Algoritmi i strukture podataka
vežbe 8

Mirko Stojadinović

30. novembar 2015

[1]

1 Algoritamske strategije

Postoji više algoritamskih strategija koje se primenjuju kada je potrebno razviti algoritam za neki problem. Ipak, ne postoje garancije da će neka strategija dati tačno rešenje, a i ako se dobije tačno rešenje ono ne mora biti efikasno. Razlog za primenu je taj što su se ove strategije pokazale uspešnim za rešavanje mnogih problema pa je verovatno da će neka od njih dati dovoljno dobro rešenje. Najpoznatije algoritamske strategije su:

1. Gruba sila (brute-force) ili iscrpna pretraga
2. Algoritmi zasnovani na razlaganju ili Podeli pa vladaj algoritmi (divide and conquer): kviksort, sortiranje spajanjem, FFT, Karacuba mnozenje, izračunavanje n-tog Fibonačijevog člana niza rekurzivno preko prethodna 2 člana niza
3. Dinamičko programiranje: problem ranca, najkraći putevi iz zadatog cvora (Dijkstring algoritam), izračunavanje n-tog Fibonačijevog člana niza iterativno, preko prethodna 2 člana niza
4. Pohlepni metodi (greedy method): minimalno povezujuće stablo, algoritam za spajanje po 2 kutije u jednu, kontinualni problem ranca, raspoređivač aktivnosti
5. Bektreking - iscrpna pretraga rešenja uz odsecanja.
6. Grananje i odsecanje (Branch and bound) - koriste se za optimizacione probleme
7. Heuristike
8. Redukcija ili svođenje problema na postojeće (NP-kompletni problemi)

1.1 Podeli pa vladaj (divide and conquer)

Ova strategija rekurzivno razbija problem na 2 ili više potproblema dok oni ne postanu dovoljno jednostavni da se mogu direktno rešiti. Rešenja potproblema se kombinuju da bi se dobilo rešenje početnog problema. Korektnost strategije se obično pokazuje matematičkom indukcijom, a vremenska složenost se dobija rešavanjem diferencnih jednačina. Primeri podeli-pa-vladaj algoritama su binarna pretraga, kviksort, Euklidov algoritam, sortiranje spajanjem itd.

Sortiranje spajanjem (Merge sort) Koristi strategiju podeli pa vladaj za sortiranje niza brojeva. I danas se ovaj algoritam koristi u bibliotekama mnogih programskih jezika. Početni poziv algoritma je Merge_sort (X, 0, n-1)

Algoritam Merge_sort (Sortiranje spajanjem)(X, l, r)
Ulaz niz X i njegove granice l i r
Izlaz sortirani niz X

```
{
  if l < r
     m = (l + r)/2
     Merge_sort(X, l, m)
     Merge_sort(X, m + 1, r)
     Merge(X, l, m, r)
}
```

[1] Materijali velikim delom preuzeti od Jelene Hadži-Purić: `www.math.rs/~jelenagr`

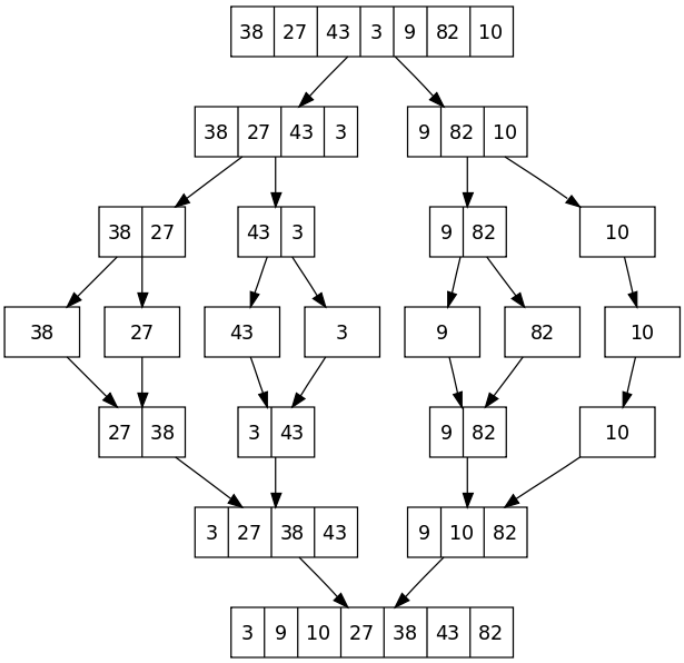
Algoritam Merge(X, l, m, r)
Ulaz niz X i granice sortiranih podnizova koje treba spojiti: l, m i $m+1$, r
Izlaz sortirani niz X

```
{
iskopiraj X[l..m] na pozicije l..m niza Y
i = l, j = m+1, k = l

while i <= m && j <= r,
    X[k++] = (Y[i] < X[j]) ? Y[i++] : X[j++]

// Ulazi se u samo jednu od naredne 2 petlje
while i <= m,
    X[k++] = Y[i++]

/* Naredni deo koda nije potreban. Zasto?
while j <= r,
    X[k++] = X[j++] */
}
```

Sortiranje spajanjem je jako stabilan algoritam i ima vremensku složenost $O(n \log n)$, a prostornu $O(n)$. Primetimo da se u algoritmu Merge kopira samo polovina niza X u niz Y.

1. Sortirati spajanjem niz brojeva 38, 27, 43, 3, 9, 82, 10.

REŠENJE:



2. Sortirati spajanjem niz brojeva 11, 4, 14, 3, 9, 10, 2, 12, 1, 15, 7, 6, 5, 13.

REŠENJE:

```
          11 4 14 3 9 10 2 12 1 15 7 6 5 13

     11 4 14 3 9 10 2          12 1 15 7 6 5 13

   11 4 14 3      9 10 2      12 1 15 7      6 5 13

  11 4    14 3    9 10    2    12 1    15 7    6 5    13

 11   4   14   3   9   10   2   12   1   15   7   6   5   13

  4 11    3 14    9 10    2    1 12    7 15    5 6    13

   3 4 11 14      2 9 10      1 7 12 15      5 6 13

     2 3 4 9 10 11 14          1 5 6 7 12 13 15

          1 2 3 4 5 6 7 9 10 11 12 13 14 15
```

Jednačine zasnovane na dekompoziciji U slučaju kada se obrada ulaza veličine n svodi na a obrada ulaza veličine $\frac{n}{b}$ i potrebno je izvršiti još $c \cdot n^k$ koraka, jednačina kojom je prestavljeno vreme izvršavanja algoritma izgleda ovako:

$$T(n) = aT(\frac{n}{b}) + c \cdot n^k$$

gde je $a, b, c, k \geq 0, b \neq 0$ i dato je $T(1)$.

Master teorema: Rešenje gornje jednačine je:

$$T(n) = \begin{cases} O(n^{\log_b a}), & a > b^k \\ O(n^k \log n), & a = b^k \\ O(n^k), & a < b^k \end{cases}$$

Dokaz: pri određivanju vremenske složenosti za dati problem koristi se *rekurzivno stablo*. Svaki čvor ovog stabla predstavlja jedan rekurzivni poziv, pri čemu koren predstavlja prvi rekurzivni poziv. Na nivou j nalazi se a^j čvorova a veličina ulaza je $\frac{n}{b^j}$. Broj koraka koji se izvrši na nivou j (bez koraka u rekurzivnim podpozivima) je $a^j \cdot c \cdot (\frac{n}{b^j})^k = c \cdot n^k \cdot (\frac{a}{b^k})^j$. U svakom sledećem nivou veličina ulaza se smanjuje tako što se deli sa b. Zato je broj nivoa u stablu $\log_b n$. Iz prethodne dve činjenice sledi da je ukupan broj koraka:

$$T(n) = \sum_{j=0}^{\log_b n} c \cdot n^k \cdot (\frac{a}{b^k})^j = c \cdot n^k \sum_{j=0}^{\log_b n} (\frac{a}{b^k})^j$$

U dokazima se koristi i jednakost: $1 + r + r^2 + \ldots + r^p = \frac{r^{p+1}-1}{r-1} = \frac{1-r^{p+1}}{1-r}$, gde je $r = \frac{a}{b^k}$. Slučajevi:

1. Koristi se da za $r > 1$ važi $1 + r + r^2 + \ldots + r^p \leq r^p \cdot \frac{r}{r-1}$. Ako uvedemo oznaku za konstantu $c_2 = \frac{r}{r-1}$, onda zamenom $p = \log_b n$ dobijamo: $\sum_{j=0}^{\log_b n} r^j \leq r^{\log_b n} \cdot \frac{r}{r-1} = (\frac{a}{b^k})^{\log_b n} \cdot c_2 = \frac{a^{\log_b n}}{n^k} \cdot c_2$ gde je pa je $T(n) \leq c \cdot n^k \cdot \frac{a^{\log_b n}}{n^k} \cdot c_2 = c_3 \cdot a^{\log_b n}$. Pošto važi da je $a^{\log_b n} = n^{\log_b a}$ (može se proveriti logaritmovanjem leve i desne strane za osnovu b), sledi da je vremenska složenost u O notaciji $O(n^{\log_b a})$. Broj $a^{\log_b n}$ je broj listova u rekurzivnom stablu, tj. u ovom slučaju se najviše posla obavlja u listovima rekurzivnog stabla.
2. Direktno sledi iz vrednosti sume. U ovom slučaju se ista količina posla obavlja na svakom nivou rekurzivnog stabla.
3. Važi da je $r < 1$ pa je prethodna suma manja od konstante $\frac{1}{1-r}$ odakle direktno sledi da je broj koraka $c_1 \cdot n^k$. U ovom slučaju najviše posla se obavlja u korenu rekurzivnog stabla.

3. Za sledeće algoritme napisati jednačinu koja predstavlja vremensku složenost tih algoritama (broj koraka je broj upoređivanja u algoritmu), i izračunati vremensku složenost tih algoritama u O notaciji:

a) binarna pretraga

b) kviksort (pod pretpostavkom da pivot uvek razdvaja niz na dva jednaka dela)

c) sortiranje spajanjem (merge sort)

REŠENJE:

a) $T(n) = T(\frac{n}{2}) + c$, koristeći master teoremu dobijamo da je složenost $O(\log n)$.

b) $T(n) = 2T(\frac{n}{2}) + c_1 \cdot n + c_2$, koristeći master teoremu dobijamo da je složenost $O(n \log n)$.

c) isto kao pod b)

4. Izračunati vremensku složenost u O notaciji algoritama čija se složenost izračunava pomoću sledećih rekurentnih relacija:

a) $T(n) = 5T(\frac{n}{2}) + c \cdot n^2$

b) $T(n) = 27T(\frac{n}{3}) + n^3$

c) $T(n) = 5T(\frac{n}{2}) + 5 \cdot n^3$

REŠENJE:

a) $O(n^{\log_2 5})$ b) $O(n^3 \log n)$ c) $O(n^3)$

5. Među n osoba superstar je osoba koja ne poznaje nikoga, a koju svi ostali poznaju. Problem: Identifikovati superstar-a (ako postoji) postavljajući pitanja oblika: Izvinite, da li poznajete onu osobu? sa ciljem da se minimizira ukupan broj pitanja. Pretpostaviti da:

1. odgovori su u formi da/ne
2. svi odgovori su tačni
3. superstar će dati odgovor

Slučaj n=2 je jednostavan. (do 2 pitanja)

Rešenje br. 1

U najgorem slučaju se može postaviti n(n-1) pitanja (ako se pitanja postavljaju bez neke mudre strategije, tj. svakome postavljamo pitanje u vezi svake druge osobe).

Rešenje br. 2

Posmatrajmo razliku problema dimenzije n-1 (sa n-1 osobom) i problema dimenzije n. Neka je induktivna hipoteza (IH) da znamo da pronađemo superstar-a među prvih n-1 osoba. Kako broj superstar-a nije veći od jedan, onda postoje tri mogućnosti: 1) superstar je među prvih n-1 osoba 2) superstar je n-ta osoba 3) ne postoji superstar.

Slučaj 1 je najjednostavniji: proveri se samo da li n-ta osoba poznaje superstar-a i da li je tačno da superstar ne poznaje ni n-tu osobu

Slučajevi 2 i 3 su teži, jer da bi se proverilo da li je n-ta osoba superstar, ukupan broj potrebnih pitanja je 2*(n-1) (bitno je da n-1 osoba poznaje superstar-a, kao i da superstar ne poznaje ostale osobe). Dakle, ako je u n-tom koraku broj potrebnih pitanja 2*(n-1), onda će ukupan broj pitanja n*(n-1), a taj slučaj je odbačen još u rešenju 1.

Rešenje br. 3

Posmatranje problema u obrnutom smeru i pokušati identifikovati osobu koja nije superstar (a broj onih koji nisu superstar je mnogo veći).

Dakle, eliminišimo neku osobu koja nije superstar iz problema i time se dimenzija problema sa n smanjuje na n-1. Nije bitno ko se eliminiše. Na primer: ako osobu A pitamo da li poznaje osobu B, onda:

- ako poznaje osobu B, eliminiše se osoba A
- u suprotnom, eliminiše se osoba B, jer nije superstar

Uočimo da je bilo dovoljno samo jedno pitanje za eliminaciju jedne osobe.

Realizacija

Algoritam se deli u dve faze. U prvoj fazi se vrši eliminacija svih osoba sem jedne-kandidata za superstar-a, a potom se u drugoj fazi testira da li kandidat jeste superstar. Podsetite se algoritma za traženje ponora grafa!!!

Radi jednostavnosti koda, smatrajmo da su u matrici susedstva na dijagonali nule. Jasno je da superstar ima sve jedinice u svojoj koloni (osim za i=j) i sve 0 u svojoj vrsti.

Algoritam Superstar(Poznaju)
ulaz: Poznaju (n*n matrica čiji elementi su:
Poznaju[i][j]=1, ako osoba i poznaje osobu j
Poznaju[i][j]=0, ako osoba i ne poznaje osobu j , gde i,j=1..n)
izlaz: Superstar (njegova vrednost, ako postoji ili 0, ako ne postoji)

```
{
  i=1; /*osoba A*/
  j=2; /*osoba B*/
  Sled=3; /*sledeci koji ce se proveravati*/

  /*prva faza*/
  while (Sled <=n+1)
  {
    if (Poznaju[i][j]==1)
        i=Sled; /*i nije superstar, eliminacija*/
    else
        j=Sled; /*j nije superstar, eliminacija*/
    Sled=Sled+1;
  }

  /*nakon izlaska iz petlje ili je i=n+1 ili je j=n+1 */
  if (i==n+1)
   kandidat=j
  else
   kandidat=i

  /*druga faza*/
  Jeste=1; /*1 ako kandidat jeste superstar*/
  k=1; /*osoba iz skupa*/
  while ( (Jeste==1) && (k <=n) )
  {
    if (Poznaju[kandidat,k]==1)
      Jeste=0; /*kandidat nije superstar*/
    if (Poznaju[k,kandidat]==0 && k!=kandidat)
      Jeste=0; /*kandidat nije superstar*/
    k=k+1;
  }

  if (Jeste==1)
    Superstar=kandidat
  else
    Superstar=0; /*nema superstar-a*/

  return superstar
}
```

Složenost

U najgorem slučaju postavlja se najviše 3*(n-1) pitanja. (U prvoj fazi se postavi n-1 pitanja radi eliminacije n-1 osobe, a u drugoj fazi se postavlja ne više od 2(n-1) pitanja).

Napomena. Postoje različita shvatanja strategije podeli-pa-vladaj pa se prethodni algoritam nekad smatra za algoritam ove strategije a nekad ne. Razlog je što se problem ne mora implementirati rekurzivno već može i iterativno (kasnije će biti reči o ovome pri radu sa repnom rekurzijom). Slično, i za binarnu pretragu postoji polemika da li jeste predstavnik strategije podeli-pa-vladaj.

6. Implementirati algoritam tipa podeli-pa-vladaj koji određuje najveći i drugi najveći elemenat niza A veličine n. Odrediti vremensku složenost algoritma.

IDEJA: Podeliti niz na dve polovine, naći dva najveća elementa u svakoj od polovina i na osnovu njih odrediti najveći i drugi najveći elemenat niza.

7. Implementirati algoritam koji odredjuje broj čvorova u binarnom stablu. Ako binarno stablo ima n čvorova odrediti vremensku i prostornu slozenost algoritma.

8. Primer *neefikasne* primene strategije podeli-pa-vladaj: Rekurzija za računanje clana Fibonačijevog niza: $F(0) = F(1) = 1; F(n) = F(n-2) + F(n-1), n > 2$. Strategija podeli-pa-vladaj daje rekurzivnu funkciju za određivanje Fibonačijevih brojeva:

```
int fib(int n){
  if ( n < 2 )
    return 1;
  else
    return fib(n-1) + fib(n-2);
}
```

Vremenska složenost algoritma: označimo sa T(n) vreme potrebno za izračunavanje F(n). Tada važi: F(n) = F(n-1) + F(n-2), početni uslov: F(1) = F(2) = 1.
Rešavanjem diferencne jednačine dobija se da je:

$F(n) = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n$
$T(n) = O((\frac{1+\sqrt{5}}{2})^n) = O(1.618^n)$

Dakle, vreme izvršavanja raste eksponencijalno s veličinom ulaza. Znate li efikasniji algoritam za računanje F(n)? Koja bi bila vremenska i prostorna složenost tog algoritma?

9. Pod pretpostavkom da je n stepen broja 2, napisati pseudokod algoritma tipa podeli-pa-vladaj koji izračunava proizvod dva cela broja A i B od po n cifara. Koristiti činjenicu da ako se ti brojevi zapišu redom u obliku: $A = A_1 10^{n/2} + A_2$, ondosno $B = B_1 10^{n/2} + B_2$ (A_1 i B_1 prestavljaju prvih $n/2$ cifara, a A_2 i B_2 prestavljaju drugih $n/2$ cifara brojeva A i B respektivno), onda se proizvod $A \cdot B$ može izračunati pomoću formule: $A_1B_1 10^n + (A_1B_2 + A_2B_1)10^{n/2} + A_2B_2$. Kao ulaz algoritmu prenose se celi brojevi A i B i broj n. Izračunati vremensku složenost napisanog algoritma pri čemu se broje samo množenja.

10. Napisati pseudokod modifikacije kviksort algoritma koji u celobrojnom nizu od n elemenata pronalazi k-ti najmanji element ($0 < k < n$). Ne treba sortirati niz do kraja. Kolika je vremenska složenost ovog algoritma?

1.2 Pohlepni algoritmi

Pohlepni algoritam u svakom koraku bira u tom trenutku naizgled najbolje rešenje (obično se bira maksimalna ili minimalna vrednost). Prednost ovih algoritama je što se lako konstruišu i što se njihova složenost obično lako izračunava. Mana je da je dokaz korektnosti ovih algoritama obično komplikovan, lako dolazi do grešaka pri njihovoj konstrukciji jer često dobra rešenja ne odgovaraju intuiciji. Jedan od dokaza korektnosti rađen je za zadatak sa kutijama kada su rađeni nizovi (pretpostavljeno je da nađeno rešenje nije optimalno i došlo se do kontradikcije). Drugi način dokazivanja korektnosti pohlepnih algoritama je pomoću indukcije. Ipak, ne postoji uniforman način za dokazivanje njihove korektnosti, već se obično pristupa trikovima i ad-hok rešenjima.

11. Primer efikasne primene pohlepnog algoritma: Trgovac treba da vrati mušteriji kusur od 62 dinara, na raspolaganju su mu novčanice od 50, 20, 10, 5 i 1 dinara. Instinktivno će većina ljudi vratiti 1x50, 1x10 i 2x1 dinar. Takav algoritam vraća tačan iznos uz najkraću moguću listu novčanica.

Algoritam: izabere se najveća novčanica koja ne prelazi ukupnu sumu (50 dinara), stavlja se na listu za vraćanje, oduzme se vraćena vrednost od ukupne kako bi se izračunao ostatak za vraćanje (12 dinara), zatim se bira sledeća novčanica koja ne prelazi ostatak koji treba vratiti (10 dinara), dodaje se na listu za vraćanje i računa novi ostatak za vraćanje (2 dinara) itd, sve dok se ne vrati tačan iznos.

Dakle strategija pohlepnog pristupa je u konkretnom primeru sa 62 dinara dovela do najefikasnijeg rešenja problema vraćanja novca.

12. Implementirati pohlepni algoritam za vraćanje kusura za dati iznos $n \geq 0$ sa monetama iz skupa 1, 5, 10, 20, 40, 50, 200, 500.

IDEJA kao u prethodnom zadatku. Neka je u algoritmu Kusur(n):
broj novčića od 1 dinara je g1
broj novčića od 5 dinara je g2
broj novčića od 10 dinara je g3
broj novčića od 20 dinara je g4
broj novčića od 40 dinara je g5
broj novčića od 50 dinara je g6
broj novčića od 200 dinara je g7
broj novčića od 500 dinara je g8

Algoritam Kusur(n)
ulaz: n (kusur koji je potrebno vratiti)
izlaz: Broj novčića koji treba vratiti za svaku monetu

```
{
g1 = 0; g2 = 0; g3 = 0; g4 = 0;
g5 = 0; g6 = 0; g7 = 0; g8 = 0;

while (n >= 500 ) { g8 = g8 + 1; n = n - 500;}

while (n >= 200 ) { g7 = g7 + 1; n = n - 200;}

while (n >= 50) { g6 = g6 + 1; n = n - 50;}

while (n >= 40) { g5 = g5 + 1; n = n - 40;}

while (n >= 20) { g4 = g4 + 1; n = n - 20;}

while {n >= 10) { g3 = g3 + 1; n = n - 10;}

while (n >= 5) { g2 = g2 + 1; n = n - 5;}

while (n >= 1) { g1 = g1 + 1; n = n - 1;}

OUTPUT: g1, g2, g3, g4, g5, g6, g7, g8 ;
}
```

Vreme izvršavanja gore navedenog algoritma $O(n)$.

13. Navesti primer skupa moneta i kusura, kada pohlepni algoritam ne nalazi rešenje, a rešenje postoji.

REŠENJE: Ako imamo na raspolaganju monete od 25, 10 i 4 dinara, pohlepni algoritam ne uspeva da nađe kombinaciju moneta za kusur od 41 dinara (dobio bi zbir 25+10+4=39 dinara). Bolji algoritam bi našao kombinaciju 25+4+4+4+4=41 dinara.

14. Potrebno je n sortiranih listi dužine w1, w2, ... , wn spojiti u jednu veliku sortiranu listu. Spajanje se obavlja u n-1 koraka tako da se u svakom koraku spoje dve odabrane liste algoritmom merge (dat kao crna kutija). Traži se optimalni plan sažimanja uz takav odabir listi u pojedinačnom koraku kako bi se postigao najmanji ukupni broj operacija.

REŠENJE: Optimalni plan spajanja se može odrediti pohlepnim pristupom i može se pokazati da ga konstruiše zapravo Hofmanov algoritam: u svakom koraku spajanja treba odabrati dve najkraće liste (drugim rečima svaki korak se izvodi tako da imamo najmanje posla u tom koraku).

15. Može se smatrati da je zadata vrednost kapacitet ranca c, da su w_i težine predmeta koje se mogu staviti u ranac, p_i vrednosti tih predmeta i x_i označava koji se deo i-tog predmeta stavlja u ranac. Problem se sastoji u određivanju predmeta koje treba staviti u ranac tako da ukupna tezina $\sum_{i=1}^{n} w_i * x_i$ ne bude veća od c, a da ukupna vrednost $\sum_{i=1}^{n} pi * xi$ bude maksimalna. Predmeti se mogu rezati.

ILI

Zadat je prirodan broj n i pozitivni realni brojevi c, w_i i $p_i (i = 1, 2, ..., n)$. Traže se realni brojevi $x_i (i = 1, 2, ..., n)$ takvi da je $\sum_{i=1}^{n} p_i * x_i$ maksimalna uz ograničenja $\sum_{i=1}^{n} w_i * x_i \leq c, 0 \leq x_i \leq 1 (i = 1, 2, ..., n)$.

TEST PRIMER:
$n = 3, w_i = (2, 3, 4), p_i = (1, 7, 8), c = 6.$

REŠENJE TEST primera: Algoritam stavlja u ranac celi drugi predmet (jer se od njega najvise profitira (uočite odnos pi / wi)), te ga popunjava s 3/4 trećeg predmeta, te je rešenje $x_i = (0, 1, 3/4)$ i maksimalna vrednost u rancu je 13.

16. Egipatski razlomci. Svaki pozitivan broj u segmentu [0,1] se može prikazati kao zbir različitih pozitivnih razlomaka sa jediničnim brojiocima. Na primer, $\frac{23}{24} = \frac{1}{2} + \frac{1}{3} + \frac{1}{8}$. Opišite efikasan pohlepni algoritam koji za dati razlomak određuje egipatsko prikazivanje razlomka. Uzmite u obzir da jedan razlomak može da ima više egipatskih prikazivanja.

17. Primer efikasne primene pohlepnog algoritma: Hofmanovo kodiranje (`www.matf.bg.ac.rs/~mirkos/Aisp/Vezbe/8/HGkod.htm`).