо да решимо проблем са скупом A'. Ако такво a не постоји, онда је пресликавање бијекција, и проблем је решен.

Суштина овог решења је да се *a мора* уклонити. Доказали смо да a не може припадати S. У томе је снага индукције: оног тренутка кад се елемент уклони и тиме смањи величина проблема, задатак је решен. При томе се мора водити рачуна да мањи проблем буде потпуно исти као полазни, изузимајући величину. Једини услов који f и A треба да задовољавају је $f(A) \subseteq A$. Овај услов задовољен је и на скупу $A' = A \setminus \{a\}$, јер се ни један елемент не пресликава у a. У тренутку кад више нема елемената који се могу уклонити, скуп S је пронађен.

Реализација. Алгоритам смо описали као рекурзивну процедуру. У сваком кораку проналазимо елемент у кога се ни један други не пресликава, уклањамо га и настављамо рекурзивно. Реализација алгоритма ипак не мора да буде рекурзивна. Сваком елементу i, $1 \leq i \leq n$ придружује се бројач $c[i]$, чија је почетна вредност једнака броју елемената који се пресликавају у i. Вредности $c[i]$ могу се израчунати у n корака пролазећи кроз вектор f и инкрементирајући (повећавајући за један) при томе одговарајуће бројаче. Затим се у ред стављају сви елементи којима је вредност бројача добила вредност 0. У сваком кораку се први елемент j из реда уклања из реда и из скупа, декрементира се $c[f[j]]$, а ако $c[f[j]]$ добије вредност 0, $f[j]$ се убацује у ред. Решавање је завршено у тренутку кад је ред празан.

Bijekcija(f, n)
улаз: f (низ природних бројева између 1 и n)
излаз: S (подскуп улазног скупа такав да је f бијекција на S)
1 $S \leftarrow A$ $\{A = \{1, 2, \ldots n\}\}$
2 **for** $j \leftarrow 1$ **to** n **do** $c[j] \leftarrow 0$
3 **for** $j \leftarrow 1$ **to** n **do** инкрементирај $c[f[j]]$
4 **for** $j \leftarrow 1$ **to** n **do**
5 **if** $c[j] = 0$ **then** стави j у *Spisak* {ред елемената за елиминацију}
6 **while** *Spisak* није празан **do**
7 скини i са почетка реда *Spisak*
8 $S \leftarrow S \setminus \{i\}$
9 $c[f[i]] = c[f[i]] - 1$
10 **if** $c[f[i]] = 0$ **then** стави $f[i]$ у *Spisak*

Није тешко проверити да се за улаз представљен на слици 4.2 добија $S = \{1, 3, 5\}$.

Сложеност. За уводни део (иницијализацију) треба извршити $O(n)$ корака. Сваки елемент се у ред може ставити највише једном, а операције потребне да се елемент уклони из реда извршавају се за време ограничено константом. Према томе, укупан број корака је $O(n)$.

4.5 Проблем проналажења звезде

Следећи пример је интересантан по томе што за решавање проблема није неопходно прегледати све улазне податке, па чак ни њихов значајан део. Међу n особа *звезда* је особа која никога не познаје, а коју сви остали познају. Проблем је идентификовати звезду (ако она постоји) постављајући питања облика "Извините, да ли познајете ону особу?" Претпоставља се да су сви одговори тачни, а да ће чак и звезда одговарати на питања. Циљ је минимизирати укупан број питања. Пошто парова особа има укупно $n(n-1)/2$, у најгорем случају треба поставити $n(n-1)$ питања, ако се питања постављају без неке стратегије. На први поглед није јасно може ли се жељени циљ постићи са гарантовано мањим бројем постављених питања.

Проблем се може преформулисати у графовски. Формирамо усмерени граф са чворовима који одговарају особама, у коме грана од особе A ка особи B постоји ако A познаје B. Звезда одговара **понору** графа, чвору у који улази $n-1$ грана, а из кога не излази ни једна грана. Јасно је да граф може имати највише један понор. Инстанца проблема се описује $n \times n$ матрицом повезаности M, чији елемент M_{ij} је једнак 1 ако особа i познаје особу j, односно 0 у противном; дијагонални елементи су једнаки нули.

Проблем. Дата је $n \times n$ матрица повезаности M. Установити да ли постоји индекс i, такав да су у M сви елементи i-те колоне (сем i-тог) једнаки 1, и да су сви елементи i-те врсте једнаки 0.

Базни случај са две особе је једноставан. Посматрајмо као и обично разлику између проблема са $n-1$ особом и проблема са n особа. Претпостављамо да знамо да пронађемо звезду међу првих $n-1$ особа индукцијом. Пошто може да постоји највише једна звезда, постоје три могућности:

1. звезда је једна од првих $n-1$ особа;
2. звезда је n-та особа;
3. звезда не постоји.

Први случај је најједноставнији: треба само проверити да ли n-та особа познаје звезду, односно да ли је тачно да звезда не познаје n-ту особу. Преостала два случаја су тежа, јер да би се проверило да ли је n-та особа звезда, треба поставити $2(n-1)$ питања. Ако поставимо $2(n-1)$ питања у n-том кораку, онда ће укупан број питања бити $n(n-1)$, што хоћемо да избегнемо. Потребно је дакле променити приступ проблему.

Идеја је да се проблем посматра "уназад". Пошто је тешко идентификовати звезду, покушајмо да пронађемо особу која није звезда: у сваком случају, број не-звезда много је већи од броја звезда. Ако елиминишемо некога из разматрања, параметар n који дефинише величину проблема

смањује се на $n-1$. При томе уопште није битно кога ћемо елиминисати! Претпоставимо да особу A питамо да ли познаје особу B. Ако A познаје B, онда A није звезда, а ако пак A не познаје B, онда B није звезда. У оба случаја се једна особа елиминише постављањем само једног питања!

Ако се сада вратимо на три могућа случаја при преласку са $n-1$ на n, видимо да је новост у томе да n-ту особу не бирамо произвољно. Елиминацијом особе A или B проблем сводимо на случај са $n-1$ особом, при чему смо сигурни да до случаја 2. неће доћи, јер елиминисана особа не може бити звезда. Ако је наступио случај 3, односно нема звезде међу првих $n-1$ особа, онда нема звезде ни међу свих n особа. Остаје први случај, који је лак: ако међу првих $n-1$ особа постоји звезда, онда се са два допунска питања може проверити да ли је то звезда и за комплетан скуп. Ако није, онда нема звезде.

Алгоритам се састоји у томе да питамо особу A да ли познаје особу B и елиминишемо или особу A или особу B, зависно од добијеног одговора. Нека је елиминисана нпр. особа A. Индукцијом (рекурзивно) се проналази звезда међу преосталих $n-1$ особа. Ако међу њима нема звезде, решавање је завршено. У противном, проверава се да ли је тачно да A познаје звезду, а да звезда не познаје A.

Реализација. Као и у случају алгоритма из претходног одељка, ефикасније је реализовати алгоритам за налажење звезде итеративно, него рекурзивно. Алгоритам се дели у две фазе. У првој фази се елиминишу све особе сем једног кандидата за звезду, а друга фаза је неопходна да се установи да ли кандидат јесте звезда. Почињемо са n кандидата, за које можемо да претпоставимо да су смештени на стек. За сваки пар кандидата ми можемо да елиминишемо једног, питањем да ли познаје другог. Почињемо узимајући прва два кандидата са стека и елиминишући једног од њих. Даље у сваком кораку формирамо пар од преосталог кандидата и (све док је стек непразан) наредне особе са стека, и елиминишемо једног од њих. У тренутку кад стек постане празан, имамо само једног кандидата. Затим се проверава да ли је тај кандидат звезда. Стек је реализован експлицитно коришћењем индекса i, j, и $Naredni$.

$Zvezda(Poznaje)$
улаз: $Poznaje$ ($n \times n$ Булова матрица са нулама на дијагонали)
излаз: $Zvezda$
1 $i \leftarrow 1$; {особа A}
2 $j \leftarrow 2$; {особа B}
3 $Naredni \leftarrow 3$; {следећи који ће се проверавати}
4 {прва фаза: елиминишу се сви кандидати осим једног}
5 **while** $Naredni \leq n+1$ **do**
6 **if** $Poznaje[i,j]$ **then** $i \leftarrow Naredni$ {елиминација i}
7 **else** $j \leftarrow Naredni$ {елиминација j}
8 $Naredni \leftarrow Naredni + 1$
9 {по изласку из петље је или $i = n+1$ или $j = n+1$}
10 **if** $i = n+1$ **then** $Kandidat \leftarrow j$
11 **else** $Kandidat \leftarrow i$
12 {друга фаза: провера да ли је $Kandidat$ звезда }
13 $Jeste \leftarrow true$ {$true$ ако $Kandidat$ јесте звезда}
14 $k \leftarrow 1$

```
15  while Jeste and k ≤ n do
16    if Poznaje[Kandidat, k] then Jeste ← false
17    if  not Poznaje[k, Kandidat] then
18      if Kandidat ≠ k then Jeste ← false
19    k ← k + 1
20  if Jeste then Zvezda ← Kandidat
21  else Zvezda ← 0 {нема звезде}
```

Сложеност. Поставља се највише $3(n-1)$ питања: $n-1$ питања у првој фази да би се елиминисала $n-1$ особа, и највише $2(n-1)$ питања за проверу да ли је преостали кандидат звезда. Приметимо да величина улаза није n, него $n(n-1)$, број елемената матрице. Приказано решење показује да је могуће одредити звезду прегледајући највише $O(n)$ елемената матрице повезаности, иако је јасно да решење мора битно да зависи од свих $n(n-1)$ елемената матрице.

4.6 Пример примене разлагања: максимум скупа правоугаоника

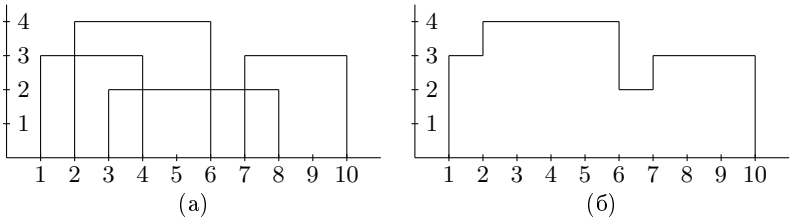
Нека су L, D и V реални бројеви такви да је $L < D$ и $V > 0$. Правоугаона функција (правоугаоник) са параметрима (L, D, V) је функција

$$f_{L,D,V}(x) = \begin{cases} 0, & x < L \text{ или } x \geq D \\ V, & L \leq x < D \end{cases} .$$

Проблем. Дато је n правоугаоника $f_{L_i,D_i,V_i}(x)$, $i = 1, 2, \ldots, n$. Потребно је одредити "контуру" овог скупа правоугаоника, односно функцију

$$f(x) = \max \{ f_{L_i,D_i,V_i}(x) \mid i = 1, 2, \ldots, n \}$$

Пример улаза и излаза приказан је на слици 4.3: улаз су четири правоугаоника $(1, 4, 3)$, $(2, 6, 4)$, $(3, 8, 2)$ и $(7, 10, 3)$, слика 4.3(а), а резултујућа контура приказана је на слици 4.3(б). Излазна функција $f(x)$ може се описати скупом интервала на којима она има константну вредност, и вредностима у тим интервалима. Другим речима, $f(x)$ се задаје низом парова (x_i, v_i), $i = 1, 2, \ldots, k$ (при чему је $v_k = 0$), тако да је $f(x) = v_i$ за $x_i \leq x < x_{i+1}$, $i = 1, 2, \ldots, k$. Због једноставности претпоставља да је још $x_0 = -\infty$, $v_0 = 0$, $x_{k+1} = \infty$. У примеру на слици 4.3(б) функција $f(x)$ описује се низом парова $(1, 3)$, $(2, 4)$, $(6, 2)$, $(7, 3)$, $(10, 0)$.



Слика 4.3: Пример улаза и излаза за проблем налажења максимума скупа правоугаоника: (а) улаз; (б) излаз.

Директни приступ решавању овог проблема заснива се на проширивању решења корак по корак, тј. проширивању решења за првих $n-1$ правоугаоника новим, n-тим правоугаоником. База индукције $n = 1$ је једноставна: правоугаоник $f_{L_1,D_1,V_1}(x)$ претвара се у функцију $f_{(L_1,V_1),(D_1,0)}(x)$. Да би се максимуму

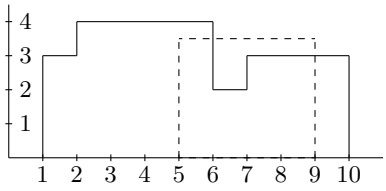
$$f_{n-1}(x) = f_{(x_1,v_1),(x_2,v_2),\ldots,(x_k,v_k)}(x)$$

прикључио n-ти правоугаоник $f_{L_n,D_n,V_n}(x)$, треба пронаћи такве индексе i, j, да је $x_i \leq L_n < x_{i+1}$, $x_j \leq D_n < x_{j+1}$, $0 \leq i \leq j \leq k$: вредност функције $f_{n-1}(x)$ може се при преласку на $f_n(x)$ променити само у оним интервалима константности са којима интервал $[L_i, D_i)$ има непразан пресек. Нови максимум је

$$f_n(x) = \begin{cases} f_{n-1}(x), & x < L_n \text{ или } x \geq D_n \\ \max\{f_{n-1}(x), V_n\}, & L_n \leq x < D_n \end{cases}.$$

Интервали константности $[x_i, x_{i+1})$ и $[x_j, x_{j+1})$ функције $f_{n-1}(x)$ у општем случају бивају тачкама L_n и D_n подељени на подинтервале $[x_i, L_n)$, $[L_n, x_{i+1})$, односно $[x_j, D_n)$, $[D_n, x_{j+1})$, у којима је вредност $f_n(x)$ константна; поред тога, у интервалима $[x_l, x_{l+1})$, $l = i+1, \ldots, j-1$ у којима $f_{n-1}(x)$ има вредност мању од V_n, $f_n(x)$ добија нову (у односу на $f_{n-1}(x)$) вредност V_n. У примеру на слици 4.4 је $n = 5$, $(L_n, D_n, V_n) = (5, 9, 3.5)$, и

$$\begin{aligned} f_{n-1}(x) &= f_{(1,3),(2,4),(6,2),(7,3),(10,0)}(x), \\ f_n(x) &= f_{(1,3),(2,4),(6,3.5),(9,3),(10,0)}(x). \end{aligned}$$

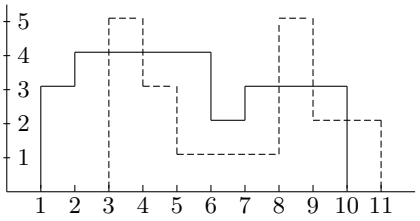


Слика 4.4: Додавање новог правоугаоника максимуму претходних $n-1$ са слике 4.3.

Овај алгоритам је коректан, али није ефикасан: додавање n-тог правоугаоника функцији $f_{n-1}(x)$ захтева $O(n)$ корака, па је укупна сложеност $O(n^2)$. Алгоритам се може усавршити применом метода разлагања (divide-and-conquer) . Уместо преласка са $n-1$ на n, ефикасније је проширивати решење са $n/2$ на n правоугаоника. Базни случај је већ размотрен. Као што смо видели раније, решење диференцне једначине $T(n) = T(n-1) + cn$ је $T(n) = O(n^2)$, док је решење једначине $T(n) = 2T(n/2) + cn$ дато са $T(n) = O(n \log n)$. Дакле, ако поделимо проблем на два једнака потпроблема и добијена решења потпроблема објединимо за линеарно време, онда је сложеност алгоритма $O(n \log n)$.

У проблему са правоугаоницима разлагање се заснива на запажању да је за додавање новог правоугаоника функцији $f_{n-1}(x)$ потребно у најгорем случају линеарно време, а да се за линеарно време могу објединити и две различите функције $f'_{n/2}(x)$ и $f''_{n/2}(x)$, решења два потпроблема за по $n/2$ правоугаоника. За асимптотски исто време постиже се много више користећи други приступ. Две у деловима константне функције могу се

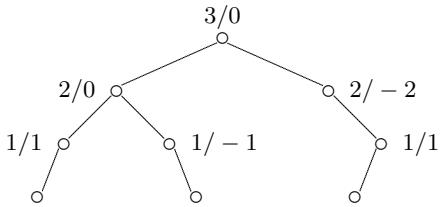
комбиновати у основи истим алгоритмом којим је комбинован један правоугаоник са једном таквом функцијом (слика 4.5). Прекидне тачке двеју функција $f'_{n/2}(x)$ и $f''_{n/2}(x)$ се пролазе истовремено слева у десно, идентификују се x-координате крајева интервала константности функције $f_n(x)$ и одређују вредности $f_n(x)$ у тим интервалима. Обједињавање се извршава за линеарно време, па је сложеност комплетног алгоритма $O(n \log n)$ у најгорем случају.



Слика 4.5: Одређивање максимума две у деловима константне функције.

4.7 Израчунавање фактора равнотеже бинарног стабла

Нека је T бинарно стабло са кореном r. **Висина** чвора v је растојање од v до најдаљег потомка. Висина стабла је висина његовог корена. **Фактор равнотеже** чвора v дефинише се као разлика висина његовог левог и десног подстабла; за стабло које недостаје, сматра се да има висину -1. На слици 4.6 приказано је бинарно стабло код кога је уз сваки чвор уписан пар h/b, где је h висина чвора, а b његов фактор равнотеже.



Слика 4.6: Бинарно стабло са ознакама h/b уз унутрашње чворове (h – висина, b – фактор равнотеже чвора).

Проблем. За задато стабло T са n чворова израчунати факторе равнотеже свих чворова.

Покушаћемо са индуктивним приступом и најједноставнијом индуктивном хипотезом.

Индуктивна хипотеза. Знамо да израчунамо факторе равнотеже свих чворова у стаблима са $< n$ чворова.

Базни случај $n = 1$ је тривијалан. Ако је дато стабло са $n > 1$ чворова, можемо да уклонимо корен, затим да решимо проблем (индуктивно) за два подстабла која су преостала. Определили смо се за уклањање корена

због тога што фактор равнотеже чвора v зависи само од чворова испод v. Због тога знамо факторе равнотеже свих чворова изузев корена. Међутим, фактор равнотеже корена зависи не од фактора равнотеже, него од висина његових синова. Закључујемо да у овом случају једноставна индукција не решава проблем. Потребно је да знамо *висине* синова корена. Идеја је да налажење висина чворова прикључимо полазном проблему.

Појачана индуктивна хипотеза. Знамо да израчунамо факторе равнотеже **и** висине свих чворова у стаблима која имају $< n$ чворова.

Базни случај је поново тривијалан. Кад сада посматрамо корен произвољног стабла, његов фактор равнотеже можемо лако да одредимо као разлику висина његових синова. Можемо да одредимо и висину корена: то је већа од висина његових синова, увећана за један.

Карактеристика овог алгоритма је да он решава нешто општији проблем. Уместо да рачунамо само факторе равнотеже, ми израчунавамо и висине чворова. Испоставља се да је општији проблем лакши, јер се висине лако рачунају. Ако је решење општије (јер је проблем проширен), онда индуктивни корак може бити једноставнији, јер поседујемо снажнија средства. Уобичајена грешка при решавању ширег проблема је да се заборави на постојање два параметра, и да се сваки од њих мора посебно израчунати.

4.8 Налажење максималног узастопног подниза

Проблем. Задат је низ x_1, x_2,..., x_n реалних бројева (не обавезно позитивних). Одредити подниз $x_i, x_{i+1},\ldots, x_j$ узастопних елемената са највећом могућом сумом.

За овакав подниз рећи ћемо да је **максимални подниз**. На пример, у низу $(2, -3, 1.5, -1, 3, -2, -3, 3)$ максимални подниз је $1.5, -1, 3$, са сумом 3.5. Може да постоји више максималних поднизова датог низа. Ако су сви чланови низа негативни, онда је максимални подниз празан (по дефиницији, сума празног низа је 0). Волели бисмо да имамо алгоритам који решава овај проблем и прегледа (пролази низ редом слева у десно) низ само једном.

Индуктивна хипотеза. Знамо да нађемо максимални подниз у низовима дужине $< n$.

За $n = 1$ максимални подниз је број x_1 ако је $x_1 \geq 0$, односно празан подниз ако је $x_1 < 0$. Посматрајмо низ $S = (x_1, x_2, \ldots, x_n)$ дужине $n > 1$. Према индуктивној хипотези ми умемо да пронађемо максимални подниз S'_M низа $S' = (x_1, x_2, \ldots, x_{n-1})$. Ако је S'_M празан подниз, онда су сви бројеви у низу S' негативни, па остаје да се размотри само x_n. Претпоставимо сада да је $S'_M = (x_i, x_{i+1}, \ldots, x_j)$ за неке индексе i и j такве да је $1 \leq i \leq j \leq n-1$. Ако је $j = n-1$ (односно максимални подниз је суфикс), онда је лако ово решење проширити на S: ако је број x_n позитиван, онда он продужује S'_M, а у противном подниз S'_M је максималан и у S. Међутим, ако је $j < n-1$, онда постоје две могућности. Или S'_M остаје максимални подниз у S, или постоји други подниз који није максималан у S', али после додавања x_n постаје максималан у S.

Кључна идеја је да се **појача индуктивна хипотеза**. Она ће бити демонстрирана на примеру налажења максималног подниза, а затим ће

нешто општије бити размотрена у следећем одељку. Проблем са размотреном индуктивном хипотезом је у томе што x_n може да продужи подниз који није максималан у S' и тако формира нови максимални подниз у S. Према томе, није довољно само познавање максималног подниза у S'. Међутим, x_n може да продужи само подниз који се завршава са x_{n-1}, односно суфикс низа S'. Претпоставимо да смо појачали индуктивну хипотезу тако да обухвати и налажење максималног суфикса, означеног са $S'_E = (x_k, x_{k+1}, \ldots, x_{n-1})$.

Појачана индуктивна хипотеза. У низовима дужине $< n$ умемо да пронађемо максимални подниз, и максимални суфикс.

Ако за подниз S' знамо оба ова подниза, алгоритам постаје јасан. Максимални суфикс проширујемо бројем x_n. Ако је добијена сума већа од суме (глобално) максималног подниза, онда имамо нови максимални подниз (такође и нови максимални суфикс). У противном, задржавамо претходни максимални подниз. Посао тиме није завршен: потребно је одредити и нови максимални суфикс. У општем случају нови максимални суфикс се не добија увек проширивањем старог бројем x_n. Може се догодити да је максимални суфикс који се завршава сабирком x_n негативан. Тада је нови максимални суфикс уствари празан подниз, са сумом 0.

$Max\_uzast\_podniz(X, n)$
улаз: X (низ дужине n)
излаз: $Glob\_max$ (сума максималног подниза)
1 $Glob\_max \leftarrow 0$ {почетна вредност глобално максималне суме}
2 $Suf\_max \leftarrow 0$ {почетна вредност суме максималног суфикса}
3 **for** $i \leftarrow 1$ **to** n **do**
4 **if** $x[i] + Suf\_max > Glob\_max$ **then**
5 $Suf\_max \leftarrow Suf\_max + x[i]$
6 $Glob\_max \leftarrow Suf\_max$
7 **elseif** $x[i] + Suf\_max > 0$ **then**
8 $Suf\_max \leftarrow Suf\_max + x[i]$
9 **else** $Suf\_max \leftarrow 0$

4.9 Појачавање индуктивне хипотезе

Појачавање индуктивне хипотезе је једна од најважнијих техника за доказивање теорема индукцијом. Приликом тражења доказа често се наилази на следећу ситуацију. Нека је теорема коју треба доказати означена са P. Индуктивна хипотеза може се означити са $P(< n)$, а доказ теореме своди се на доказивање тачности тврђења $P(< n) \Rightarrow P(n)$. У много случајева може се увести допунска претпоставка Q, која олакшава доказ. Другим речима, лакше је доказати $[P \wedge Q](< n) \Rightarrow P(n)$ него $P(< n) \Rightarrow P(n)$. Претпоставка може да изгледа коректна, али није јасно како се може доказати. Идеја се састоји у томе да се Q укључи у индуктивну хипотезу. Сада је потребно доказати да $[P \wedge Q](< n) \Rightarrow [P \wedge Q](n)$. Конјункција $P \wedge Q$ је јаче тврђење него само P, али се често јача тврђења лакше доказују. Процес се наставља, и после одређеног броја уведених претпоставки може да доведе до завршетка доказа. Проблем налажења максималног подниза је добар пример како се овај принцип може искористити за усавршавање алгоритама.

Најчешћа грешка приликом коришћења ове технике је превид чињенице да је додата допунска претпоставка у тренутку кад треба кориговати доказ. Другим речима, ми доказујемо тврђење $[P \wedge Q](< n) \Rightarrow P(n)$, заборављајући да је уведена допунска претпоставка Q. У примеру са максималним поднизом до ове грешке би дошло ако не бисмо израчунали нови максимални суфикс. Слично, у примеру са налажењем фактора равнотеже чворова бинарног стабла, оваква грешка била би ако се не израчуна висина корена стабла. Према томе, важно је прецизно регистровати промене индуктивне хипотезе.

4.10 Уобичајене грешке

Осврнућемо се на неке могуће грешке приликом индуктивне конструкције алгоритама. На пример, често се заборавља на базу индукције. У случају рекурзивне процедуре базни случај је од суштинског значаја, јер се кроз њега излази из рекурзије. Друга честа грешка је проширивање решења за n на решење специјалног, уместо произвољног случаја проблема за $n+1$.

Ненамерна промена индуктивне хипотезе је такође честа грешка. Демонстрираћемо то на једном примеру. Граф G је **бипартитни** ако се његов скуп чворова може представити као дисјунктна унија два подскупа, тако да не постоји грана између два произвољна чвора из истог поскупа. Може се доказати да је за повезан бипартитни граф овакво разлагање у дисјунктну унију једнозначно.

Проблем. За задати повезани неусмерени граф G установити да ли је бипартитни, а ако јесте, пронаћи одговарајуће разлагање његових чворова у дисјунктну унију.

Погрешно решење: Уклањамо чвор v и разлажемо остатак графа индукцијом, ако је могуће. Први подскуп зваћемо *бели*, а други *црни*. Ако је v повезан само са белим чворовима, прикључујемо га црном скупу, и обрнуто — ако је v повезан само са црним чворовима, прикључујемо га белом скупу. Ако је v повезан са чворовима из оба подскупа, онда граф није бипарититан, јер је разлагање једнозначно.

Основна грешка у овом решењу (а то је грешка коју смо намеравали да илуструјемо) је у томе што после уклањања чвора v граф не мора да остане повезан. Према томе, мањи пример проблема није исти као полазни, па се индукција не може користити. Ако бисмо уклонили чвор који не "разбија" граф, решење би било коректно.

4.11 Резиме

У овом поглављу приказано је више техника за конструкцију алгоритама, које су у ствари варијанте једног истог приступа. Нове технике и бројни примери биће дати у наредним поглављима. Најбољи начин да се те технике савладају је да се оне користе за решавање проблема. Приликом решавања проблема корисно је имати на уму следеће сугестије:

- Принцип индукције користи се да се улаз за проблем сведе на један или више мањих. Ако се свођење може увек извести, а базни случај се може решити, онда је алгоритам дефинисан индукцијом, односно добијен је рекурзивни алгоритам. Основна идеја је да се пажња сконцентрише на смањивање проблема, а не на његово директно решавање.

- Један од најлакших начина да се смањи проблем је елиминација неких његових елемената, што се може покушати на више начина. Поред елиминације елемената који су очигледно сувишни (као у одељку 4.3), могуће је спајање два елемента у један, проналажење елемената који се могу обрадити на посебан начин, или увођење новог елемента који преузима улогу два или више полазних елемената (као у случају Хафмановог кодирања).

- Величина проблема може се смањити на више начина. Међутим, не добија се сваком од тих редукција алгоритам исте ефикасности. Зато је корисно размотрити различите редоследе примене индуктивне хипотезе.

- Један од најефикаснијих начина за смањивање величине проблема је његова подела на два или више подједнаких делова. Разлагање је ефикасно ако се проблем може поделити, тако да се од решења потпроблема лако добија решење целог проблема.

- Приликом свођења на мање потпроблеме, треба тежити да потпроблеми буду што независнији. На пример, проблем сортирања може се свести на проналажење и уклањање најмањег елемента; редослед осталих не зависи од уклоњеног елемента.

- На крају, све набројане технике треба користити заједно, комбинујући их на разне начине. На пример, може се користити разлагање са појачавањем индуктивне хипотезе, тако да се добијени потпроблеми лакше комбинују.

Глава 5

Стратегије конструкције алгоритама

5.1 Увод

Постоји неколико познатих стратегија конструкције алгоритама међу којима су алгоритми грубе силе, похлепни алгоритми, претрага са враћањем, гранање са одсецањем, разлагање, динамичко програмирање.

Алгоритми грубе силе подразумевају прегледање свих могућих решења и најчешће се примењују на оптимизационе проблеме. Ова стратегија је применљива само за врло мале улазе, док се за велике улазе углавном трага за неким ефикаснијим алгоритмом. На пример, уколико би проблем био да се за дати природан број n одреде сви његови делиоци, алгоритам грубе силе би за сваки цео број од 1 до $n/2$ проверавао да ли дели број n.

Идеја *похлепних алгоритама* је да увек праве избор који делује најбољи у том тренутку. Дакле, прави се локално оптимални избор у нади да ће овај избор водити и глобално најбољем решењу. Похлепни алгоритми за неке (не све) проблеме проналазе оптимално решење. На пример, ако је проблем да за дати низ новчаница које имамо на располагању вратимо кусур са најмањим могућим бројем новчаница, похлепни алгоритам би у сваком кораку бирао највећу новчаницу која је на располагању, а која је мања од текуће вредности преосталог кусура који треба вратити. Овакав приступ у општем случају не гарантује добијање оптималног решења.

Претрага са враћањем или скраћено *претрага* представља једну од најопштијих техника конструкције алгоритама и многи проблеми који се тичу тражења скупа решења или који имају за циљ одређивање оптималног решења које задовољава неки скуп ограничења могу се решити на овај начин. Алгоритми претраге са враћањем траже решење проблема систематичном претрагом простора решења. Проблем који се може решити на овај начин је решавање судоку задатака, дакле за дату делимично попуњену 9×9 матрицу доделити бројеве празним пољима матрице тако да сваки ред, врста и 3×3 подматрица садржи по једном све цифре од 1 до 9. Алгоритам грубе силе би генерисао све могуће конфигурације и проверавао да ли је пронађена конфигурација валидна, док би алгоритам претраге пре сваке доделе вредности проверавао да ли је та додела дозвољена и у случају да

јесте рекурзивно настављао даље, односно у случају да није проверавао би наредну могућу вредност за доделу.

Гранање са одсецањем је варијација претраге, која се може применити ако се тражи минимум (или максимум) неке циљне функције. У овим алгоритмима важно је израчунавање добрих доњих или горњих граница које служе за раније напуштање неперспективних парцијалних решења. Дакле, ако се на основу текућег парцијалног решења не може добити боље решење од текућег најбољег решења, онда се ово парцијално решење искључује из даљег разматрања и врши се одсецање. На пример, у проблему бојења графа минималним бројем боја, могуће је вршити одсецање у ситуацији када је потребно увести нову боју за бојење графа, чиме би број употребљених боја постао једнак броју боја којима је граф већ успешно обојен.

У наставку ћемо детаљније размотрити стратегију динамичког програмирања и видети још један пример стратегије разлагања.

5.2 Динамичко програмирање

Слично као разлагање, **динамичко програмирање** решава проблеме свођењем на потпроблеме. Стратегија разлагања дели проблеме на дисјунктне потпроблеме, које рекурзивно решава и чија решења комбинује у решење полазног проблема. За разлику од тога, динамичко програмирање се примењује када се потпроблеми преклапају, тј. када потпроблеми имају заједничке потпроблеме. У таквој ситуацији метод разлагања обавља више посла него што је неопходно, решавајући више пута исте потпроблеме. Динамичко програмирање решава сваки потпроблем само једном и резултати се записују у одговарајућу табелу.

Динамичко програмирање се најчешће примењује на проблеме оптимизације. Приликом развоја алгоритама овог типа, разликују се четири корака:

1. карактеризација структуре оптималног решења
2. рекурзивно дефинисање *вредности* оптималног решења
3. израчунавање *вредности* оптималног решења, најчешће редоследом одоздо на горе
4. реконструкција оптималног решења на основу израчунатих података

Кораци 1. – 3. су основа решења применом динамичког програмирања. Ако само решење није потребно, корак 4. може се изоставити. Ако се пак извршава и корак 4., понекад је потребно у оквиру корака 3. прикупљати додатне информације, да би се олакшала реконструкција оптималног решења.

У наставку ћемо видети како се динамичко програмирање примењује на решавање неких проблема, као што су проблем расецања жице, збир подскупа и израчунавање едит-растојања два низа.

5.2.1 Проблем расецања жице

Гвожђара продаје комаде жице дужине i по цени p_i $1, 2, \ldots$ у складу са ценовником. Пример ценовника приказан је у табели 5.1.

дужина i	1	2	3	4	5	6	7	8	9	10
цена p_i	1	5	8	9	10	17	17	20	24	30

Табела 5.1: Пример ценовника са ценама p_i комада жице различите дужине i

У гвожђари имају колут жице дужине n. *Проблем расецања жице* гласи: како исећи жицу на комаде тако да збир цена свих комада буде највећи могући? На пример, ако је $n = 4$, жица се може исећи на пет суштински различитих начина (пошто је број партиција броја 4 једнак 5):

партиција $n = 4$	4	$3+1$	$2+2$	$2+1+1$	$1+1+1+1$
збир цена	9	9	10	7	4

Табела 5.2: Збир цена за различите партиције броја $n = 4$

Као што се може видети из табеле 5.2, збир цена комада је највећи (10) ако се жица исече на два комада дужине 2.

Ако се приликом расецања жице води рачуна о редоследу, онда је укупан број пресецања жице највише $n - 1$. Пошто се на сваком од тих места жица може независно пресећи или не, број начина на које се то може урадити је 2^{n-1}. Број суштински различитих начина расецања жице дужине n једнак је броју партиција броја n, који се асимптотски понаша као $\frac{1}{4n\sqrt{3}} e^{\pi\sqrt{2n/3}}$, што расте брже од било ког полинома по n. Због тога је овакав начин решавања проблема неефикасан за велике n.

Ако је оптимално решење расећи жицу на k комада дужине $i_1, i_2, \ldots, i_k$ (дакле $n = i_1 + i_2 + \ldots + i_k$), онда је одговарајућа максимална сума

$$r_n = p_{i_1} + p_{i_2} + \ldots + p_{i_n}$$

За $n \leq 10$ није тешко непосредно израчунати вредност максималне суме r_n и одговарајуће партиције за коју се постиже та сума:

n	1	2	3	4	5	6	7	8	9	10
r_i	1	5	8	10	13	17	18	22	25	30
партиције	1	2	3	$2+2$	$2+3$	6	$1+6$ $2+2+3$	$2+6$	$3+6$	10

Вредности r_n за $n \geq 1$ могу се израчунати на основу претходних вредности:

$$r_n = max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1\} \tag{5.1}$$

Први аргумент функције max, p_n одговара варијанти када се цела жица дужине n продаје као један комад. Осталих $n - 1$ аргумената одговарају

максималном збиру ако се жица на почетку најпре пресече на два комада дужине i, односно $n-i$, $i=1,\ldots,n-1$, после чега се та два комада *оптимално* расеку на мање комаде чиме се добијају збирови r_i, односно r_{n-i}. Пошто се унапред не зна коју вредност i је најбоље изабрати, разматрају се све могуће вредности i и бира се она којој одговара највећи збир $r_i + r_{n-i}$. Последњи сабирак у једначини (5.1) могао би да буде $r_{\lfloor n/2 \rfloor} + r_{\lfloor n/2 \rfloor}$, због симетрије.

Приметимо да у току решавања полазног проблема величине n, ми решавамо мање проблеме истог типа. Кад се уради прво расецање, $i+(n-i)$, два добијена комада могу се сматрати инстанцама проблема расецања жице. Укупно оптимално решење у себи садржи оптимална решења два одговарајућа потпроблема. Због тога кажемо да проблем расецања жице има *оптималну подструктуру*: оптимално решење проблема садржи оптимална решења одговарајућих потпроблема који се могу независно решавати.

Могућ је сличан, али једноставнији начин уређења рекурзивне структуре проблема расецања жице, тако да се од жице најпре одсече један комад дужине i, а да се даље расеца само остатак дужине $n-i$. На овај начин може се обухватити и варијанта без сечења жице: први комад је тада $i=n$, са ценом p_n, а остатак је 0 са ценом $r_0 = 0$. Тако се добија једнакост једноставнија од (5.1):

$$r_n = max_{1 \le i \le n}(p_i + r_{n-i}) \tag{5.2}$$

5.2.2 Рекурзивни алгоритам одозго на доле

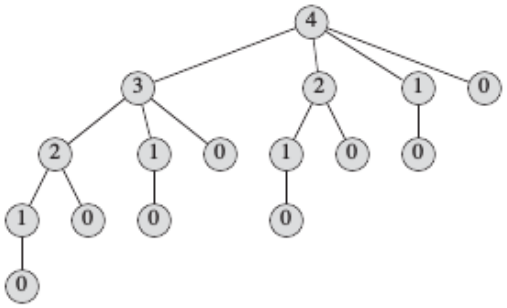
Једнакост (5.2) директно се реализује следећим кодом:

```
Rasecanje(p, n)
улаз: p = p_1, p_2 ... p_n (вектор цена комада жице) и n (дужина жице)
излаз: q (максимална вредност)
1 if n = 0 return 0
2 q ← −∞
3 for i ← 1 to n
4     q ← max{q, p_i + Rasecanje(p, n − i)}
5 return q
```

Оваква рекурзивна реализација је неефикасна, јер се рекурзивни позиви са истим аргументом извршавају више пута. На пример, за $n=4$ добија се стабло рекурзивних позива приказано на слици 5.1.

Нека је $T(n)$ укупан број позива алгоритма *Rasecanje* када је основни позив са другим аргументом једнаким n. Број чворова у подстаблу чији је корен означен са n једнак је $T(n)$, при чему је ту урачунат и основни позив који одговара корену. Рекурзивном алгоритму *Rasecanje* одговара рекурентна релација

$$T(n) = \begin{cases} 1, & n = 0 \\ 1 + \sum_{j=0}^{n-1} T(j), & n > 0 \end{cases} \tag{5.3}$$



Слика 5.1: Стабло рекурзивних позива алгоритма $Rasecanje(p, n)$ за $n = 4$. У сваком чвору приказана је величина n одговарајућег потпроблема, тако да грана од оца са ознаком s ка сину са ознаком t одговара расецању на почетни комад величине $s - t$, после чега преостаје потпроблем величине t. Пут од корена до неког листа одговара неком од 2^{n-1} начина за расецање жице дужине n. У општем случају ово стабло има 2^n чворова и 2^{n-1} листова. Ознаке на путу од корена до неког листа једнаке су величини остатка жице после одсецања. Другим речима, ознаке одређују тачке сечења рачунајући са десног краја жице.

Сабирак 1 одговара позиву у корену, а члан $T(j)$ једнак је броју позива (укључујући рекурзивне позиве) покренутих позивом $Rasecanje(p, n - i)$, где је $j = n - i$. Непосредно се проверава да је $T(1) = 2, T(2) = 4, T(3) = 8$, а индукцијом се лако доказује да је $T(n) = 2^n$, што значи да је сложеност алгоритма $Rasecanje$ експоненцијална по n. Ово није неочекивано, пошто алгоритам разматра свих 2^{n-1} начина разлагања броја n у сабирке.

5.2.3 Коришћење динамичког програмирања за оптимално расецање

Показаћемо сада како се алгоритам $Rasecanje$ може учинити ефикасним применом динамичког програмирања. Запажа се да је обичан рекурзиван алгоритам неефикасан јер више пута решава исте потпроблеме. Проблем се може решити тако да се сваки потпроблем решава само *једном*, записујући то решење. Ако се касније наиђе на исти потпроблем може се прочитати записани резултат, уместо да се поново израчунава. Према томе, динамичко програмирање користи допунску меморију да би се убрзало израчунавање, што га чини једним од примера *размене простор-време* (енг. time-memory trade-off). Уштеде могу бити значајне, тако да се од алгоритма експоненцијалне сложености може доћи до алгоритма полиномијалне сложености.

Постоје два еквивалентна начина за реализацију алгоритма који користи динамичко програмирање:

- први начин је приступ *одозго на доле са записивањем*, односно *меморизацијом* (енг. top-down with memoization). Процедура се пише рекурзивно на природан начин, али са додатком да се записује резултат

сваког потпроблема (обично у низу или хеш табели). Нова процедура најпре проверава да ли је претходно већ решавала тај проблем; ако јесте, враћа сачувани резултат и прескаче даље рачунање; у противном процедура израчунава резултат на уобичајен начин.

- други приступ је приступ *одоздо на горе* (енг. bottom-up). Приступ је обично повезан са неким природним појмом "величине" потпроблема, тако решавање било ког потпроблема зависи само од решења "мањих" потпроблема. Потпроблеми се најпре сортирају према величини, па се решавају тим редоследом, полазећи од мањих. У тренутку када се решава неки потпроблем, сви мањи потпроблеми од којих он зависи су већ решени и њихова решења су записана. Сваки потпроблем решава се само једном, а кад наиђемо на њега, већ смо пре тога решили све потпроблеме од којих он зависи.

Ова два приступа обично воде алгоритмима исте асимптотске сложености. Приступ одоздо на горе обично има мање константне факторе јер не троши време на рекурзивне позиве. Следи псеудокод алгоритма за оптимално расецање, приступ одозго на доле са записивањем.

Rasecanje_z(p, n)
улаз: $p = p_1, p_2 \dots p_n$ (вектор цена комада жице) и n (дужина жице)
излаз: q (максимална вредност)
1 Нека је $r_0, \dots, r_n$ нови низ
2 **for** $i \leftarrow 0$ **to** n
3 $\quad r_i \leftarrow -\infty$
4 **return** *Rasecanje_z_pom*(p, n, r)

Rasecanje_z_pom(p, n, r)
улаз: $p = p_1, p_2 \dots p_n$ (вектор цена комада жице) и n (дужина жице)
$\quad r = r_0, \dots, r_n$ (вектор у коме вредност r_i садржи максималну вредност за улаз i)
излаз: q (максимална вредност)
1 **if** $r_n \geq 0$ **then**
2 $\quad$ **return** r_n
3 **if** $n = 0$ **then**
4 $\quad q \leftarrow 0$
5 **else**
6 $\quad q \leftarrow -\infty$
7 $\quad$ **for** $i \leftarrow 1$ **to** n **do**
8 $\quad\quad q \leftarrow \max\{q, p_i + Rasecanje\_z\_pom(p, n-i, r)\}$
9 $r_n \leftarrow q$
10 **return** q

Овде основни алгоритам *Rasecanje_z* иницијализује нови помоћни низ $r_0, \dots, r_n$ вредностима $-\infty$, што је погодан начин за означавање "непознатог" (познате вредности су увек ненегативне). После тога покреће се помоћни алгоритам *Rasecanje_z_pom*, који је у ствари само нова верзија претходног рекурзивног алгоритма са додатим записивањем израчунатих резултата. У линији 1 проверава се да ли је тражена вредност већ позната, а ако јесте, она се враћа у линији 2. У противном, у линијама 3-7 израчунава се

тражена вредност q на обичан начин, у линији 8 она се записује у r_n, а у линији 9 враћа се као резултат. Верзија одоздо на горе је једноставнија.

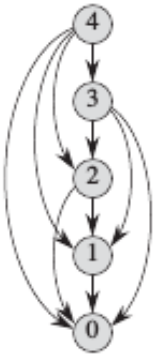
Rasecanje_DG(p, n)
улаз: $p = p_1, p_2 \dots p_n$ (вектор цена комада жице) и n (дужина жице)
излаз: q (максимална вредност)
1 Нека је $r_0, \dots, r_n$ нови низ
2 $r_0 \leftarrow 0$
3 **for** $j \leftarrow 1$ **to** n **do**
4 $\quad q \leftarrow -\infty$
5 $\quad$ **for** $i \leftarrow 1$ **to** j **do**
6 $\quad\quad q \leftarrow \max\{q, p_i + r_{j-i}\}$
7 $\quad r_j \leftarrow q$
8 **return** r_n

Овде се користи природно уређење потпроблема: проблем величине i је мањи од проблема величине j, ако је $i < j$. Према томе, процедура решава потпроблеме редом величине $j = 0, 1, \dots, n$.

У линији 1 креира се помоћни низ $r_0, \dots, r_n$ за записивање резултата за одговарајуће потпроблеме, а линија 2 иницијализује r_0 нулом. Петља у линијама 3–7 решава све потпроблеме величине $j, j = 1, 2, \dots, n$. Приступ који се користи за решавање проблема конкретне величине j је исти као у алгоритму *Rasecanje*, изузев што се у линији 6 директно чита вредност r_{j-i} уместо да се рекурзивним позивом решава потпроблем величине $j - i$. У линији 7 чува се решење потпроблема величине r_j. Коначно, у линији 8 враћа се r_n, што је једнако оптималној вредности r_n. Обе верзије алгоритма, одозго на доле са записивањем и одоздо на горе имају исту асимптотску сложеност $\Theta(n^2)$, јер садрже двоструке петље. За прву верзију алгоритма то ипак није сасвим очигледно. Због тога што се рекурзивни позив за претходно решени потпроблем одмах завршава враћањем резултата, алгоритам сваки потпроблем решава само једном. За проблем величине n кроз петљу у линијама 6–7 пролази се n пута. Због тога је укупан број итерација обе петље за све рекурзивне позиве збир аритметичке прогресије, дакле $\Theta(n^2)$.

5.2.4 Графови потпроблема

Када се разматра решавање проблема применом динамичког програмирања, потребно је имати представу о скупу потпроблема и какве су њихове међузависности. *Граф потпроблема* садржи управо те информације. На слици 5.2 приказан је граф потпроблема за проблем расецања жице за $n = 4$. То је усмерени граф са по једним чвором за сваки потпроблем. У графу постоји усмерена грана од чвора одговарајућег за потпроблем x ка чвору који је одговарајући за потпроблем y ако одређивање оптималног решења за потпроблем x директно у себе укључује разматрање оптималног решења за потпроблем y. На пример, грана у графу од чвора x до чвора y постоји ако рекурзивни одозго на доле алгоритам покренут за x директно позива самог себе за y. Граф потпроблема може се схватити као упрошћена верзија стабла рекурзије за рекурзивни метод одозго на доле, тако да се сви чворови за исти потпроблем обједине у само један чвор.



Слика 5.2: Граф потпроблема за проблем расецања жице ако је $n = 4$. Ознаке чворова одговарају величинама потпроблема. Усмерена грана (x, y) показује да је приликом решавања потпроблема x потребно решити потпроблем y. Граф је редукована верзија стабла са слике 5.1, при чему су сви чворови са истом ознаком скупљени у један чвор и све гране воде од оца ка сину.

Метод одоздо на горе чворове у графу потпроблема разматра таквим редоследом да смо пре разматрања чвора x размотрили све чворове y такве да постоји грана (x, y). Другим речима, ако бисмо све гране графа усмерили супротно, редослед чворова добија се тополошким сортирањем, о чему ће бити речи касније. На сличан начин види се да метод одозго на доле са записивањем обилази граф потпроблема у складу са алгоритмом претраге у дубину. Величина графа потпроблема може да помогне при одређивању сложености алгоритма заснованог на динамичком програмирању. Пошто се сваки потпроблем решава само једном, време извршавања једнако је збиру времена извршавања свих потпроблема. Обично је време потребно за решавање једног потпроблема пропорционално степену одговарајућег чвора (броју грана које излази из тог чвора), а број потпроблема једнак је броју чворова у графу потпроблема. У таквом случају време извршавања динамичког програмирања је линеарно у односу на број чворова и грана графа потпроблема.

5.2.5 Реконструкција решења

Размотрени алгоритми за решавање оптималног расецања жице динамичким програмирањем враћају само цену оптималног решења, али не и само решење. Алгоритми се могу допунити тако да поред оптималне *вредности* за сваки потпроблем врате и *избор* који је довео до те оптималне вредности. У наставку је проширена верзија одоздо на горе алгоритма *Rasecanje_z* која за сваку величину j комада жице враћа не само најбољу цену r_j него и s_j, оптималну величину комада жице који се прво одсеца:

Rasecanje_DG_E(p, n)
улаз: $p = p_1, p_2 \ldots p_n$ (вектор цена комада жице) и n (дужина жице)
излаз: q (максимална вредност)

```
1   Нека су r_0, ..., r_n и s_0, ... s_n нови низови
2   r_0 ← 0
3   for j ← 1 to n do
4     q ← −∞
5     for i ← 1 to j do
6       if q < p_i + r_{j−i} then
7         q ← p_i + r_{j−i}
8         s_j ← i
9     r_j ← q
10  return r, s
```

Овај алгоритам сличан је алгоритму *Rasecanje_DG* с тим што се у линији 1 креира и низ $s_1, \ldots, s_n$, а у линији 8 се ажурира s_j, тако да садржи оптималну величину i првог комада жице који се одсеца при решавању потпроблема величине j.

Наредна процедура на основу цена p и величине жице n позива алгоритам *Rasecanje_DG_E* за израчунавање низа $s_1, \ldots, s_n$ оптималних величина првих одсечених комада жице, а онда штампа комплетан списак величина комада жице и њихово оптимално расецање за жицу дужине n:

Rasecanje_Stampa(p, n)
улаз: $p = p_1, p_2 \ldots p_n$ (вектор цена комада жице) и n (дужина жице)
излаз: одштампан вектор који даје максималну вредност

```
1 (r, s) = Rasecanje_DG_E(p, n)
2 while n > 0
3   одштампај s_n
4   n ← n − s_n
```

За цене комада наведене у ценовнику алгоритам *Rasecanje_DG_E*$(p, 10)$ вратио би следеће низове:

n	0	1	2	3	4	5	6	7	8	9	10
r_i	0	1	5	8	10	13	17	18	22	25	30
s_i	0	1	2	3	2	2	6	1	2	3	10

Позив *Rasecanje_Stampa*$(p, 10)$ одштампао би само 10, а позив за $n = 7$ одштампао би 1 и 6, што одговара оптималном расецању r_7 које је раније наведено.

5.2.6 Проблем ранца

Претпоставимо да ранац треба напунити стварима. Ствари могу да буду различитог облика и величине, а циљ је да се у ранац упакује што је могуће више ствари. Уместо ранца то може да буде камион, брод или силицијумски чип, а проблем је паковање елемената. Постоји много варијанти овог проблема; овде ће бити разматрана варијанта са једнодимензионалним предметима.

Проблем. Дат је природан број K и n предмета различитих величина (тежина), тако да i-ти предмет има величину k_i, $1 \leq i \leq n$. Пронаћи подскуп предмета чија је сума величина једнака тачно K, или установити да такав подскуп не постоји.

Означимо проблем са $P(n, K)$, где n означава број предмета, а K величину (носивост) ранца. Величине предмета подразумевају се имплицитно, односно њихове величине нису део експлицитне ознаке проблема. На тај начин $P(i, k)$ означава проблем са првих i предмета и ранцем величине k. Због једноставности ограничавамо са на проблем одлучивања да ли решење постоји. Започињемо са најједноставнијим индуктивним приступом.

Индуктивна хипотеза (први покушај)**.** Умемо да решимо $P(n-1, K)$.

Базни случај (када је $n = 1$) је једноставан: решење постоји акко је $k_1 = K$. Ако постоји решење проблема $P(n-1, K)$, онда је то и решење $P(n, K)$: у то решење не улази предмет величине k_n. У противном, ако не постоји решење $P(n-1, K)$, поставља се питање — може ли се искористити овај негативни резултат? Најпре закључујемо да n-ти предмет мора бити укључен у збир. Тада остали предмети морају стати у мањи ранац величине $K - k_n$. Проблем је тиме сведен на два мања проблема $P(n-1, K)$ и $P(n-1, K-k_n)$. Да би се решење комплетирало, требало би појачати индуктивну хипотезу. Потребно је имати решења не само за ранац величине K, него и за ранце величине мање од K.

Индуктивна хипотеза (други покушај)**.** Умемо да решимо $P(n-1, k)$ за све k, $0 \leq k \leq K$.

Ова индуктивна хипотеза омогућује нам да решимо $P(n, k)$ за $0 \leq k \leq K$. Базни случај $P(1, k)$ се лако решава: за $k = 0$ увек постоји решење (тривијално, нула сабирака), а у противном решење постоји акко је $k_1 = k$. Проблем $P(n, k)$ своди се на два проблема $P(n-1, k)$ и $P(n-1, k-k_n)$ (други проблем отпада ако је $k - k_n < 0$). Оба ова проблема се могу решити индукцијом. Пошто је свођење комплетно, алгоритам је конструисан. Међутим, овај алгоритам је неефикасан. Проблем величине n сведен је на *два* проблема величине $n-1$ (истина, у једном од потпроблема смањена је вредност k). Сваки од нових проблема своди се на по два наредна проблема, итд, што доводи до експоненцијалног алгоритма.

Ефикасност алгоритма може се повећати применом динамичког програмирања уместо рекурзије. Запажа се да укупан број различитих проблема не мора бити превелики: први параметар може да има највише n, а други највише K различитих вредности. Укупно је број различитих проблема највише nK. Експоненцијално време извршавања алгоритма је последица дуплирања броја проблема после сваког свођења. Пошто укупно има nK различитих проблема, јасно је да су неки од њих (ако је $2^n > nK$) више пута решавани. Корисна идеја је памтити сва нађена решења да би се избегло поновљање решавања истих проблема. Овај приступ је комбинација појачавања индуктивне хипотезе и коришћења потпуне индукције. Размотрићемо сада како се овај приступ може практично реализовати.

За смештање свих израчунатих резултата користи се посебна матрица димензије $n \times K$. Елемент (i, k) матрице садржи информације о решењу

$P(i,k)$. Свођење по другој варијанти индуктивне хипотезе еквивалентно је израчунавању једне врсте на основу претходне врсте матрице. Сваки елемент се израчунава на основу два елемента из претходне врсте. Ако је потребно одредити и подскуп са задатим збиром, онда уз сваки елемент матрице треба сачувати и информацију о томе да ли је одговарајући елемент скупа укључен у збир у том кораку. Пратећи ове информације уназад од елемента (n,K), може се реконструисати подскуп са збиром K. Алгоритам је приказан у наставку, а у табели 5.3 приказан је један пример. Матрица P је због удобности димензије $(n+1)\times(k+1)$ и садржи елементе $P[i,k]$, $0\le i\le n$, $0\le k\le K$.

$Ranac(S,K)$
улаз: S (вектор дужине n са величинама предмета) и K (величина ранца)
излаз: P (матрица, тако да је $P[i,k].Postoji = true$ акко постоји решење проблема ранца са првих i предмета, за величину ранца k;
$P[i,k].Pripada = true$ акко i-ти предмет припада том решењу)

```
1  P[0,0].Postoji ← true
2  for k ← 1 to K do
3     P[0,k].Postoji ← false {елементи P[i,0] за i ≥ 1 рачунају се на основу P[0,0]}
4  for i ← 1 to n do
5     for k ← 0 to K do  {израчунавање елемента P[i,k]}
6        P[i,k].Postoji ← false {полазна вредност}
7        if P[i − 1,k].Postoji then
8           P[i,k].Postoji ← true
9           P[i,k].Pripada ← false
10       elseif k − S[i] ≥ 0 then
11          if P[i − 1,k − S[i]].Postoji then
12             P[i,k].Postoji ← true
13             P[i,k].Pripada ← true
```

	0	1	2	3	4	5	6	7	8	9	10	11
$k_1=8$	O	-	-	-	-	-	-	-	I	-	-	-
$k_2=5$	O	-	-	-	-	I	-	-	O	-	-	-
$k_3=4$	O	-	-	-	I	O	-	-	O	I	-	-
$k_4=3$	O	-	-	I	O	O	-	I	I	O	-	I

Табела 5.3: Пример табеле формиране при решавању проблема ранца. Улаз су $n=4$ предмета величина $8,5,4$ и 3, а величина ранца је $K=11$. Симбол "I" значи да постоји решење које обухвата предмет из одговарајуће врсте; "O": постоји решење, али само без тог предмета; "-": не постоји решење.

Метод коришћен за решавање овог проблема је специјални случај динамичког програмирања, и то приступа одоздо на горе, код кога се формирају велике табеле (у општем случају вишедимензионалне) са свим претходним резултатима. Табеле се конструишу итеративно. Сваки елемент се израчунава на основу већ израчунатих елемената. Основни проблем је организовати израчунавање елемената табеле на најефикаснији начин.

Сложеност. У табели има nK елемената. Сваки од њих израчунава се за константно време на основу друга два елемента, па је укупна временска

сложеност алгоритма $O(nK)$. Ако предмети нису превелики, онда K не може бити превелико, па је $nK \ll 2^n$. Ако је K јако велико, или су величине предмета реални бројеви, онда је овај приступ неприменљив. Ако је потребно само установити да ли решење постоји, онда је одговор садржан у елементу $P[n, K]$. Ако пак треба одредити подскуп са збиром K, онда треба прећи пут уназад, полазећи од позиције (n, K), користећи поље *Postoji* елемената матрице из програма. Предмет тежине k_n припада скупу акко је $P[n, K].Pripada$ тачно; ако је $P[n, K].Pripada$ тачно, односно нетачно, онда се даље на исти начин посматра елеменат $P[n-1, K-k_n]$, односно $P[n-1, K]$ из $(n-1)$-е врсте, итд. Подскуп се реконструише за $O(n)$ корака.

Просторна сложеност овог алгоритма је $O(nK)$. Ако се захтева само одговор на питање да ли постоји подскуп са збиром K, лако је модификовати алгоритам тако да му просторна сложеност буде $O(K)$: наредна врста се израчунава на основу претходне, па је за израчунавање елемента $P[n, K]$ довољно имати простор за смештање две узастопне врсте матрице.

5.2.7 Упоређивање низова

Проблему упоређивања низова тек се од недавно посвећује већа пажња. Основни разлог су примене у молекуларној биологији. Овде ћемо се бавити проблемом проналажења минималног броја едит операција које преводе један стринг у други.

Нека су $A = a_1a_2 \ldots a_n$ и $B = b_1b_2 \ldots b_m$ два стринга, низа чији су елементи знакови из коначног алфабета. Циљ је трансформисати A у B, знак по знак. Дозвољене су три врсте елементарних трансформација (или **едит операција**), које имају цену 1:

- *уметање* — уметање знака;
- *брисање* — брисање знака, и
- *замена* — замена једног знака другим знаком.

Проблем. За задате стрингове $A = a_1a_2 \ldots a_n$ и $B = b_1b_2 \ldots b_m$ пронаћи низ едит операција најмање цене који A трансформише у B.

На пример, да се стринг *abbc* трансформише у стринг *babb*, може се прво обрисати a (добија се *bbc*), уметнути a између два b (добија се *babc*), и на крају заменити c са b — укупно три промене. Исти резултат може се постићи са само две промене: уметањем новог b на почетку (добија се *babbc*) и брисањем последњег c. Циљ је пронаћи трансформацију која се састоји од минималног броја едит операција (таква трансформација у општем случају није јединствена).

Најмањи број едит операција који трансформише стринг A у стринг B зваћемо **едит растојањем** $d(A, B)$ стрингова A и B. Функција $d(A, B)$ је метрика: очигледно је $d(A, B) = 0$ ако и само ако је $A = B$, и $d(A, B) = d(B, A)$ за произвољне стрингове A и B (ако постоји низ едит операција дужине k који трансформише A у B, онда инверзних k операција, примењених обрнутим редоследом, трансформишу B у A). Ако су A, B и C произвољни стрингови, тада важи неједнакост троугла: $d(A, B) + d(B, C) \geq d(A, C)$.

Заиста, постоји низ од $d(A,B)$ едит операција које преводе A у B, и постоји низ од $d(B,C)$ едит операција које преводе B у C. Посматране заједно, ових $d(A,B)+d(B,C)$ едит операција преводе A у C; из чињенице да је $d(A,C)$ најмањи број едит операција које A преводе у C следи тражена неједнакост. Растојање $d(A,B)$ познато је као *Левенштајново растојање* (Levenshtein).

Размотримо како може да изгледа минимални низ едит операција који A трансформише у B. На неке знакове a_{i_1}, a_{i_2},..., a_{i_k} из A примењује се операција замене знацима b_{j_1}, b_{j_2},..., b_{j_k} из B. Искључивањем ових знакова, стрингови A и B су подељени на $k+1$ (евентуално празних) група знакова, тако да се i-та (празна) група у B добија брисањем знакова из i-те групе у A, или се од i-те (празне) групе у A уметањем добија i-та група у B, $i=1,2,\ldots,k+1$. Због минималности су све едит операције у оквиру једне групе истог типа — или брисање или уметање, јер се брисање једног знака, па уметање другог знака на исто место (две едит операције са укупном ценом 2), може заменити само једном заменом, цене 1.

Узимајући у обзир ове чињенице, низ едит операција који је кандидат за оптимални, може се представити на следећи прегледан начин. Уведимо нови знак ϕ. Наспрам обрисаних знакова из A у B се умеће знак ϕ, а наспрам знакова у B, добијених уметањем, у A се умеће знак ϕ. Тиме су A, B продужени до низова $\bar{A}$, $\bar{B}$ једнаке дужине, а све едит операције сведене су на замену: брисању знака a из A одговара замена a из $\bar{A}$ са ϕ у $\bar{B}$, а уметању знака b у B одговара замена ϕ у $\bar{A}$ знаком b у $\bar{B}$. Знак ϕ сматра се различитим од свих осталих знакова, па је цена замене истим, односно различитим знаком и даље 0, односно 1. Два горе наведена примера низова едит операција могу се представити са

a	b	ϕ	b	c	
ϕ	b	a	b	b	
1+	0+	1+	0+	1	$=3,$

односно

ϕ	a	b	b	c	
b	a	b	b	ϕ	
1+	0+	0+	0+	1	$=2.$

Проблем трансформисања стрингова има такође примену при упоређивању датотека и при чувању различитих верзија датотеке. Претпоставимо да имамо текстуалну датотеку (нпр. програм) и другу датотеку који је модификација прве. Згодно је издвојити разлике између две датотеке. То могу бити различите верзије истог програма, па ако су верзије сличне и треба да буду архивиране, ефикасније је чувати само једну верзију и разлике уместо обе верзије програма. У таквим случајевима (овде знацима стринга одговарају линије у датотеци) дозвољене операције су само уметања и брисања. У другим ситуацијама могу се различитим едит операцијама доделити различите цене.

Могућих низова едит операција које A преводе у B, има много, па је на први поглед тешко међу њима наћи најбољи. За почетак, ограничићемо се на налажење едит растојања $d(A,B)$, односно најмањег броја едит операција које преводе A у B (а не и самог низа едит операција). Као и обично, покушаћемо да проблем решимо индукцијом. За задати стринг A означимо са $A(i)$ његов префикс дужине i. Проблем је одредити растојање $d(A(n),B(m))$.

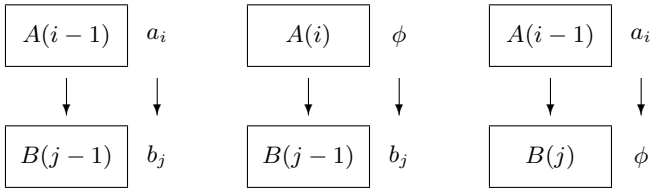
За $n = 0$ потребно је празан префикс $A(0)$ трансформисати у стринг $B(m)$ дужине m. Очигледно је за то потребно најмање m операција уметања знакова; према томе, $d(A(0), B(m)) = m$ за $m \geq 0$. Слично, $d(A(n), B(0)) = n$ за $n \geq 0$, јер се најкраћи низ едит операција које стринг $A(n)$ дужине n преводи у празан стринг $B(0)$ састоји од n брисања. Да бисмо проблем решили индукцијом, потребно је да израчунавање растојања $d(A(i), B(j))$ између неких префикса сведемо на израчунавање растојања између неких "краћих" префикса. Овај циљ може се постићи ако размотримо која едит операција може бити последња у низу едит операција које преводе $A(i)$ у $B(j)$. Постоје три могућности:

(а) замена знака a_i (истим или различитим) знаком b_j, пошто је претходно префикс $A(i-1)$ трансформисан у $B(j-1)$,

(б) уметање знака b_j пошто је претходно префикс $A(i)$ трансформисан у $B(j-1)$;

(ц) брисање знака a_i, пошто је претходно префикс $A(i-1)$ трансформисан у $B(j)$,

што је илустровано на слици 5.3.



(а) замена (б) уметање (б) брисање

Слика 5.3: Низови едит операција које трансформишу A у B разврстани према последњој операцији.

Нека је

$$c(i,j) = \begin{cases} 0, & \text{за} \quad a_i = b_j \\ 1, & \text{за} \quad a_i \neq b_j \end{cases} .$$

Најмањи број едит операција које $A(i)$ преводе у $B(j)$ једнак најмањем од три израза $d(A(i-1), B(j-1)) + c(i,j)$, $d(A(i), B(j-1)) + 1$ и $d(A(i-1), B(j)) + 1$. Нека је C матрица чији су елементи растојања свих подстрингова A од свих подстрингова B, тј. $C[i,j] = d(A(i), B(j))$, $0 \leq i \leq n$, $0 \leq j \leq m$. Елементи ове матрице задовољавају диференцну једначину

$$C[i,j] = \min\{C[i-1, j-1] + c(i,j), C[i, j-1] + 1, C[i-1, j] + 1\},$$

која њен произвољан елемент изражава преко три друга: изнад, лево и горе–лево:

$$\begin{array}{cccc} & & & j \\ & & & \downarrow \\ & & & \vdots \\ & \cdots & C[i-1, j-1] & C[i-1, j] \quad \ldots \\ i\rightarrow & \cdots & C[i, j-1] & C[i, j] \quad \ldots \end{array}$$

Ова једнакост омогућује израчунавање свих елемената матрице C, пошто се знају њена 0-та колона и 0-та врста. Тиме је решен проблем израчунавања едит растојања стрингова A и B: $d(A,B) = C[n,m]$. Описани алгоритам је пример стратегије динамичког програмирања.

Реализација. За израчунавања се користи матрица C димензије $(n+1) \times (m+1)$, при чему први, односно други индекс узимају вредности из опсега $[0,n]$, односно $[0,m]$. Нека $M[i,j]$ означава последњу едит операцију која доводи до минималне вредности $C[i,j]$. Довољно је запамтити само ту последњу едит операцију (брисање $A[i]$, уметање $B[j]$ или замена $A[i]$ са $B[j]$), јер се до комплетног низа едит операција долази пролазећи елементе матрице "уназад", полазећи од $M[n,m]$. За израчунавање $C[i,j]$, потребно је знати $C[i-1,j]$, $C[i,j-1]$ и $C[i-1,j-1]$. Последња промена $M[i,j]$ одређена је тиме која од три могућности даје минималну вредност за $C[i,j]$; избор није увек једнозначан!

Edit_rastojanje(A,n,B,m)
улаз: A (стринг дужине n) и B (стринг дужине m)
излаз: C (матрица растојања подстрингова A и B)
1 **for** $i \leftarrow 0$ **to** n **do** $C[i,0] \leftarrow i$
2 **for** $j \leftarrow 0$ **to** m **do** $C[0,j] \leftarrow j$
3 **for** $i \leftarrow 1$ **to** n **do**
4 **for** $j \leftarrow 1$ **to** m **do**
5 $x \leftarrow C[i-1,j]+1$
6 $y \leftarrow C[i,j-1]+1$
7 **if** $a_i = b_j$ **then**
8 $z \leftarrow C[i-1,j-1]$
9 **else**
10 $z \leftarrow C[i-1,j-1]+1$
11 $C[i,j] \leftarrow \min(x,y,z)$ {изабрана едит операција памти се у $M[i,j]$}

Пример 5.1. Размотримо рад алгоритма на горе наведеном примеру стрингова $A = abbc$ и $B = babb$. Матрица растојања подстрингова A и B, заједно са стрелицама које приказују могуће путеве добијања $C[i,j]$ ($\searrow$: замена $A[i]$ са $B[j]$, $\rightarrow$: уметање $B[j]$ или $\downarrow$: брисање $A[i]$), дата је следећом табелом:

		B	0		1		2		3		4
A					b		a		b		b
0			**0**	$\rightarrow$	**1**	$\rightarrow$	2	$\rightarrow$	3	$\rightarrow$	4
1	a		$\downarrow$ 1	$\searrow$	1	$\searrow$	**1**	$\rightarrow$	2	$\rightarrow$	3
2	b		$\downarrow$ 2	$\searrow$	1	$\searrow$ $\rightarrow$	$\downarrow$ 2	$\searrow$	**1**	$\searrow$ $\rightarrow$	2
3	b		$\downarrow$ 3	$\searrow$	$\downarrow$ 2	$\searrow$	2	$\searrow$	$\downarrow$ 2	$\searrow$	**1**
4	c		$\downarrow$ 4		3	$\searrow$	$\downarrow$ 3	$\searrow$	$\downarrow$ 3		$\downarrow$ **2**

У доњем десном углу табеле проналазимо $d(A,B) = 2$. До доњег десног угла долази се само из поља изнад њега операцијом $\downarrow$. На сличан начин идући уназад, видимо да се до горњег левог угла табеле уназад може доћи

само на један начин, низом операција (уназад) $\downarrow\searrow\searrow\searrow\rightarrow$. Према томе, у овом примеру постоји само један низ од две едит трансформације који преводи $A = abbc$ у $B = babb$.

Сложеност. Сваки елемент матрице C израчунава се за константно време, па је временска сложеност алгоритма $O(nm)$. Проблем са овом верзијом алгоритма је у томе што је и његова просторна сложеност $O(nm)$. Ако се матрица C попуњава врсту по врсту, пошто врста зависи само од претходне, овај алгоритам може се прерадити тако да му просторна сложеност буде $O(m)$.

Примедба. Динамичко програмирање је корисно у ситуацији кад се решење проблема изражава преко неколико решења нешто мањих проблема. Коришћење табеле за смештање претходних резултата је уобичајено код динамичког програмирања. Табела се обично пролази неким редоследом (обично по врстама), па је сложеност алгоритма најмање квадратна.

Варијанта овог проблема је трансформација стринга A у стринг B, тако да уметање испред или иза A има цену 0. Тиме се постиже да се рачунају само едит операције које A претварају у неки подстринг B. На пример, "дабар" се на овај начин може трансформисати у "абракадабра" тако што се испред, односно иза "дабар" уметне "абрака", односно "а", а из речи "дабар" обрише се само једно слово, друго "а". Пошто се једино ово брисање рачуна, види се да у тексту "абракадабра" постоји реч "дабр" која је на растојању само 1 од речи "дабар".

Како се овај проблем може ефикасно решити? Уметањима испред речи у табели C одговара прва врста табеле: уметање испред речи је претварање њеног празног префикса у непразан префикс текста. Према томе, кретање десно у првој врсти табеле C има цену 0. На сличан начин, кретању десно у последњем реду табеле C одговара дописивање иза краја речи A, па и том кретању одговара цена 0. У свим осталим ситуацијама елементи табеле C рачунају се на описани начин. Према томе, важи:

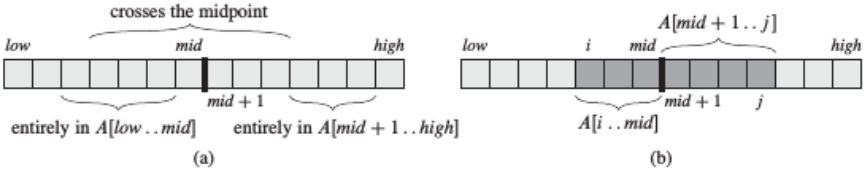
$C[0, j] = 0,\ j = 0, 1, \ldots, m,$
$C[i, 0] = i,\ i = 0, 1, \ldots, n$ и

$$C[i,j] = \begin{cases} \min\{C[i-1,j-1] + c(i,j), C[i,j-1]+1, C[i-1,j]+1\}, & 1 \le i < n \\ \min\{C[i-1,j-1] + c(i,j), C[i,j-1], C[i-1,j]+1\}, & i = n \end{cases}$$

5.3 Стратегија разлагања

Проблем налажења максималног узастопног подниза датог низа $A[1..n]$ (видети поглавље 4.8) може се ефикасно решити и применом стратегије разлагања. Нека је потребно одредити максимални подниз низа $A[levi..desni]$. Први корак је поделити низ на два подниза приближно једнаке дужине $A[levi..srednji]$ и $A[srednji + 1..desni]$, где је $srednji = \lfloor (levi + desni)/2 \rfloor$. Било који узастопни подниз $A[i..j]$ низа $A[levi..desni]$ може да заузме један од наредне три врсте положаја (видети слику 5.4):

- може цео да лежи у поднизу $A[levi..srednji]$, ако је $levi \le u \le i \le srednji$,
- може цео да лежи у поднизу $A[srednji + 1..desni]$, ако је $srednji < i \le j \le desni$,
- може да делом лежи у левом, а делом у десном поднизу, ако је $levi \le i \le srednji < j \le desni$.



Слика 5.4: (а) Могући положаји поднизова $A[levi..desni]$; цео у $A[levi..srednji]$, цео у $A[srednji+1..desni]$ или се протеже кроз оба подниза. (б) Било који подниз $A[levi..srednji]$ који садржи индексе $srednji$ и $srednji + 1$ састоји се од два подниза $A[i..srednji]$ и $A[srednji + 1..desni]$, где је $levi \le i \le srednji$ и $srednji < j \le desni$

Према томе, и максимални подниз може да заузима један од наведене три врсте положаја. Максимални подниз низа $A[levi..desni]$ је или максимални подниз низа $A[levi..srednji]$ или максимални подниз низа $A[srednji+1..desni]$ или садржи два средња индекса. Према томе, потребно је одредити рекурзивно максималне поднизове у левом и десном поднизу, одредити највећи подниз који садржи индексе $srednji$ и $srednji + 1$ и од те три вредности изабрати највећу.

Максимални подниз који пресеца индекс $srednji$ може се одредити за време $O(desni - levi + 1)$. Као што се може видети на слици 5.4(б), такав подниз је конкатенација два подниза $A[i..srednji]$ и $A[srednji+1..j]$, где је $levi \le i \le srednji$ и $srednji < j \le desni$. Према томе, довољно је одредити

највеће поднизове облика $A[i..srednji]$ и $A[srednji+1..j]$ и сабрати њихове збирове. Тај посао се може обавити алгоритмом у наставку:

$Max\_srednji\_zbir(A, levi, srednji, desni)$
улаз: A (низ у границама од $levi$ до $desni$) и $srednji$ (индекс средњег елемента)
излаз: max_leva и max_desna (индекси почетка и краја максималног средњег низа) и његова сума

```
1   leva_suma ← −∞
2   suma ← 0
3   for i ← srednji downto levi do
4     suma ← suma + A[i]
5     if suma > leva_suma then
6       leva_suma ← suma
7       max_leva ← i
8   desna_suma ← −∞
9   suma ← 0
10  for j ← srednji + 1 to desni do
11    suma ← suma + A[j]
12    if suma > desna_suma then
13      desna_suma ← suma
14      max_desna ← j
15  return (max_leva, max_desna, leva_suma + desna_suma)
```

Алгоритам налази посебно највећи збир подниза $A[max\_levi..srednji]$ међу поднизовима $A[i..srednji]$ и највећи збир подниза $A[srednji{+}1..max\_desni]$ међу поднизовима $A[srednji+1..j]$.

Ако је дужина низа $A[levi..desni]$ једнака n онда је сложеност алгоритма $Max\_srednji\_zbir$ једнака $\Theta(n)$, јер је збир бројева пролаза кроз две **for** петље $\Theta(n)$, а тела обе петље се извршавају за време $O(1)$. Рекурзивни алгоритам за налажење максималног узастопног подниза описан је наредним кодом:

$Max\_podniz(A, levi, desni)$
улаз: A (низ у границама од $levi$ до $desni$
излаз: индекси почетка и краја максималног подниза и његова вредност

```
1  if levi = desni then
2    return (levi, desni, A[levi]) {базни случај: само један елемент}
3  else
4    srednji ← ⌊(levi + desni)/2⌋
5    (levi1, desni1, suma1) ← Max_podniz(A, levi, srednji)
6    (levi2, desni2, suma2) ← Max_podniz(A, srednji + 1, desni)
7    (levi3, desni3, suma3) ← Max_srednji_zbir(A, levi, srednji, desni)
8    if suma1 ≥ suma2 and suma1 ≥ suma3 then
9      return (levi1, desni1, suma1)
10   elseif suma2 ≥ suma1 and suma2 ≥ suma3 then
11     return (levi2, desni2, suma2)
12   else
13     return (levi3, desni3, suma3)
```

Позив $Max\_podniz(A, 1, n)$ проналази максимални узастопни подниз. Сложеност алгоритма $T(n)$ задовољава рекурентну релацију:

$$T(n) = 2T(n/2) + O(n)$$

чије је решење (према мастер теореми) $T(n) = O(n \log n)$. Видимо да је сложеност алгоритма већа од линеарне сложености инкременталног алгоритма из поглавља 4.8. Проблем је у томе што је сложеност комбиновања резултата два рекурзивна позива $\Theta(n)$. Испоставља се да појачавањем индуктивне хипотезе (о томе шта је резултат рекурзивних позива) може добити ефикаснији алгоритам, линеарне сложености. Са идејом сличном оној из поглавља 4.8 можемо да претпоставимо да се из рекурзивних позива за леви, односно десни подниз добијају максимални збир, префикс и суфикс M_1, P_1, S_1, односно M_2, P_2, S_2. Такође, потребно је одредити и збир свих елемената у левом и десном поднизу и означимо те вредности са Z_1 и Z_2. Ови збирови су потребни у ситуацији када се максимални префикс целог низа састоји од свих елемената левог подниза на који се надовеже максимални префикс десног низа (и аналогно за максимални суфикс целог низа). На основу ових вредности могу се израчунати максимални збир M, префикс P, суфикс S и укупан збир свих елемената низа Z.

$$M = \max\{M_1, M_2, S_1 + P_2\}$$

$$P = \max\{P_1, Z_1 + P_2\}$$

$$S = \max\{S_2, Z_2 + S_1\}$$

$$Z = Z_1 + Z_2$$

Сложеност овог алгоритма задовољава рекурентну релацију $T(n) = 2T(n/2) + O(1)$, чије је решење $T(n) = O(n)$.

5.4 Пробабилистички алгоритми

Алгоритми које смо до сада разматрали били су детерминистички — сваки наредни корак је унапред одређен. Кад се детерминистички алгоритам извршава два пута са истим улазом, начин извршавања је оба пута исти, као и добијени излази. Пробабилистички алгоритми су другачији. Они садрже кораке који зависе не само од улаза, него и од неких случајних догађаја. Постоји много варијација пробабилистичких алгоритама. Овде ћемо размотрити две од њих.

5.4.1 Одређивање броја из горње половине

Претпоставимо да је дат скуп бројева $x_1, x_2, \ldots, x_n$ и да међу њима треба изабрати неки број из "горње половине", односно број који је већи или једнак од бар $n/2$ осталих. На пример, потребно је изабрати "доброг" студента, при чему је критеријум просечна оцена. Једна могућност је узети највећи број (који је увек у горњој половини). Већ смо видели да је за

одређивање максимума потребно $n-1$ упоређивања. Друга могућност је започети са извршавањем алгоритма за налажење максимума, и зауставити се кад се прође половина бројева. Број који је већи или једнак од једне половине бројева је сигурно у горњој половини. Алгоритам захтева око $n/2$ упоређивања. Може ли се овај посао обавити ефикасније? Није тешко показати да је немогуће гарантовати да број припада горњој половини ако је извршено мање од $n/2$ упоређивања. Према томе, описани алгоритам је оптималан.

Овај алгоритам је, међутим, оптималан само ако инсистирамо на *гаранцији*. У много случајева гаранција није неопходна, довољна је пристојна вероватноћа да је решење тачно. На пример, код хеш табела није могуће гарантовати да до колизија неће доћи, али постоји начин за решавање проблема изазваних појавом колизија (операције са хеш табелама се такође могу сматрати пробабилистичким алгоритмима). Ако одустанемо од гаранције, онда постоји бољи алгоритам за налажење елемента из горње половине. Изаберимо на случајан начин два броја x_i и x_j из скупа, тако да је $i \neq j$. Претпоставимо да је $x_i \geq x_j$. Вероватноћа да случајно изабрани број из скупа припада горњој половини је бар $1/2$ (она ће бити већа од $1/2$ ако је више бројева једнако медијани). Вероватноћа да ни један од бројева x_i, x_j не припада горњој половини је највише $1/4$. Због $x_i \geq x_j$ ова вероватноћа једнака је вероватноћи да x_i не припада горњој половини. Према томе, вероватноћа да x_i припада горњој половини је бар $3/4$.

Вероватноћа $3/4$ да је добијени резултат тачан обично није довољна. Међутим, описани приступ може се уопштити. На случајан начин бирамо k бројева из скупа и одређујемо највећи од њих. На исти начин као у специјалном случају, закључујемо да највећи од k елемената припада горњој половини са вероватноћом најмање $1-2^{-k}$ (он не припада горњој половини акко горњој половини не припада ни један од изабраних бројева, односно са вероватноћом највише 2^{-k}). На пример, ако је $k=10$, вероватноћа успеха је приближно 0.999, а за $k=20$ та вероватноћа је око 0.999999. Ако је пак $k=100$, онда је вероватноћа грешке за све практичне сврхе занемарљива. Имамо дакле алгоритам који бира број из горње половине са произвољно великом вероватноћом, а који извршава највише 100 упоређивања независно од величине улаза. При томе се претпоставља да се случајни избор једног броја може извршити за константно време; генерисање случајних бројева биће размотрено у одељку 5.4.2.

За овакав алгоритам обично се каже да је **Монте Карло** алгоритам. Нетачан резултат може се добити са јако малом вероватноћом, али је време извршавања овог Монте Карло алгоритма боље него за најбољи детерминистички алгоритам. Други тип пробабилистичког алгоритма је онај који никад не даје погрешан резултат, али му време извршавања није гарантовано. Овакав алгоритам се може извршити брзо, али се може извршавати и произвољно дуго. Овај тип алгоритма, који са обично зове **Лас Вегас**, користан је ако му је *очекивано* време извршавања мало. У одељку 5.4.3 биће размотрен Лас Вегас алгоритам који решава један проблем бојења елемената скупа.

Идеја пробабилистичких алгоритама тесно је повезана са извођењем доказа. Коришћење вероватноће за доказивање комбинаторних тврђења је моћна техника. У основи, ако се докаже да неки објекат из скупа објеката има вероватноћу већу од нуле, онда је то индиректан доказ да постоји

објекат са тим особинама. Ова идеја може се искористити за конструкцију пробабилистичког алгоритма. Претпоставимо да тражимо објекат са неким особинама, а знамо да ако генеришемо случајни објекат, он ће задовољавати жељени услов са вероватноћом већом од нуле (што је пробабилистички доказ да тражени објекат постоји). Ми затим пратимо пробабилистички доказ, генеришући случајне догађаје кад је потребно, и на крају налазимо жељени објекат се неком позитивном вероватноћом. Овај поступак може се понављати више пута, све до успешног проналажења жељеног објекта.

5.4.2 Случајни бројеви

Битан елемент пробабилистичких алгоритама је генерисање случајних бројева. Потребно је имати ефикасне методе за решавање овог проблема. Детерминистичка процедура генерише бројеве на фиксирани начин, па тако добијени бројеви не могу бити *случајни* у правом смислу те речи. Ти бројеви су међусобно повезани на сасвим одређени начин. На срећу, то у пракси није велики проблем: довољно је користити тзв. **псеудослучајне бројеве**. Ти бројеви генеришу се детерминистичком процедуром (па дакле нису прави случајни бројеви), али процедура генерисања је довољно сложена, да друге апликације не "осећају" међузависности између тих бројева.

Овде нећемо детаљније разматрати добијање случајних бројева. Један од начина за добијање псеудослучајних бројева је **линеарни конгруентни метод**. Први корак је избор целог броја X_0 као првог члана низа, случајног броја изабраног на неки независан начин (на пример, текуће време у микросекундама). Остали бројеви израчунавају се на основу диференцне једначине $X_n = aX_{n-1} + b \pmod{m}$, где су a, b и m константе, које се морају пажљиво изабрати. И поред пажљивог избора, овакви низови нису довољно квалитетни ("случајни"). Алтернатива су диференцне једначине вишег реда (са зависношћу од више претходних чланова). На овај начин добија се низ бројева из опсега од 0 до $m - 1$. Ако су потребни случајни бројеви из опсега од 0 до 1, онда се чланови овог низа могу поделити са m.

5.4.3 Један проблем са бојењем

Нека је S скуп од n елемената, и нека је $S_1, S_2, \ldots S_k$ колекција сачињена од k његових различитих подскупова, од којих сваки садржи тачно r елемената, при чему је $k \leq 2^{r-2}$.

Проблем. Обојити сваки елемент скупа S једном од две боје, црвеном или плавом, тако да сваки подскуп S_i садржи бар један плави и бар један црвени елемент.

Бојење које задовољава тај услов зваћемо *исправним бојењем*. Испоставља се да под наведеним условима исправно бојење увек постоји. Једноставан пробабилистички алгоритам добија се преправком пробабилистичког доказа постојања таквог бојења:

> *Обојити сваки елемената S случајно изабраном бојом, плавом или црвеном, независно од бојења осталих елемената.*

Јасно је да овај алгоритам не даје увек исправно бојење. Израчунаћемо вероватноћу неуспеха. Вероватноћа да су сви елементи S_i обојени црвено је 2^{-r}, а вероватноћа да су сви обојени истом бојом (црвеном или плавом) је $2 \cdot 2^{-r} = 2^{1-r}$. Случајни догађај (подскуп скупа свих 2^n случајних бојења — елементарних догађаја) A: "неки од скупова S_i је неисправно обојен" је унија свих случајних догађаја A_i: "скуп S_i је неисправно обојен", за $i = 1, 2, \ldots, k$, па важи неједнакост

$$P(A) \leq \sum_{i=1}^{k} P(A_i) = \sum_{i=1}^{k} 2^{1-r} = k2^{1-r} \leq 2^{r-2}2^{1-r} = \frac{1}{2}.$$

Тиме је доказано да исправно бојење постоји (у противном би вероватноћа неисправног бојења била тачно 1). Поред тога, види се да је овај пробабилистички алгоритам добар. Исправност задатог бојења лако се проверава: проверавају се елементи сваког подскупа док се не пронађу два различито обојена елемента. Вероватноћа успешног бојења $p = 1 - P(A)$ је бар $1/2$. Ако се у једном покушају не добије неисправно бојење, поступак (бојење) се понавља. Очекивани број покушаја бојења је мањи или једнак од два. Заиста, вероватноћа да се исправно бојење пронађе у j-том покушају је $(1-p)^{j-1}p$, па је математичко очекивање броја покушаја

$$\sum_{j=1}^{\infty} j(1-p)^{j-1}p = \frac{1}{p} \leq 2.$$

Описани алгоритам бојења је очигледно Лас Вегас алгоритам, јер се бојења проверавају једно за другим, а са тражењем се завршава кад се наиђе на исправно бојење. Не постоји гаранција успешног бојења у било ком фиксираном броју покушаја, али је овај алгоритам у пракси ипак врло ефикасан.

Глава 6

Структуре података

До сада смо се упознали са великим бројем различитих структура података. Стек и ред су структуре података код којих је унапред одређен елемент који се избацује: код стека то је елемент који је последњи додат у структуру, а код реда елемент који је први додат. Обе ове структуре ефикасно имплементирају операције за додавање новог елемента у структуру и избацивање елемента из структуре.

Размотрили смо, такође, бинарна стабла претраге као пример апстрактног типа података речник: операције које подржава речник су додавање новог елемента, изацивање произвољног елемента из структуре и испитивање присуства неког елемента у структури података. Као посебан пример видели смо АВЛ стабла код којих се све наведене операције изводе ефикасно.

Видели смо и хип као једну имплементацију реда са приоритетом. Ред са приоритетом подржава две операције: додавање елемента у структуру и изацивање елемента са највећим или најмањим приоритетом из структуре. Као пример структуре података разматрали смо и хеш табеле које представљају још једну могућу реализацију речника код које се у просеку операције изводе доста ефикасно.

У наставку ћемо размотрити нову структуру података, која се користи за ефикасно извођење две операције са фамилијама дисјунктних скупова.

6.1 Проблем формирања унија

6.1.1 Структура података за дисјунктне скупове

Проблем формирања унија може да послужи као пример како добро изабрана структура података може да побољша ефикасност алгоритма. Нека је дато n елемената $x_1, x_2, \ldots, x_n$, који су подељени у дисјунктне подскупове. На почетку су подскупови једночлани. Над елементима и подскуповима могу се произвољним редоследом извршавати следеће две операције:

- **podskup**(i): даје име подскупа који садржи елемент x_i;
- **unija**(A, B): формира унију подскупова A и B, подскуп са јединственим именом.

Циљ је наћи структуру података која омогућује што ефикасније извршавање ове две операције.

Први кандидат за жељену структуру података је вектор. Пошто су сви елементи познати, могу се сместити у вектор X дужине n. Најједноставније решење постављеног проблема је сместити у $X[i]$ име подскупа који садржи елемент x_i. У оваквој реализацији налажење подскупа коме припада елемент x_i је тривијално. Међутим, формирање уније два подскупа је сложеније. Претпоставимо да је **unija**(A, B) нови подскуп са именом A. Тада свим елементима вектора X из подскупа B треба променити име у A, па је сложеност ове операције у најгорем случају $O(n)$.

Пример 6.1. Систему дисјунктних подскупова $\{1\}, \{2, 4, 6\}, \{3, 5\}, \{7\}$ одговара низ

i	1	2	3	4	5	6	7
$X[i]$	1	2	3	2	3	2	7

Да би се формирала унија $\{2, 4, 6\} \cup \{3, 5\}$ требало би $X[3]$ и $X[5]$ заменити са 2.

Могућ је и други приступ овом проблему, заснован на индиректном адресирању, при чему је налажење уније битно једноставније. Он такође користи целобројни низ X дужине n. За представљање подскупа $\{i_1, i_2, ..., i_k\}$ користе се локације $i_1, i_2, ..., i_k$ у низу X. На свакој од тих локација, сем једној, записује се индекс неког другог елемента подскупа - оца тог елемента у на тај начин представљеном стаблу; на локацији која одговара једном елементу записује се 0 - тај елемент је корен стабла које одговара подскупу.

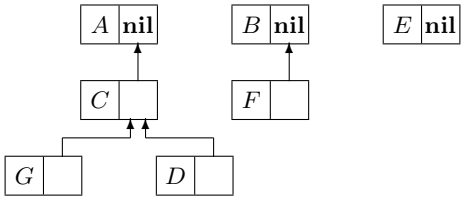
На пример, споменути систем дисјунктних скупова може се представити низом:

i	1	2	3	4	5	6	7
$X[i]$	0	6	0	6	3	0	0

Корени стабала – подскупова су елементи 1, 3, 6 и 7 којима одговарају вредности 0 у низу X; подскуп $\{2, 4, 6\}$ је стабло са три чвора, од којих је 6 корен, а 2 и 4 су његови синови; дакле 2, 4 и 6 су у подскупу 6.

На почетку су сви подскупови једночлани, сто значи да су сви елементи корени стабала са по једним чвором; сви елементи низа X су дакле нуле. Претпоставимо да су A, B два елемента који су представници својих подскупова, тј. $X[A] = X[B] = 0$. Операција **unija**(A, B) изводи се тако што се $X[A]$ замени са B или обрнуто тако што се $X[B]$ замени са A. После неколико формирања унија, вектор X имплицитно задаје неколико стабала; на слици 6.1 та стабла су приказана као експлицитно представљана стабла. Свако стабло одговара подскупу, а сваки чвор елементу. Елемент у корену стабла служи као име подскупа. Да би се пронашло име подскупа коме припада елемент G, треба пратити ланац показивача до корена стабла, слога са показивачем **nil**. Ово подсећа на ситуацију кад неко промени адресу: уместо да свима јавља да је променио адресу, он на старој адреси оставља своју нову адресу (са молбом да му достављају приспелу пошту)

— што је једноставније. Наравно, налажење праве адресе је сада нешто сложеније, односно операција налажења подскупа коме елемент припада је мање ефикасна (нарочито ако су стабла која представљају подскупове издужена).



Слика 6.1: Структура података погодна за представљање дисјунктних подскупова.

Идеја на којој се заснива ефикасна структура за налажење унија је да се стабла уравнотеже, односно скрате. Унија подскупова A и B на слици 6.1 може се формирати усмеравањем показивача A ка B, или усмеравањем показивача B ка A. Очигледно је да се у другом случају добија уравнотеженије стабло. Испоставља се да следећа хеуристика решава проблем: показивач корена стабла са мање елемената (односно произвољног од два стабла ако им је број елемената једнак) треба усмерити ка корену стабла са већим бројем елемената. Јасно је да при формирању сваке нове уније успут треба израчунавати и број елемената новог подскупа. Тај податак се смешта као део слога у корену стабла. Применом овог правила се при формирању унија постиже да је висина стабла са m елемената увек мања или једнака од $\log_2 m$, што показује следећа теорема.

Теорема 6.1. *Ако се при формирању унија користи уравнотежавање, свако стабло висине h садржи бар 2^h елемената.*

Доказ. Доказ се изводи индукцијом по броју формирања унија. Тврђење је очигледно тачно за прву унију: добија се стабло висине један са два елемента. Посматрајмо унију подскупова A и B. Нека су висине одговарајућих стабала $h(A)$ и $h(B)$ и нека су бројеви чворова у њима $n(A)$, $n(B)$; без смањења општости може се претпоставити да је $n(A) \geq n(B)$. Висина комбинованог стабла једнака је $\max\{h(A), h(B)+1\}$ већем од бројева $h(A)$ и $h(B)+1$. Ако је висина једнака $h(a)$, онда је тврђење тачно, јер ново стабло има за исту висину више елемената од стабла A. У противном, ново стабло има бар два пута више елемената од $n(B)$ (због $n(A) \geq n(B)$), а висина му је за један већа од $h(B)$, па је тврђење теореме такође тачно.

Последица ове теореме је да се при тражењу подскупа коме припада елемент никад не пролази више од $\log_2 n$ показивача. Унија подскупова се формира за константно време. Према томе, за извршење низа од $m \geq n$ операција прве или друге врсте, број корака је највише $O(m \log n)$. Роберт Тарџан (Tarjan) био је први који је дискутовао горње границе временске сложености низа операција и он је осмислио како се ефикасност ове структуре може даље побољшати.

Посматрајмо још једном аналогију са преусмеравањем пошиљки. У случају вишеструке промене адресе, пошиљка ће ићи од једне до друге

адресе, док не доспе на коначно одредиште. Разумна идеја је да се по испоруци пошиљке уназад, на све успутне адресе, достави последња адреса. Дакле, кад се при одређивању подскупа неког елемента пређе пут до корена стабла, тај пут се још једном пролази у обратном смеру, а показивачи свих успутних чворова преусмеравају се ка корену. Сложеност тражења тиме је повећана највише за константни фактор (тј. асимптотска сложеност тражења се не мења), а постиже се скраћивање путева при каснијим тражењима подскупа за све успутне чворове.

Глава 7

Графовски алгоритми

7.1 Увод

У овом поглављу разматраћемо компликованије односе између објеката за чије моделирање користимо графове. Графови могу да моделирају много ситуација, и користе се у различитим областима, од картографије до социјалне психологије. Приказаћемо више важних основних алгоритама за обраду графова и за израчунавање неких њихових карактеристика.

Погледајмо најпре неке примере моделирања помоћу графова.

1. Налажење доброг пута до неког одредишта у граду је графовски проблем. Улице одговарају гранама (усмерене гране у случају једносмерних улица), а раскрснице чворовима. Сваком чвору и свакој грани (делу улице) може се придружити време потребно за пролазак, па се проблем своди на налажење "најбржег" пута између два чвора.

2. Процес извршавања неког програма може се поделити на стања. Из сваког стања може се наставити на неколико начина. Нека стања сматрају се непожељним. Проблем утврђивања која стања могу довести у непожељна стања је графовски проблем у коме стањима одговарају чворови, а гране указују на могуће преласке из једног у друго стање.

3. Прављење распореда часова на факултету може се посматрати као графовски проблем. Чворови одговарају групама, а две групе су повезане ако постоји студент који намерава да их обе похађа, или ако имају заједничког предавача. Проблем је како направити распоред по групама тако да се минимизирају конфликти. То је тежак проблем, за који није лако наћи добра решења.

За представљање графова најчешће ћемо користити представу листом повезаности, која је ефикаснија за ретке графове (односно графове са релативно малим бројем грана). Увешћемо најпре неопходне појмове. Граф $G = (V, E)$ састоји се од скупа V **чворова**, и скупа E **грана**. Свака грана одговара пару различитих чворова (понекад су дозвољене петље, односно гране које воде од чвора ка њему самом; ми ћемо претпостављати да оне нису дозвољене). Граф може бити **усмерен** или **неусмерен**. Гране усмереног графа су уређени парови чворова (који се зову крајеви

гране); редослед два чвора које повезује грана је битан. Гране усмереног графа цртају се као стрелице усмерене од једног чвора (почетка) ка другом чвору (крају). Гране неусмереног графа су неуређени парови. **Степен** $d(v)$ чвора v је број грана суседних чвору v (односно број грана које v повезују са неким другим чвором). У усмереном графу разликујемо **улазни степен**, број грана за које је чвор v крај, и **излазни степен**, број грана за које је чвор v почетак.

Пут од v_1 до v_k је низ чворова $v_1, v_2, \ldots, v_k$ повезаних гранама (v_1, v_2), (v_2, v_3), $\ldots$, (v_{k-1}, v_k); ове гране се обично такође сматрају делом пута. Пут је **прост**, ако се сваки чвор у њему појављује само једном. За чвор u се каже да је **достижан** из чвора v ако постоји пут (усмерен, односно неусмерен, зависно од графа) од v до u. По дефиницији је чвор v достижан из v. **Циклус** је пут чији се први и последњи чвор поклапају. Циклус је **прост** ако се, сем првог и последњег чвора, ни један други чвор у њему не појављује два пута. **Неусмерени облик** усмереног графа $G = (V, E)$ је исти граф, без смерова на гранама. За граф се каже да је **повезан** ако (у његовом неусмереном облику) постоји пут између произвољна два чвора. **Шума** је граф који (у свом неусмереном облику) не садржи циклусе. **Стабло** или **дрво** је повезана шума. **Коренско стабло** је усмерено стабло са једним посебно издвојеним чвором, који се зове **корен** при чему су све гране усмерене од корена.

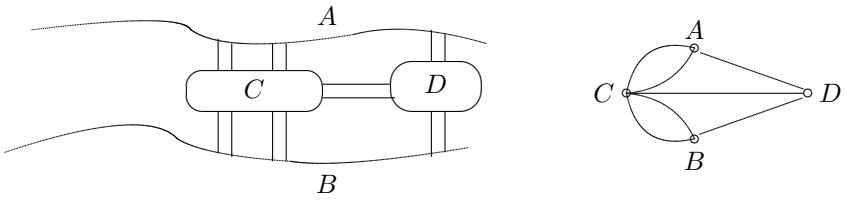
Граф $H = (U, F)$ је **подграф** графа $G = (V, E)$ ако је $U \subseteq V$ и $F \subseteq E$. **Повезујуће стабло** неусмереног графа G је његов подграф који је стабло и садржи све чворове G. **Индуковани подграф** графа $G = (V, E)$ је његов подграф $H = (U, F)$ такав да је $U \subseteq V$ и F садржи све гране из E чија су оба краја у U. Ако граф $G = (V, E)$ није повезан, онда се он може на јединствен начин разложити у скуп повезаних подграфова, који се зову **компоненте повезаности** графа G. Скуп чворова компоненте повезаности је класа еквиваленције у односу на релацију достижности за чворове: компоненту повезаности G којој припада чвор v чине сви чворови достижни из v. Компонента повезаности графа G је максимални повезани подграф G (односно такав подграф који није прави подграф ни једног другог повезаног подграфа G). **Повезујућа шума** неусмереног графа G је његов подграф који се састоји од повезујућих стабала компоненти повезаности G. **Бипартитни граф** је граф чији се чворови могу поделити на два дисјунктна подскупа тако да у графу постоје само гране између чворова из различитих подскупова. **Тежински граф** је граф чијим су гранама придружени реални бројеви (тежине, цене, дужине, енг. weight, cost, length).

Многе дефиниције за усмерене и неусмерене графове су сличне, изузев неких очигледних разлика. На пример, усмерени и неусмерени путеви дефинишу се на исти начин, сем што се код усмерених графова специфицирају и смерови грана. Значење таквих термина зависи од контекста; ако се, на пример, ради о путевима у усмереном графу, онда се мисли на усмерене путеве.

Размотрићемо најпре једноставан проблем, који се сматра првим проблемом теорије графова — обилазак Кенингсбершких мостова. Затим ћемо се бавити уређењем чворова графа, налажењем најкраћих путева у графу и другим проблемима.

7.2 Ојлерови графови

Појам Ојлерових графова у вези је са, како се сматра, првим решеним проблемом теорије графова. Швајцарски математичар Леонард Ојлер наишао је на следећи задатак 1736. године. Град Кенигсберг, данас Калињинград, лежи на обалама и на два острва на реци Прегел, као што је приказано на слици 7.1. Град је био повезан са седам мостова. Питање (које је мучило многе тадашње грађане Кенигсберга) било је да ли је могуће почети шетњу из било које тачке у граду и вратити се у полазну тачку, прелазећи при томе сваки мост тачно једном. Решење је добијено уопштавањем проблема. "Граф" на истој слици, бар што се овог проблема тиче, еквивалентан је са планом града (наводници су употребљени јер овај "граф" има вишеструке гране, па строго гледано по дефиницији и није граф; то је тзв. мултиграф). Проблем се може еквивалентно формулисати као следећи проблем из теорије графова: да ли је могуће у повезаном графу пронаћи циклус, који сваку грану садржи тачно једном (*Ојлеров циклус*). Или: да ли је могуће нацртати граф са слике 7.1 не дижући оловку са папира, тако да оловка свој пут заврши на месту са кога је и кренула. Ојлер је решио проблем, доказавши да је овакав обилазак могућ *ако и само ако* је граф повезан и сви његови чворови имају паран степен. Такви графови зову се **Ојлерови графови**. Пошто "граф" на слици 7.1 има чворове непарног степена, закључујемо да проблем Кенигсбершких мостова нема решење. Доказ теореме индукцијом, који следи, даје ефикасан алгоритам за налажење Ојлеровог циклуса у графу.



Слика 7.1: Проблем Кенигсбершких мостова, и одговарајући граф.

Проблем. У задатом неусмереном повезаном графу $G = (V, E)$ чији сви чворови имају паран степен, пронаћи затворени пут P, такав да се у њему свака грана из E појављује тачно једном.

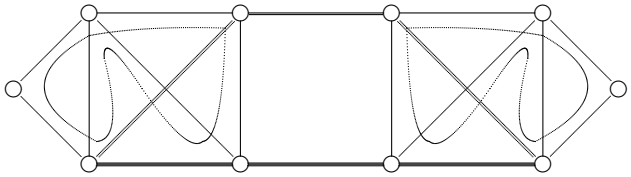
Лако је показати да ако у графу постоји Ојлеров циклус, онда сви чворови графа морају имати паран степен. За време обиласка затвореног циклуса, у сваки чвор се улази исто толико пута колико пута се из њега излази. Пошто се свака грана пролази тачно једном, број грана суседних произвољном чвору мора бити паран. Да бисмо индукцијом доказали да је услов довољан, морамо најпре да изаберемо параметар по коме ће бити изведена индукција. Тај избор треба да омогући смањивање проблема, без његове промене. Ако уклонимо чвор из графа, степени чворова у добијеном графу нису више сви парни. Требало би да уклонимо такав скуп грана S, да за сваки чвор v графа број грана из S суседних са v буде паран (макар и 0). Произвољан циклус задовољава овај услов, па се поставља питање да ли Ојлеров граф увек садржи циклус. Претпоставимо да смо започели

обилазак графа из произвољног чвора v произвољним редоследом. Сигурно је да ће се током обиласка раније или касније наићи на већ обиђени чвор, јер кад год уђемо у неки чвор, смањујемо његов степен за један, чинимо га непарним, па га увек можемо и напустити. Наравно, овакав обилазак не мора да садржи све гране графа. То показује да у сваком Ојлеровом графу мора да постоји неки циклус.

Сада можемо да формулишемо индуктивну хипотезу и докажемо теорему.

Индуктивна хипотеза. Повезани граф са $< m$ грана чији сви чворови имају паран степен, садржи Ојлеров циклус, који се може пронаћи.

Посматрајмо граф $G = (V, E)$ са m грана. Нека је P неки циклус у G, и нека је G' граф добијен уклањањем грана које чине P из графа G. Степени свих чворова у G' су парни, јер је број уклоњених грана суседних било ком чвору паран. Ипак се индуктивна хипотеза не може применити на граф G', јер он не мора бити повезан, видети пример на слици 7.2. Нека су G'_1, $G'_2, \ldots, G'_k$ компоненте повезаности графа G'. У свакој компоненти степени свих чворова су парни. Поред тога, број грана у свакој компоненти је мањи од m (њихов укупан број грана мањи је од m). Према томе, индуктивна хипотеза може се применити на све компоненте: у свакој компоненти G'_i постоји Ојлеров циклус P'_i, и ми знамо да га пронађемо. Потребно је сада све ове циклусе објединити у један Ојлеров циклус за граф G. Полазимо из било ког чвора циклуса P ("магистралног пута") све док не дођемо до неког чвора v_j који припада компоненти G'_j. Тада обилазимо компоненту G'_j циклусом P_j ("локалним путем") и враћамо се у чвор v_j. Настављајући на тај начин, обилазећи циклусе компоненти у тренутку наиласка на њих, на крају ћемо се вратити у полазну тачку. У том тренуку све гране графа G пређене су тачно једном, што значи да је конструисан Ојлеров циклус.



Слика 7.2: Пример конструкције Ојлеровог пута индукцијом. Пуном линијом извучене су гране помоћног циклуса. Избацивањем грана овог циклуса из графа, добија се граф са две компоненте повезаности.

7.3 Претрага у дубину усмерених графова – нови појмови и особине

У току извођења DFS усмереног графа $G = (V, E)$ чворовима $v \in V$ графа могу се приписати:

- боје $v.boja$
  - бела – за неозначене чворове
  - сива – за чвор из кога је покренут DFS

  - црна – за чворове из којих је завршен рекурзивни позив DFS.

- отац $v.\pi$ у DFS шуми. При томе је $v.\pi =$ **NULL** ако је чвор v корен неког од стабала DFS шуме графа G

- времена отварања $v.d$ и затварања $v.f$ чвора $v \in V$. При томе је *време* вредност посебног бројача, који пре покретања DFS има вредност 0, а инкрементира се сваки пут када се:

  - покрене рекурзивни позив из неког чвора $w \in V$; његова вредност се тада уписује у $w.d$
  - заврши рекурзивни позив из неког чвора $w \in V$; његова вредност се тада уписује у $w.f$

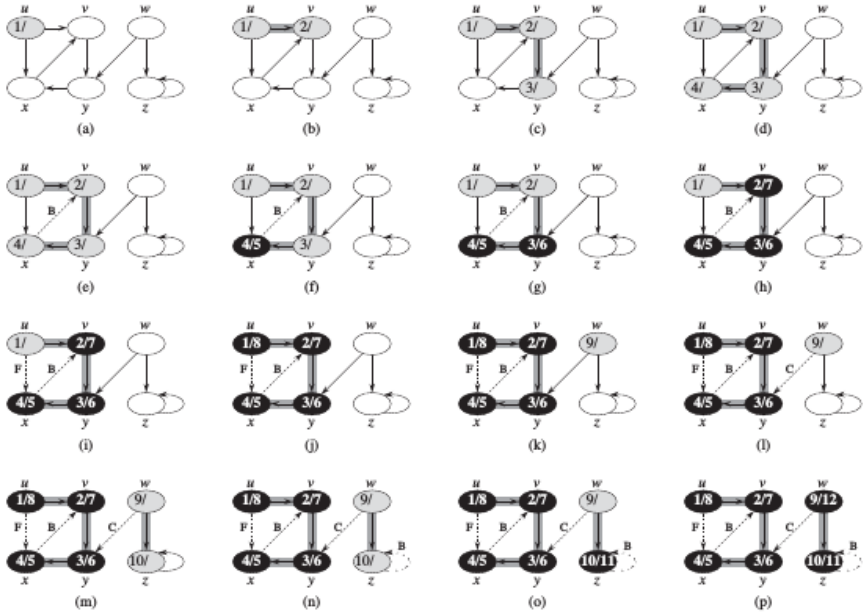
Јасно је да за сваки чвор $w \in V$ важи $1 \leq w.d < w.f \leq 2|V|$. Поред тога, чвор w је био до тренутка $w.d$, сив до тренутка $w.f$ и црн после тог тренутка.

Задатак: Ако се знају вредности $u.d$ и $u.f$ за све чворове $u \in V$, одредити одлазну и долазну нумерацију графа. Да ли познавање одлазне и долазне нумерације чворова једнозначно одређује њихова времена $u.d$ и $u.f$? Ако да, како?

У наставку је приказан кôд алгоритма DFS са овим допунама:

```
DFS(G)
улаз: G = (V, E) (усмерени граф)
излаз: извршен DFS обилазак стабла
1  for each v ∈ V do
2     v.boja ← bela
3     v.π ← NULL
4  t ← 0
5  for each v ∈ V
6     if v.boja = bela
7        DFS_obilazak(G, v)
```

```
DFS_obilazak(G, u)
улаз: G = (V, E) (усмерени граф), u (чвор графа)
излаз: извршен DFS обилазак стабла
1     t ← t + 1 {бели чвор u је управо отворен}
2     u.d ← t
3     u.boja ← siva
4     for each (u, v) ∈ E {откривамо грану (u, v)}
5        if v.boja = bela
6           v.π ← u
7           DFS_obilazak(G, v)
8     u.boja ← crna {чвор u зацрњујемо}
9     t ← t + 1
10    u.f ← t
```



Слика 7.3: Пример извршавања алгоритма DFS на усмереном графу. Гране графа у току претраге постају подебљане ако су гране стабла, односно испрекидане (остале). Гране које не припадају стаблу означене су словима B, C или F ако су редом повратне, попречне (улево), односно директне гране. Унутар чвора u уписани су бројеви $u.d$ и $u.f$.

Пример извршавања алгоритма DFS приказан је на слици 7.3.

Алгоритам DFS пре првог позива $DFS\_obilazak$ све чворове боји у бело и показивач на оца у шуми поставља на **NULL** . Глобални бројач t (време) поставља на 0. После сваког позива $DFS\_obilazak$ у линији 7 чвор u постаје корен стабла у DFS шуми. По завршетку алгоритма DFS свим чворовима u придружено је време отварања $u.d$ и затварања $u.f$.

На почетку позива $DFS\_obilazak(G, u)$ чвор u је бео; после инкрементирања бројача t (линија 1), у линији 2 се та вредност уписује као време отварања $u.d$, а у линији 3 чвор u постаје сив. У петљи у линијама 4–7 пролазе се редом сви суседи v чвора u и из њих се рекурзивно покреће $DFS\_obilazak$ ако је v у том тренутку бео (неозначен). По завршетку петље чвор u добија црну боју.

Запажа се да резултат претраге зависи од редоследа чворова у линији 5 алгоритма DFS, односно од редоследа суседа v чвора u у линији 4 алгоритма $DFS\_obilazak$ (тај редослед одређен је листом повезаности графа G).

Запажа се да је $u = v.\pi$ (односно u је отац v) акко је позив $DFS\_obilazak(G, v)$ покренут за време проласка кроз списак суседа чвора u у петљи. Поред тога, чвор v је потомак чвора u у шуми акко је v отворен у току временског интервала док је u сив.

Наредна теорема представља важну карактеризацију када је један чвор потомак другог чвора у DFS шуми.

Теорема [теорема о белим путевима]: У DFS шуми графа $G = (V, E)$

(усмереног или неусмереног) чвор v је потомак чвора u акко у тренутку $u.d$ покретања претраге из u постоји пут од u до v преко (искључиво) белих чворова.

Доказ: ($\Rightarrow$): У специјалном случају када је $v = u$, пут од v до u састоји се од само једног чвора u, који је још увек бео у тренутку приписивања времена $u.d$. Претпоставимо сада да је v прави потомак u у DFS шуми. Тада је $v.d > u.d$, односно у тренутку $u.d$ чвор v је бео. Ово важи за све потомке чвора u па важи за све чворове на јединственом путу кроз стабло шуме од u до v.

($\Leftarrow$): Претпоставимо да у тренутку $u.d$ постоји пут од белих чворова од u до v, а да v није потомак u у DFS шуми. Без смањења општости може се претпоставити да су сви чворови на том путу сем v потомци u (у противном се за v може узети први чвор на том путу који није потомак u). Нека w претходи чвору v на путу (специјално, w може да буде чвор u). Пошто (w, v) није грана стабла, за време претраге из чвора w у тренутку када се у петљи наиђе на чвор v, чвор v је већ означен и црн (зато што пре наиласка на чвор v приликом претраге из w морају бити завршене претраге свих подстабала са претходним суседима чвора w), тј. до њега води пут од w (а тиме и пут од u) преко грана стабла, односно v је потомак u, супротно претпоставци.

7.4 Тополошко сортирање

Претпоставимо да је задат скуп послова у вези са чијим редоследом извршавања постоје нека ограничења. Неки послови зависе од других, односно не могу се започети пре него што се ти други послови заврше. Све зависности су познате, а циљ је направити такав редослед извршавања послова који задовољава сва задата ограничења; другим речима, тражи се такав распоред за који важи да сваки посао започиње тек кад буду завршени сви послови од којих он зависи. Потребно је конструисати ефикасни алгоритам за формирање таквог распореда. Проблем се зове **тополошко сортирање**. Задатим пословима и њиховим међузависностима може се на природан начин придружити граф. Сваком послу придружује се чвор, а усмерена грана од посла x до посла y постоји ако се посао y не може започети пре завршетка посла x. Јасно је да граф мора бити ациклички (без усмерених циклуса), јер се у противном неки послови никада не би могли започети.

Проблем. У задатом усмереном ацикличком графу $G = (V, E)$ са n чворова нумерисати чворове бројевима од 1 до n, тако да ако је произвољан чвор v нумерисан са k, онда сви чворови до којих постоји усмерени пут из v имају број већи од k.

Природна је следећа индуктивна хипотеза.

Индуктивна хипотеза. Умемо да нумеришемо на захтевани начин чворове свих усмерених ацикличких графова са мање од n чворова.

Базни случај једног чвора, односно посла, је тривијалан. Као и обично, посматрајмо произвољни граф са n чворова, уклонимо један чвор, применимо индуктивну хипотезу и покушајмо да проширимо нумерацију на полазни граф. Имамо слободу избора n-тог чвора. Требало би га изабрати

тако да остатак посла буде што једноставнији. Потребно је нумерисати чворове. Који чвор је најлакше нумерисати? То је очигледно чвор (посао) који не зависи од других послова, односно чвор са улазним степеном нула; њему се може доделити број 1. Да ли се увек може пронаћи чвор са улазним степеном нула? Интуитивно се намеће потврдан одговор, јер се са означавањем негде мора започети. Следећа лема потврђује ову чињеницу.

Лема 7.1. *Усмерени ациклички граф увек има чвор са улазним степеном нула.*

Доказ. Ако би сви чворови графа имали позитивне улазне степене, могли бисмо да кренемо из неког чвора "уназад" пролазећи гране у супротном смеру. Међутим, број чворова у графу је коначан, па се у том обиласку мора у неком тренутку наићи на неки чвор по други пут, што значи да у графу постоји циклус. Ово је међутим супротно претпоставци да се ради о ацикличком графу. [1]

Претпоставимо да смо пронашли чвор са улазним степеном нула. Нумеришимо га са 1, уклонимо све гране које воде из њега, и нумеришимо остатак графа (који је такође ациклички) бројевима од 2 до n (према индуктивној хипотези они се могу нумерисати од 1 до $n-1$, а затим се сваки редни број може повећати за један). Види се да је после избора чвора са улазним степеном нула, остатак посла једноставан.

Реализација. Једини проблем при реализацији је како пронаћи чвор са улазним степеном нула и како поправити улазне степене чворова после уклањања гране. Сваком чвору може се придружити променљива (поље) $UlStepen$, тако да је на почетку $v.UlStepen$ једнако улазном степену чвора v. Почетне вредности промењивих $UlStepen$ могу се добити проласком кроз скуп свих грана произвољним редоследом (све гране су наведене у листи повезаности) и повећавањем за један $w.UlStepen$ сваки пут кад се наиђе на грану (v, w). Чворови са улазним степеном нула стављају се у ред (или стек, што је једнако добро). Према леми 7.1 у графу постоји бар један чвор v са улазним степеном нула. Чвор v се као први у реду лако проналази; он се уклања из реда. Затим се за сваку грану (v, w) која излази из v бројач $w.UlStepen$ смањује за један. Ако бројач при томе добије вредност нула, чвор w ставља се у ред. После уклањања чвора v граф остаје ациклички, па у њему према леми 7.1 поново постоји чвор са улазним степеном нула. Алгоритам завршава са радом кад ред који садржи чворове степена нула постане празан, јер су у том тренутку сви чворови нумерисани. Алгоритам је дат у наставку текста.

Top_sort(G)
улаз: $G = (V, E)$ (усмерени ациклички граф)
излаз: $v.Rb$ за сваки чвор v добија вредност у складу са тополошким сортирањем G.
1 Иницијализовати $v.UlStepen$ за све чворове {нпр. помоћу DFS}
2 $G\_rb \leftarrow 0$
3 $Q \leftarrow \emptyset$ {ред за чворове са улазним степеном 0 }
4 **for** $i \leftarrow 1$ **to** n **do**

[1] На сличан начин закључује се да у графу мора постојати и чвор са излазним степеном нула.

5 **if** $v_i.UlStepen = 0$ **then** уписати v_i у ред Q
6 **while** ред Q није празан
7 извадити чвор v из реда Q
8 $G\_rb \leftarrow G\_rb + 1$
9 $v\_Rb \leftarrow G\_rb$
10 **for** све гране (v, w) **do**
11 $w.UlStepen \leftarrow w.UlStepen - 1$
12 **if** $w.UlStepen = 0$ **then** уписати w у ред Q

Сложеност. Сложеност израчунавања почетних вредности промењљивих *UlStepen* је $O(|V| + |E|)$. За налажење чвора са улазним степеном нула потребно је константно време (приступ реду). Свака грана (v, w) разматра се тачно једном, у тренутку кад се v уклања из реда. Према томе, број промена вредности *UlStepen* једнак је броју грана у графу. Временска сложеност алгоритма је дакле $O(|V| + |E|)$, односно линеарна је функција од величине улаза.

7.5 Јако повезане компоненте усмереног графа

На скупу чворова усмереног графа $G = (V, E)$ може се дефинисати релација $\sim$ *обостране достижности*: $u \sim v$ ако је чвор u достижан из чвора v и чвор v достижан из чвора u. Ова релација је релација еквиваленције:

- рефлексивна – за сваки чвор $u \in V$ је $u \sim u$,
- симетрична – за свака два чвора $u, v \in V$ важи $u \sim v$ акко $v \sim u$,
- транзитивна – за свака три чвора $u, v, w \in V$ из $u \sim v$ и $v \sim w$ следи и $u \sim w$.

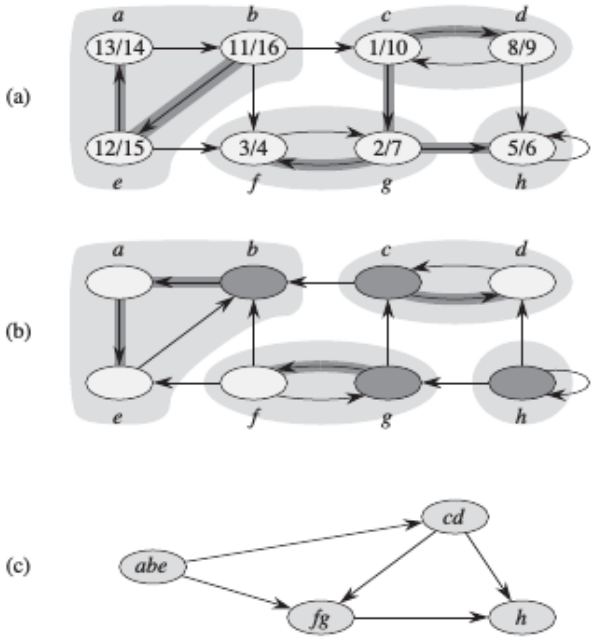
Ова релација разлаже скуп чворова V у класе еквиваленције које се зову **јаке компоненте повезаности** графа G (енгл. strong connected components, SCC). На слици 7.4(a) приказан је пример графа и његовог разлагања на јаке компоненте повезаности, односно простије компоненте. У наставку ћемо на јаке компоненте повезаности референсати као на само компоненте.

Јаким компонентама повезаности усмерених графова одговарају компоненте повезаности неусмерених графова. У алгоритму SCC за одређивање јаких компоненти повезаности графа G користи се транспоновани граф $G^T = (V, E^T)$ графа G. Граф G^T добија се усмеравањем свих грана G у супротном смеру, $E^T = \{(u, v)|(v, u) \in E\}$. Запажа се да графови G и G^T имају исте јаке компоненте повезаности: два чвора су обострано достижна у G акко су обострано достижна у G^T. На слици 7.4(b) приказан је транспоновани граф графа са слике 7.4(a), са шрафираним јаким компонентама повезаности.

У наставку је дат кôд алгоритма SCC за одређивање јаких компоненти повезаности графа $G = (V, E)$.

SCC(G)

1. покренути DFS(G) и тако одредити времена $u.f$ за све чворове $u \in V$



Слика 7.4: (a) Усмерени граф G у коме су јаке компоненте повезаности шрафиране. Чворови графа означени су временом отварања и затварања, а гране које припадају стаблима DFS шуме су подебљане. (b) Граф G^T, добијен транспоновањем графа G, са DFS шумом одређеном у линији 3 алгоритма SCC, у коме су гране шуме подебљане. Свака јака компонента повезаности одговара једном стаблу DFS шуме. Затамњени чворови b, c, g и h су корени стабала добијених претрагом G^T. (c) Ациклички граф компоненти G^{SCC} добијен поистовећивањем чворова у оквиру сваке јаке компоненте повезаности.

2. одредити граф G^T

3. покренути DFS(G^T), при чему се у основној петљи DFS чворови разматрају редом према опадајућим вредностима $u.f$ (из линије 1)

4. излистати чворове сваког стабла DFS шуме из линије 3 као јаке компоненте повезаности

Напомена: редослед према опадајућим вредностима $u.f$ одговара редоследу према опадајућим DFS одлазним бројевима.

Сложеност алгоритма SCC је очигледно $O(|V| + |E|)$.

Доказ коректности алгоритма SCC заснива се на особинама **графа компоненти** $G^{SCC} = (V^{SCC}, E^{SCC})$ дефинисаног на следећи начин. Нека су $C_1, C_2, \ldots, C_k$ јаке компоненте повезаности графа G. Скуп чворова $V^{SCC} = \{v_1, v_2, \ldots, v_k\}$ садржи по један чвор v_i за сваку јаку компоненту повезаности C_i графа G. Грана (v_i, v_j) је грана графа компоненти, тј. $(v_i, v_j) \in E^{SCC}$ ако у графу G постоји усмерена грана (x, y) за нека два чвора $x \in C_i, y \in$

C_j. Другим речима, G^{SCC} добија се скраћивањем (на нулу) свих грана које повезују два чвора у оквиру исте компоненте.

Основна особина графа компоненти је да је он ациклички, што је и тврђење следеће леме:

Лема: Нека су C и C' две различите јаке компоненте повезаности усмереног графа $G = (V, E)$, нека $u, v \in C, u', v' \in C$ и претпоставимо да у G постоји пут од u до u'. Тада у G не може да постоји пут од v' до v.

Доказ: Ако би у G постојао пут од v' до v, онда би постојали путеви од u до v' и од v' до u, тј. u и v' припадали би истој компоненти повезаности, супротно претпоставци.

Показаћемо сада да се разматрањем чворова у другој претрази редом према опадајућим временима затварања постиже тополошко сортирање графа компоненти.

Да се избегне забуна, времена $u.d$ и $u.f$ односе се на прву DFS претрагу у алгоритму SCC. Појмове времена отварања и времена затварања уопштићемо на скупове чворова: за скуп $U \subset V$ дефинишемо $d(U) = \min_{u \in U} u.d$, $f(U) = \max_{u \in U} u.f$. Другим речима, $d(U)$, односно $f(U)$, представљају прво време отварања, односно последње време затварања неког чвора из скупа чворова U.

Наредна лема повезује јако повезане компоненте графа и времена затварања чворова у току прве DFS претраге.

Лема: Нека су C и C' различите јаке компоненте повезаности графа $G = (V, E)$. Претпоставимо да постоји грана $(u, v) \in E$ таква да је $u \in C, v \in C'$. Тада је $f(C) > f(C')$.

Доказ: Размотримо два случаја у односу на то да ли се први отворени чвор налази у C или у C'.

- случај $d(C) < d(C')$. Нека је x чвор који је први отворен у C. У тренутку $x.d$ сви чворови у C и C' су бели (неозначени). У том тренутку у G постоји пут од x до свих чворова у C преко белих чворова. Пошто постоји грана $(u, v) \in E$, за сваки чвор $w \in C'$ постоји пут од x до w (преко u и v). Према теореми о белим путевима, сви чворови у C и C' су потомци x у DFS шуми графа. Због тога се претрага из x последња завршава, односно $x.f = f(C) > f(C')$.

- случај $d(C) > d(C')$. Нека је y чвор који је први отворен у C'. У тренутку $y.d$ сви чворови у C' су бели и у G постоје путеви преко белих чворова до свих чворова у C'. Према теореми о белим путевима сви чворови у C' су потомци чвора y у DFS стаблу, па је y последњи затворени чвор у C', односно $y.f = f(C')$. У тренутку $y.d$ сви чворови у C су бели. Због тога што постоји грана (u, v) из C у C', не може да постоји пут из C' у C, тј. ни један чвор у C није достижан из y. Према томе, у тренутку $y.f$ сви чворови у C су још увек бели. Због тога је за сваки чвор $w \in C$ $w.f > y.f$, па је $f(C) > f(C')$.

Наредна лема тврди да свака грана из G^T која повезује различите јаке компоненте повезаности иде од компоненте са мањим временом затварања ка компоненти са већим временом затварања.

Последица: Нека су C и C' различите јаке компоненте повезаности усмереног графа $G = (V, E)$. Претпоставимо да постоји грана $(u, v) \in E^T$, где је $u \in C, v \in C'$. Тада је $f(C) < f(C')$.

Ова последица је кључна за разумевање зашто алгоритам SCC ради коректно. Размотримо другу DFS претрагу примењену на граф G^T. Почиње се са јаком компонентом повезаности C са највећим временом затварања (завршетка) $f(C)$. Претрага почиње од неког чвора x и означава све чворове у C. На основу последице у графу G^T не може да постоји грана ка некој другој јакој компоненти повезаности. Према томе, DFS стабло са кореном у x садржи тачно чворове из скупа C. Пошто су означени сви чворови из C, претрага се наставља из неке друге јаке компоненте повезаности C', са највећим временом завршетка од свих компоненти различитих од C. Поново, та претрага означава све чворове из C'; међутим, према наведеној последици, из компоненте C' грана може да води само ка неком чвору у компоненти C, који је у том тренутку већ означен. Према томе, свако DFS стабло обухвата тачно једну компоненту; гране из те компоненте могу да воде само ка компонентама које су већ означене. Тиме је доказано да је алгоритам SCC коректан.

Теорема: Алгоритам SCC одређује јаке компоненте повезаности графа G.

Друга DFS претрага постаје јаснија ако се схвати као DFS графа компоненти $(G^T)^{SCC}$ графа G^T. Ако сваку компоненту посећену у току друге DFS претраге пресликамо у чвор графа $(G^T)^{SCC}$, друга DFS претрага означава чворове $(G^T)^{SCC}$ редоследом обрнутим од тополошког редоследа. Ако се обрну гране графа $(G^T)^{SCC}$, добија се граф $((G^T)^{SCC})^T$ који је једнак G^{SCC} (ово је потребно доказати), па друга DFS претрага пролази чворове G^{SCC} тополошким редоследом.

7.6 Најкраћи путеви из задатог чвора

У овом одељку бавићемо се **тежинским графовима**. Нека је $G = (V, E)$ усмерени граф са ненегативним **тежинама** придруженим гранама. Овде ћемо тежине звати *дужинама*, пошто оперишемо **дужинама путева** као збировима дужина грана (не бројевима грана). Ако је граф неусмерен, можемо га сматрати усмереним, при чему свакој његовој неусмереној грани одговарају две усмерене гране исте дужине, у оба смера. Према томе, разматрање у овом одељку односи се и на неусмерене графове.

Проблем. За дати усмерени граф $G = (V, E)$ и задати његов чвор v пронаћи најкраће путеве од v до свих осталих чворова у G.

Због једноставности ћемо се бавити само налажењем дужина најкраћих путева. Алгоритми се могу проширити тако да проналазе и саме најкраће путеве. Постоји много ситуација у којима се појављује овај проблем. На пример, граф може одговарати ауто–карти: чворови су градови, а дужине грана су дужине директних путева између градова (или време потребно да се тај пут пређе, или изгради, итд, зависно од проблема).

7.6.1 Ациклички случај

Претпоставимо најпре да је граф G ациклички. У том случају проблем је лакши, и његово решење помоћи ће нам да га решимо у општем случају. Покушаћемо индукцијом по броју чворова. Базни случај је тривијалан.

Нека је $|V| = n$. Можемо да искористимо тополошко сортирање графа из претходног одељка. Ако је редни број чвора v једнак k, онда се чворови са редним бројевима мањим од k не морају разматрати: не постоји начин да се до њих дође из v. Поред тога, редослед добијен тополошким сортирањем је погодан за примену индукције. Посматрајмо последњи чвор, односно чвор z са редним бројем n. Претпоставимо (индуктивна хипотеза) да знамо најкраће путеве од v до свих осталих чворова, сем до z. Означимо дужину најкраћег пута од v до w са $w.SP$. Да бисмо одредили $z.SP$, довољно је да проверимо само оне чворове w из којих постоји грана до z. Пошто се најкраћи путеви до осталих чворова већ знају, $z.SP$ једнако је минимуму збира $w.SP + dužina(w, z)$, по свим чворовима w из којих води грана до z. Да ли је тиме проблем решен? Питање је да ли додавање чвора z може да скрати пут до неког другог чвора. Међутим, пошто је z последњи чвор у тополошком редоследу, ни један други чвор није достижан из z, па се дужине осталих најкраћих путева не мењају. Дакле, уклањање z, налажење најкраћих путева без њега, и враћање z назад су основни делови алгоритма. Другим речима, следећа индуктивна хипотеза решава проблем.

Индуктивна хипотеза. Ако се зна тополошки редослед чворова, умемо да израчунамо дужине најкраћих путева од v до првих $n - 1$ чворова.

Кад је дат ациклички граф са n чворова (тополошки уређених), уклањамо n-ти чвор, индукцијом решавамо смањени проблем, налазимо најмању међу вредностима $w.SP+dužina(w, z)$, за све чворове w такве да $(w, z) \in E$. Алгоритам је дат у наставку текста.

Acikl_najkr_putevi(G, v, n)
улаз: $G = (V, E)$ (тежински ациклички граф), v (чвор), n (број чворова)
излаз: за сваки чвор $w \in V$, $w.SP$ је дужина најкраћег пута од v до w.
{Претпостављамо да је већ извршено тополошко сортирање.}
{Пре позива ове процедуре ставља се $z.SP \leftarrow \infty$ за све чворове $z \neq v$.}

```
1   нека је z чвор са редним бројем n у тополошком редоследу
2   if z ≠ v then
3      Acikl_najkr_putevi(G − z, v, n − 1)
4      {G − z добија се уклањањем z са свим суседним гранама из G}
5      for све чворове w такве да је (w, z) ∈ E do
6         if w.SP + duzina(w, z) < z.SP then
7            z.SP ← w.SP + duzina(w, z);
8   else v.SP ← 0;
```

Сада ћемо покушати да усавршимо алгоритам тако да се тополошко сортирање обавља истовремено са налажењем најкраћих путева. Другим речима, циљ је објединити два пролаза (за тополошко сортирање и налажење најкраћих путева) у један.

Размотримо начин на који се алгоритам рекурзивно извршава (после налажења тополошког редоследа). Претпоставимо, због једноставности, да је редни број чвора v у тополошком редоследу 1 (чворови са редним бројем мањим од редног броја v ионако нису достижни из v). Први корак је позив рекурзивне процедуре. Процедура затим позива рекурзивно саму себе, све док се не дође до чвора v. У том тренутку се дужина најкраћег пута до v

поставља на 0, и рекурзија почиње да се "размотава". Затим се разматра чвор u са редним бројем 2; дужина најкраћег пута до њега изједначује се са дужином гране (v, u), ако она постоји; у противном, не постоји пут од v до u. Следећи корак је провера чвора x са редним бројем 3. У овом случају у x улазе највише две гране (од v или u), па се упоређују дужине одговарајућих путева. Уместо оваквог извршавања рекурзије уназад, покушаћемо да исте кораке извршимо преко низа чворова са растућим редним бројевима.

Индукција се примењује према растућим редним бројевима почевши од v. Овај редослед ослобађа нас потребе да редне бројеве унапред знамо, па ћемо бити у стању да извршавамо истовремено оба алгоритма. Претпоставимо да су дужине најкраћих путева до чворова са редним бројевима од 1 до m при тополошком сортирању познати, и размотримо чвор са редним бројем $m+1$, који ћемо означавати са z. Да бисмо пронашли најкраћи пут до z, морамо да проверимо све гране које воде у z. Тополошки редослед гарантује да све такве гране полазе из чворова са мањим редним бројевима. Према индуктивној хипотези ти чворови су већ разматрани, па се дужине најкраћих путева до њих знају. За сваку грану (w, z) знамо дужину $w.SP$ најкраћег пута до w, па је дужина најкраћег пута до z преко w једнака $w.SP + dužina(w, z)$. Поред тога, као и раније, не морамо да водимо рачуна о евентуалним променама најкраћих путева ка чворовима са мањим редним бројевима, јер се до њих не може доћи из z. Побољшани алгоритам приказан је у наставку.

Acikl_najkr_putevi2(G, v)
улаз: $G = (V, E)$ (тежински ациклички граф), v (чвор графа G)
излаз: за сваки чвор $w \in V$, $w.SP$ је дужина најкраћег пута од v до w.

```
1   for w ∈ V do
2     w.SP ← ∞
3   иницијализуј v.UlStepen за све чворове
4   for i ← 1 to n do
5     if v_i.UlStepen = 0 then уписати v_i у ред Q
6   v.SP ← 0
7   while ред Q није празан
8     скини чвор w из реда Q
9     for све гране (w, z) do
10      if w.SP + duzina(w, z) < z.SP then
11        z.SP ← w.SP + duzina(w, z)
12      z.UlStepen ← z.UlStepen − 1
13      if z.UlStepen = 0 then уписати z у ред Q
```

Сложеност. Свака грана се по једном разматра у току иницијализације улазних степенова, и по једном у тренутку кад се њен полазни чвор уклања из листе. Приступ листи захтева константно време. Сваки чвор се разматра тачно једном. Према томе, временска сложеност алгоритма у најгорем случају је $O(|V| + |E|)$.

7.6.2 Општи случај

Кад граф није ациклички, не постоји тополошки редослед, па се разматрани алгоритми не могу директно применити. Међутим, основне идеје се могу искористити и у општем случају. Једноставност размотрених алгоритама последица је следеће особине тополошког редоследа:

> Ако је z чвор са редним бројем k, онда (1) не постоје путеви од z до чворова са редним бројевима мањим од k, и (2) не постоје путеви од чворова са редним бројевима већим од k до z.

Ова особина омогућује нам да нађемо најкраћи пут од v до z, не водећи рачуна о чворовима који су после z у тополошком редоследу. Може ли се некако дефинисати редослед чворова произвољног графа (који није нужно ациклички) који би омогућио нешто слично?

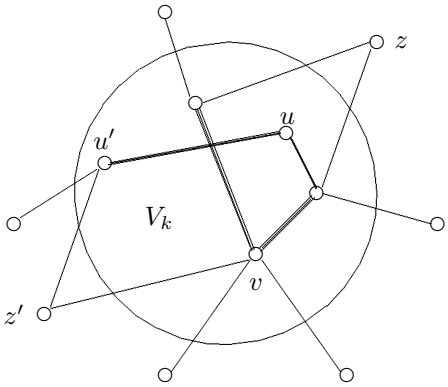
Идеја је разматрати чворове графа редом према дужинама најкраћих путева до њих од v. Те дужине се на почетку, наравно, не знају; оне се израчунавају у току извршавања алгоритма. Најпре проверавамо све гране које излазе из v. Нека је (v, x) најкраћа међу њима. Пошто су по претпоставци све дужине грана позитивне, најкраћи пут од v до x је грана (v, x). Дужине свих других путева до x су веће или једнаке од дужине ове гране. Чвор x је притом најближи од свих чворова чвору v. Према томе, знамо најкраћи пут до x, и то може да послужи као база индукције. Покушајмо да направимо следећи корак. Како можемо да пронађемо најкраћи пут до неког другог чвора? Бирамо чвор који је други најближи до v (x је први најближи). Једини путеви које треба узети у обзир су друге гране из v или путеви који се састоје од две гране: прва је (v, x), а друга је грана из чвора x. Нека је са $дужина(u, w)$ означена дужина гране (u, w). Бирамо најмањи од израза $дужина(v, y)$ $(y \neq x)$ или $дужина(v, x) + дужина(x, z)$ $(z \neq v)$. Још једном закључујемо да се други путеви не морају разматрати, јер је ово најкраћи пут за одлазак из v (изузев до x). Може се формулисати следећа индуктивна хипотеза.

Индуктивна хипотеза. За задати граф и његов чвор v, умемо да пронађемо k чворова најближих чвору v, као и дужине најкраћих путева до њих.

Запазимо да је индукција по броју чворова до којих су дужине најкраћих путева већ израчунате, а не по величини графа. Поред тога, претпоставља се да су то чворови најближи чвору v, и да умемо да их пронађемо. Ми умемо да пронађемо први најближи чвор, па је база (случај $k = 1$) решена. Кад k добије вредност $|V| - 1$, решен је комплетан проблем.

Означимо са V_k скуп који се састоји од k најближих чворова чвору v, укључујући и v. Проблем је пронаћи чвор w који је најближи чвору v међу чворовима ван V_k, и пронаћи најкраћи пут од v до w. Најкраћи пут од v до w може да саржи само чворове из V_k. Он не може да садржи неки чвор y ван V_k, јер би y био ближи чвору v од w. Према томе, да бисмо пронашли чвор w, довољно је да проверимо гране које спајају чворове из V_k са чворовима који нису у V_k; све друге гране се за сада могу игнорисати. Нека је (u, z) грана таква да је $u \in V_k$ и $z \notin V_k$. Таква грана одређује пут од v до z који се састоји од најкраћег пута од v до u (према индуктивној

хипотези већ познат) и гране (u, z). Довољно је упоредити све такве путеве и изабрати најкраћи међу њима, видети илустрацију на слици 7.5.



Слика 7.5: Налажење следећег најближег чвора задатом чвору v.

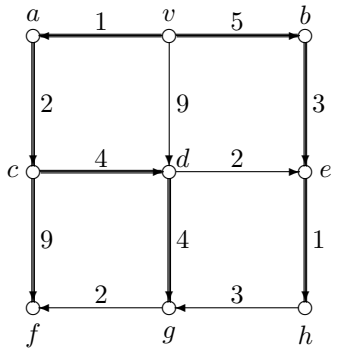
Алгоритам одређен овом индуктивном хипотезом извршава се на следећи начин. У свакој итерацији додаје се нови чвор. То је чвор w за који је најмања дужина

$$\min \{u.SP + dužina(u, w) | \, u \in V_k\} \tag{7.1}$$

међу свим чворовима $w \notin V_k$. Из већ изнетих разлога, w је заиста $(k+1)$-ви (следећи) најближи чвор чвору v. Према томе, његово додавање продужује индуктивну хипотезу.

Алгоритам је сада у потпуности прецизиран, али му се ефикасност може побољшати. Основни корак алгоритма је проналажење следећег најближег чвора. То се остварује израчунавањем најкраћег пута према (7.1). Међутим, није неопходно у сваком кораку проверавати све вредности $u.SP + dužina(u, w)$. Већина тих вредности не мења се при додавању новог чвора: могу се променити само оне вредности које одговарају путевима кроз новододати чвор. Ми можемо да памтимо дужине познатих најкраћих путева до свих чворова ван V_k, и да им поправљамо вредности само при проширивању V_k. Једини начин да се добије нови најкраћи пут после додавања w у V_k је да тај пут пролази кроз w. Према томе, треба проверити све гране од w ка чворовима ван V_k. За сваку такву грану (w, z) упоређујемо дужину $w.SP + dužina(w, z)$ са вредношћу $z.SP$, и по потреби поправљамо $z.SP$. Свака итерација обухвата налажење чвора са најмањом вредношћу SP, и поправку вредности SP за неке од преосталих чворова. Овај алгоритам познат је као Дијкстрин алгоритам (Dijkstra).

Пример 7.1. Пример извршавања алгоритма *Najkr_putevi* за налажење најкраћих путева од чвора v у графу дат је на слици 7.6. Прва врста односи се само на путеве од једне гране из v. Бира се најкраћи пут, у овом случају он води ка чвору a. Друга врста показује поправке дужина путева укључујући сада све путеве од једне гране из v или a, и најкраћи пут сада води до c. У свакој линији бира се нови чвор, и приказују се дужине тренутних најкраћих путева од v до свих чворова. Подебљана су растојања за која се поуздано зна да су најкраћа.



	v	a	b	c	d	e	f	g	h
a	0	1	5	∞	9	∞	∞	∞	∞
c	0	**1**	5	3	9	∞	∞	∞	∞
b	0	**1**	5	**3**	7	∞	12	∞	∞
d	0	**1**	**5**	**3**	7	8	12	∞	∞
e	0	**1**	**5**	**3**	**7**	8	12	11	∞
h	0	**1**	**5**	**3**	**7**	**8**	12	11	9
g	0	**1**	**5**	**3**	**7**	**8**	12	11	**9**
f	0	**1**	**5**	**3**	**7**	**8**	12	**11**	**9**

Слика 7.6: Пример извршавања алгоритма за налажење најкраћих путева од задатог чвора.

Реализација. Потребно је да проналазимо најмању вредност у скупу дужина путева и да често поправљамо дужине путева. Добра структура података за налажење минималних елемената и за поправке дужина елемената је хип. Пошто је потребно да пронађемо чвор са најмањом дужином пута до њега, све чворове ван скупа V_k чувамо у хипу, са кључевима једнаким дужинама тренутно најкраћих путева од v до њих. На почетку су све дужине путева сем једне једнаке ∞, па редослед елемената у хипу није битан, сем што v мора бити на врху. Налажење чвора w је једноставно: он се узима са врха хипа. После тога за сваку грану (w, u) проверава се да ли коришћење те гране скраћује пут до чвора u. Међутим, кад се промени дужина пута до неког чвора u, може се променити положај u у хипу. Према томе, потребно је на одговарајући начин поправљати хип (проблем је у томе што хип као структура података не омогућује ефикасно проналажење задатог елемента). Лоцирање чвора u у хипу може се извести тако што се одржава низ са индексима чворова у хипу. Налажење чвора у хипу је због тога еквивалентно приступању елементу низа. Пошто су елементи хипа чворови графа, просторна сложеност је $O(|V|)$, што је прихватљиво. Дужине путева могу само да опадају. Ако неки елеменат хипа постане мањи од оца, он се може замењивањем померати навише док му се не пронађе одговарајући положај. То је исти поступак као и при обичном поступку поправке хипа (при уметању елемента у хип, на пример). Дијкстрин алгоритам за налажење најкраћих путева од задатог чвора приказан је у наставку.

$Najkr\_putevi2(G, v)$
улаз: $G = (V, E)$ (тежински усмерени граф), v (чвор графа G)
излаз: за сваки чвор $w \in V$, $w.SP$ је дужина најкраћег пута од v до w.
{Претпоставка је да су све дужине грана ненегативне. }
1 **for** све чворове w **do**
2 $w.Oznaka \leftarrow false$ {означени чворови су у V_k}
3 $w.SP \leftarrow +\infty$;
4 $v.SP \leftarrow 0$;
5 **while** постоји неозначен чвор **do**
6 нека је w неозначени чвор са најмањом вредношћу $w.SP$

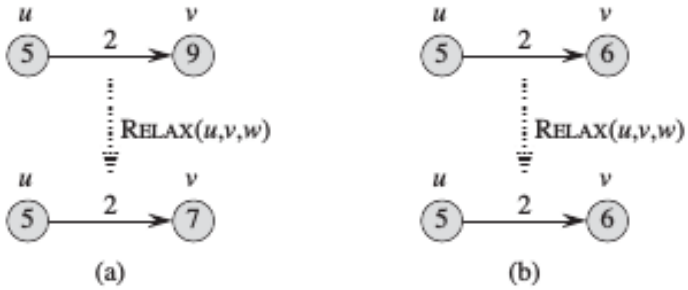
```
7       w.Oznaka ← true {укључивање w у V_k}
8       for све гране (w, z) такве да је z неозначен do
9         if w.SP + duzina(w, z) < z.SP then
10          z.SP ← w.SP + duzina(w, z)
```

Сложеност. Поправка дужине пута захтева $O(\log m)$ упоређивања, где је m величина хипа. Укупно има $|V|$ итерација, и због тога $|V|$ брисања из хипа. Поправке треба извршити највише $|E|$ пута (јер свака грана може да проузрокује највише једну поправку), па је потребно извршити највише $O(|E| \log |V|)$ упоређивања у хипу. Према томе, временска сложеност алгоритма је $O((|E| + |V|) \log |V|)$. Запажа се да је алгоритам спорији него алгоритам који исти проблем решава за ацикличке графове: у другом случају следећи чвор се узима из (произвољно уређене) листе, и никакве поправке нису потребне.

Најкраће путеве од v до свих осталих чворова нашли смо тако што смо путеве проналазили један по један. Сваки нови пут је одређен једном граном, која продужује претходно познати најкраћи пут до новог чвора. Све те гране формирају стабло са кореном v. Ово стабло зове се **стабло најкраћих путева**, и важно је за решавање многих проблема са путевима. Ако су тежине свих грана једнаке, онда је стабло најкраћих путева у ствари BFS стабло са кореном у чвору v. У примеру на слици 7.6 подебљане су гране које припадају стаблу најкраћих путева.

7.7 Белман-Фордов алгоритам

Основни корак у Белман-Фордовом алгоритму је *релаксација* гране (u, v), односно провера да ли се вредност $v.SP$ може заменити мањом вредношћу $u.SP + duzina(u, v)$; ако је то тачно, поправља се вредност $v.SP$ и памти да најкраћи пут води кроз чвор u. На слици 7.7 илустровано је извршавање овог корака на два примера.



Слика 7.7: Релаксација гране (u, v) дужине 2. Вредност SP за сваки чвор приказана је унутар кружића. (а) Пошто је $u.SP + duzina(u, v) < v.SP$, вредност $v.SP$ се смањује. (b) Овде је $v.SP \leq u.SP + duzina(u, v)$, па вредност $v.SP$ остаје непромењена.

7.7.1 Особине најкраћих путева и корака релаксације

Особине које ће бити наведене користе се у доказу коректности Белман-Фордовог алгоритма. Овде $\delta(u, w)$ означава дужину најкраћег пута од u до w; $duzina(u, w)$ је ознака за дужину гране (u, w); $u.SP$ означава тренутну процену дужине најкраћег пута од v до u.

Неједнакост троугла: За сваку грану $(u, w) \in E$ важи $\delta(v, w) \leq \delta(v, u) + duzina(u, w)$

Горња граница: Увек важи да је $u.SP \geq \delta(v, u)$ за све чворове $u \in V$, а кад $u.SP$ достигне $\delta(v, u)$, онда се $u.SP$ више не мења.

Карактеризација недостижности: Ако u није достижан из v, онда је увек $v.SP = \delta(v, u) = +\infty$

Особина конвергенције: Ако је $v \hookrightarrow u \rightarrow w$ најкраћи пут у G за неке чворове $u, w \in V$ и ако је $u.SP = \delta(v, u)$ у било ком тренутку пре релаксације гране (u, w), онда је после релаксације $w.SP = \delta(v, w)$.

Особина релаксације пута: Ако је $p = (v_0, v_1, \ldots, v_k)$ најкраћи пут од чвора $v = v_0$ до чвора v_k и гране пута P се релаксирају редом $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, онда је $v_k.SP = \delta(v, v_k)$. Ово важи без обзира на евентуалне друге кораке релаксације.

Особина графа претходника: У тренутку када је $u.SP = \delta(v, u)$ за било који чвор $u \in V$, подграф од грана ка претходницима чворова је стабло најкраћих путева са кореном у v.

7.7.2 Белман-Фордов алгоритам

Проблем одређивања најкраћих путева у графу $G = (V, E)$ од датог чвора $v \in V$ решава и Белман-Фордов алгоритам:

$Bellman - Ford(G, v)$
улаз: $G = (V, E)$ (дати граф), v (чвор графа)
тежина гране $e \in E$ је $duzina(e)$
излаз: за свако $u \in V$ израчуната вредност најкраћег растојања од v

```
1   for each w ∈ V do
2     w.SP ← ∞
3   v.SP ← 0
4   for i ← 1 to |V| − 1 do
5     for each (u, v) ∈ E do
6       if u.SP + duzina(u, v) < v.SP then
7         v.SP ← u.SP + duzina(u, v)
8   for each (u, v) ∈ E do
9     if u.SP + duzina(u, v) < v.SP then
10      return false
11  return true
```

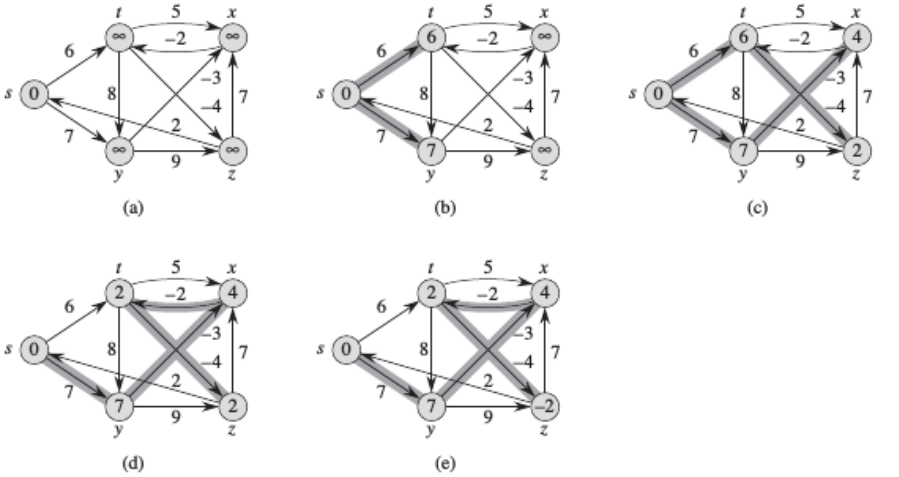
Алгоритам поправља процене $w.SP$ дужина најкраћих путева $\delta(v, w)$ постепено их смањујући, док не достигну стварну дужину најкраћих путева $\delta(v, w)$.

За разлику од Дијкстриног алгоритма, овај алгоритам ради и ако граф има гране негативне дужине. Алгоритам враћа логичку вредност $false$ ако

у G постоји циклус са негативним збиром дужина грана, односно *true* у противном. У првом случају дужине најкраћих путева се не могу одредити.

На слици 7.8 приказано је извршавање алгоритма на графу са пет чворова. После иницијализације у линијама 1–3, алгоритам $|V| - 1$ пут пролази све гране **for** петље у линијама 5–7. На сликама 7.8(б)-(е) приказано је стање алгоритма после сваког од четири проласка кроз гране. После тих $|V|-1$ пролазака у линијама 8–11 проверава се да ли у G постоји циклус негативне дужине и враћа се одговарајућа логичка вредност (у наставку ћемо видети зашто је та провера коректна).

Сложеност алгоритма је $O(|V||E|)$, због две уметнуте **for** петље.



Слика 7.8: Пример извршавања Белман-Фордовог алгоритма. Вредности SP приказане су унутар чворова, а шрафиране гране воде ка претходницима чворова на (тренутно) најкраћим путевима: ако је грана (u, v) шрафирана, онда се до чвора v најкраћим путем долази преко чвора u. У примеру у сваком пролазу се гране релаксирају у следећем редоследу: $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. (а) Ситуација пре првог проласка кроз гране. (б)-(е) Ситуација након сваког наредног проласка кроз гране. Вредности SP и гране ка претходницима су коначне на слици (е). У овом примеру, алгоритам враћа вредност $TRUE$.

Лема: Нека је $G = (V, E)$ тежински усмерени граф, $v \in V$ и за свако $e \in E$ је $duzina(e)$ дужина гране e. Претпоставимо да у G не постоји циклус негативне дужине достижан из чвора v. Тада је после $|V| - 1$ извршавања **for** петље у линијама 5–7 алгоритма $Bellman - Ford$ $u.SP = \delta(v, u)$ за све чворове $u \in V$ достижне из v.

Доказ: Доказ се заснива на особини релаксације пута. Посматрајмо било који чвор u који је достижан из v и нека је $p = (v_0, v_1, \ldots, v_k)$ где је $v_0 = v, v_k = u$ најкраћи пут од v до u. Пошто су најкраћи путеви прости, p има највише $|V| - 1$ грана, па је $k \leq |V| - 1$. Свака од $|V| - 1$ итерација релаксира свих $|E|$ грана. Међу гранама релаксираним у итерацији i је $(v_{i-1}, v_i), i = 1, 2, \ldots, k$. Због тога је на основу особине релаксације путева $u.SP = v_k.SP = \delta(v, u)$.

Последица: Нека је $G = (V, E)$ тежински усмерени граф, $v \in V$, $tezina(u, w)$ је тежина гране $(u, w) \in E$. Претпоставимо да у G не постоји циклус негативне дужине достижан из v. Тада за сваки чвор $u \in V$ постоји пут од v до u акко алгоритам $Bellman - Ford$ покренут за v даје $u.SP < \infty$.

Теорема (Коректност алгоритма $Bellman - Ford$): Ако G не садржи циклус негативне дужине достижан из v онда алгоритам враћа $true$, добија се $u.SP = \delta(v, u)$ за све чворове $u \in V$ и подграф претходника у G је стабло најкраћих путева у G са кореном у v. Ако G садржи циклус негативне дужине достижан из v онда алгоритам враћа $false$.

Доказ: Претпоставимо најпре да G не садржи циклус негативне дужине достижан из v. Доказаћемо најпре да је на крају $u.SP = \delta(v, u)$ за све чворове $u \in V$. Ако је чвор u достижан из v, онда је тврђење тачно на основу леме. У противном тврђење следи из карактеризације недостижности. Тиме је тврђење доказано. Особина графа претходника заједно са доказаним тврђењем има за последицу да је подграф претходника стабло најкраћих путева. Остаје да се докаже да алгоритам враћа резултат $true$. На завршетку се за сваку грану $(u, w) \in E$ на основу неједнакости троугла добија $w.SP = \delta(v, w) \leq \delta(v, u) + duzina(u, w) = u.SP + duzina(u, w)$, па ниједан тест у линији 9 не враћа $false$. Дакле, алгоритам враћа $true$.

Претпоставимо сада да G садржи циклус негативне дужине $c = (v_0, v_1, \ldots, v_k), v_k = v_0$, достижан из v. Тада је:

$$\sum_{i=1}^{k} duzina(v_{i-1}, v_i) < 0 \tag{7.2}$$

Претпоставимо супротно, да је алгоритам вратио $true$, тј. $v_i.SP \leq v_{i-1}.SP + duzina(v_{i-1}, v_i)$ за $i = 1, 2, \ldots, k$. Сабирањем ових неједнакости за све гране циклуса добија се:

$\sum_{i=1}^{k} v_i.SP \leq \sum_{i=1}^{k} v_{i-1}.SP + duzina(v_{i-1}, v_i) = \sum_{i=1}^{k} v_{i-1}.SP + \sum_{i=1}^{k} duzina(v_{i-1}, v_i)$

Пошто је $v_0 = v_k$, сваки чвор улази у збирове $\sum_{i=1}^{k} v_i.SP$ и $\sum_{i=1}^{k} v_{i-1}.SP$, тј. ови збирови су једнаки.

Поред тога, на основу последице, вредности $v_i.SP$ су коначне за $i = 1, 2, \ldots, k$ па се добија да је

$$0 \leq \sum_{i=1}^{k} duzina(v_{i-1}, v_i)$$

супротно неједнакости 7.2. Закључујемо да алгоритам $Bellman - Ford$ враћа $true$ ако граф G не садржи циклус негативне дужине, односно $false$ у противном.

7.8 Минимално повезујуће стабло

Размотримо систем рачунара које треба повезати оптичким кабловима. Потребно је обезбедити да постоји веза између свака два рачунара. Познати су трошкови постављања кабла између свака два рачунара. Циљ је пројектовати мрежу оптичких каблова тако да цена мреже буде минимална. Систем рачунара може бити представљен графом чији чворови одговарају рачунарима, а гране – потенцијалним везама између рачунара, са одговарајућом (позитивном) ценом. Проблем је пронаћи повезани подграф (са

гранама које одговарају постављеним оптичким кабловима), који садржи све чворове, такав да му сума цена грана буде минимална. Није тешко видети да тај подграф мора да буде стабло. Ако би подграф имао циклус, онда би се из циклуса могла уклонити једна грана — тиме се добија подграф који је и даље повезан, а има мању цену, јер су цене грана позитивне. Тражени подграф зове се **минимално повезујуће (разапињуће) стабло** (MCST, скраћеница од minimum–cost spanning tree) и има много примена. Наш циљ је конструкција ефикасног алгоритма за налажење MCST. Због једноставности, претпоставимо да су цене грана различите. Ова претпоставка има за последицу да је MCST јединствено, што олакшава решавање проблема. Без ове претпоставке алгоритам остаје непромењен, изузев што, кад се наиђе на гране једнаке тежине, произвољно се бира једна од њих.

Проблем. За задати неусмерени повезани тежински граф $G = (V, E)$ конструисати повезујуће стабло T минималне цене.

У овом контексту тежине грана тежинског графа G су у ствари њихове цене. Природно је користити следећу индуктивну хипотезу.

Индуктивна хипотеза (1)**.** Умемо да конструишемо MCST за повезани граф са мање од m грана.

Базни случај је тривијалан. Ако је задат проблем MCST са m грана, како се он може свести на проблем са мање од m грана? Тврдимо да грана најмање тежине мора бити укључена у MCST. Ако она не би била укључена, онда би њено додавање стаблу MCST затворило неки циклус; уклањањем произвољне друге гране из тог циклуса поново се добија стабло, али мање тежине — што је у супротности са претпоставком о минималности MCST. Дакле, ми знамо једну грану која мора да припада MCST. Можемо да је уклонимо из графа и применимо индуктивну хипотезу на остатак графа, који сада има мање од m грана. Да ли је ово регуларна примена индукције?

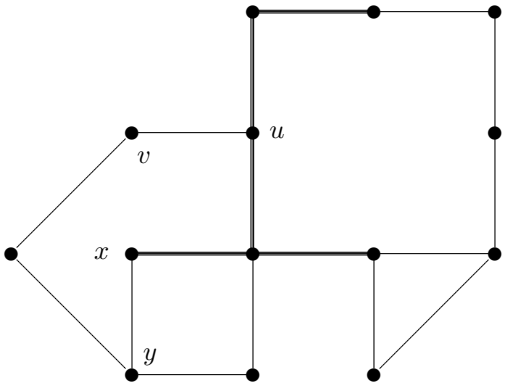
Проблем је у томе што после уклањања гране, преостали проблем више није еквивалентан полазном. Прво, избор једне гране ограничава могућности избора других грана. Друго, после уклањања гране граф не мора да остане повезан.

Решење насталог проблема је у прецизирању индуктивне хипотезе. Ми знамо како да изаберемо прву грану, али не можемо да је уклонимо и просто заборавимо на њу, јер остали избори зависе од ње. Дакле, уместо да грану уклонимо, треба да је означимо, и да ту чињеницу, њен избор, користимо даље у алгоритму. Алгоритам се извршава тако што се једна по једна грана бира и додаје у MCST. Према томе, индукција је не према величини графа, него према броју изабраних грана у задатом (фиксираном) графу.

Индуктивна хипотеза (2)**.** За задати повезан граф $G = (V, E)$ умемо да пронађемо подграф – стабло T са k грана ($k < |V| - 1$), тако да је стабло T подграф MCST графа G.

Базни случај за ову хипотезу смо већ размотрили — он се односи на избор прве гране. Претпоставимо да смо пронашли стабло T које задовољава индуктивну хипотезу и да је потребно да T проширимо наредном граном.

Како да пронађемо нову грану за коју ћемо бити сигурни да припада MCST? Применићемо сличан приступ као и при избору прве гране. За T се већ зна да је део коначног MCST. Због тога у MCST мора да постоји бар једна грана која повезује неки чвор из T са неким чвором у остатку графа. Покушаћемо да пронађемо такву грану. Нека је E_k скуп свих грана које повезују T са чворовима ван T. Тврдимо да грана са најмањом ценом из E_k припада MCST. Означимо ту грану са (u, v) (видети слику 7.9; гране стабла T су подебљане). Пошто је MCST повезујуће стабло, оно садржи тачно један пут од u до v (између свака два чвора у стаблу постоји тачно један пут). Ако грана (u, v) не припада MCST, онда она не припада ни том путу од u до v. Међутим, пошто u припада, а v не припада T, на том путу мора да постоји бар једна грана (x, y) таква да $x \in T$ и $y \notin T$. Цена ове гране већа је од цене (u, v), јер је цена (u, v) најмања међу ценама грана које повезују T са остатком графа. Сада можемо да применимо слично закључивање као при избору прве гране. Ако додамо (u, v) стаблу MCST, а избацимо (x, y), добијамо повезујуће стабло мање цене, што је контрадикција.



Слика 7.9: Налажење следеће гране минималног повезујућег стабла (MCST).

Реализација. Описани алгоритам (Примов алгоритам) сличан је алгоритму за налажење најкраћих путева од задатог чвора из претходног одељка. Прва изабрана грана је грана са најмањом ценом. T се дефинише као стабло са само том једном граном. У свакој итерацији проналази се грана која повезује T са неким чвором ван T, а има најмању цену. У алгоритму за налажење најкраћих путева од задатог чвора тражили смо најкраћи *пут* до чвора ван T. Према томе, једина разлика између MCST алгоритма и алгоритма за налажење најкраћих путева је у томе што се минимум тражи не по дужини пута, него по цени гране. Остатак алгоритма преноси се практично без промене. За сваки чвор w ван T памтимо цену гране минималне цене до w од неког чвора из T, односно ∞ ако таква грана не постоји. У свакој итерацији ми на тај начин бирамо грану најмање цене и повезујемо одговарајући чвор w са стаблом T. Затим проверавамо све гране суседне чвору w. Ако је цена неке такве гране (w, z) (за $z \notin T$) мања од цене тренутно најјефтиније познате гране до z, онда поправљамо цену чвора z и грану која кроз стабло води до њега. Овај алгоритам познат је

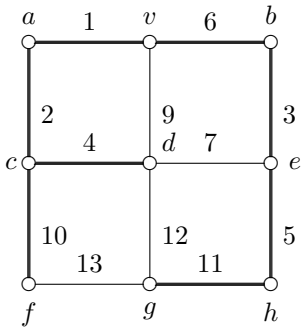
под називом Примов алгоритам.

MCST_Prim(G)
улаз: $G = (V, E)$ (неусмерени тежински граф)
излаз: T (минимално повезујуће стабло графа G)
{На почетку је T празан скуп}
1 **for** све чворове w **do**
2 $w.Oznaka \leftarrow false$; {$w.Oznaka$ је $true$ ако је w у T}
3 $w.Cena \leftarrow \infty$;
4 нека је (x, y) грана са најмањом ценом у G
5 $x.Oznaka \leftarrow true$; {y ће бити означено у главној петљи}
6 **for** све гране (x, z) **do**
7 $z.Grana \leftarrow (x, z)$; {грана најмање цене од T до z}
8 $z.Cena \leftarrow cena(x, z)$; {цена гране $z.Grana$}
9 **while** постоји неозначен чвор **do**
10 {нека је w неозначени чвор са најмањом вредношћу $w.Cena$;}
11 **if** $w.Cena = \infty$ **then**
12 *print* "G није повезан"; *halt*;
13 **else**
14 $w.Oznaka \leftarrow true$
15 {додај $w.Grana$ у T}
16 {сада поправљамо цене неозначених чворова повезаних са w}
17 **for** све гране (w, z) **do**
18 **if not** $z.Oznaka$ **then**
19 **if** $cena(w, z) < z.Cena$ **then**
20 $z.Grana \leftarrow (w, z)$
21 $z.Cena \leftarrow cena(w, z)$;

Сложеност. Сложеност Примовог алгоритма идентична је сложености алгоритма за налажење најкраћих растојања од задатог чвора из претходног одељка, $O((|E| + |V|) \log |V|)$.

Пример 7.2. Алгоритам за конструкцију MCST илустроваћемо примером на слици 7.10. Чвор у првој колони табеле је онај који је додат у одговарајућем кораку. Први додати чвор је v, и у првој врсти наведене су све гране из v са својим ценама. У свакој врсти бира се грана са најмањом ценом. Списак тренутно најбољих грана и њихових цена поправља се у сваком кораку (приказани су само крајеви грана). На слици су гране графа које припадају MCST подебљане.

Други ефикасан алгоритам за одређивање MCST графа $G = (V, E)$ је такође похлепан, али до MCST не долази додавањем нових грана на тренутно стабло, него на тренутну шуму. Додавањем сваке нове гране смањује се број стабала у шуми, тако да се на крају долази до само једног стабла, за које се испоставља да је MCST. При томе се гране за додавање разматрају редоследом према растућим тежинама; ако грана која је на реду повезује два чвора у различитим стаблима тренутне шуме, онда се та грана укључује у шуму, чиме се два стабла спајају у једно. У противном, ако грана повезује два чвора у истом стаблу тренутне шуме, грана се прескаче.



	v	a	b	c	d	e	f	g	h
v	–	v(1)	v(6)	∞	v(9)	∞	∞	∞	∞
a	–	–	v(6)	a(2)	v(9)	∞	∞	∞	∞
c	–	–	v(6)	–	c(4)	∞	c(10)	∞	∞
d	–	–	v(6)	–	–	d(7)	c(10)	d(12)	∞
b	–	–	–	–	–	b(3)	c(10)	d(12)	∞
e	–	–	–	–	–	–	c(10)	d(12)	e(5)
h	–	–	–	–	–	–	c(10)	h(11)	–
f	–	–	–	–	–	–	–	h(11)	–
g	–	–	–	–	–	–	–	–	–

Слика 7.10: Пример извршавања Примовог алгоритма за налажење MCST.

Да се установи да ли су крајеви u, v тренутне гране (u, v) у истом или различитим стаблима тренутне шуме, згодно је користити структуру података за дисјунктне скупове (видети одељак 6.1): тренутна стабла шуме су дисјунктни подскупови скупа чворова. Операције $podskup(u)$ и $podskup(v)$ проналазе представнике u', v' два подскупа (корене стабала којим су они представљени), па су u и v у истом подскупу акко је $u' = v'$. Ако је $u \neq v$, онда се та два подскупа замењују својом унијом, тј. примењује се операција $unija(u', v')$. У наставку је приказан кôд овог алгоритма – Крускаловог алгоритма.

$MCST\_Kruskal(G)$
улаз: $G = (V, E)$ (неусмерени тежински граф, тежина гране $e \in E$ је $e.w$)
излаз: A (минимално повезујуће стабло графа G)
1 $A \leftarrow \emptyset$
2 **for each** $v \in V$ **do**
3 формирати подскуп $\{v\}$
4 сортирати гране $e \in E$ у неопадајући низ према тежинама $e.w$
5 **for each** $(u, v) \in E$ датим у неопадајућем редоследу према $(u, v).w$ **do**
6 **if** $u' = podskup(u) \neq podskup(v) = v'$ **then**
7 $A \leftarrow A \cup \{(u, v)\}$
8 $unija(u', v')$
9 **return** A

Пример извршавања Крускаловог алгоритма на графу са слике 7.10 приказан је у табели 7.1. С обзиром на то да су дужине свих грана различите, резултат је исто MCST као на слици 7.10.

Сложеност алгоритма је $O(|V|)+O(|E|\log|E|)+O(|E|\log|V|) = O(|E|\log|V|)$ – ови сабирци одговарају редом петљи у линијама 2, 3, сортирању грана у линији 4 и **for** петљи у линијама 5–8. При томе се користи чињеница да је $O(\log|E|) = O(\log|V|)$ због $|E| \leq |V|^2$.

7.9 Сви најкраћи путеви

Сада ћемо размотрити проблем израчунавања најкраћих путева између *свака два* чвора у графу.

грана	дужина	укључена?	шума
			$v, a, b, c, d, e, f, g, h$
av	1	да	$\underline{a}v, b, \underline{c}, d, e, f, g, h$
ac	2	да	$acv, \underline{b}, c, d, \underline{e}, f, g, h$
be	3	да	$a\underline{c}v, be, \underline{d}, e, f, g, h$
cd	4	да	$acdv, b\underline{e}, f, g, \underline{h}$
eh	5	да	$acd\underline{v}, \underline{b}eh, f, g$
bv	6	да	$abc\underline{d}\underline{e}hv, f, g$
de	7	не	$abc\underline{d}eh\underline{v}, f, g$
dv	8	не	$ab\underline{c}dehv, \underline{f}, g$
cf	9	да	$abcdef\underline{h}v, \underline{g}$
gh	10	да	$abc\underline{d}e\underline{f}ghv$
dg	11	не	$abcde\underline{f}\underline{g}hv$
fg	12	не	$abcdefghv$

Табела 7.1: Пример извршавања Крускаловог алгоритма за граф са слике 7.10

Проблем. Дат је тежински граф $G = (V, E)$ (усмерени или неусмерени) са ненегативним тежинама (дужинама) грана. Пронаћи путеве минималне дужине између свака два чвора.

Поново, пошто говоримо о најкраћим путевима, тежине грана зовемо дужинама. Ово је проблем налажења **свих најкраћих путева**. Због једноставности ћемо се задовољити налажењем дужина свих најкраћих путева, уместо самих путева. Претпостављамо да је граф усмерен; све што ће бити речено важи и за неусмерене графове. Поред тога, претпоставља се да су дужине грана ненегативне.

Као и обично, покушајмо са директним индуктивним приступом. Може се користити индукција по броју грана или чворова.

Како се мењају најкраћи путеви у графу после додавања нове гране (u, w)? Нова грана може пре свега да представља краћи пут између чворова u и w. Поред тога, може се променити најкраћи пут између произвољна друга два чвора v_1 и v_2. Да би се установило има ли промене, треба са претходно познатом најмањом дужином пута од v_1 до v_2 упоредити збир дужина најкраћег пута од v_1 до u, гране (u, w) и дужине најкраћег пута од w до v_2. Укупно, за сваку нову грану потребно је извршити $O(|V|^2)$ провера, па је сложеност оваквог алгоритма у најгорем случају $O(|E||V|^2)$. Пошто је број грана највише $O(|V|^2)$, сложеност овог алгоритма је $O(|V|^4)$.

Како се мењају најкраћи путеви ако се у граф дода нови чвор u? Потребно је најпре пронаћи дужине најкраћих путева од u до свих осталих чворова, и од свих осталих чворова до u. Пошто су дужине најкраћих путева који не садрже u већ познате, најкраћи пут од u до w можемо да пронађемо на следећи начин. Потребно је да одредимо само прву грану на том путу. Ако је то грана (u, v), онда је дужина најкраћег пута од u до w једнака збиру дужине гране (u, v) и дужине најкраћег пута од v до w, која је већ позната. Потребно је дакле да упоредимо ове збирове за све гране суседне са u, и да међу њима изаберемо најмању. Најкраћи пут од w до u може се пронаћи на сличан начин. Али то све није довољно.

Поново је потребно да за сваки пар чворова проверимо да ли између њих постоји нови краћи пут кроз нови чвор u. За свака два чвора v_1 и v_2, да би се установило има ли промене, треба са претходно познатом најмањом дужином пута од v_1 до v_2 упоредити збир дужина најкраћег пута од v_1 до u и дужине најкраћег пута од u до v_2. То је укупно $O(|V|^2)$ провера и сабирања после додавања сваког новог чвора, па је сложеност оваквог алгоритма у најгорем случају $O(|V|^3)$. Испоставља се да је индукција по броју чворова ефикаснија него по броју грана. Међутим, постоји још боља индуктивна конструкција за решавање овог проблема.

Идеја је да се не мења број чворова или грана, него да се уведу ограничења на тип дозвољених путева. Индукција се изводи по опадајућем броју таквих ограничења, тако да на крају долазе у обзир сви могући путеви. Нумеришимо чворове од 1 до $|V|$. Пут од u до w зове се *k-пут* ако су редни бројеви свих чворова на путу (изузев u и w) мањи или једнаки од k. Специјално, 0-пут се састоји само од једне гране (пошто се ни један други чвор не може појавити на путу).

Индуктивна хипотеза. Умемо да одредимо дужине најкраћих путева између свака два чвора, при чему су дозвољени само k-путеви, за $k < m$.

База индукције је случај $m = 1$, кад се разматрају само директне гране и решење је очигледно. Претпоставимо да је индуктивна хипотеза тачна и да хоћемо да је проширимо на $k \leq m$. Једини нови путеви које треба да размотримо су m-путеви. Треба да пронађемо најкраће m-путеве између свака два чвора и да проверимо да ли они побољшавају k-путеве за $k < m$. Нека је v_m чвор са редним бројем m. Произвољан најкраћи m-пут садржи v_m највише једном (претпоставља се да су цене грана позитивне). Најкраћи m-пут између u и v може да се састоји од најкраћег $(m-1)$-пута од u до v_m, и најкраћег $(m-1)$-пута од v_m до v. Према индуктивној хипотези ми већ знамо дужине најкраћих k-путева за $k < m$, па је довољно да саберемо ове две дужине (и збир упоредимо са дужином најкраћег $(m-1)$-пута од u до v) да бисмо пронашли дужину најкраћег m-пута од u до v. Овај алгоритам познат је под називом Флојд-Варшалов алгоритам. Он је нешто бржи од претходног алгоритма (за константни фактор) и лакше га је и реализовати.

Svi_Najkr_putevi(W)
улаз: W ($n \times n$ матрица повезаности која представља тежински граф)
{ $W[x, y]$ је тежина гране (x, y) ако она постоји, односно ∞ у противном}
{ $W[x, x]$ је 0 за све чворове x}
излаз: на крају матрица W садржи дужине најкраћих путева.

```
1 for m ← 1 to n do  {индукција је по параметру m}
2   for x ← 1 to n do
3     for y ← 1 to n do
4       if W[x,m] + W[m,y] < W[x,y] then
5         W[x,y] ← W[x,m] + W[m,y]
```

Унутрашње две петље користе се за проверу свих *парова* чворова. Запажа се да се ова провера може извршавати са паровима чворова произвољним редоследом, јер је свака провера независна од осталих.

Сложеност. За свако m алгоритам извршава једно сабирање и једно упоређивање за сваки пар чворова. Број корака индукције је $|V|$, па је укупан број сабирања, односно упоређивања, највише $|V|^3$. Присетимо се да је временска сложеност алгоритма за налажење дужина најкраћих путева од једног чвора $O(|E|\log|V|)$. Ако је граф густ, па је број грана $\Omega(|V|^2)$, онда је описани алгоритам ефикаснији од извршавања за сваки чвор алгоритма за најкраће путеве од датог чвора. Иако је могуће реализовати алгоритам за најкраће путеве од једног чвора сложености $O(|V|^2)$, а тиме и алгоритам сложености $O(|V|^3)$ за налажење свих најкраћих растојања, алгоритам из овог одељка бољи је за густе графове због своје једноставне реализације. С друге стране, ако граф није густ (па има на пример $O(|V|)$ грана), онда је боља временска сложеност $O(|E||V|\log|V|)$ која потиче од $(|V|)$ пута употребљеног алгоритма за најкраће путеве од једног чвора.

7.10 Транзитивно затворење

За задати усмерени граф $G = (V, E)$ његово **транзитивно затворење** $C = (V, F)$ је усмерени граф у коме грана (u, w) између чворова u и w постоји ако и само ако у G постоји усмерени пут од u до w. Постоји много примена транзитивног затворења, па је важно имати ефикасни алгоритам за његово налажење.

Проблем. Пронаћи транзитивно затворење задатог усмереног графа $G = (V, E)$.

Овај проблем решићемо **редукцијом** (свођењем) на други проблем. Другим речима показаћемо како се може произвољни улаз за проблем транзитивно затворење свести на улаз за други проблем, који умемо да решимо. После тога решење другог проблема трансформишемо у решење проблема транзитивног затворења. Проблем о коме је реч је налажење свих најкраћих путева.

Нека је $G' = (V, E')$ комплетни усмерени граф (граф код кога за сваки пар чворова постоје обе гране, у оба смера). Грани $e \in E'$ додељује се дужина 0 ако је $e \in E$, односно 1 у противном. Сада за граф G' решавамо проблем налажења свих најкраћих путева. Ако у G постоји пут између v и w, онда је у G' његова дужина 0. Шта више, пут између v и w у G постоји ако и само ако је дужина најкраћег пута између v и w у G' једнака 0. Другим речима, решење проблема свих најкраћих путева непосредно се трансформише у решење проблема транзитивног затворења.

Није тешко преправити алгоритам за све најкраће путеве, тако да директно решава проблем транзитивног затворења.

Tranzitivno_zatvorenje(A)
улаз: A ($n \times n$ матрица повезаности која представља усмерени граф)
{ $A[x, y]$ је *true* ако грана (x, y) припада графу, односно $false$ у противном}
{ $A[x, x]$ је *true* за све чворове x}
излаз: на крају матрица A представља транзитивно затворење графа.

```
1 for m ← 1 to n do  {индукција је по параметру m}
2   for x ← 1 to n do
3     for y ← 1 to n do
```

```
4        if A[x,m] and A[m,y] then A[x,y] ← true
5        {овај корак је поправљен у следећој варијанти алгоритма}
```

Чињеница да можемо да сведемо један проблем на други значи да је први проблем општији од другог, односно да је други проблем специјални случај првог. Обично су општија решења скупља, сложенија. Ми смо видели много примера где је општији проблем лакше решити; ипак, да бисмо више добили, морамо више и да платимо. После коришћења редукције препоручљиво је покушати са поправком добијеног решења, користећи специјалне особине проблема.

Размотримо основни корак алгоритма, наредбу **if**. Она се састоји од две провере, $A[x,m]$ и $A[m,y]$. Нешто се предузима само ако су оба услова испуњена. Ова **if** наредба извршава се n пута за сваки пар чворова. Свака поправка ове наредбе водила би битној поправци алгоритма. Морају ли се сваки пут проверавати оба услова? Прва провера зависи само од x и m, а друга зависи само од m и y. Због тога се прва провера може за фиксиране x и m извршити само једном (уместо n пута). Ако први услов није испуњен, онда се други не мора проверавати ни за једну вредност y. Ако је пак први услов испуњен, онда се његова испуњеност не мора поново проверавати. Ова промена је уграђена у побољшани алгоритам дат у наставку. Асимптотска сложеност остаје непромењена, али се алгоритам у просеку извршава два пута брже.

*Tranzitivno_zatvorenje*2(A)
улаз: A ($n \times n$ матрица повезаности која представља усмерени граф)
{ $A[x,y]$ је *true* ако грана (x,y) припада графу, односно *false* у противном}
{ $A[x,x]$ је *true* за све чворове x}
излаз: на крају матрица A представља транзитивно затворење графа.

```
1  for m ← 1 to n do  {индукција је по параметру m}
2    for x ← 1 to n do
3      if A[x,m] then
4        for y ← 1 to n do
5          if A[m,y] then A[x,y] ← true
```

Овај алгоритам може се даље усавршити. Линија
`if A[m,y] then A[x,y] <- true`
може се еквивалентно заменити линијом
`A[x,y] = A[x,y] or A[m,y]`.

Запажа се да се после ове замене у унутрашњој петљи алгоритма операција `or` примењује на врсту m и врсту x матрице A, а резултат је нова врста m. Због тога, ако је $n \leq 64$, врсте матрице A могу се представити n-битним целим бројевима, па се примена операције `or` на врсте замењује битском `or` операцијом два цела броја, што је n пута брже. Ако је n велико, онда се врста може представити низом 64-битних целих бројева, па се алгоритам извршава приближно 64 пута брже. Асимптотска сложеност је и даље $O(n^3)$, али убрзање за фактор 64 није занемарљиво.

Глава 8

Алгебарски алгоритми

8.1 Увод

Кад год извршимо неку алгебарску операцију, ми у ствари извршавамо један алгоритам. До сада смо те операције прихватали као нешто обично, као алгоритме који постоје сами по себи. Међутим, без обзира да ли је у питању множење, дељење или нека сложенија алгебарска операција, уобичајени алгоритам није увек најбољи ако се ради са врло великим бројевима или низовима бројева. Иста појава на коју смо наилазили у претходним поглављима испољава се и овде: алгоритми који су добри за мали улаз постају неефикасни за веће улазе.

Сложеност алгоритма мери се бројем "операција" које алгоритам извршава. У највећем броју случајева претпостављамо да се основне аритметичке операције (као што су сабирање, множење, дељење) извршавају за јединично време. То је разумна претпоставка кад се аргументи могу представити машинском речи (на пример, не превелики цели бројеви, реални бројеви једноструке или двоструке тачности). Постоје међутим ситуације кад су аргументи огромни, на пример бројеви са 2000 цифара. Тада се мора узмети у обзир и величина аргумената, а основне операције престају да буду једноставне. Ако се игноришу величине аргумената, могуће је да на први поглед добар алгоритам буде у ствари врло неефикасан.

Значење "величине улаза" понекад није сасвим јасно. Нека је дат цели број n са којим треба извршити неку аритметичку операцију. На први поглед изгледа природно да се за величину улаза сматра сама вредност n. Ово се, међутим, не уклапа у неформалну дефиницију величине улаза као мери величине меморијског простора потребног за његово смештање. Разлика је врло велика. Сабирање два броја од по 100 цифара може се извршити врло брзо, чак и ручно. С друге стране, бројање до вредности представљене 100-цифреним бројем практично је немогуће, чак и на најбржем рачунару. Пошто се број n може представити са $\lceil \log_2 n \rceil$ бита, за његову **величину** може се сматрати $\lceil \log_2 n \rceil = O(\log n)$. Тако се алгоритам који захтева $O(\log n)$ операција кад је улаз n сматра линеарним (на пример израчунавање $2n$), јер је $O(\log n)$ линеарна функција величине улаза. Алгоритам који захтева извршавање $O(\sqrt{n})$ операција за улаз n (на пример факторизација броја n провером дељивости броја n свим бројевима мањим

или једнаким од $\sqrt{n}$) има експоненцијалну сложеност.

Као и обично, у овом поглављу ћемо пажњу усмерити на интересантне технике конструкције алгоритама. Најпре се разматра степеновање датог броја, затим вероватно најстарији нетривијални алгоритам, Еуклидов алгоритам за израчунавање највећег заједничког делиоца два броја. Врло је интересантно да модерни рачунари користе 2200 година стар алгоритам. Ова два алгоритма користе се рецимо у познатом асиметричном шифарском систему RSA. Поглавље се завршава једним од најважнијих и најлепших алгоритама — брзом Фуријеовом трансформацијом.

8.2 Степеновање

Започињемо једном од основних аритметичких операција.

Проблем. Дата су два природна броја n и k. Израчунати n^k.

Проблем се лако може свести на израчунавање n^{k-1}, јер је $n^k = n \cdot n^{k-1}$. Према томе, проблем се може решити индукцијом по k; добијени директни алгоритам приказан је на слици 8.1. Смањена је вредност k, али не и његова величина. Тривијални алгоритам захтева k множења. Пошто је величина броја k приближно $\log_2 k$, број итерација је експоненцијална функција величине k ($k = 2^{\log_2 k}$). Ово није лоше за мале, али је неприхватљиво за велике вредности k.

Алгоритам Stepen(n, k); {први покушај}
Улаз: n и k (два природна броја).
Излаз: P (вредност израза n^k).
begin
    $P := n$;
    for $i := 1$ **to** $k - 1$ **do**
        $P := n \cdot P$
end

Слика 8.1: Тривијални алгоритам за степеновање.

Други начин да се проблем реши је свођење на проблем са двоструко мањим експонентом помоћу једнакости $n^k = (n^{k/2})^2$. Половљење k одговара смањењу његове величине за константу. Због тога је број множења линеарна функција од величине k. Најједноставнији случај је $k = 2^j$, за неки природан број j:

$$n^k = n^{2^j} = \overbrace{\left(\left(n^2\right)^2\right)^{\cdots 2}}^{j \text{ пута}}.$$

Шта ако k није степен двојке? Размотримо још једном примењени поступак редукције. Пошавши од параметара n и k, проблем је сведен на мањи, са параметрима n и $k/2$. Ако $k/2$ није цели број, онда $(k-1)/2$ јесте, па се може применити слична редукција:

$$n^k = n\left(n^{(k-1)/2}\right)^2.$$

Сада је алгоритам комплетиран. Ако је k парно, једноставно квадрирамо резултат степеновања изложиоцем $k/2$. Ако је пак k непарно, квадрирамо резултат степеновања изложиоцем $(k-1)/2$ и множимо га са n. Број множења је највише $2\log_2 k$. Алгоритам је приказан на слици 8.2.

Алгоритам Stepen_kvadriranjem(n, k);
Улаз: n и k (два природна броја).
Излаз: P (вредност израза n^k).
begin
  if $k = 1$ **then** $P := n$
  else
    $z := Stepen\_kvadriranjem(n, k \operatorname{div} 2)$;
    if $k \bmod 2 = 0$ **then**
      $P := z \cdot z$
    else
      $P := n \cdot z \cdot z$
end

Слика 8.2: Степеновање квадрирањем.

Сложеност. Укупан број множења је $O(\log k)$. Међутим, како се напредује са извршавањем алгоритма, међурезултати постају све већи, па множења постају све компликованија. Ако се множење обавља у прстену остатака по модулу m (односно после сваке операције резултат се замењује својим остатком при дељењу са m), онда међурезултати не расту у току извршења алгоритма и временска сложеност алгоритма је производ сложености једног множења и броја множења, $O(logk)$.

8.3 Еуклидов алгоритам

Највећи заједнички делилац два природна броја n и m (означава се са $\mathrm{NZD}\,(n, m)$) је јединствени природни број d који 1) дели m и n, и 2) већи је или једнак од сваког другог природног броја d' који дели n и m. На пример, $NZD(24, 33) = 3$.

Проблем. Одредити највећи заједнички делилац два дата природна броја.

Као и обично, идеја је свести полазни проблем на проблем са мањим улазом. Могу ли се n или m некако смањити, а да се резултат не промени? Еуклид је приметио да је то могуће: ако d дели n и m, онда дели и њихову разлику. Важи и обрнуто; ако је на пример $n \geq m$, а d дели m и $n-m$, онда d дели и збир $m+(n-m) = n$. Другим речима, $\mathrm{NZD}\,(n, m) = \mathrm{NZD}\,(n-m, m)$, и добијен је мањи улаз. Међутим, одузимањем су смањене *вредности* бројева са којима се ради, али не (битно, у општем случају) и њихове *величине*. Да би алгоритам био ефикасан, морају се смањити величине бројева. На пример, ако је n врло велики број (нпр. 1000 цифара) и $m = 24$, број одузимања 24 од n је огроман, око $n/24$. Ово рачунање састоји се од $O(n)$ корака, што је експоненцијална функција од величине n.

Размотримо ову идеју још једном. После одузимања m од n исти алгоритам треба применити на $n-m$ и m. Ако је $n-m$ и даље веће од m, од њега треба одузети m. Са одузимањем m наставља се све док разлика не постане мања од m. Број оваквих одузимања може бити огроман; срећом, резултат који се тако добија је једнак остатку $n \bmod m$ при дељењу n са m. Према томе, $\mathrm{NZD}(n, m) = \mathrm{NZD}(n \bmod m, m)$. Дељење са остатком извршава се ефикасно. На пример, $\mathrm{NZD}(33, 24) = \mathrm{NZD}(24, 33 \mod 24) = \mathrm{NZD}(24, 9)$. Настављајући на исти начин, добија се $\mathrm{NZD}(24, 9) = \mathrm{NZD}(9, 6) = \mathrm{NZD}(6, 3) = \mathrm{NZD}(3, 0) = 3$.

У општем случају, полазећи од бројева $r_0 = n$ и $r_1 = m$, израчунава се остатак $r_2 = r_0 \bmod r_1$, затим остатак $r_3 = r_1 \bmod r_2$, итд. Тако се добија опадајући низ остатака

$$r_{i-1} = q_i r_i + r_{i+1}, \quad 0 \leq r_{i+1} < r_i, \text{ за } i = 1, 2, \ldots, k \tag{8.1}$$

(при дељењу r_{i-1} са r_i количник је q_i, а остатак r_{i+1}). Добијени низ је коначан јер је опадајући и састоји се од природних бројева. Нека је нпр. $r_{k+1} = 0$ и нека је $r_k \neq 0$ последњи члан овог низа различит од нуле. Како је

$$\mathrm{NZD}(r_0, r_1) = \mathrm{NZD}(r_1, r_2) = \ldots = \mathrm{NZD}(r_{k-1}, r_k) = \mathrm{NZD}(r_k, 0) = r_k,$$

видимо да је $d = \mathrm{NZD}(n, m)$ управо једнако r_k, последњем остатку различитом од нуле. Описани поступак за израчунавање највећег заједничког делиоца два броја зове се Еуклидов алгоритам.

$NZD(m, n)$
улаз: m и n (два природна броја)
излаз: nzd (највећи заједнички делилац бројева m и n)
1 $a \leftarrow \max(n, m)$
2 $b \leftarrow \min(n, m)$
3 $r \leftarrow 1$ {вредност која омогућује улазак у петљу}
4 **while** $r > 0$ **do** {r је остатак}
5 $\quad r \leftarrow a \bmod b$
6 $\quad a \leftarrow b$
7 $\quad b \leftarrow r$
8 **return** a

Сложеност. Тврдимо да је број дељења у Еуклидовом алгоритму једнак $O(\log(m + n))$ (што не значи да је сложеност алгоритма линеарна, јер величине бројева који се деле опадају како се одмиче са извршавањем алгоритма). Да се то докаже, довољно је доказати да вредност остатка a у алгоритму постаје бар два пута мања после две итерације. После прве итерације пар (a, b) $(a > b)$ замењује се паром $(b, a \bmod b)$, а после друге итерације паром $(a \bmod b, b \bmod (a \bmod b))$. Према томе, после две итерације је број a замењен бројем $a \bmod b$, који је увек мањи од $a/2$. Заиста, ако $b \leq a/2$ онда је $a \bmod b < b \leq a/2$; у противном, за $b > a/2$, такође се добија $a \bmod b = a - b = a/2 - (b - a/2) < a/2$.

Уз малу допуну, Еуклидов алгоритам се може искористити и за решавање следећег проблема.

Проблем. Највећи заједнички делилац d два природна броја n и m изразити као њихову целобројну линеарну комбинацију. Другим речима, одредити целе бројеве x, y, тако да важи $d = \mathrm{NZD}\,(n, m) = nx + my$.

Циљ је d изразити у облику линеарне комбинације $d = r_0x + r_1y$ остатака $r_0 = n$ и $r_1 = m$. Проблем се може решити индукцијом. Полази се од базе, израза за d у облику линеарне комбинације остатака $r_{k-1} = n$ и $r_{k-2} = m$: $d = r_k = r_{k-2} - q_{k-1}r_{k-1}$; овај израз је еквивалентан претпоследњем дељењу у Еуклидовом алгоритму (израз (8.1) за $i = k-1$). Корак индукције био би изражавање d у облику линеарне комбинације остатака r_{i-1} и r_i, полазећи од израза $d = x'r_i + y'r_{i+1}$ — целобројне линеарне комбинације r_i и r_{i+1}. Заиста, заменом r_{i+1} у овом изразу са $r_{i+1} = r_{i-1} - q_ir_i$, добија се

$$d = x'r_i + y'(r_{i-1} - q_ir_i) = y'r_{i-1} + (x' - q_iy')r_i,$$

тј. израз за d у облику целобројне линеарне комбинације остатака r_{i-1} и r_i. Дакле, индукцијом по i, $i = k-2, k-1, \ldots, 1, 0$ доказано је да се d може изразити као целобројна линеарна комбинација било која два узастопна члана r_i, r_{i+1} низа остатака. Специјално, за $i = 0$, добија се тражени израз.

Пример 8.1. Нека је задатак одредити целе бројеве x и y тако да важи $3 = 33x + 24y$. С обзиром на то да је $\mathrm{NZD}\,(33, 24) = 3$ можемо искористити претходни поступак за одређивање бројева x и y. Приликом израчунавања највећег заједничког делиоца имамо наредни низ једнакости: $\mathrm{NZD}\,(33, 24) = \mathrm{NZD}\,(24, 9) = \mathrm{NZD}\,(9, 6) = \mathrm{NZD}\,(6, 3) = \mathrm{NZD}\,(3, 0)$. Пратећи овај низ једнакости добијамо: $d = 3 = 9-6 = 9-(24-2\cdot 9) = 3\cdot 9-24 = 3\cdot(33-24)-24 = 3\cdot 33 - 4\cdot 24$, односно $x = 3, y = -4$.

Сложеност. Број операција је пропорционалан са бројем дељења у Еуклидовом алгоритму, тј. $O(m + n)$.

Значај ове допуне Еуклидовог алгоритма је у томе што она омогућује решавања тзв. *линеарних Диофантових једначина* $ax + by = c$, где су a, b, c дати цели бројеви, а x и y су цели бројеви које треба одредити. Без смањења општости може се претпоставити да је $a, b > 0$. Да би наведена једначина имала бар једно решење, потребно је да $d = \mathrm{NZD}\,(a, b)$ дели c (пошто d дели леву страну једначине, мора да дели и десну). Ако је овај услов испуњен, једно од решења лако се добија описаним поступком. Пошто се d изрази у облику $d = ax' + by'$, множењем са целим бројем c/d добија се $c = a(x'c/d) + b(y'c/d)$, тј. види се да је једно решење једначине пар $(x, y) = (x'c/d, y'c/d)$.

8.4 Брза Фуријеова трансформација

Брза Фуријеова трансформација (или FFT, што је скраћеница од fast Fourier transform) је важна из више разлога. Она ефикасно решава важан практичан проблем, елегантна је, и отвара нове, неочекиване области примене. Због тога је она један од најважнијих алгоритама од свог открића средином шездесетих година.

Алгоритам FFT није једноставан, и до њега се не долази директно. Ограничићемо се на само једну његову примену, множење полинома.

Проблем. Израчунати производ два задата полинома $p(x)$ и $q(x)$.

Формулација проблема је прецизна само на први поглед, јер није прецизиран начин представљања полинома. Обично се полином

$$P = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$$

представља низом својих коефицијената уз 1, x, x^2, $\ldots$, x^{n-1}; али то није једина могућност. Алтернатива је представљање полинома степена $n-1$ својим вредностима у n различитих тачака: те вредности једнозначно одређују полином. Други начин представљања је интересантан због једноставности множења. Производ два полинома степена $n-1$ је полином степена $2n-2$, па је одређен својим вредностима у $2n-1$ тачака. Ако претпоставимо да су вредности полинома – чинилаца дате у $2n-1$ тачака, онда се производ полинома израчунава помоћу $2n-1$, односно $O(n)$ обичних множења.

Нажалост, представљање полинома вредностима за неке примене није погодно. Пример је израчунавање вредности полинома у задатим тачкама; при репрезентацији вредностима, ово је много теже него ако су задати коефицијенти полинома. Међутим, ако бисмо могли да ефикасно преводимо полиноме из једне у другу представу, добили бисмо одличан алгоритам за множење полинома. Управо то се постиже применом FFT.

Прелаз од представе полинома коефицијентима на представу вредностима у тачкама, решава се израчунавањем вредности полинома. Вредност полинома $p(x)$ (задатог коефицијентима) у било којој тачки може се помоћу Хорнерове шеме (одељак 4.2) израчунати помоћу n множења. Израчунавање вредности $p(x)$ у n произвољних тачака изводљиво је дакле помоћу n^2 множења. Прелаз од представе полинома вредностима на представу коефицијентима зове се **интерполација**. Интерполација у општем случају такође захтева $O(n^2)$ операција. Овде је кључна идеја (као и у многим другим примерима које смо видели) да се не користи *произвољних* n тачака: ми имамо слободу да по жељи изаберемо *погодан* скуп од n различитих тачака. Брза Фуријеова трансформација користи специјалан скуп тачака, тако да се обе трансформације, израчунавање вредности и интерполација, могу ефикасно извршавати.

8.4.1 Директна Фуријеова трансформација

Размотримо проблем израчунавања вредности полинома. Потребно је израчунати вредности два полинома степена $n-1$ у $2n-1$ тачака, да би се њихов производ, полином степена $2n-2$, могао интерполирати. Међутим, полином степена $n-1$ може се представити као полином степена $2n-2$

изједначавањем са нулом водећих $n-1$ коефицијената. Због тога се без губитка општости може претпоставити да је проблем израчунати вредности произвољног полинома $P = \sum_{j=0}^{n-1} a_j x^j$ степена $n-1$ у n различитих тачака. Циљ је пронаћи таквих n тачака, у којима је лако израчунати вредности полинома. Због једноставности претпостављамо да је n степен двојке.

Користићемо матричну терминологију да бисмо упростили означавање. Израчунавање вредности полинома P у n тачака $x_0, x_1, \ldots, x_{n-1}$ може се представити као израчунавање производа матрице и вектора:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \ldots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P(x_0) \\ P(x_1) \\ \ldots \\ P(x_{n-1}) \end{pmatrix}.$$

Питање је како изабрати вредности $x_0, x_1, \ldots, x_{n-1}$, тако да се ово множење упрости. Посматрајмо две произвољне врсте r и s. Волели бисмо да их учинимо што сличнијим, да бисмо уштедели на множењима. Не може се ставити $x_r = x_s$, јер су тачке различите, али се $x_r^2 = x_s^2$ може постићи стављајући $x_s = -x_r$. Ово је добар избор, јер је сваки паран степен x_r једнак одговарајућем парном степену x_s; непарни степени разликују се само по знаку. Исто се може урадити и са осталим паровима врста. Наслућује се у ком правцу треба тражити n специјалних врста, за које би се горњи производ сводио на само $n/2$ производа врста матрице са колоном коефицијената. Резултат би био половљење величине улаза, а тиме и врло ефикасан алгоритам. Покушајмо да поставимо овај проблем као два одвојена проблема двоструко мање величине.

Подела полазног проблема на два потпроблема величине $m = n/2$ може се описати следећим изразом

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_{m-1} & x_{m-1}^2 & \ldots & x_{m-1}^{n-1} \\ 1 & -x_0 & (-x_0)^2 & \ldots & (-x_0)^{n-1} \\ 1 & -x_1 & (-x_1)^2 & \ldots & (-x_1)^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & -x_{m-1} & (-x_{m-1})^2 & \ldots & (-x_{m-1})^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \ldots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P(x_0) \\ P(x_1) \\ \ldots \\ P(x_{m-1}) \\ P(-x_0) \\ \ldots \\ P(-x_{m-1}) \end{pmatrix}. \quad (8.2)$$

Полазна $n \times n$ матрица подељена је на две врло сличне подматрице димензија $n/2 \times n$. За свако $j = 0, 1, \ldots, n/2 - 1$, важи $x_{n/2+j} = -x_j$. Згодно је дакле написати изразе за $P(x_j)$ и $P(-x_j)$, односно уопште $P(x)$, са раздвојеним члановима парног и непарног степена:

$$P(x) = \sum_{j=0}^{n/2-1} a_{2j} x^{2j} + \sum_{j=0}^{n/2-1} a_{2j+1} x^{2j+1}.$$

Ако са $P_0(x) = \sum_{j=0}^{n/2-1} a_{2j} x^j$, односно $P_1(x) = \sum_{j=0}^{n/2-1} a_{2j+1} x^j$ означимо полиноме степена $n/2-1$ са коефицијентима полинома P парног, односно непарног индекса, долазимо до једнакости

$$P(x) = P_0(x^2) + x P_1(x^2). \quad (8.3)$$

Заменом x са $-x$ добијамо $P(-x) = P_0(x^2) + (-x)P_1(x^2)$. Израчунавање $P(x_j)$, $j = 0, 1, \ldots, n-1$ своди се на рачунање $P(x_j)$ и $P(-x_j)$ за $j = 0, 1, \ldots, n/2-1$, односно на израчунавање само $n/2$ вредности $P_0(x_j^2)$, $n/2$ вредности $P_1(x_j^2)$, и допунских $n/2$ сабирања, $n/2$ одузимања и n множења. Дакле, имамо два потпроблема величине $n/2$ и $O(n)$ допунских операција.

Може ли се наставити рекурзивно на исти начин? Ако би нам то пошло за руком, дошли бисмо до познате диференцне једначине $T(n) = 2T(n/2) + O(n)$, чије је решење $T(n) = O(n \log n)$. Проблем израчунавања $P(x)$ (полинома степена $n-1$) у n тачака свели смо на израчунавање $P_0(x^2)$ и $P_1(x^2)$ (два полинома степена $n/2-1$) у $n/2$ тачака. То је регуларна редукција, изузев једног детаља: вредности x у $P(x)$ могу се произвољно бирати, али вредности x^2 у изразу (на пример) $P_0(x^2)$ могу бити само позитивне. Пошто смо до редукције дошли коришћењем негативних бројева, ово представља проблем. Издвојимо из (8.2) матрицу која одговара израчунавању вредности $P_0(x^2)$:

$$\begin{pmatrix} 1 & x_0^2 & x_0^4 & \ldots & x_0^{n-2} \\ 1 & x_1^2 & x_1^4 & \ldots & x_1^{n-2} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_{n/2-1}^2 & x_{n/2-1}^4 & \ldots & x_{n/2-1}^{n-2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_2 \\ \ldots \\ a_{n-2} \end{pmatrix} = \begin{pmatrix} P_0(x_0^2) \\ P_0(x_1^2) \\ \ldots \\ P_0(x_{n/2-1}^2) \end{pmatrix}.$$

Да бисмо још једном извели редукцију на исти начин, морали бисмо да ставимо нпр. $x_{n/4}^2 = -(x_0)^2$. Пошто су квадрати реалних бројева увек позитивни, ово је немогуће, бар ако се ограничимо на реалне бројеве. Потешкоћа се превазилази преласком на комплексне бројеве. Проблем се може опет поделити на два дела стављајући $x_{j+n/4} = ix_j$, за $j = 0, 1, \ldots, n/4-1$ (i је овде корен из -1, комплексан број). Ово раздвајање задовољава исте услове као и претходно. Према томе, проблем величине $n/2$ може се решити свођењем на два проблема величине $n/4$, изводећи $O(n)$ допунских операција.

За следеће раздвајање потребан нам је број z такав да је $z^8 = 1$ и $z^j \neq 1$ за $0 < j < 8$, односно примитивни осми корен из јединице; тада је $z^4 = -1$ и $z^2 = i$. Општије, потребан нам је примитивни n-ти корен из јединице. Означимо га са ω (због једноставности се n не спомиње експлицитно; у оквиру овог одељка ради се увек о једном истом n). Примитивни n-ти корен из јединице ω задовољава следеће услове:

$$\omega^n = 1,\ \omega^j \neq 1 \text{ за } 0 < j < n. \tag{8.4}$$

За n тачака $x_0, x_1, \ldots, x_{n-1}$ бирамо бројеве $1, \omega, \omega^2, \ldots, \omega^{n-1}$. Према томе, израчунава се следећи производ:

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ 1 & \omega^2 & \omega^{2\cdot 2} & \ldots & \omega^{2\cdot(n-1)} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & \omega^{n-1} & \omega^{(n-1)\cdot 2} & \ldots & \omega^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \ldots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} P(1) \\ P(\omega) \\ \ldots \\ P(\omega^{n-1}) \end{pmatrix}. \tag{8.5}$$

Овај производ се зове **Фуријеова трансформација** вектора $(a_0, a_1, \ldots, a_{n-1})$. Запазимо најпре да је испуњен услов

$$x_{n/2+j} = \omega^{n/2+j} = \omega^{n/2}\omega^j = -x_j, \quad j = 0, 1, \ldots, n/2-1.$$

Према томе, прва редукција проблема величине n на два мања је и даље исправна. Даље, два потпроблема произашла из ове редукције имају по $n/2$ тачака $1, \omega^2, \omega^4, \ldots, \omega^{n-2}$, што је управо проблем величине $n/2$, у коме уместо ω фигурише ω^2 — примитивни $n/2$-ти корен из јединице. Да је ω^2 примитивни $n/2$-ти корен из јединице непосредно следи из услова (8.4). Према томе, даље се може наставити рекурзивно. Сложеност алгоритма задовољава диференцну једначину $T(n) = 2T(n/2) + O(n)$, чије је решење $O(n \log n)$. Алгоритам омогућује ефикасно израчунавање Фуријеове трансформације вектора коефицијената полинома, па је добио име брза Фуријеова трансформација, односно FFT.

$FFT(n, a_0, a_1, \ldots, a_{n-1}, \omega)$
улаз: n (природни број), $a_0, a_1, \ldots, a_{n-1}$ (низ *елемената* типа који зависи од примене)
ω (примитивни n-ти корен из јединице)
излаз: V (низ излазних елемената, са индексима од 0 до $n-1$).
{претпоставља се да је n степен двојке}
1 **if** $n = 1$ **then**
2 $V[0] \leftarrow a_0$
3 **else**
4 $U \leftarrow FFT(n/2, a_0, a_2, \ldots, a_{n-2}, \omega^2)$;
5 $W \leftarrow FFT(n/2, a_1, a_3, \ldots, a_{n-1}, \omega^2)$;
6 **for** $j \leftarrow 0$ **to** $n/2 - 1$ **do** {према (8.3) за $x = \omega^j$}
7 $V[j] \leftarrow U[j] + \omega^j W[j]$;
8 $V[j + n/2] := U[j] - \omega^j W[j]$;
9 **return** V

Пример 8.2. Демонстрираћемо израчунавање FFT на примеру полинома са коефицијентима $(0, 1, 2, 3, 4, 5, 6, 7)$. Да бисмо избегли забуну, потпроблеме ћемо означавати са $P_{j_0, j_1, \ldots, j_k}(x_0, x_1, \ldots, x_k)$, где $j_0, j_1, \ldots, j_k$ означавају коефицијенте полинома, а $x_0, x_1, \ldots, x_k$ тачке у којима се израчунавају вредности полинома. Задатак је дакле решити проблем $P_{0,1,2,3,4,5,6,7}(1, \omega, \omega^2, \ldots, \omega^7)$, и то на основу (8.3).

Први корак је свођење $P_{0,1,2,3,4,5,6,7}(1, \omega, \omega^2, \ldots, \omega^7)$ на $P_{0,2,4,6}(1, \omega^2, \omega^4, \omega^6)$ и $P_{1,3,5,7}(1, \omega^2, \omega^4, \omega^6)$. Настављамо рекурзивно и сводимо $P_{0,2,4,6}(1, \omega^2, \omega^4, \omega^6)$ на $P_{0,4}(1, \omega^4)$ и $P_{2,6}(1, \omega^4)$. $P_{0,4}(1, \omega^4)$ се затим своди на $P_0(1) = 0$, и $P_4(1) = 4$. Комбиновањем ових резултата добијамо

$$\begin{aligned} P_{0,4}(1) &= P_0(1) + 1 \cdot P_4(1) = 0 + 1 \cdot 4 = 4, \\ P_{0,4}(\omega^4) &= P_0(\omega^8) + \omega^4 P_4(\omega^8) = 0 + \omega^4 \cdot 4. \end{aligned}$$

Пошто је $\omega^4 = -1$, добијамо $P_{0,4}(\omega^4) = -4$, односно после обједињавања $P_{0,4}(1, \omega^4) = (4, -4)$. На исти начин добијамо $P_{2,6}(1, \omega^4) = (8, -4)$.

Сада комбиновањем ова два вектора добијамо $P_{0,2,4,6}(1, \omega^2, \omega^4, \omega^6)$:

$$\begin{aligned} P_{0,2,4,6}(1) &= P_{0,4}(1) + 1 \cdot P_{2,6}(1) = 4 + 8 = 12, \\ P_{0,2,4,6}(\omega^2) &= P_{0,4}(\omega^4) + \omega^2 P_{2,6}(\omega^4) = -4 + \omega^2(-4), \\ P_{0,2,4,6}(\omega^4) &= P_{0,4}(\omega^8) + \omega^4 P_{2,6}(\omega^8) = P_{0,4}(1) - 1P_{2,6}(1) = 4 - 8 = -4, \\ P_{0,2,4,6}(\omega^6) &= P_{0,4}(\omega^{12}) + \omega^6 P_{2,6}(\omega^{12}) = P_{0,4}(\omega^4) - \omega^2 P_{2,6}(\omega^4) = -4 - \omega^2(-4), \end{aligned}$$

односно

$$P_{0,2,4,6}(1,\omega^2,\omega^4,\omega^6) = (12,-4(1+\omega^2),-4,-4(1-\omega^2)).$$

На сличан начин добија се

$$P_{1,3,5,7}(1,\omega^2,\omega^4,\omega^6) = (16,-4(1+\omega^2),-4,-4(1-\omega^2)).$$

Преостаје још израчунавање 8 вредности $P_{0,1,2,3,4,5,6,7}(1,\omega,\omega^2,\ldots,\omega^7)$. На пример, $P_{0,1,2,3,4,5,6,7}(1) = 12 + 1 \cdot 16 = 28$, и слично $P_{0,1,2,3,4,5,6,7}(\omega^4) = 12 - 1 \cdot 16 = -4$; $P_{0,1,2,3,4,5,6,7}(\omega) = (-4(1+\omega^2)) + \omega \cdot (-4(1+\omega^2))$, и слично $P_{0,1,2,3,4,5,6,7}(\omega^5) = (-4(1+\omega^2)) - \omega \cdot (-4(1+\omega^2))$, итд. Резултат је

$$\begin{aligned} &P_{0,1,2,3,4,5,6,7}(1,\omega,\omega^2,\ldots,\omega^7) = \\ &4(7,-1-\omega-\omega^2-\omega^3,-1-\omega^2,-1+\omega^2-\omega^3+\omega^5, \\ &-1,-1+\omega-\omega^2+\omega^3,-1+\omega^2,-1+\omega^2+\omega^3-\omega^5). \end{aligned}$$

8.4.2 Инверзна Фуријеова трансформација

Брза Фуријеова трансформација решава само пола проблема: вредности задатих полинома $p(x)$ и $q(x)$ могу се ефикасно израчунати у тачкама $1,\omega,\ldots,\omega^{n-1}$, измножити парови добијених вредности, и тако наћи вредности полинома $p(x)q(x)$ у наведеним тачкама. Остаје проблем интерполације, односно одређивања коефицијената производа полинома на основу вредности у тачкама. На срећу, испоставља се да је проблем интерполације врло сличан проблему израчунавања вредности, и да га решава практично исти алгоритам.

Вратимо се матричној нотацији. Нека је A^T транспонована матрица матрице A. Означимо вектор коефицијената полинома са $\mathbf{a} = (a_0,a_1,\ldots,a_{n-1})^T$, а вектор вредности полинома са $\mathbf{v} = (P(1),P(\omega),\ldots,P(\omega^{n-1}))^T$. Нека је $V(\omega)$ матрица из једнакости (8.5). Ако су задати коефицијенти полинома $\mathbf{a}$, његове вредности $\mathbf{v}$ у n тачака $1,\omega,\ldots,\omega^{n-1}$ добијају се према (8.5) израчунавањем производа $\mathbf{v} = V(\omega)\mathbf{a}$. С друге стране, ако су задате вредности полинома $\mathbf{v} = (P(1),P(\omega),\ldots,P(\omega^{n-1}))^T = (v_0,v_1,\ldots,v_{n-1})^T$, а потребно је израчунати његове коефицијенте, једнакост (8.5), односно $V(\omega)\mathbf{a} = \mathbf{v}$ је систем линеарних једначина по $\mathbf{a}$. Решавање система једначина своди се на рачунање инверзне матрице $V(\omega)^{-1}$ алгоритмом сложености $O(n^3)$. Непосредно се проверава да је

$$V(\omega)V(\omega^{-1}) = nE,$$

где је са E означена јединична матрица реда n. Заиста, ако је $r \neq s$, онда је производ $(r+1)$-е врсте матрице $V(\omega)$ и $(s+1)$-е колоне матрице $V(\omega^{-1})$ једнак

$$\sum_{k=0}^{n-1} \omega^{rk}\omega^{-sk} = \sum_{k=0}^{n-1} \omega^{k(r-s)} = \frac{1-\omega^{n(r-s)}}{1-\omega^{r-s}} = 0.$$

Ако је пак $r = s$, онда је тај производ $\sum_{k=0}^{n-1} \omega^{rk}\omega^{-rk} = n$. Тиме је доказана следећа теорема.

Теорема 8.1. *Инверзна матрица матрице $V(\omega)$ Фуријеове трансформације је*

$$V(\omega)^{-1} = \frac{1}{n}V(\omega^{-1}).$$

Решавање система једначина $\mathbf{v} = V(\omega)\mathbf{a}$ своди се дакле на израчунавање производа

$$\mathbf{a} = \frac{1}{n} V(\omega^{-1})\mathbf{v}.$$

Посао се даље поједностављује захваљујући следећој теореми.

Теорема 8.2. *Ако је ω примитивни n-ти корен из јединице, онда је ω^{-1} такође примитивни n-ти корен из јединице.*

Према томе, производ $\frac{1}{n} V(\omega^{-1})\mathbf{v}$ може се израчунати применом брзе Фуријеове трансформације, замењујући ω са ω^{-1}. Ова трансформација зове се **инверзна Фуријеова трансформација**.

Сложеност. Узимајући све у обзир, производ два полинома може се израчунати изводећи $O(n \log n)$ операција (са комплексним бројевима).

Глава 9

Редукције

9.1 Увод

Започећемо ово поглавље једном анегдотом. Математичару је објашњено како може да скува чај: треба да узме чајник, напуни га водом из славине, стави чајник на штедњак, сачека да вода проври, и на крају стави чај у воду. А онда су га питали: како може да скува чај ако има чајник пун вреле воде? Једноставно, каже он; просуће воду и тако свести проблем на већ решен.

У овом поглављу позабавићемо се идејом редукције, односно свођења једног проблема на други. Показаћемо да редукције, иако понекад нерационалне, могу бити и врло корисне. На пример, ако убаците у сандуче поште на Новом Београду писмо за човека који станује одмах поред те поште, писмо ће, без обзира на близину одредишта, прећи пут до Главне поште у Београду, па назад до поште на Новом Београду, и тек онда ће га поштар однети вашем познанику. Чести су случајеви кад није лако препознати специјални случај и за њега скројити ефикасније специјално решење. У пракси је често ефикасније све специјалне случајеве третирати на исти начин. На такву ситуацију често се наилази и при конструкцији алгоритама. Кад наиђемо на проблем који се може схватити као специјални случај другог проблема, користимо познато решење. То решење понекад може бити превише опште или превише скупо. Међутим, у многим случајевима је коришћење општег решења најлакши, најбржи и најелегантнији начин да се проблем реши. Овај принцип се често користи. За неке рачунарске проблеме, на пример рад са неком базом података, обично није неопходно написати програм који решава само тај проблем. Опште решење не мора бити најефикасније, али је много једноставније искористити управо њега.

Претпоставимо да је дат проблем P, који изгледа компликовано, али личи на познати проблем Q. Може се независно (испочетка) решавати проблем P, или се може искористити неки од метода за решавање Q и применити га на P. Постоји, међутим, и трећа могућност. Може се покушати са налажењем **редукције**, односно свођења једног проблема на други. Неформално, редукција је решавање једног проблема коришћењем "црне кутије" која решава други проблем. Редукцијом се може постићи један

од два циља, зависно од смера. Решење P, које користи црну кутију за решавање Q, може се трансформисати у алгоритам за решавање P, ако се зна алгоритам за решавање Q. С друге стране, ако се за P зна да је тежак проблем, и зна се доња граница сложености за алгоритме који решавају P, онда је то истовремено и доња граница сложености за проблем Q. У првом случају је редукција искоришћена за добијање информације о P, а у другом — о Q.

У овом поглављу видећемо један пример редукције. Налажење редукције једног проблема на други може бити корисно чак и ако не даје нову доњу или горњу границу сложености проблема. Редукција омогућује боље разумевање оба проблема. Она се може искористити за налажење нових техника за напад на проблем или његове варијанте.

И раније смо већ сретали са примерима редукција — на пример, редукција проблема налажења транзитивног затворења на проблем налажења свих најкраћих путева (одељак 7.10).

9.2 Налажење троуглова у неусмереном графу

Постоји тесна веза између графова и матрица. Граф $G = (V, E)$ са n чворова $v_1, v_2, \ldots v_n$, може се представити својом **матрицом повезаности** — квадратном матрицом $A = (a_{ij})$ реда n, таквом да је $a_{ij} = 1$ ако $(v_i, v_j) \in E$, односно $a_{ij} = 0$ у осталим случајевима. Ако је G неусмерени граф, матрица A је **симетрична**. Ако је G тежински граф, онда се такође може представити квадратном матрицом $A = (a_{ij})$ реда n, при чему је елеменат a_{ij} једнак **тежини** гране (v_i, v_j), односно ∞, ако те гране нема у графу. Постоје и други начини да се матрица придружи графу. Тако се графу $G = (V, E)$ са n чворова и m грана може придружити $n \times m$ матрица у којој је (i, j) елеменат једнак 1 ако и само ако је i-ти чвор суседан са j-том граном.

Везе графова и матрица не задржавају се само на репрезентацији. Многе особине графова могу се боље разумети анализом одговарајућих матрица. Слично, многе особине матрица откривају се посматрањем одговарајућих графова. Није изненађујуће да се многи алгоритамски проблеми могу решити коришћењем ове аналогије. То ћемо илустровати једним примером.

Проблем. Нека је $G = (V, E)$ неусмерени повезани граф са n чворова и m грана. Потребно је установити да ли у G постоји **троугао**, односно таква три чвора да између свака два од њих постоји грана.

Директно решење обухвата проверу свих трочланих подскупова скупа чворова. Подскупова има $\binom{n}{3} = n(n-1)(n-2)/6$, а пошто се сваки од њих може проверити за константно време, временска сложеност оваквог алгоритма је $O(n^3)$. Може се конструисати и алгоритам сложености $O(mn)$, где је m број грана у графу (заснован на провери свих парова $(grana, čvor)$ — да ли граде троугао), који је бољи ако граф није густ. Може ли се ово даље побољшати? Приказаћемо сада алгоритам који је асимптотски бржи, и суштински је другачији. Сврха примера је да илуструје везу између графовских и матричних алгоритама.

Нека је A матрица повезаности графа G. Пошто је граф G неусмерен, матрица A је симетрична. Размотримо везу елемената матрице $B = A^2 =$

AA (производ је обичан производ матрица) и графа G. Према дефиницији производа матрица је

$$B[i,j] = \sum_{k=1}^{n} A[i,k] \cdot A[k,j].$$

Из ове једнакости следи да је услов $B[i,j] > 0$ испуњен акко постоји индекс k, такав да су оба елемента $A[i,k]$ и $A[k,j]$ јединице. Другим речима, $B[i,j] > 0$ је испуњено ако и само ако постоји чвор v_k, такав да је $k \neq i$, $k \neq j$ и да су оба чвора v_i и v_j повезана са v_k (претпостављамо да граф нема петљи, односно $A[i,i] = 0$ за $i = 1, 2, \ldots, n$). Према томе, у графу постоји троугао који садржи темена v_i и v_j акко је v_i повезан са v_j и $B[i,j] > 0$. Коначно, у G постоји троугао акко постоје такви индекси i, j да је $A[i,j] = 1$ и $B[i,j] > 0$.

Наведена анализа сугерише алгоритам. Треба израчунати матрицу $B = A^2$ и проверити испуњеност услова $A[i,j] = 1$, $B[i,j] > 0$ за свих n^2 парова (i,j). Сложеност провере овог услова је $O(n^2)$, па преовлађујући део временске сложености алгоритма потиче од множења матрица. Тиме је проблем налажења троугла у графу сведен на проблем квадрирања матрица. За множење матрица може се искористити Штрасенов алгоритам и тако добити алгоритам за налажење троугла у графу сложености $O(n^{2.81})$. Прецизније, описана редукција показује да је сложеност овог графовског проблема $O(M(n))$, где је $M(n)$ сложеност множења Булових матрица реда n.

Глава 10

NP-комплетност

10.1 Увод

У претходним поглављима разматране су технике за решавање алгоритамских проблема и њихове примене на конкретне проблеме. Било би лепо кад би сви проблеми имали елегантне ефикасне алгоритме, до којих се долази коришћењем малог скупа техника. Међутим, постоји много проблема који се не покоравају до сада размотреним техникама. Могуће је да приликом њиховог решавања није уложено довољно напора, али се са доста разлога може претпоставити да постоје проблеми који *немају* ефикасно опште решење. У овом поглављу описаћемо технике за препознавање неких таквих проблема.

Временска сложеност већине до сада разматраних алгоритама ограничена је неким полиномом од величине улаза. За такве алгоритме кажемо да су **ефикасни** алгоритми, а за одговарајуће проблеме да су **решиви**. Другим речима, за алгоритам кажемо да је ефикасан ако је његова временска сложеност $O(P(n))$, где је $P(n)$ полином од величине проблема n. Подсетимо се да је величина улаза дефинисана као број битова потребних за представљање улаза. Ово није строго формална дефиниција: могуће је исте податке представити битовима на више начина; међутим, дужине свих довољно ефикасних представљања истих података не могу се значајно разликовати. Класу свих проблема који се могу решити ефикасним алгоритмом означаваћемо са P (полиномијално време). Ова дефиниција може да изгледа чудно на први поглед. Тако, на пример, алгоритам сложености $O(n^{10})$ се ни по којим мерилима не може сматрати ефикасним; слично, алгоритам са временом извршавања $10^7 n$ тешко је сматрати ефикасним, иако је линеарне сложености. Ипак, ова дефиниција има смисла из два разлога. Прво, она омогућује развој теорије којом ћемо се сада позабавити; друго, и важније, ову дефиницију је лако применити. Испоставља се да огромна већина решивих проблема има практично употребљива решења. Другим речима, временска сложеност полиномијалних алгоритама на које се у пракси наилази је најчешће полином малог степена, ретко изнад квадратног. Обрнуто је обично такође тачно: алгоритми са временском сложеношћу већом од произвољног полинома обично се не могу практично извршавати за велике улазе.

Постоји доста проблема за које се не зна ни један алгоритам полиномијалне

сложености. Неки од тих проблема ће једном можда бити решени ефикасним алгоритмима. Међутим, има разлога за веровање да се многи проблеми не могу решити ефикасно. Волели бисмо да будемо у стању да препознамо такве проблеме, да не бисмо губили време на тражење непостојећег алгоритма. У овом поглављу размотрићемо како приступати проблемима за које се не зна да ли су у класи P. Специјално, размотрићемо једну потклasу таквих проблема, класу такозваних NP-**комплетних проблема**. Ови проблеми могу се груписати у једну класу јер су сви међусобно строго еквивалентни — *ефикасан алгоритам за неки* NP-*комплетан проблем постоји ако и само ако за сваки* NP-*комплетан проблем постоји ефикасан алгоритам*. Широко је распрострањено веровање да не постоји ефикасан алгоритам за било који NP-комплетан проблем, али се не зна доказ оваквог тврђења. Чак и кад би постојали ефикасни алгоритми за решавање NP-комплетних проблема, они би сигурно били врло компликовани, јер се таквим проблемима много истраживача бавило током дугог низа година. До овог тренутка се за стотине (односно хиљаде, у зависности од начина бројања тих проблема) проблема зна да су NP-комплетни, што ову област чини врло значајном.

Ово поглавље састоји се из два дела. У првом се дефинише класа NP-комплетних проблема и наводе примери доказа припадности неких проблема тој класи. Затим се разматра неколико техника за *приближно* решавање NP-комплетних проблема алгоритмима полиномијалне сложености.

10.2 Редукције полиномијалне временске сложености

У овом одељку ограничићемо се на **проблеме одлучивања**, тј. разматраћемо само оне проблеме на које се после извршавања одређеног алгоритма може одговорити са "да" или "не". Ово ограничење упрошћава разматрања. Већи део проблема лако се може превести у проблеме одлучивања. На пример, уместо да тражимо максималну клику (потпуно повезани подраф графа, тј. подскуп чворова графа такав да између свака два чвора у подскупу постоји грана), можемо да поставимо питање да ли за задато k постоји клика величине бар k. Ако умемо да решимо проблем одлучивања, обично можемо да решимо и полазни проблем — бинарном претрагом, на пример.

Проблем одлучивања може се посматрати као **проблем препознавања језика**. Нека је U скуп могућих улаза за проблем одлучивања. Нека је $L \subseteq U$ скуп свих улаза за које је решење проблема "да". За L се каже да је **језик** који одговара проблему, па појмови *проблем* и *језик* могу да се користе равноправно. Проблем одлучивања је установити да ли задати улаз припада језику L. Сада ћемо увести појам редукције полиномијалне сложености између језика, као основни алат у овом поглављу.

Дефиниција 1. Нека су L_1 и L_2 два језика, подскупа редом скупова улаза U_1 и U_2. Кажемо да је L_1 **полиномијално сводљив** на L_2, ако постоји алгоритам полиномијалне временске сложености, који дати улаз $u_1 \in U_1$ преводи у улаз $u_2 \in U_2$, тако да $u_1 \in L_1$ ако и само ако $u_2 \in L_2$. Алгоритам је полиномијалан у односу на величину улаза u_1. Претпостављамо да је појам величине добро дефинисан у просторима улаза U_1 и U_2, тако да је, на пример, величина u_2 ограничена полиномом од величине u_1.

Алгоритам из дефиниције своди један проблем на други. Ако знамо алгоритам за препознавање L_2, онда га можемо *суперпонирати* са алгоритмом редукције и тако добити алгоритам за решавање L_1. Означимо алгоритам редукције са AR, а алгоритам за препознавање L_2 са AL_2. Произвољни улаз $u_1 \in U_1$ може се применом AR трансформисати у улаз $u_2 \in U_2$, и применом AL_2 установити да ли $u_2 \in L_2$, а тиме и да ли $u_1 \in L_1$. Специјално, ако је $L_2 \in P$, закључујемо да је и $L_1 \in P$.

Теорема 10.1. *Ако је језик L_1 полиномијално сводљив на језик L_2, и $L_2 \in P$, онда је и $L_1 \in P$.*

Релација полиномијалне сводљивости није симетрична: полиномијална сводљивост L_1 на L_2 не повлачи полиномијалну сводљивост L_2 на L_1. Ова асиметрија потиче од чињенице да дефиниција сводљивости захтева да се *произвољан* улаз за L_1 може трансформисати у еквивалентан улаз за L_2, али не и обрнуто. Могуће је да улази за L_2, добијени на овај начин, представљају само мали део свих могућих улаза за L_2. Према томе, ако је L_1 полиномијално сводљив на L_2, онда можемо сматрати да је проблем L_2 *тежи*.

Два језика L_1 и L_2 су **полиномијално еквивалентни**, или једноставно еквивалентни, ако је сваки од њих полиномијално сводљив на други. Специјално, сви нетривијални проблеми из класе P су еквивалентни. Докажимо ово тврђење. Нека су B и C два нетривијална проблема из P (тј. за оба постоје улази са одговором "да" и улази са одговором "не"). Нека су u_0 и u_1 два улаза за проблем C, таква да је за u_0 решење "не", а за u_1 "да" (односно $u_0 \notin L_C$ и $u_1 \in L_C$). За свођење проблема B на проблем C може се искористити следећи полиномијални алгоритам: улазу v за проблем B придружује се $\phi(v) = u_0$ ако $v \notin L_B$, односно $\phi(v) = u_1$ ако $v \in L_B$. Тада за произвољан улаз v за проблем B важи $v \in L_B$ акко $\phi(v) \in L_C$.

Релација "полиномијалне сводљивости" је транзитивна, као што показује следећа теорема.

Теорема 10.2. *Ако је L_1 полиномијално сводљив на L_2 и L_2 је полиномијално сводљив на L_3, онда је L_1 полиномијално сводљив на L_3.*

Доказ. Нека су језици L_1, L_2 и L_3 подскупови скупова могућих улаза редом U_1, U_2 и U_3. Суперпонирањем алгоритама редукције L_1 на L_2, односно L_2 на L_3 добија се алгоритам редукције L_1 на L_3. Произвољан улаз $u_1 \in U_1$ конвертује се најпре у улаз $u_2 \in U_2$, који се затим конвертује у улаз $u_3 \in U_3$. Пошто су редукције полиномијалне сложености, а композиција две полиномијалне функције је полиномијална функција, резултујући алгоритам редукције је такође полиномијалне сложености (то је један од разлога зашто су за меру сложености решивих проблема изабрани полиноми).

Суштина метода који ћемо примењивати у овом поглављу је да ако се за неки проблем не може наћи ефикасан алгоритам, онда покушавамо да установимо да ли је он еквивалентан неком од проблема за које знамо да су тешки. Класа NP-комплетних проблема садржи стотине таквих међусобно еквивалентних проблема.

10.3 Недетерминизам и Кукова теорема

Сада ћемо дефинисати другу важну класу језика (односно проблема одлучивања) — класу NP. Размотримо као пример проблем трговачког путника.

Проблем. Задат је тежински граф $G = (V, E)$ са $|V| = n$ чворова (тако да је за произвољне чворове – градове $v_i, v_j \in V$, тежина гране $(v_i, v_j) \in E$ једнака $d(v_i, v_j)$) и број $B \in \mathbf{Z}^+$. Установити да ли у G постоји Хамилтонов циклус са збиром тежина грана мањим или једнаким од B. Другим речима, постоји ли такав низ чворова $(v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)})$ да је

$$\sum_{i=1}^{n-1} d(v_{\pi(i)}, v_{\pi(i+1)}) + d(v_{\pi(n)}, v_{\pi(1)}) \leq B?$$

Полиномијални алгоритам за решавање овог проблема није познат. Претпоставимо, међутим, да је за неки конкретан улаз за овај проблем добијен одговор "да". Ако у то посумњамо, можемо захтевати "доказ" тог тврђења — низ чворова који има тражену особину. Кад имамо овакав низ чворова, лако можемо да проверимо да ли је то Хамилтонов циклус, да израчунамо његову дужину и упоредимо је са задатом границом B. Поред тога, очигледно је временска сложеност оваквог алгоритма провере полиномијална у односу на величину улаза.

Управо појам *полиномијалне проверљивости* карактерише класу NP. Приметимо да проверљивост за полиномијално време не повлачи и могућност решавања за полиномијално време. Утврђујући да се за полиномијално време може проверити одговор "да" за проблем трговачког путника, ми не узимамо у обзир време које нам може бити потребно за проналажење једног од могућих Хамилтонових циклуса, чији број расте експоненцијално са бројем чворова графа. Ми само тврдимо да се за сваки задати низ чворова и за конкретан улаз u, за полиномијално време може проверити да ли тај низ "доказује" да је за улаз u одговор "да".

Класа NP се на други начин неформално може дефинисати помоћу појма *недетерминистичког алгоритма*. Такав алгоритам састоји се од две различите фазе: *фазе погађања* и *фазе провере*. За задати улаз u, у првој фази се изводи просто "погађање" неке структуре S. Затим се u и S заједно предају као улаз фази провере, која се изводи на обичан (детерминистички) начин, па се завршава одговором "да" или "не", или се извршава бесконачно дуго. Недетерминистички алгоритам "решава" проблем одлучивања П, ако су за произвољни улаз $u \in U_{\text{П}}$ за овај проблем испуњена следећа два услова:

1. Ако $u \in L_{\text{П}}$, онда постоји таква структура S, чије би погађање за улаз u довело до тога да се фаза провере са улазом (u, S) заврши одговором "да".

2. Ако $u \notin L_{\text{П}}$, онда *не* постоји таква структура S, чије би погађање за улаз u обезбедило завршавање фазе провере са улазом (u, S) одговором "да".

На пример, недетерминистички алгоритам за решавање проблема трговачког путника могао би се конструисати користећи као фазу погађања

избор произвољног низа градова (чворова), а као фазу провере — горе поменути алгоритам "провере доказа" за проблем трговачког путника. Очигледно је да за произвољан конкретан улаз u постоји такво погађање S, да се као резултат рада фазе провере са улазом (u, S) добија "да", онда и само онда, ако за улаз u постоји Хамилтонов циклус тражене дужине.

Каже се да недетерминистички алгоритам који решава проблем одлучивања П ради за "полиномијално време", ако постоји полином p такав да за сваки улаз $u \in U_{\Pi}$ постоји такво погађање S, да се фаза провере са улазом (u, S) завршава са одговором "да" за време $p(|u|)$. Из тога следи да је величина структуре S обавезно ограничена полиномом од величине улаза, јер се на проверу погађања S може утрошити највише полиномијално време.

Класа NP, дефинисана неформално, — то је класа свих проблема одлучивања који при разумном кодирању могу бити решени недетерминистичким (N – nondeterministic) алгоритмом за полиномијално (P – polynomial) време. На пример, проблем трговачког путника припада класи NP.

У сличним неформалним дефиницијама термин "решава" треба опрезно користити. Треба да буде јасно да је основни смисао "полиномијалног недетерминистичког алгоритма" у томе да објасни појам "проверљивости за полиномијално време", а не у томе да буде реални метод решавања проблема одлучивања. За сваки конкретан улаз такав алгоритам има не једно, него неколико могућих извршавања — по једно за свако могуће погађање.

Недетерминистички алгоритми су врло моћни, али њихова снага није неограничена. Постоје проблеми који се не могу ефикасно решити недетерминистичким алгоритмом.

На пример, посматрајмо следећи проблем: да ли је величина максималне клике у задатом графу једнака тачно k? Недетерминистичким алгоритмом лако се може пронаћи клика величине k, ако она постоји, али се не може лако установити (чак ни недетерминистички) да не постоји већа клика.

Изгледа разумно сматрати да су недетерминистички алгоритми моћнији од детерминистичких, али да ли је то тачно? Један начин да се то докаже био би да се пронађе неки проблем који је у класи NP а који није у P. То до сада никоме није пошло за руком. У противном, да би се доказало да су ове две класе једнаке (односно P=NP), требало би показати да сваки проблем из класе NP може да буде решен детерминистичким алгоритмом полиномијалне временске сложености. Ни овакво тврђење нико није успео да докаже (и мало је оних који верују у његову тачност). Проблем утврђивања односа између P и NP познат је као **проблем** P=NP.

Сада ћемо дефинисати две класе, које не само да садрже бројне важне проблеме за које се не зна да ли су у P, него садрже **најтеже** проблеме у NP.

Дефиниција 2. Проблем X је NP-**тежак проблем** ако је сваки проблем из класе NP полиномијално сводљив на X.

Дефиниција 3. Проблем X је NP-**комплетан проблем** ако (1) припада класи NP, и (2) X је NP-тежак.

Последице дефиниције класе NP-тешких проблема је да, ако се за било који NP-тежак проблем докаже да припада класи P, онда је P=NP.

Кук је 1971. године доказао да *постоје* NP-комплетни проблеми. Специјално, навео је пример једног таквог проблема, који ћемо ускоро приказати. Када се зна један NP-комплетан проблем, докази да су други проблеми NP-комплетни постају једноставнији. Да се за нови проблем X докаже да је NP-тежак, довољно је доказати да је неки NP-комплетан проблем полиномијално сводљив на X. То следи из следеће леме.

Лема 10.1. *Проблем X је* NP*-комплетан ако (1) X припада класи* NP*, и (2') постоји* NP*-комплетан проблем Y који је полиномијално сводљив на X.*

Доказ. Проблем Y је према услову (2) NP-тежак, па је сваки проблем у класи NP полиномијално сводљив на Y. Пошто је Y полиномијално сводљив на X, а полиномијална сводљивост је транзитивна релација, сваки проблем у класи NP је полиномијално сводљив и на проблем X.

Много је лакше доказати да су два проблема полиномијално сводљива, него директно доказати да је испуњен услов (2). Због тога је Куков резултат камен темељац целе теорије. Даље, са појавом нових проблема за које се зна да су NP-комплетни, расте број могућности за доказивање услова (2'). Убрзо пошто је Куков резултат постао познат, Карп (R. M. Karp) је за 24 важна проблема показао да су NP-комплетни. Од тог времена је за стотине проблема (можда хиљаде, зависно од начина бројања варијација истог проблема) доказано да су NP-комплетни. У следећем одељку приказаћемо пет примера NP-комплетних проблема, са доказима њихове NP-комплетности. Навешћемо такође (без доказа) више NP-комплетних проблема. Најтежи део доказа је обично (не увек) доказ испуњености услова (2').

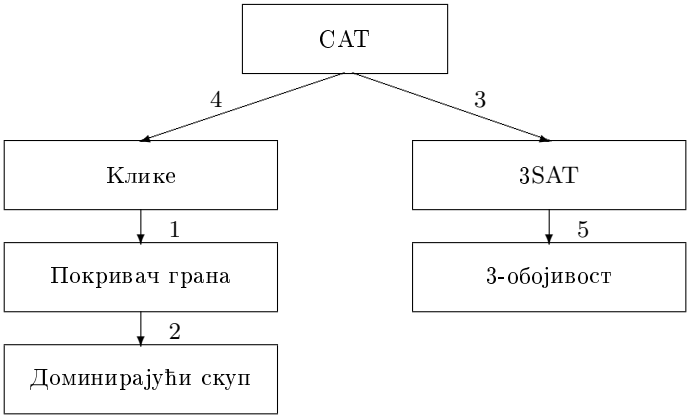
Изложићемо сада проблем за који је Кук доказао да је NP-комплетан. Проблем је познат као **проблем задовољивости** (SAT, скраћеница од satisfiability). Нека је S Булов израз у **конјуктивној нормалној форми** (КНФ). Другим речима, S је конјункција више *клауза* (дисјункција група *литерала* — симбола променљивих или њихових негација). Као пример може да послужи израз $S = (x + y + \bar{z}) \cdot (\bar{x} + y + z) \cdot (\bar{x} + \bar{y} + \bar{z})$, где сабирање, односно множење одговарају дисјункцији, односно конјункцији (*или*, односно *и*), а свака променљива има вредност 0 (нетачно) или 1 (тачно). Познато је да се свака Булова функција може представити изразом у КНФ. За Булов израз се каже да је **задовољив**, ако постоји такво додељивање нула и јединица променљивим, да израз има вредност 1. Проблем SAT састоји се у утврђивању да ли је задати израз задовољив (при чему није неопходно пронаћи одговарајуће вредности променљивих). У наведеном примеру израз S је задовољив, јер за $x = 1$, $y = 1$ и $z = 0$ има вредност $S = 1$.

Проблем SAT је у класи NP јер се за (недетерминистички) изабране вредности променљивих за полиномијално време (од величине улаза — укупне дужине формуле) може проверити да ли је израз тачан. Проблем SAT је и NP-тежак. То није лако доказати.

Теорема 10.3 (Кукова теорема)**.** *Проблем* SAT *је* NP*-комплетан.*

10.4 Примери доказа NP-комплетности

У овом одељку доказаћемо да су NP-комплетни следећих пет проблема: покривач грана, доминирајући скуп, 3 SAT, 3-обојивост и проблем клика. Сваки од ових проблема биће детаљније изложен. Доказивање NP-комплетности заснива се на техникама које ће бити резимиране на крају одељка. Да би се доказала NP-комплетност неког проблема, мора се најпре доказати да он припада класи NP, што је обично (али не увек!) лако, а затим треба пронаћи редукцију полиномијалне временске сложености на наш проблем неког проблема за који се зна да је NP-комплетан. Редослед редукција у доказима NP-комплетности пет наведених проблема приказан је на слици 10.1. Да би се ови докази лакше разумели, наведени су редоследом према тежини, а не према растојању од корена стабла на слици 10.1; редослед доказа приказан је бројевима уз гране.



Слика 10.1: Редослед доказа NP-комплетности у тексту.

10.4.1 Покривач грана

Нека је $G = (V, E)$ неусмерени граф. **Покривач грана** графа G је такав скуп чворова, да је свака грана G суседна бар једном од чворова из скупа.

Проблем. Задат је неусмерени граф $G = (V, E)$ и природни број k. Установити да ли у G постоји покривач грана са $\leq k$ чворова.

Теорема 10.4. *Проблем покривач грана је* NP-*комплетан.*

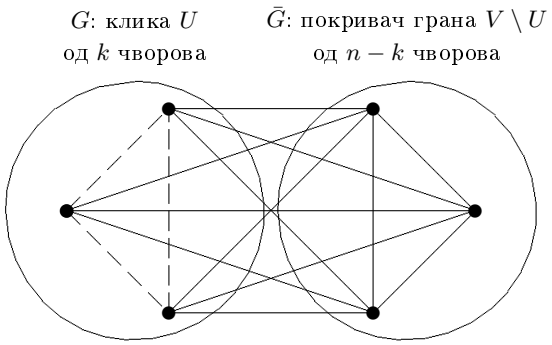
Доказ. Проблем покривач грана је у класи NP, јер се за претпостављени подскуп од $\leq k$ чворова лако проверава за полиномијално време да ли је покривач грана графа. Да бисмо доказали да је проблем покривач грана NP-комплетан, треба да на њега сведемо неки NP-комплетан проблем. У ту сврху искористићемо проблем клика (доказ да је проблем клика NP-комплетан биће дат у одељку 10.4.4). У неусмереном графу $G = (V, E)$ **клика** је такав подграф C графа G у коме су сви чворови међусобно повезани гранама из G. Другим речима, клика је комплетан подграф.

Проблем клика гласи: за задати граф G и природни број k, установити да ли G садржи клику величине k. Потребно је трансформисати произвољан улаз за проблем клика у улаз за проблем покривач грана, тако да је решење проблема клика "да" акко је "да" решење одговарајућег проблема покривач грана. Нека је $G = (V, E)$, k произвољан улаз за проблем клика. Нека је $\bar{G} = (V, \bar{E})$ **комплемент графа** G, односно граф са истим скупом чворова као и G, и комплементарним скупом грана у односу на G (односно између произвољна два чвора у $\bar{G}$ грана постоји акко између та два чвора у G *не* постоји грана). Нека је $n = |V|$. Тврдимо да је проблем клика (G, k) сведен на проблем покривач грана графа, са улазом $\bar{G}$, $n - k$ (видети пример на слици 10.2, где су испрекиданим линијама приказане гране из G).

- Претпоставимо да је $C = (U, F)$ клика у G. Скуп чворова $V \setminus U$ покрива све гране у $\bar{G}$, јер у $\bar{G}$ нема грана које повезују чворове из U (све те гране су у G). Према томе, $V \setminus U$ је покривач грана за $\bar{G}$. Другим речима, ако G има клику величине k, онда $\bar{G}$ има покривач грана величине $n - k$.

- Обрнуто, нека је D покривач грана у $\bar{G}$. Тада D покрива све гране у $\bar{G}$, па у $\bar{G}$ не може да постоји грана која повезује нека два чвора из $V \setminus D$. Према томе, $V \setminus D$ је клика у G. Закључујемо да ако у $\bar{G}$ постоји покривач грана величине $n - k$, онда у G постоји клика величине k.

Ова редукција се очигледно може извршити за полиномијално време: потребно је само конструисати граф $\bar{G}$ полазећи од графа G и израчунати разлику $n - k$.



Слика 10.2: Редукција проблема клика на проблем покривач грана.

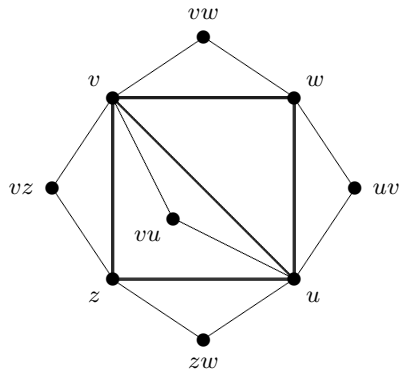
10.4.2 Доминирајући скуп

Нека је $G = (V, E)$ неусмерени граф. **Доминирајући скуп** је скуп $D \subset V$, такав да је сваки чвор G у D, или је суседан бар једном чвору из D.

Проблем. Дат је неусмерени граф $G = (V, E)$ и природни број k. Установити да ли у G постоји доминирајући скуп са највише k чворова.

Теорема 10.5. *Проблем доминирајући скуп је* NP*-комплетан.*

Доказ. Проблем доминирајући скуп је у класи NP, јер се за претпостављени подскуп од највише k чворова лако за полиномијално време проверава да ли је доминирајући скуп. Извешћемо редукцију проблема покривач грана на проблем доминирајући скуп. Ако је задат произвољан улаз (G, k) за проблем покривач грана, циљ је конструисати нови граф G' који има доминирајући скуп одређене величине акко G има покривач грана величине највише k. При томе се, без смањења општости, може претпоставити да G нема изолованих чворова (они не утичу на покривач грана, али морају бити укључени у доминирајући скуп). Полазећи од графа G, додајемо му $|E|$ нових чворова и $2|E|$ нових грана на следећи начин. За сваку грану (v, w) из G додајемо нови чвор vw и две нове гране (v, vw) и (w, vw) (видети пример на слици 10.3). Другим речима, сваку грану трансформишемо у троугао. Означимо нови граф са G'. Граф G' је лако конструисати за полиномијално време.



Слика 10.3: Редукција проблема покривач грана на проблем доминирајући скуп.

Тврдимо да G има покривач грана величине m акко G' има доминирајући скуп величине m.

- Нека је D доминирајући скуп графа G'. Ако D садржи било који од нових чворова vw, онда се такав чвор може заменити било чвором v, било чвором w, после чега ће и нови скуп бити доминирајући скуп (v и w покривају све чворове које покрива vw). Према томе, без смањења општости може се претпоставити да D садржи само чворове из G. Међутим, пошто D "доминира" над свим новим чворовима, он мора за сваку грану из G да садржи бар један њен крај, па је због тога D истовремено и покривач грана графа G.

- Обрнуто, ако је C покривач грана у G, онда је свака грана G суседна неком чвору из C, па је и сваки нови чвор из G' суседан неком чвору из C. Стари чворови су такође покривени чворовима из C, јер по претпоставци чворови из C покривају све гране.

10.4.3 3SAT

Проблем 3SAT је упрошћена верзија обичног проблема SAT. Улаз за проблем 3SAT је КНФ у којој свака клауза има тачно три литерала.

Проблем. Задат је Булов израз у КНФ, у коме свака клауза садржи тачно три литерала. Установити да ли је израз задовољив.

Теорема 10.6. *Проблем* 3SAT *је* NP*-комплетан.*

Доказ. Овај проблем је на први поглед лакши од обичног проблема SAT, због допунског ограничења да свака клауза има по три променљиве. Показаћемо да алгоритам који решава 3SAT може да се искористи да реши обичан проблем SAT (односно да се SAT може свести на 3SAT). Пре тога, јасно је да 3SAT припада класи NP. Могу се изабрати ("погодити") вредности променљивих и за полиномијално време проверити да ли је израз тачан. Нека је E произвољан улаз за SAT. Сваку клаузу у E заменићемо са неколико клауза од по тачно три литерала. Нека је $C = (x_1 + x_2 + \cdots + x_k)$ произвољна клауза из E, таква да је $k \geq 4$. Овде је због удобности са x_i означен литерал, односно било променљива, било негација променљиве. Показаћемо како се C може еквивалентно заменити са неколико клауза од по тачно три литерала. Идеја је увести нове променљиве $y_1, y_2, \ldots, y_{k-3}$, које клаузу трансформишу у део улаза за 3SAT, не мењајући задовољивост израза. За сваку клаузу из E уводе се нове, различите променљиве; C се замењује конјункцијом клауза C', тако да је

$$C' = (x_1+x_2+y_1)(x_3+\bar{y}_1+y_2)(x_4+\bar{y}_2+y_3)\cdots(x_{k-2}+\bar{y}_{k-4}+y_{k-3})(x_{k-1}+x_k+\bar{y}_{k-3}).$$

Тврдимо да је израз, добијен од E заменом C са C', задовољив акко је задовољив израз E.

- Ако је израз E задовољив, онда бар један од литерала x_i мора имати вредност 1. У том случају се могу изабрати вредности променљивих y_i у C' тако да све клаузе у C' буду тачне. На пример, ако је $x_3 = 1$, онда се може ставити $y_1 = 1$ (што чини тачном прву клаузу), $y_2 = 0$ (друга клауза је тачна због $x_3 = 1$), и $y_i = 0$ за све $i > 2$. Уопште, ако је $x_i = 1$, онда стављамо $y_1 = y_2 = \cdots = y_{i-2} = 1$ и $y_{i-1} = y_i = \cdots = y_{k-3} = 0$, што обезбеђује да буде $C' = 1$.
- Обрнуто, ако израз C' има вредност 1, тврдимо да бар један од литерала x_i мора имати вредност 1. Заиста, ако би сви литерали x_i имали вредност 0, онда би израз C' имао исту тачност као и израз $C'' = (y_1) \cdot (\bar{y}_1 + y_2) \cdot (\bar{y}_2 + y_3) \cdots (\bar{y}_{k-4} + y_{k-3}) \cdot (\bar{y}_{k-3})$, који очигледно није задовољив (да би било $C'' = 1$, морало би да буде редом $y_1 = 1$, па $y_2 = 1$, итд, $y_{k-4} = 1$, $y_{k-3} = 0$, и $y_{k-3} = 1$, што је контрадикција).

Помоћу ове редукције све клаузе са више од три литерала могу се заменити са неколико клауза од по тачно три литерала. Остаје да се трансформишу клаузе са једним или два литерала. Клауза облика $C = (x_1 + x_2)$ замењује се еквивалентном изразом

$$C' = (x_1 + x_2 + z)(x_1 + x_2 + \bar{z}),$$

где је z нова променљива. Лако се показује да је почетни израз задовољив акко је задовољив израз добијен заменом C са C'.

Коначно, клауза облика $C = x_1$ може се заменити изразом

$$C' = (x_1 + y + z)(x_1 + \bar{y} + z)(x_1 + y + \bar{z})(x_1 + \bar{y} + \bar{z}),$$

у коме су y и z нове променљиве. Лако се показује да је почетни израз задовољив акко је задовољив израз добијен заменом C са C'.

Према томе, произвољни улаз за проблем SAT може се свести на улаз за проблем 3 SAT, тако да је први израз задовољив акко је задовољив други. Јасно је да се ова редукција изводи за полиномијално време.

10.4.4 Клике

Проблем клика дефинисан је у одељку 10.4.1, у коме је разматран проблем покривач грана.

Проблем. Дат је неусмерени граф $G = (V, E)$ и природни број k. Установити да ли G садржи клику величине бар k.

Теорема 10.7. *Проблем клика је* NP*-комплетан.*

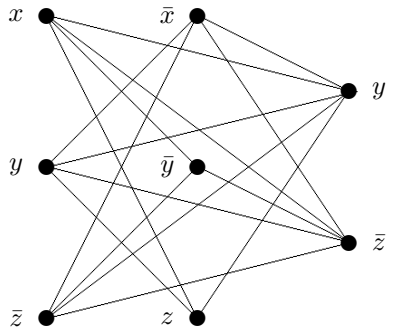
Доказ. Проблем клика је у класи NP, јер се за сваки претпостављени подскуп од k чворова за полиномијално време може проверити да ли је клика. Показаћемо сада да се проблем SAT може свести на проблем клика. Нека је E произвољни Булов израз у КНФ, $E = E_1 \cdot E_2 \cdots E_m$. Посматрајмо, на пример, клаузу $E_i = (x + y + z + w)$. Њој придружујемо "колону" од четири чвора, означена литералима из E_i (без обзира што се неки од њих можда појављују и у другим клаузама). Другим речима, граф G који конструишемо имаће по један чвор за сваку појаву било које променљиве. Остаје питање како повезати ове чворове, тако да G садржи клику величине бар k акко је израз E задовољив. Приметимо да се вредност k може изабрати произвољно, јер је потребно свести проблем SAT на проблем клика, односно решити проблем SAT користећи алгоритам за решавање проблема клика. Наравно, алгоритам за решавање проблема клика мора да ради за сваку вредност k. Ово је важна флексибилност, која се често користи у доказима NP-комплетности. У овом случају за k ћемо изабрати вредност једнаку броју клауза m.

Гране у графу G могу се задати на следећи начин. Чворови из исте колоне (односно чворови придружени литералима из исте клаузе) не повезују се гранама. Чворови из различитих колона су скоро увек повезани: изузетак је случај два чвора од којих један одговара променљивој, а други комплементу те исте променљиве. Пример графа који одговара изразу

$$E = (x + y + \bar{z}) \cdot (\bar{x} + \bar{y} + z) \cdot (y + \bar{z})$$

приказан је на слици 10.4. Јасно је да се G може конструисати за полиномијално време.

Тврдимо да G има клику величине бар m, акко је израз E задовољив. Најпре запажамо да због конструкције максимална клика не може имати више од m чворова, независно од E.



Слика 10.4: Редукција проблема SAT са улазом $(x+y+\bar{z})\cdot(\bar{x}+\bar{y}+z)\cdot(y+\bar{z})$ на проблем клика.

- Претпоставимо да је израз E задовољив. Тада постоји такво додељивање вредности променљивим, да у свакој клаузи постоји бар један литерал са вредношћу 1. Чвор који одговара том литералу прикључује се клики (ако има више таквих литерала, бира се произвољан од њих). Добијени подграф јесте клика, јер једини начин да два чвора из различитих колона не буду повезана је да један од њих буде комплемент другог — што је немогуће, јер је свим изабраним литералима додељена вредност 1.
- Обрнуто, претпоставимо да G садржи клику величине бар m. Клика мора да садржи тачно један чвор из сваке колоне (јер по конструкцији чворови из исте колоне нису повезани). Одговарајућим литералима додељујемо вредност 1. Ако на овај начин некој променљивој није додељена вредност, то се може учинити на произвољан начин. Изведено додељивање вредности променљивима је непротивречно: ако би некој променљивој x и њеном комплементу $\bar{x}$, укљученим у клику, истовремено била додељена вредност 1, то би значило да одговарајући чворови (по конструкцији) нису повезани — супротно претпоставци да су оба чвора у клики.

10.4.5 3-обојивост

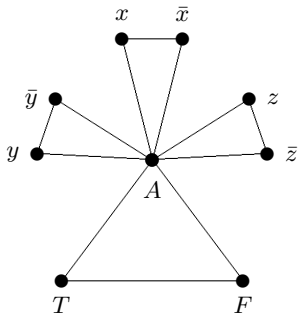
Нека је $G = (V, E)$ неусмерени граф. **Исправно бојење** (или само бојење) графа G је такво придруживање боја чворовима, да је сваком чвору придружена нека боја, а да су суседним чворовима увек придружене различите боје.

Проблем (3-обојивост)**.** Дат је неусмерени граф $G = (V, E)$. Установити да ли се G може обојити са три боје.

Теорема 10.8. *Проблем 3-обојивост је* NP*-комплетан.*

Доказ. Проблем 3-обојивост припада класи NP, јер се може претпоставити произвољно бојење графа са 3 боје, а затим за полиномијално време проверити да ли је претпостављено бојење исправно. Извешћемо редукцију

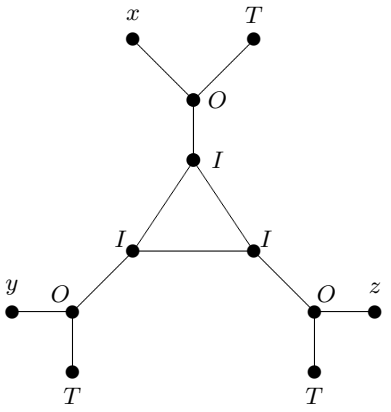
проблема 3SAT на проблем 3-обојивост. Доказ је нешто компликованији из два разлога. Најпре, проблеми се односе на различите објекте — Булове изразе у КНФ, односно графове. Друго, не може се просто заменити један објекат (на пример чвор или грана) другим (на пример клаузом); мора се водити рачуна о комплетној структури. Идеја је да се искористе саставни елементи, који се повезују у целину. Нека је E произвољан улаз за проблем 3SAT. Треба конструисати граф G, тако да је израз E задовољив акко је G 3-обојив. Најпре конструишемо *основни троугао* M. Пошто је M троугао (комплетни граф са три чвора), за његово бојење потребне су тачно три боје. Означимо те боје са T (тачно), F (нетачно) и A, видети доњи троугао на слици 10.5. Поред тога, за сваку променљиву x додајемо нови троугао M_x, чије чворове означавамо са x, $\bar{x}$ и A, при чему се чвор означен са A поклапа са чвором из M са истом ознаком. Према томе, ако се у изразу појављује k променљивих, имамо $k+1$ троуглова са заједничким чвором A (слика 10.5). Ово за сада обезбеђује да, ако је, на пример, чвор x обојен бојом T, онда чвор $\bar{x}$ мора бити обојен бојом F (јер су оба чвора суседна чвору обојеном бојом A), и обрнуто. Ово је у складу са значењем $\bar{x}$.



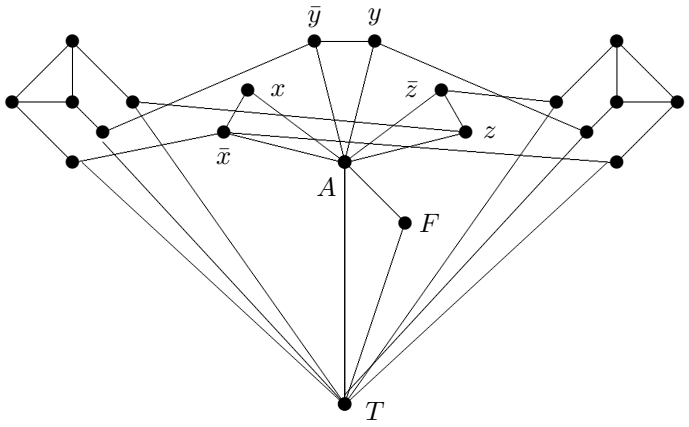
Слика 10.5: Први део конструкције за редукцију 3SAT на 3-обојивост графа.

Потребно је додати услов, који би обезбеђивао да у свакој клаузи бар један литерал има вредност 1. То се може обезбедити на следећи начин. Претпоставимо, на пример, да се ради о клаузи $(x+y+z)$. Овде се x, y, z могу сматрати литералима, тј. недостатак негација над симболима x, y, z не смањује општост разматрања. За ову клаузу уводимо шест нових чворова и повезујемо их са постојећим чворовима на начин приказан на слици 10.6. Назовимо три нова чвора повезана са T и x, y или z *спољашњим чворовима* (означени су са O на слици), а три нова чвора у троуглу — *унутрашњим чворовима* (означени су са I на слици). Тврдимо да ова конструкција обезбеђује (ако је могуће бојење са три боје) да бар један од чворова x, y или z мора бити обојен бојом T. Ни један од чворова x, y, z не може бити обојен бојом A, јер су сви они повезани са чвором A (видети слику 10.5). Ако би сва три чвора x, y, z били обојени бојом F, онда би три нова спољашња чвора повезана са њима морали бити обојени бојом A, па се унутрашњи троугао не би могао обојити са три боје! Комплетан граф који одговара изразу $(\bar{x}+y+\bar{z})\cdot(\bar{x}+\bar{y}+z)$ приказан је на слици 10.7.

Сада можемо да комплетирамо доказ, и то у два смера: (1) ако је израз E задовољив, G се може обојити са три боје, и (2) ако се G може обојити



Слика 10.6: Подграфови који одговарају клаузама у редукцији 3 SAT на 3-обојивост.



Слика 10.7: Граф који одговара изразу $(\bar{x}+y+\bar{z})\cdot(\bar{x}+\bar{y}+z)$ у редукцији 3 SAT на 3-обојивост.

са три боје, онда је израз E задовољив.

- Ако је израз E задовољив, онда постоји такво додељивање вредности променљивим, да у свакој клаузи бар један литерал има вредност 1. Обојимо чворове графа у складу са њиховим вредностима (са T ако је вредност 1, односно са F у противном). Троугао M обојен је бојама T, F и A на описани начин. У подграфу који одговара клаузи бар један литерал има вредност 1; одговарајући спољашњи чвор бојимо са F, а остале спољашње чворове са A, после чега је бојење унутрашњих чворова лако извести. Према томе, G се може обојити са три боје.

- Обрнуто, ако се G може обојити са три боје, назовимо боје у складу са бојењем троугла M (који мора бити обојен са три боје). Због троуглова на слици 10.5, боје променљивих омогућују непротивречно додељивање вредности променљивим. Конструкција блока са слике 10.6 обезбеђује да је бар један литерал у свакој клаузи обојен са T.

Коначно, јасно је да се граф G може конструисати за полиномијално време, чиме је доказ завршен.

10.4.6 Општа запажања

Размотрићемо укратко неколико општих метода за доказивање NP-комплетности неког проблема Q. Први услов — да Q припада класи NP — обично се лако доказује (не увек). После тога треба изабрати неки проблем за који се зна да је NP-комплетан, а за који изгледа да је повезан, или сличан са Q. Тешко је дефинисати ову "сличност", јер понекад изабрани проблем и Q изгледају јако различито (на пример, проблем клика и SAT). Избор правог проблема који ће бити сведен на Q је понекад изузетно тежак, и захтева велико искуство. Мора се покушати са неколико редукција, док се не дође до погодног проблема.

Важно је да се изведе редукција на Q, полазећи од *произвољног улаза* познатог NP-комплетног проблема. Најчешћа грешка у оваквим доказима је извођење редукције у обрнутом смеру. Прави смер обезбеђује нпр. следеће правило: треба обезбедити да се познати NP-комплетан проблем може решити црном кутијом, која извршава алгоритам за решавање Q. То можда мало противречи интуицији. Природно би било покушати са решавањем проблема Q, пошто је то задати проблем. Овде пак покушавамо да решимо други проблем (познати NP-комплетан проблем), користећи решење проблема Q. Ми и не покушавамо да решимо Q!

Постоји више "степени слободе" који се могу користити при редукцији. На пример, ако Q садржи неки параметар, онда се његова вредност може фиксирати на произвољан начин (супротно од параметара у проблему који се своди на Q, који се не смеју фиксирати!). Пошто је Q само алат за решавање познатог NP-комплетног проблема, може се искористити на произвољан начин. Поред фиксирања параметара, могу се користити рестрикције Q на специјалне случајеве добијене и на друге начине. На пример, могу се користити само неки типови улаза за Q (ако се ради о графовима — бипартитни графови, стабла и слично). Други важан извор флексибилности је чињеница да је ефикасност редукције небитна — довољно је да се редукција може извести за полиномијално време. Могу се игнорисати не само константе, тако што би се, на пример, дуплирала величина проблема: исто тако може се и квадрирати величина проблема! Може се увести полиномијално много нових променљивих, може се заменити сваки чвор у графу новим великим графом и слично. Не постоји потреба за ефикасношћу (све док се остаје у границама полиномијалног), јер сврха редукције није да се трансформише у алгоритам.

Постоје неке уобичајене технике за добијање редукција. Најједноставнија је доказати да је познати NP-комплетан проблем специјалан случај проблема Q. Ако је тако, доказ је директан, јер је решавање Q истовремено и решавање познатог NP-комплетног проблема. Посматрајмо, на пример, проблем **покривање скупова**. Улаз је колекција подскупова $S_1, S_2, \ldots, S_n$ задатог скупа U и природни број k. Проблем је установити да ли постоји подскуп $W \subseteq U$ са највише k елемената, који садржи бар по један елеменат сваког од подскупова S_i. Запажамо да је проблем покривач грана специјални случај проблема покривања скупова у коме U одговара скупу чворова V, а

сваки скуп S_i одговара једној грани и садржи два чвора којима је та грана суседна. Према томе, ако знамо да решимо проблем покривања скупова за произвољне скупове, онда можемо да решимо и проблем покривач грана.

Морамо, међутим, бити пажљиви кад користимо овај приступ. У општем случају није тачно да додавање нових захтева чини проблем тежим. Посматрајмо проблем покривач грана. Претпоставимо да смо додали ограничење да покривач грана не сме да садржи два суседна чвора. Другим речима, тражимо мали скуп чворова који је истовремено и покривач грана и независан скуп (независан скуп је подскуп чворова графа, такав да између његова два произвољна елемента не постоји грана). Овај проблем је на први поглед тежи и од проблема покривач грана и од проблема независан скуп, јер треба бринути о више захтева. Испоставља се, међутим, да је ово лакши проблем, и да се може решити за полиномијално време. Разлог овој појави је у томе што допунски захтеви толико сужавају фамилију подскупова кандидата, да се минимум лако налази.

Друга релативно једноставна техника користи **локалне редукције**. Објекат, улаз за један проблем, пресликава се у објекат који је улаз за други проблем. Пресликавање се изводи на локални начин, независно од осталих објеката. Доказ NP-комплетности проблема доминирајућег скупа изведен је на овај начин. Свака грана у једном графу замењена је троуглом у другом графу. Потешкоћа са овом техником је како на најбољи начин дефинисати одговарајуће објекте.

Најкомпликованија техника је употреба саставних елемената – блокова, као што је то учињено при доказу NP-комплетности проблема 3-обојивост. Блокови обично зависе један од другог, па је њихово независно пројектовање неизводљиво. Морају се имати на уму сви циљеви проблема, да би се могло координирати пројектовање различитих блокова.

10.4.7 Још неки NP-комплетни проблеми

Следећи списак садржи још неколико NP-комплетних проблема, који могу бити корисни као полазна основа за нове редукције. Проналажење правог проблема за редукцију обично је више од половине доказа NP-комплетности.

Хамилтонов циклус: Хамилтонов циклус у графу је прост циклус који садржи сваки чвор графа тачно једном. Проблем је установити да ли задати граф садржи Хамилтонов циклус. Проблем је NP-комплетан и за неусмерене и за усмерене графове. (Редукција проблема покривач грана.)

Хамилтонов пут: Хамилтонов пут у графу је прости пут који садржи сваки чвор графа тачно једном. Проблем је установити да ли задати граф садржи Хамилтонов пут. Проблем је NP-комплетан и за неусмерене и усмерене графове. (Редукција проблема покривач грана.)

Проблем трговачког путника: Нека је задат тежински комплетан граф G и број W. Установити да ли у G постоји Хамилтонов циклус са збиром тежина грана $\leq W$. (Директна редукција проблема Хамилтонов циклус.)

Независан скуп: Независан скуп у графу је подскуп чворова графа, такав да између његова два произвољна елемента не постоји грана. Ако је задат граф G и природни број k, установити да ли G садржи независни

скуп са бар k чворова. (Директна редукција проблема клика.)

3-димензионално упаривање: Нека су X, Y и Z дисјунктни скупови од по k елемената, и нека је M задати скуп тројки (x, y, z) таквих да је $x \in X$, $y \in Y$ и $z \in Z$. Проблем је установити да ли постоји такав подскуп скупа M који сваки елеменат садржи тачно једном. Одговарајући дводимензионални проблем упаривања је обичан проблем бипартитног упаривања. (Редукција проблема 3 SAT.)

Партиција: Улаз је скуп X чијем је сваком елементу x придружена његова величина $s(x)$. Проблем је установити да ли је могуће поделити скуп на два дисјунктна подскупа са једнаким сумама величина. (Редукција проблема 3-димензионално упаривање.)

Приметимо да се овај и следећи проблем могу ефикасно решити алгоритмом *Ranac*, ако су величине мали цели бројеви. Међутим, пошто је величина улаза сразмерна броју бита потребних да се представи улаз, овакви алгоритми (који се зову **псеудополиномијални алгоритми**) су уствари експоненцијални у односу на величину улаза.

Проблем ранца: Улаз су два броја S, V и скуп X чијем је сваком елементу x придружена величина $s(x)$ и вредност $v(x)$. Проблем је установити да ли постоји подскуп $B \subseteq X$ са укупном величином $\leq S$ и укупном вредношћу $\geq V$. (Редукција проблема партиције.)

Проблем паковања: Улаз је низ бројева $a_1, a_2, \ldots, a_n$ и два броја b, k. Проблем је установити да ли се овај скуп може разложити у k подскупова, тако да је сума бројева у сваком подскупу $\leq b$. (Редукција проблема партиције.)

10.5 Технике за рад са NP-комплетним проблемима

Појам NP-комплетности је основа за елегантну теорију која омогућује препознавање проблема за које највероватније не постоји полиномијални алгоритам. Међутим, доказивањем да је проблем NP-комплетан, сам проблем није елиминисан! И даље је потребно решити га. Технике за решавање NP-комплетних проблема су понекад другачије од техника које смо до сада разматрали. Ни један NP-комплетан проблем се (највероватније) не може решити тачно и комплетно алгоритмом полиномијалне временске сложености. Због тога смо принуђени на компромисе. Најчешћи компромиси односе се на оптималност, гарантовану ефикасност, или комплетност решења. Постоје и друге алтернативе, од којих свака понешто жртвује. Исти алгоритам се може користити у различитим ситуацијама, примењујући различите компромисе.

Алгоритам који не даје увек оптималан (или тачан) резултат зове се **приближан алгоритам**. Посебно су занимљиви приближни алгоритми који могу да гарантују границу за степен одступања од тачног решења. Нешто касније видећемо три примера таквих алгоритама.

У одељку 5.4 разматрали смо пробабилистичке алгоритме који могу да погреше. Најпознатији у тој класи је алгоритам за препознавање простих бројева. За проблем препознавања простих бројева се до пре десет година није знало да ли је у P, па се сматрало да није NP-комплетан. Испоставило се да је овај проблем у класи P, што су 2002. године доказали Agrawal, Kayal и Saxena. Овде се нећемо бавити алгоритмима за препознавање простих бројева, јер захтевају предзнање из теорије бројева. Раширено

је веровање да се NP-комплетни проблеми не могу решити алгоритмом полиномијалне временске сложености, код којих је вероватноћа грешке мала за све улазе. Такви алгоритми су по свему судећи ефикасни за проблеме, за које се не зна да ли су у класи P, али се не верује да су NP-комплетни. Такви проблеми нису чести. Пробабилистички алгоритми се могу користити као део других стратегија — на пример, као део приближних алгоритама.

Други компромис могућ је у вези са захтевом да полиномијално буде време извршавања у *најгорем случају*. Може се покушати са решавањем NP-комплетних проблема за полиномијално *средње време*. Потешкоћа са овим приступом је како дефинисати *просечно време*. На пример, тешко је искључити улазе за које је конкретан проблем тривијалан (као што је граф који има само изоловане чворове) из израчунавања просека. Такви тривијални улази могу знатно да утичу на просек. Алгоритми предвиђени за поједине типове случајних улаза могу да буду корисни ако је стварна расподела вероватноћа улаза у складу са претпостављеном. Међутим, обично је налажење тачне расподеле врло тешко. Најтежи део посла при конструкцији алгоритама, који у просеку добро раде, је најчешће њихова анализа.

Коначно, могу се правити компромиси у вези са комплетношћу алгоритама; наиме, може се дозволити да алгоритам ради ефикасно само за неке специјалне улазе. На пример, проблем покривач грана може се решити за полиномијално време за бипартитне графове. Према томе, кад се формулише апстрактни проблем полазећи од ситуације из реалног живота, треба обезбедити да сви допунски услови које улаз задовољава буду укључени у апстрактну дефиницију. Други пример су алгоритми са експоненцијалном временском сложеношћу, који се ипак могу извршавати за мале улазе, што је често потпуно задовољавајуће.

10.5.1 Приближни алгоритми са гарантованим квалитетом решења

У овом одељку размотрићемо приближне алгоритме за три NP-комплетна алгоритма: покривач грана, паковање и еуклидски проблем трговачког путника. Сва три алгоритма имају гарантовани квалитет решења; другим речима, може се доказати да решења која они дају нису предалеко од оптималних решења.

Покривач грана

Најпре ћемо размотрити једноставан приближни алгоритам за налажење покривача грана датог графа. Алгоритам гарантује налажење покривача са највише два пута више чворова него у минималном покривачу. За задати неусмерени граф $G = (V, E)$ **упаривање** је скуп дисјунктних грана (грана без заједничких чворова). **Максимално упаривање** је упаривање које се не може проширити додавањем нове гране. Максимално упаривање може се наћи простим додавањем грана све дотле док је то могуће (односно док постоје нека два неупарена, а суседна чвора).

Нека је G граф, и нека је M максимално упаривање у G. Пошто је M упаривање, његове гране немају заједничких тачака, а пошто је M максимално упаривање, све остале гране имају бар један заједнички чвор са неком граном из M.

Теорема 10.9. *Скуп чворова суседних гранама максималног упаривања M је покривач грана, са највише два пута више чворова него што их има минимални покривач.*

Доказ. Скуп крајева грана из M је покривач грана, јер је M максимално упаривање (ако би постојала непокривена грана, та грана могла би да се укључи у упаривање, што је немогуће, јер је претпостављено да је упаривање максимално). Сваки покривач грана, па и оптимални, мора да покрије све гране — специјално, све гране упаривања M. Пошто је M упаривање, било који чвор из M може да покрије највише једну грану упаривања M. Према томе, бар један крај сваке гране из M мора да припада оптималном покривачу грана.

Једнодимензионално паковање

Једнодимензионални проблем паковања односи се на паковање објеката различите величине у *кутије* тако да се искористи најмањи могући број кутија. На пример, приликом селидбе је циљ пренети све ствари, користећи камион најмањи могући број пута, пакујући ствари што је могуће боље. То је наравно тродимензионални проблем; овде ћемо се позабавити његовом једнодимензионалном верзијом. Због једноставности се претпоставља да сви сандуци имају величину 1.

Проблем. Нека је $x_1, x_2, \ldots, x_n$ скуп реалних бројева између 0 и 1. Поделити их у најмањи могући број подскупова, тако да сума бројева у сваком подскупу буде највише 1.

На једнодимензионални проблем паковања наилази се, на пример, у проблемима управљања меморијом, кад постоје захтеви за доделом меморијских блокова различите величине, а додељивање се врши из неколико једнаких великих блокова меморије. Једнодимензионални проблем паковања је NP-комплетан.

Један од могућих хеуристичких алгоритама за решавање овог проблема је ставити x_1 у прву кутију, а затим, за свако i, ставити x_i у прву кутију у којој има довољно места, или започети са новом кутијом, ако нема довољно места ни у једној од коришћених кутија. Овај алгоритам зове се **први одговарајући** и, као што показује следећа теорема, довољно је добар у најгорем случају.

Теорема 10.10. *Алгоритам први одговарајући захтева највише $2m$ кутија, где је m најмањи могући број кутија.*

Доказ. По завршетку алгоритма први одговарајући не постоје две кутије са искоришћењем мањим од $1/2$. Према томе, ако са k означимо број употребљених кутија, биће $m \geq \sum_{i=1}^{n} x_i > (k-1)/2$, одакле је $k < 2m + 1$, односно $k \leq 2m$.

Испоставља се да је граница дефинисана овом теоремом прилично груба. Константа 2 из теореме може се смањити на 1.7 после нешто компликованије анализе. Константа 1.7 не може се даље смањити, јер постоје примери

у којима алгоритам први одговарајући захтева 1.7 пута више кутија од оптималног алгоритма.

Описани алгоритам може се једноставно побољшати. До најгорег случаја долази кад се много малих бројева појављује на почетку. Уместо да се бројеви пакују у реду којим наилазе, они се најпре сортирају у нерастући низ. Промењени алгоритам зове се **опадајући први одговарајући**, и у најгорем случају троши највише око 1.22 пута више кутија од оптималног алгоритма (теорему дајемо без доказа).

Теорема 10.11. *Алгоритам опадајући први одговарајући захтева највише* $\frac{11}{9}m + 4$ *кутија, где је* m *најмањи могући број кутија.*

Константа 11/9 је такође најбоља могућа. Први одговарајући и опадајући први одговарајући су једноставне хеуристике. Постоје други методи који гарантују још мање константе. Њихова анализа је у већини случајева компликована.

Стратегије које смо описали типичне су за хеуристичке алгоритме. Оне одражавају природне приступе, односно одговарају начину на који би неко ручно решавао ове проблеме. Међутим, видели смо много случајева у којима се директни приступи понашају лоше за велике улазе. Због тога је веома важно анализирати понашање таквих алгоритама.

Еуклидски проблем трговачког путника

Проблем трговачког путника (TSP, скраћеница од traveling salesman problem) је важан проблем са много примена. Размотрићемо овде варијанту TSP са допунским ограничењем да тежине грана одговарају еуклидским растојањима.

Проблем. Нека је $C_1, C_2, \ldots, C_n$ скуп тачака у равни које одговарају положајима n градова. Пронаћи Хамилтонов циклус минималне дужине (маршруту трговачког путника) међу њима.

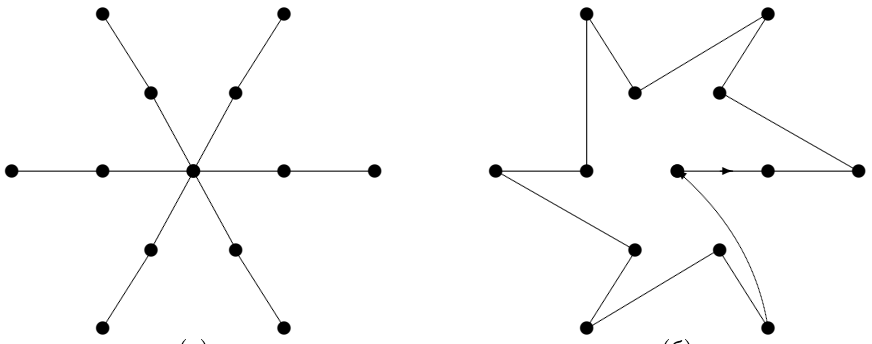
Проблем је и даље NP-комплетан (ово тврђење наводимо без доказа), али видећемо да претпоставка да су растојања еуклидска омогућује конструкцију приближног алгоритма за његово решавање. Прецизније, ова претпоставка може се уопштити, замењујући је претпоставком да растојања задовољавају **неједнакост троугла**, односно да је растојање између два чвора увек мање или једнако од дужине произвољног пута између њих преко осталих чворова.

Најпре се конструише минимално повезујуће стабло (MCST; овде су цене грана њихове дужине), што је много лакши проблем (видети одељак 7.8). Тврдимо да цена стабла није већа од дужине најбољег циклуса TSP. Заиста, циклус TSP садржи све чворове, па га уклањање једне гране чини повезујућим стаблом, чија цена је већа или једнака од цене MCST.

Од повезујућег стабла се, међутим, не добија директно циклус TSP. Посматрајмо најпре циклус који се добија претрагом у дубину овог стабла (полазећи од произвољног чвора), и садржи грану у обрнутом смеру увек кад се грана приликом враћања пролази у супротном смеру (овај циклус одговара, на пример, обиласку галерије у облику стабла, са сликама на оба зида сваког ходника, идући увек удесно). Свака грана се тако пролази

тачно два пута, па је цена овог циклуса највише два пута већа од цене MCST. На крају се овај циклус преправља у циклус TSP, идући пречицом увек кад треба проћи кроз грану већ укључену у циклус (видети слику 10.8). Другим речима, уместо повратка старом граном, идемо директно до првог непрегледаног чвора. Претпоставка да су растојања еуклидска је важна, јер обезбеђује да је увек директни пут између два града бар толико добар, колико било који заобилазни пут.

Теорема 10.12. *Дужина добијеног циклуса* TSP *је мања од двоструке дужине минималног циклуса* TSP.



Слика 10.8: (а) Минимално повезујуће стабло (MCST) (б) циклус TSP добијен од MCST полазећи од средњег чвора и идући увек удесно.

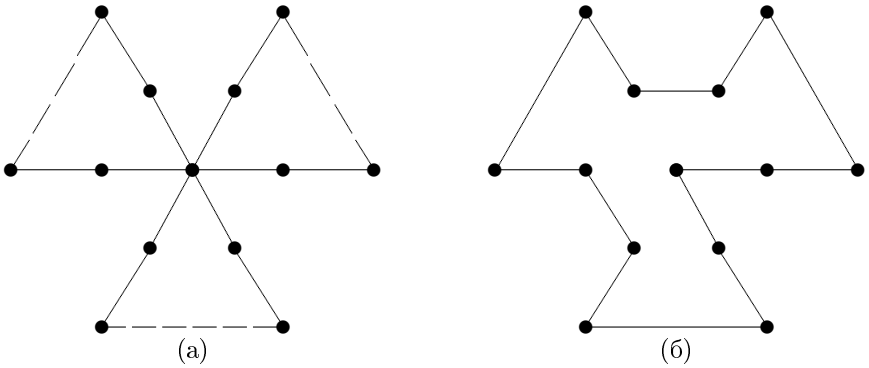
Сложеност. Временска сложеност овог алгоритма одређена је бројем корака у алгоритму за конструкцију MCST.

Побољшање

Алгоритам који смо управо описали може се побољшати на следећи начин. Његов "најгрубљи" део је претварање обиласка стабла у циклус TSP. Та конверзија може се посматрати и на други начин: она формира Ојлеров циклус од стабла у коме је свака грана удвостручена. После тога се конструише циклус TSP коришћењем пречица у Ојлеровом циклусу. Конверзија стабла у Ојлеров циклус може се извести много рационалније. Ојлеров граф може да садржи само чворове парног степена. Посматрајмо све чворове непарног степена у стаблу. Број таквих чворова мора бити паран (у противном би сума степена чворова била непарна, што је немогуће, јер је та сума једнака двоструком броју грана). Ојлеров граф се од стабла може добити додавањем довољног броја грана, чиме се постиже да степени свих чворова постану парни. Пошто се циклус TSP састоји од Ојлеровог циклуса (са неким пречицама), волели бисмо да минимизирамо збир дужина додатих грана. Размотримо сада тај проблем.

Дато је стабло у равни, а циљ је додати му неке гране, са минималном сумом дужина, тако да добијени граф буде Ојлеров. Сваком чвору непарног степена мора се додати бар једна грана. Покушајмо да постигнемо циљ додавањем тачно једне гране сваком таквом чвору. Претпоставимо да има

$2k$ чворова непарног степена. Ако додамо k грана, тако да свака од њих спаја два чвора непарног степена, онда ће степени свих чворова постати парни. Проблем тако постаје проблем *упаривања*. Потребно је пронаћи упаривање минималне дужине које покрива све чворове непарног степена. Налажење оптималног упаривања може се извести за $O(n^3)$ корака за произвољни граф (ово тврђење наводимо без доказа). Коначни циклус TSP се затим добија од Ојлеровог графа (који обухвата минимално повезујуће стабло и упаривање минималне дужине) коришћењем пречица. Циклус TSP добијен овим алгоритмом од стабла са слике 10.8 приказан је на слици 10.9.

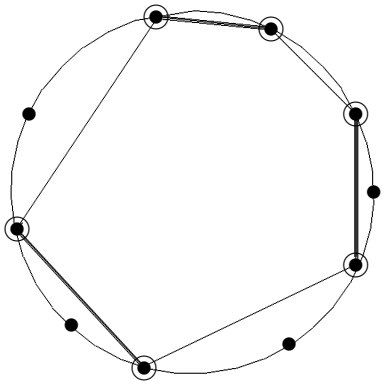


Слика 10.9: Минимални Ојлеров циклус и одговарајући циклус TSP (а) минимално повезујуће стабло са упаривањем, (б) циклус TSP добијен од Ојлеровог циклуса.

Теорема 10.13. *Побољшани алгоритам даје циклус* TSP *чија је дужина највише* 1.5 *пута већа од дужине минималног циклуса* TSP*.*

Доказ. Потребно је оценити дужину Ојлеровог циклуса, јер се дужина циклуса после увођења евентуалних пречица може само смањити. Ојлеров циклус се састоји од стабла и упаривања. Означимо са Q минимални циклус TSP, а са $|Q|$ његову дужину. Видели смо већ да је дужина стабла мања или једнака од $|Q|$; према томе, довољно је доказати да дужина упаривања не прелази $|Q|/2$. Циклус Q садржи све чворове. Нека је D скуп чворова непарног степена у полазном стаблу. За скуп D могу се формирати два дисјунктна упаривања са збиром дужина $\leq |Q|$ на следећи начин (видети слику 10.10, на којој су заокружени чворови из D). Започињемо са произвољним чвором $v \in D$ и упарујемо га са са најближим чвором (у смеру казаљке на сату) дуж циклуса Q. После тога наставља се са упаривањем у истом смеру. Ако упарени чворови нису суседи у Q, онда је растојање између њих мање или једнако од дужине пута који их повезује у Q (због неједнакости троугла). Овај процес даје једно упаривање. Друго упаривање добија се понављањем истог процеса у смеру супротном од казаљке на сату. Сума дужина оба упаривања је највише $|Q|$, видети слику 10.10. Међутим, пошто је M упаривање минималне тежине у D, његова дужина је мања

или једнака од дужине бољег од два споменута упаривања (са збиром $|Q|$), па је дакле мања или једнака од $|Q|/2$.



Слика 10.10: Два упаривања чија сума дужина не прелази дужину минималног циклуса TSP.

Налажење упаривања минималне тежине знатно је сложеније од налажења минималног повезујућег стабла, али се тиме добија боља граница. Наведени пример илуструје основну карактеристику овог типа алгоритама: уопштава се лакши проблем — или се ослабљују (релаксирају) неки елементи полазног проблема — и тако се формира хеуристика.

10.6 Резиме

Претходна поглавља су можда изазвала оптимизам у вези са могућношћу конструкције добрих алгоритама. Ово поглавље треба да нас приближи реалности. Има много важних проблема који се, нажалост, не могу решити елегантним, ефикасним алгоритмима. Потребно је да препознајемо такве проблеме и да им проналазимо неоптимална, приближна решења. При нападу на неки проблем на располагању су два приступа. Може се покушати са техникама редукције да би се проблем решио, или могу искористити технике уведене у овом поглављу, и показати да је проблем NP-комплетан. Да би се избегли многобројни безуспешни покушаји пре избора исправног приступа, потребно је развити интуицију за оцену тежине проблема.

Литература

[1] K. Rosen, Discrete Mathematics and Its Applications, McGraw Hill, 2006.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, Second Edition, MIT Press, 2002.

[3] М. Живковић, Алгоритми, Математички факултет, 2000.