Konstrukcija i analiza algoritama
vežbe 10

Nina Radojičić
nina@matf.bg.ac.rs

Matematčki fakultet

decembar 2016.

Sadržaj

Sadržaj

Sadržaj

Traženje uzorka u tekstu - naivni (elementarni) algoritam

Uzorak ili obrazac (p) predstavlja nisku karaktera dužine m.

Tipično je da je tekst mnogo duži od uzorka ($n >> m$)

Traženje uzorka u tekstu sleva udesno

Traženje uzorka u tekstu - naivni (elementarni) algoritam

```
//Funkcija proverava da li string s sadrzi string p.
//Vraca poziciju na kojoj p pocinje, odnosno -1 ukoliko ga nema
int Sravni_niske(char s[], char p[])
{
  int i, j;
  /* Proveravamo da li p pocinje na svakoj poziciji i */
  for (i = 0; s[i]; i++)
      /* Poredimo p sa s pocevsi od znaka p[0] sve dok
         ne naidjemo na razliku */
      for (j = 0; s[i+j] == p[j]; j++)
        /* Nismo naisli na razliku, a ispitali smo
            sve karaktere niske p */
        if (p[j+1]=='\0')
          return i;

  /* Uzorak nije nadjen. */
  return -1;

}
```

KMP algoritam (Knuth-Morris-Pratt)

IDEJA:

Analizirati uzorak!!!

Broj poređenja u ovom algoritmu je $O(n+m)$ što je značajno efikasnije od elementarnog algoritma.

KMP algoritam se sastoji iz dva dela

1. popunjavanje niza h (tabele pomeranja uzorka) koja zavisi samo od uzorka,
2. sam algoritam traženja uzorka u tekstu.

KMP algoritam (Knuth-Morris-Pratt)

```
Algoritam Pomaci
Ulaz: P (uzorak, string duzine m).
Izlaz: h (niz duzine m).
begin
  i := 1;
  j := 0;
  h[1] := 0; g[1] := 0;

  while i < m do
    while j > 0 cand P[i] != P[j]
      j := h[j];

    i := i+1; j := j+1;
    g[i] := j;

    if P[i] == P[j]
      h[i] := h[j];
    else
      h[i] := j;
end
```

KMP algoritam (Knuth-Morris-Pratt)

```
Algoritam KMP
Ulaz: S (tekst, string duzine n) i P (uzorak duzine m).
Izlaz: Start (indeks pocetka prvog podstringa S jednakog P,
  ako takav postoji, odnosno 0 u protivnom
begin
  Start := 0;
  i := 1;                          // pokazivac na uzorak
  j := 1;                          // pokazivac na tekst

  while i <= m and j <= n do       // dok ima nade da se nadje podstring
                                                  // jednak uzorku
    while i > 0 cand P[i] != S[j] do
      i := h[i];                   // pomeranje uzorka udesno za i - h[i]
    i := i + 1; j := j + 1;        // napredovanje u tekstu i u uzorku

  if i > m then
    Start := j - i + 1;            // pronadjen je podstring jednak uzorku
end
```

KMP - zadaci

1

Traži se prva pojava uzorka p=abcabcacab u tekstu s=babcbabcabcaabca.

(a) Izračunati brojeve pomeranja uzorka - tabelu koja se koristi za algoritam KMP.

(b) Izračunati broj pokušaja i broj poređenja karaktera?

Rešenje:

Uzorak p sa indeksima:

```
1 2 3 4 5 6 7 8 9 10
a b c a b c a c a b
```

Brojači u prvom algoritmu i nizovi g i h koji se formiraju:

```
i: 1 2   3   4 5 6 7 8     9 10
j: 0 1 0 1 0 1 2 3 4 5 1 0 1 2
```

```
i:    1 2 3 4 5 6 7 8 9 10
h[i]: 0 1 1 0 1 1 0 5 0 1
g[i]: 0 1 1 1 2 3 4 5 1 2
```

Rešenje:

Tekst s sa indeksima zajedno sa pokušajima:

```
indeks:     1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
s:          b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
pokusaj 1: a
```

pokusaj 1 pa pomeraj za 1-h[1] = 1 poziciju;

Rešenje:

Tekst s sa indeksima zajedno sa pokušajima:

```
indeks:     1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
s:          b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
pokusaj 1: a
pokusaj 2:    a  b  c  a
```

```
pokusaj 2 pa pomeraj za 4-h[4] = 4 pozicije;
```

Rešenje:

Tekst s sa indeksima zajedno sa pokušajima:

```
indeks:     1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
s:          b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
pokusaj 1: a
pokusaj 2:    a  b  c  a
pokusaj 3:                  a  b  c  a  b  c  a  c
```

pokusaj 3 pa pomeraj za 8-h[8] = 3 pozicije;

Rešenje:

Tekst s sa indeksima zajedno sa pokušajima:

```
indeks:     1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
s:          b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
pokusaj 1: a
pokusaj 2:    a  b  c  a
pokusaj 3:                  a  b  c  a  b  c  a  c
pokusaj 4:                           a  b  c  a  b
```

pokusaj 4 pa pomeraj za 5-h[5] = 4 pozicije;

Rešenje:

Tekst s sa indeksima zajedno sa pokušajima:

```
indeks:      1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16
s:           b  a  b  c  b  a  b  c  a  b  c  a  a  b  c  a
pokusaj 1: a
pokusaj 2:    a  b  c  a
pokusaj 3:                  a  b  c  a  b  c  a  c
pokusaj 4:                           a  b  c  a  b
pokusaj 5:                                       a  b  c  a
```

Brojači u algoritmu KMP:

```
i: 1 0 1 2 3 4 0 1 2 3 4 5  6  7  8  5 1 2  3  4
j: 1   2 3 4 5   6 7 8 9 10 11 12 13    14 15 16
```

Rešenje:

Pokušaja: 5 (za 1 više od broja pomeraja).

Poređenja karaktera: 18 (jedan način da se ovo vidi je broj vrednosti koje uzima brojač i ne računajući nule, drugi je iz gornjeg prikaza pokušaja).

Algoritamske strategije - 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 složenost se dobija rešavanjem diferencnih jednačina.

Primeri podeli-pa-vladaj algoritama su FFT, kviksort, sortiranje spajanjem, Euklidov algoritam, binarna pretraga itd.

Kviksort

Složenost ovog algoritma je u najgorem sluăju $O(n^2)$ (kada se za pivot bira uvek najmanji element što je slučaj i u algoritmu koji je ovde naveden).

U praksi se koristi algoritam koji za pivot biva proizvoljan (random) element niza i tada je *prosečna* složenost algoritma $O(n \log n)$.

Dodatan memorijski prostor koji se koristi je veličine $O(1)$.

Napomena: Kada se ispituje složenost algoritma sortiranja izračunava se broj poređenja tokom njegovog izvršavanja.

Algoritam Razdvajanje (X, *Levi*, *Desni*)

Ulaz X (niz), *Levi* (leva granica niza), *Desni* (desna granica niza)
Izlaz indeks S takav da je $X[i] \leq X[S]$ za sve $i \leq S$ i $X[j] > X[S]$ za sve $j > S$

```
{
    pivot = X[Levi];
    L = Levi;     D = Desni;
    while L < D
    {
        while X[L] ≤ pivot && L ≤ D
            L = L + 1;
        if L > Desni
            L = Desni;
        while X[D] > pivot && D ≥ Levi
            D = D − 1;
        if L < D
            zameni X[L], X[D];
    }
    S = D;
    zameni X[Levi], X[S];
    return S;
}
```

Algoritam Q_Sort (X, $Levi$, $Desni$)

Ulaz X, leva i desna granica dela niza koji se sortira
Izlaz sortirani deo niza X

```
{
    if Levi < Desni
    {
        S = Razdvajanje(X, Levi, Desni);
        Q_sort(X, Levi, S − 1);
        Q_sort(X, S + 1, Desni);
    }
}
```

Algoritam Kviksort (X, n)

Ulaz X (niz od n brojeva)
Izlaz sortirani niz X
{
    $Q\_sort(X, 1, n)$;
}

Primer

Algoritmom Kviksort prikazati postupak sortiranja brojeva 8, 11, 5, 14, 3, 9, 10, 2, 12, 1, 15, 7, 6, 4, 13.

8	11	5	14	3	9	10	2	12	1	15	7	6	4	13
8	4	5	14	3	9	10	2	12	1	15	7	6	11	13
8	4	5	6	3	9	10	2	12	1	15	7	14	11	13
8	4	5	6	3	7	10	2	12	1	15	9	14	11	13
8	4	5	6	3	7	1	2	12	10	15	9	14	11	13
2	4	5	6	3	7	1	**8**	12	10	15	9	14	11	13
2	4	5	6	3	7	1								
2	1	5	6	3	7	4								
1	**2**	5	6	3	7	4								
		5	6	3	7	4								
		5	4	3	7	6								
		3	4	**5**	7	6								
					7	6								
					6	**7**								

12	10	15	9	14	11	13
12	10	11	9	14	15	13
9	10	11	**12**	14	15	13
9	10	11				

Sortiranje spajanjem (Merge sort)

Sortiranje spajanjem 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)
}
```

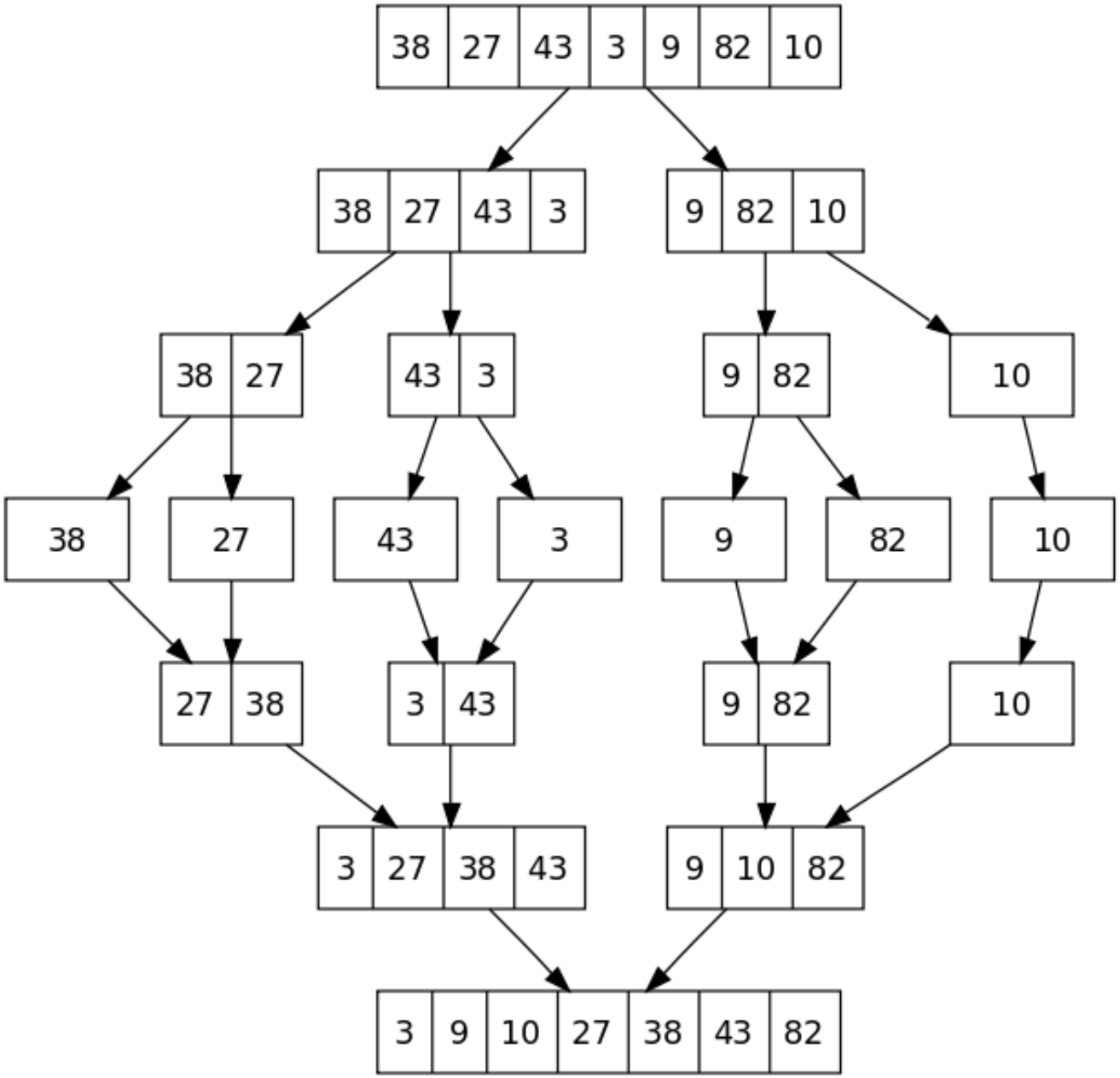
Algoritam Merge(X, l, m, r)

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++] */
}
```

Primer

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


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

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 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:

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

Primer

Za sledeće algoritme napisati jednačinu koja predstavlja složenost tih algoritama (broj koraka je broj upoređivanja u algoritmu), i izračunati 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)

Primer

Za sledeće algoritme napisati jednačinu koja predstavlja složenost tih algoritama (broj koraka je broj upoređivanja u algoritmu), i izračunati 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 $O(\log n)$.

Primer

Za sledeće algoritme napisati jednačinu koja predstavlja složenost tih algoritama (broj koraka je broj upoređivanja u algoritmu), i izračunati 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) $T(n) = 2T(\frac{n}{2}) + c \cdot n$, koristeći master teoremu dobijamo da je složenost $O(n \log n)$.

Primer

Za sledeće algoritme napisati jednačinu koja predstavlja složenost tih algoritama (broj koraka je broj upoređivanja u algoritmu), i izračunati 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) $T(n) = 2T(\frac{n}{2}) + c \cdot n$, koristeći master teoremu dobijamo da je složenost $O(n \log n)$.
c) isto kao pod b)

Algoritamske strategije - bektreking

Bektreking algoritmi tragaju za svim rešenjima nekog problema. Najpoznatiji problem ove vrste je problem osam kraljica (dama).

Ovi algoritmi su posebno korisni kada se lako mogu odbaciti mnoge mogućnosti za rešenja.

Neki probliemi koji se rešavaju na ovaj način su rešavanje ukrštenica, slagalica, Sudokua.

Ovi algoritmi nalaze se u osnovi logičkih programskih jezika, npr. Prologa.

Primer

Napisati funkciju u C-u koja na ulazu prima dva celobrojna niza A i B, oba dimenzije n, i na izlaz štampa sve moguće nizove dimenzije n koji na i-toj poziciji imaju ili broj A[i] ili broj B[i], i čija se svaka dva uzastopna člana razlikuju najviše za 5.

Primer

Napisati funkciju u C-u koja na ulazu prima dva celobrojna niza A i B, oba dimenzije n, i na izlaz štampa sve moguće nizove dimenzije n koji na i-toj poziciji imaju ili broj A[i] ili broj B[i], i čija se svaka dva uzastopna člana razlikuju najviše za 5.

Npr. za nizove

```
A: 3  1  7
B: 6  0  4
```

na ekran treba da se odštampa

```
3 1 4
3 0 4
6 1 4
```

REŠENJE 1 - gruba sila, LOŠE (neefikasno) rešenje:

```
void mesavina (int a[], int b[], int c[], int i, int n){
  int j;
  if (i == n){
    for (j=0; j<n-1; j++)
      if (abs(c[j] - c[j+1]) > 5)
        break;
    if (j == n-1){
      for (j=0; j<n; j++)
        printf("%d ",c[j]);
      printf("\n");
    }
  }
  else {
    c[i] = a[i];
    mesavina (a, b, c, i+1, n);
    c[i] = b[i];
    mesavina (a, b, c, i+1, n);
  }
}
```

REŠENJE 2 - korišćenje odsecanja, DOBRO (efikasnije) rešenje:

```
void mesavina (int a[], int b[], int c[], int i, int n){
  int j;
  if (i == n){
    for (j=0; j<n; j++)
      printf("%d ",c[j]);
    printf("\n");
  }
  else {
    if (i==0 || abs (a[i] - c[i-1]) <= 5){ // odsecanje
      c[i] = a[i];
      mesavina (a, b, c, i+1, n);
    }
    if (i==0 || abs (b[i] - c[i-1]) <= 5){ // odsecanje
      c[i] = b[i];
      mesavina (a, b, c, i+1, n);
    }
  }
}
```

Primer

Napisati C program koji za datu sumu S i dati niz v koji sadrži vrednosti novčanica pronalazi sva rešenja za rasitnjavanja sume S. Pretpostaviti da svake novčanice ima proizvoljno mnogo.