

Simulacije (binarna pretraga po rešenju, DFS, IDDFS)

Pri rešavanju zadatka, česta tehnika je simulirati neki od postupaka opisanog u zadatku poznatom strukturon podataka ili poznatom algoritamskom strategijom.

Binarna pretraga po rešenju

1. Deca su se uhvatila za ruke u lanac i trče livadom. Kako je neko dete brže, a neko dete sporije, dešava se da brža deca vuku sporiju decu. U lanac se uhvatilo n ($1 \leq n \leq 10^5$) dece tako da svako dete drži za ruke dva susedna deteta (sem prvog i poslednjeg deteta u lancu kojima je po jedna ruka slobodna). Svako dete ima svoju maksimalnu brzinu x_i ($0 \leq x_i \leq 10^3$) kojom može trčati i deo brzine y_i ($1 \leq x_i \leq 10^3$) koju gubi za svako dete koje ga vuče ili koga on vuče (jer postaje nestabilan i može pasti). Deca se mogu međusobno vući tako što deo svoje snage utroše na povlačenje drugog deteta. Koliko najbrže može trčati lanac dece?

Unos se sastoji od celih brojeva. U prvom redu standardnog ulaza se unosi broj n, a potom sledi unos ne redova sa po n parova brojeva x i y koji opisuju pojedino dete. Deca se drže za ruke onim redom kojim su navedeni u unosu. Ispisati na standardni izlaz najveću brzinu (na 3 decimale) kojom deca mogu trčati u lancu.

ULAZ	IZLAZ
5	77.750
50 4	
111 2	
76 1	
85 1	
110 50	

Analiza zadatka:

Primer 1:

Ako, na primer, trči dvoje dece tako da je dato $x_1=20$, $y_1=3$, $x_2=28$, $y_2=2$, onda

2. dete može preneti 4.5 od svoje brzine na 1. dete. Tada 2. dete izgubi 4.5 svoje brzine i još zbog nestabilnosti izgubi $y_2=2$ brzine, te ukupno trči brzinom $28-4.5-2=21.5$

1. dete dobija 4.5 brzine, tem u je brzina $x_1+4.5=24.5$, ali zbog nestabilnosti izgubi $y_1=3$ brzine, te ukupno trči brzinom $x_1+4.5-y_1=21.5$

Dakle, oba deteta mogu zajedno trčati brzinom 21.5

Primer 2:

Ako, na primer, trči dvoje dece tako da je dato $x_1=20$, $y_1=3$, **$x_2=24$** , $y_2=2$, onda oni ako ne prenose brzinu, mogu zajedno trčati brzinom 20. A ako bi prenosili brzinu, x_1 bi svakako bilo manje, te bi prenosom brzine bili još sporiji.

Primer 3:

Ako, na primer, trči troje dece tako da je dato $x_1=16$, $y_1=2$, $x_2=19$, $y_2=1$, $x_3=32$, $y_3=3$ onda oni ako prenose brzinu, 3. dete može da prenese 2. detetu 9 brzina i tada brzina 3. deteta je $x_3-9-y_3=32-9-3=20$

2. dete primi 9 brzina od 3. deteta i izgubi zbog nestabilnosti sa 3. detetom $y_2=1$ brzinu, te je ukupna brzina $x_2+9-1=27$

ako 2. dete da 1. detetu 6 brzina, onda 2. dete ima brzinu $27-6-1$ (zbog nestabilnosti sa 1. detetom izgubi $y_2=1$ brzinu)=20

1. dete kad primi 6 brzina i izgubi $y_1=2$ zbog nestabilnosti prema 2. detetu, onda 1. dete ima brzinu $x_1+6-y_1=20$
Dakle, 3. dete vuče dvoje dece i zajedno trče brzinom 20

Konstrukcija resenja:

Koliku složenost resenja mozemo da ocekujemo?

Mozemo li u linearnom vremenu odgovoriti na pitanje: Da li deca mogu da trče brzinom x?

Analizirajmo dat primer i kako mozemo u linearном vremenu odgovoriti na pitanje: Da li deca mogu da trče brzinom 77.6?

Moramo, dakle analizirati najpre ili 1. dete ili 5. dete.

1. detetu je potrebno $(77.6-50)+4+2=33.6$ brzine od 2. deteta da bi trcalo brzinom od 77.6

2. dete sada ima $111-33.6=77.4$ brzine <77.6, te je 2. detetu potrebno dati brzinu od 3. deteta

Dakle, 2. detetu treba od 3. deteta $(77.6-77.4)+2+1=3.2$ brzine

Time 3. detetu ostane $76-3.2=72.8$ brzine. Zato 3. detetu od 4. deteta treba $(77.6-72.8)+1+1=6.8$ brzine.

Tada 4. detetu ostane $85-6.8=78.2$ brzine. Dakle, onda 4. dete ima $78.2-77.6=0.6$ brzine viska, ali mu se ne isplati slati brzinu 5. detetu dokle god visak koji se salje je manji od ukupne nestabilnosti oba deteta $1+50=51$, a vazi da visak $0.6<51$

Poslednje dete je 5. dete i ima $110-77.6=32.4$ brzine viska, te znamo da deca mogu trcati brzinom 77.6

Diskusija:

Ako bismo redom isprobavali mogu li deca trcati redom brzinom 0.001, 0.002,...onda slozenost resenja bi bila $O(\text{brojDece} * \text{resenje}/0.001)$, sto je za $n=10^5$, resenje=1000 u najgorem slucaju 10^{11}

Ali, ako zadatku resimo binarnom pretragom po brzini kojom deca trce ($0 \leq x_i \leq 10^3$), onda je slozenost $O(\text{brojDece} * \log(1000/0.001))$, sto je za $n=10^5$, u najgorem slucaju $2 \cdot 10^6$ sto odgovara vremenskom ogranicenju od 1s na masinama 100MIPS

Resenje:

```
#include <stdio.h>
#define MAXN 1000
int n;
int deca[MAXN][2];
int proveri (double x);
//provera da li dete moze da trci brzinom x
int main()
{
    int i;
    double donja, gornja, sred;
    scanf("%d", &n);
    for(i=0;i<n;i++)
        scanf("%d%d", &deca[i][0],&deca[i][1]);

    donja=0; gornja=1000; //inicijalizacija bin.pretrage
    while(gornja-donja>0.0005) //tacnost na 3 decimale
    {
        sred=(donja+gornja)/2;
        if(proveri(sred)) donja=sred;
        else gornja=sred;
    }

    printf("%.3lf\n", (donja+gornja)/2);
    return 0;
}

int proveri(double x)
{
    int i;
    double kolicina=0;
    // Promenljiva kolicina govori da li imamo visak ili manjak brzine i koliko.

    for(i=0;i<n;i++)
    {
        kolicina+=deca[i][0]; //brzina i.-og deteta
        kolicina-=x; //testiramo visak brzine u odnosu na resenje x
        if(i==n-1)
        {
            if (kolicina<0) return 0; //x nije resenje
            else return 1; //x moze biti resenje
        }
    }
}
```

```

}
if (kolicina<0)
    kolicina-=deca[i][1]+deca[i+1][1];
else if (kolicina<=deca[i][1]+deca[i+1][1])
    kolicina=0;
else
    kolicina-=deca[i][1]+deca[i+1][1];
}
return 0; //x ne moze biti resenje, ova linije je samo da compile ne bi generisao warning da nista ne vracamo
//povratni rezultat funkcije se svakako formira 11 redova iznad
}

```

Diskusija implementacije:

14: donja=0; gornja=1000;

U 14. redu inicijalizujemo donju i gornju granicu na min i max resenje zadatka.

15: while(gornja-donja>0.0005) //tacnost na 3 decimale

U 15. liniji zapocinje binarna pretraga cije resenje se ispisuje u 22. liniji

22: printf("%.3lf\n", (donja+gornja)/2);

U funkciji proveri se proverava da li vrednost x moze biti resenje zadatka.

Promenljiva kolicina govori da li imamo visak ili manjak brzine i koliko.

U linijama

kolicina+=deca[i][0]; //brzina i.-og deteta

kolicina-=x; //testiramo visak brzine u odnosu na resenje x

najpre za svako dete dodamo njegovu brzinu, a potom oduzmemmo zeljenu brzinu x.

Ako smo dosli do poslednjeg deteta

if(i==n-1)

{

if (kolicina<0) return 0; //x nije resenje

else return 1; //x moze biti resenje

}

donosimo odluku da li mogu deca trcati brzinom x.

Ako dete nije poslednje, gledamo da li nam nedostaje brzina

if (kolicina<0)

i ako nam nedostaje uzimamo je od sledeceg deteta

kolicina-=deca[i][1]+deca[i+1][1];

Ako nam je ostalo premalo brzine da bi se isplatilo da je prenesemo sledecem detetu, onda je postavimo na 0.

else if (kolicina<=deca[i][1]+deca[i+1][1])

kolicina=0;

Ako nam je ostalo dovoljno visaka brzine da bi je poslali dalje, izracunamo koliko ce ostati sledecem detetu

else

kolicina-=deca[i][1]+deca[i+1][1];

DFS

2. Ako su dati brojevi a,b,c, d ($1 \leq a,b,c,d \leq 8$) koji predstavljaju dve lokacije konja (skakača u šahu koji se kreće u obliku slova L) na šahovskoj tabli i ako je dat broj g ($1 \leq g \leq 10$), proverite može li skakač doći na polje c-d u ne više od g poteza.

ULAZ

1 1 8 8 5

IZLAZ

Ne moze

Ideja rešenja:

DFS (depth first search, pretraga u dubinu) je algoritam pretrage koji potencijalna rešenja obilazi grananjem u dubinu. DFS pretražuje sve situacije (stanja) koja su potencijalno rešenje problema.

Zadatak čemo rešiti DFS-om tj. rekurzivnom funkcijom void obidji(int x, int y, int potez)

Funkcija obidji kao argument ima poziciju polja na tabli (x,y) i broj poteza da bismo došli do tog polja na tabli.

Konj se kreće u obliku slova L, tj. dva polja u proizvolnjem smeru (horizontalnom i vertikalnom) i jedno polje u normalnom pravcu u odnosu na prethodni pravac.

Analiza zadatka i primera:

Dakle, skakač sa polja 5-4 može doći na polje 3 3, 3 5, 4 2, 4 6, 6 2, 6 6, 7 3, 7 5

	X		X					
X				X				
		K						
X				X				
	X		X					

Skakač sa polja x-y može doći na polje

x-1,y-2
x-1,y+2
x+1,y-2
x+1,y+2
x-2,y-1
x-2,y+1
x+2,y-1
x+2,y+1

Za ulaz 1 1 8 8 5 možemo DFSom upisati u koliko poteza smo stigli do neke pozicije na tabli

0	3	2	3	2	3	4	5
3	4	1	2	3	4	3	4
2	1	4	3	2	3	4	5
3	2	3	2	3	4	3	4
2	3	2	3	4	3	4	5
3	4	3	4	3	4	5	4
4	3	4	3	4	5	4	5
5	4	5	4	5	4	5	-1

Implementacija

```
#include <stdio.h>

int tabla[8][8];
int granica;

void obidji (int x,int y,int potez)
{ if(x<0 || x>7 || y<0 || y>7) return; //lose polje
if(tabla[x][y]!=-1 && tabla[x][y]<=potez) return;
```

```

//tu smo vec bili u isto ili manje poteza
if(potez>granica) return; //potrosili smo poteze
tabla[x][y]=potez;
obidji(x-1,y-2,potez+1);
obidji(x-1,y+2,potez+1);
obidji(x+1,y-2,potez+1);
obidji(x+1,y+2,potez+1);
obidji(x-2,y-1,potez+1);
obidji(x-2,y+1,potez+1);
obidji(x+2,y-1,potez+1);
obidji(x+2,y+1,potez+1);
}

int main()
{ int a,b,c,d,i,j;
for(i=0;i<8;i++)
    for(j=0;j<8;j++) tabla[i][j]=-1;
scanf("%d%d%d%d%d", &a,&b,&c,&d,&granica);
a--;b--;c--;d--; //pomicanje polja
//tako da gornje levo polje bude 0-0
obidji(a,b,0);
if(tabla[c][d]!=-1) printf("Moze\n");
else printf("Ne moze!\n");
return 0;
}

```

Diskusija:

Promenljiva granica opisuje koliko najviše poteza skakač sme napraviti.

U glavnoj funkciji main postavimo sva polja 8*8 table na -1 kako bismo označili da su sva polja neposećena na početku rada programa.

U svako polje matrice tabla upisaćemo u koliko poteza smo stigli do neke pozicije na tabli. Kako sa 0 poteza dolazimo do startne pozicije, zato neposećena mesta postavljamo na početku na -1, a ne na 0.

U glavnoj funkciji, naredbama

a--;b--;c--;d--; //pomicanje polja

postavljamo da gornje levo polje table ima poziciju 00, a donje desno polje da ima poziciju 77.

Funkcija obidji kao argument ima poziciju polja na tabli i broj poteza da bismo došli do tog polja na tabli. Rekurziju zaustavljamo:

1. Ako smo izašli van polja šahovske table
if(x<0 || x>7 || y<0 || y>7) return; //lose polje
2. Ako smo potrošili dozvoljen broj poteza
if(potez>granica) return; //potrosili smo poteze
3. Ako smo na polje xy došli sa ranije, sa ne više od *potez* poteza
if(tabla[x][y]!=-1 && tabla[x][y]<=potez) return;

U funkciji obidji, u liniji tabla[x][y]=potez;

Zapisujemo u koliko poteza je skakač došao na polje xy.

Da li je ovaj uslov izlaska bio nužan, tj. da li smo mogli izbaciti ovaj uslov

3. Ako smo na polje xy došli sa ranije, sa ne više od *potez* poteza
if(tabla[x][y]!=-1 && tabla[x][y]<=potez) return;

ODGOVOR: Da, ali bi program bio veoma spor, jer bi se rekurzija ponavljala sa istim ili lošijim parametrima
Na primer, za ulaz 1 1 8 8 5 program sa linijom

```
if(tabla[x][y]!=-1 && tabla[x][y]<=potez) return;  
napravi 729 poziva rekurzijem a bez tog poziva 22681 poziva rekurzije.
```

Program bez linije

```
if(tabla[x][y]!=-1 && tabla[x][y]<=potez) return;  
faktor DFS grananja programa je 8, te je složenost programa O(8g)
```

Program sa linijom

```
if(tabla[x][y]!=-1 && tabla[x][y]<=potez) return;  
može najviše g=granica puta promeniti vrednost nekog polja matrice tabla, te je složenost programa O(64*g), jer  
tabla ima 64 polja
```

3. Napisati program koji unosi sa standardnog ulaza cele brojeva ($1 \leq n, m \leq 50$), a zatim n redova sa po m znakova (samo 'x' i '.'). Uneseni znakovi opisuju zemljiste cije ivice moraju biti oivcene znakovima 'x', tako da 'x' predstavlja kamen, a '.' predstavlja plodno tlo. Nakon unesenog opisa zemljista unose se dva broja koja predstavljaju red i kolone polja (nije kamen) gde se seje trava. Vas program mora da sve znakove '.' zameni sa '+' ako se na odgovarajuce mesto moze prosiriti trava. Trava se siri samo na susedna polja na kojima je plodno tlo. Susedna polja dele zajednicku ivicu. Ispisite izgled zemljista nakon sto se trava prosiri koliko god moze.

ULAZ	IZLAZ
4 10	xxxxxxxxxx
xxxxxxxxxx	x++++++x.xx
x.....x.xx	xxx+++x.xx
xxx...x.xx	x..xxx..xx
x..xxx..xx	
2 2	

Koristicemo rekurzivni DFS, koji nece raditi nikakvu pretragu, nego ce se jednostavno siriti. Koristicemo za memoizaciju (obelezavanje obilaska plusicem) matricu zemlja i obeleziti znakom + svako polje matrice u koje dodjemo 1. put, a da to polje nije kamen, tj. jednako '.'

```
#include <stdio.h>

char zemlja[50][51];
void DFS(int x, int y)
{
    if(zemlja[x][y]!='.') return;
    zemlja[x][y]='+';
    DFS(x-1,y);
    DFS(x+1,y);
    DFS(x,y-1);
    DFS(x,y+1);
}
int main()
{
    int n,m,i,x,y;
    scanf("%d%d", &n,&m);
    for(i=0;i<n;i++) scanf("%s", zemlja[i]);
    scanf("%d%d", &x,&y);
    DFS(x-1,y-1);
    for(i=0;i<n;i++)
        printf("%s\n",zemlja[i]);
    return 0;
}
```

Kad zadatak ne bi jos u formulaciji ogranicio da zemljiste bude omedjeno kamenom, morali bismo u rekurziju dodati ogranicenje koje vodi racuna da ne istupimo van polja.

Vremenska slozenost ovog programa zbog memoizacije je $O(m^*n)$.

Slicno ovom zadatku se resavaju zadaci koji proveravaju moze li se od nekog mesta na zemljistu doci do nekog drugog mesta.

IDDFS

4. Resite zadatak 2, odnosno problem pronalaska minimalnog broj poteza da se skakacem od polja a-b dodje do polja c-d, ali tako na ne koristite matrice.

Ideja resenja: Koristicemo IDDFS tj. iterative deeping DFS kao dopunu tradicionalnoj DFS strategiji. Ideja se sastoji od uzastopnog pozivanja DFSa sa sve vecom granicom dubina do kojih se sme vrsiti grananje.

Ideja IDDFS je da prvo ogranicimo dubinu grananja na 1 i kad rekurzija zavrsi, pozovemo je ponovo ali sa granicom dubine 2, a zatim sa granicom 3,... sve dok ne uspemo doci do ciljne tacke pretrage. Tada znamo da je pronadjeno resenje na najmanjoj dubini, jer prethodni poziv koji je bio za jedan nivo manje dubine nije uspeo doci do resenja.

Ako je faktor grananja manji od 2, nije efikasno koristiti IDDFS.

U zadataku sa skakacem, faktor grananja je 8.

```
#include <stdio.h>

int granica,c,d;
int nasao=0;

void obidji (int x,int y,int potez)
{ if (x==c && y==d) nasao=1;
  if(x<0 || x>7 || y<0 || y>7) return; //lose polje
  if(potez>granica) return; //potrosili smo poteze
  obidji(x-1,y-2,potez+1);
  obidji(x-1,y+2,potez+1);
  obidji(x+1,y-2,potez+1);
  obidji(x+1,y+2,potez+1);
  obidji(x-2,y-1,potez+1);
  obidji(x-2,y+1,potez+1);
  obidji(x+2,y-1,potez+1);
  obidji(x+2,y+1,potez+1);
}

int main()
{ int a,b;
  scanf("%d%d%d%d", &a,&b,&c,&d);
  a--;b--;c--;d--; //pomicanje polja
  //tako da gornje levo polje bude 0-0
  for(granica=0;nasao==0;granica++)
    obidji(a,b,0);

  printf("Potrebno je minimalno %d poteza\n", granica);
  return 0;
}
```

Ovaj program za unos

1 1 8 8

otkriva da je potrebno minimalno 6 poteza.

U IDDFS verziji ovaj program koristi 27334 puta rekurziju.

Da smo program prekinuli u 7. redu tako sto smo odmah nakon if selekcije pozvali exit(0), broj rekurzivnih poziva bi bio i do 8 puta manji, tj. 5799

5. Napisati program koji ce uz minimalan broj poteza sloziti rotacijama slagalicu 3×3 . Potrebno je ispisati sva stanja slagalice obrnutim redosledom. Slagalica 3×3 treba da se slozi u sledeci oblik

1	2	3
4	5	6
7	8	9

Pri tom se mogu koristiti 4 razlicite vrste rotacija: rotacijom se 4 broja koja se medjusobno dodiruju (kvadrat 2×2) zamene u smeru kazaljke na satu.

Od stanja

9	8	7
6	5	4
3	2	1

moze se preci u sledeca stanja

6	9	7	9	5	8	9	8	7
5	8	4	6	4	7	3	6	4
3	2	1	3	2	1	2	5	1

ULAZ

1 2 3

9 6 8

4 7 5

IZLAZ

1 2 3

4 5 6

7 8 9

1 2 3

4 6 9

7 5 8

1 2 3

4 9 8

7 6 5

1 2 3

9 6 8

4 7 5

```
#include <stdio.h>
```

```
int IDDFS(int a, int b, int c, int d, int e,
          int f, int g, int h, int i, int dubina)
{ if (a==1 && b==2 && c==3 && d==4 && e==5
     && f==6 && g==7 && h==8 && i==9)
{
    printf("\n%d %d %d\n%d %d %d\n%d %d %d\n",
           a,b,c,d,e,f,g,h,i);
    return 1;
}
if (dubina==0) return 0;
if (IDDFS(d,a,c,e,b,f,g,h,i,dubina-1)
 || IDDFS(a,e,b,d,f,c,g,h,i,dubina-1)
```

```
|| IDDFS(a,b,c,g,d,f,h,e,i,dubina-1)
|| IDDFS(a,b,c,d,h,e,g,i,f,dubina-1) )
{ printf("\n%d %d %d\n%d %d %d\n%d %d %d\n",
      a,b,c,d,e,f,g,h,i);
  return 1;
}
return 0;
}
int main()
{ int a,b,c,d,e,f,g,h,i,dubina;
  scanf("%d %d %d %d %d %d %d %d",
        &a,&b,&c,&d,&e,&f,&g,&h,&i);
  for(dubina=0;dubina<=12;dubina++)
    if(IDDFS(a,b,c,d,e,f,g,h,i,dubina)) break;
  return 0;
}
```