RAČUNARSTVO
Sveska 1, Broj 2, 1992

UDK 519.688

ALGORITMI ZA SRAVNJIVANJE NISKOVNIH OBRAZACA[1]

CVETANA KRSTEV

1 Uvod

Podaci koje treba računarski obrađivati se često ne mogu logički predstaviti pomoću nezavisnih zapisa koji se sastoje od malih prepoznatljivih delova. Ovakva vrsta podataka se najčešće zapisuje u obliku, obično veoma dugačke, niske.

Ovakva vrsta podataka je osnovna, na primer, kod takozvanih sistema za „obradu reči" koji pružaju obilje mogućnosti za manipulaciju tekstom na nekom prirodnom jeziku. U tom slučaju govorimo o tekstualnim niskama. Ovakve tekstualne niske nastaju nad azbukom koja sadrži slova, brojeve, interpunkcijske i specijalne znake. Tekstualne niske kojima manipulišu sistemi za obradu reči mogu biti veoma dugačke: tekst neke knjige može da sadrži i preko milion karaktera.

Poseban primer niski su binarne niske koje se tvore nad azbukom od samo dva simbola $\{0, 1\}$. Ovakve binarne niske se koriste u sistemima za računarsku grafiku koji sliku predstavljaju u obliku binarne niske: 0 predstavlja neosvetljen a 1 osvetljen piksel na ekranu ili nekom drugom izlaznom uređaju. Premda suštinskih razlika između tekstualnih i binarnih niski nema, veličina azbuke nad kojom se one tvore utiče na izbor najefikasnijeg algoritma za njihovo sravnjivanje.

Fundamentalna operacija nad niskama je sravnjivanje niskovnih obrazaca. Tako, na primer, svi sistemi za obradu reči obezbeđuju funkciju „pronađi" koja u tekstu pronalazi zadati niskovni obrazac. U bibliografskim sistemima je sravnjivanje niskovnih obrazaca uključeno u pretraživanje teksta pomoću ključnih reči. Sravnjivanje niskovnih obrazaca je esencijalno i u raznim leksičkim i lingvističkim analizama.

Sam niskovni obrazac može biti različitog oblika [1]: jedna ključna reč, konačan skup ključnih reči, regularni izraz ili može biti opisan na neki drugi proceduralni ili

[1]Rad finansiran od Fonda za nauku Republike Srbije preko Matematičkog instituta, projekat 0401A.

neproceduralni način (kakav omogućuje, na primer, SNOBOL-ski program [5]). U ovom radu biće opisani fundamentalni algoritmi za sravnjivanje jedne niske (ključne reči) sa tekstom.

Problem se može definisati na sledeći način: za dati tekst x dužine N i za dati obrazac (ključnu reč) p dužine M treba pronaći prvo pojavljivanje obrasca unutar teksta. Pretpostavljamo, takođe, da je $M > 0$, tj. da obrazac nije prazna niska. Većina algoritma za rešavanje ovog problema prolazi sekvencijalno kroz tekst pa se lako može uopštiti da pronađe sva pojavljivanja obrasca unutar teksta: kada se obrazac sravni, algoritam ponovo počinje sa skaniranjem teksta počev od pozicije u tekstu koja neposredno sledi početak poslednjeg sravnjenog obrasca.

2 Algoritam „grube sile“

Očigledan metod za sravnjivanje zadatog obrasca sa tekstom je isprobavanje redom svih pozicija u tekstu i utvrđivanje da li se obrazac na tim pozicijama podudara sa tekstom [8]. Funkcija *GrubaMetoda* na takav način pronalazi prvo pojavljivanje obrasca $p[1..M]$ u tekstu $x[1..N]$.

```
function GrubaMetoda: integer;
  var i, j: integer;
  begin
  i := 1; j := 1;
  repeat
    if x[i] = p[j]
          then begin i := i + 1; j := j + 1 end
          else begin i := i - j + 2; j := 1 end;
  until (j > M) or (i > N);
  if j > M then GrubaMetoda:= i - M
          else GrubaMetoda:= i
  end;
```

i i j su pokazivači tekućeg karaktera teksta, odnosno obrasca. Sve dok su tekući karakter teksta i tekući karakter obrasca jednaki, ovi pokazivači se uvećavaju. Ako se stigne do kraja obrasca ($j > M$), onda je obrazac pronađen u tekstu a *GrubaMetoda* dobija vrednost pozicije početka obrasca u tekstu. Ako se tekući karakter teksta i tekući karakter obrasca razlikuju, onda se obrazac na poziciji $i - j + 1$ ne može sravniti sa tekstom pa se obrazac pomera za jednu poziciju udesno u odnosu na tekst ($i = i - j + 2$) a j se postavlja na početak obrasca ($j = 1$). Ako se stigne do kraja teksta ($i > N$), obrazac u tekstu ne postoji a *GrubaMetoda* dobija vrednost $N + 1$.

U primenama ovog algoritma na tekst u prirodnom (ili programskom) jeziku, na primer u uređivačima teksta, unutrašnja petlja u kojoj se uvećavaju brojači i i j se retko izvršava. Pogledajmo sledeći primer: treba pronaći nisku PRAKSI u tekstu

JEDAN PRIMER KOJI POTVRDJUJE LINEARNOST METODE U PRAKSI

Pokazivač j će se uvećati samo tri puta: dva puta za PR iz PRIMER i jednom za P iz POTVRDJUJE. Međutim, razmotrimo slučaj kada je $p = a^{M-1}b$ a $x = a^N$. U tom slučaju se pokazivač j pomera $(m-1)n$ puta da bi se na kraju utvrdilo da se obrazac ne može sravniti sa tekstom.

Šema 1. prikazuje kako algoritam radi u slučaju kada je obrazac $p = abcabcacab$ a $x = babcbabcabcaabcabcabcacabc$.

		neuspeh	nastavak
	babcbabcabcaabcabcabcacabc		
1.	*abc*	$i = 5,\ j = 4$	$i = 3,\ j = 1$
2.	*abcabca*	$i = 13,\ j = 8$	$i = 7,\ j = 1$
3.	*abca*	$i = 13,\ j = 5$	$i = 10,\ j = 1$
4.	*a*	$i = 13,\ j = 2$	$i = 13,\ j = 1$
5.	*abcabca*	$i = 20,\ j = 8$	$i = 14,\ j = 1$
6.	*abcabcacab*		

Šema 1. Algoritam „grube sile“

Svaki red u ovoj šemi odgovara ulasku u unutrašnju petlju algoritma a svaki karakter u redu uvećanju pokazivača j. Ako pogledamo malo bliže šta se događa između 2. i 3. reda, videćemo da, pošto propadne pokušaj sravnjivanja obrasca na poziciji 6 u tekstu, isprobavaju se i pozicije 7 i 8 dok se ne dođe do delimičnog slaganja iz 3. reda na poziciji 9. To su, očigledno, nepotrebni pokušaji, jer nam delimično sravnjen obrazac iz 2. reda daje dovoljno informacija o tekstu na ovim pozicijama pa njih ne bi trebalo ni isprobavati (na poziciji 7 je b a na poziciji 8 je c). Pokušaj iz 4. reda je takođe nepotreban i mogao bi se izbeći. Podniska *abc* se ponavlja dva puta na početku obrasca. Ako je u 3. redu propao pokušaj sravnjivanja druge podniske *abc* obrasca, nema svrhe na istom mestu pokušavati da se sravni prva podniska *abc* obrasca. Cilj je, očigledno, da se izbegnu ovi nepotrebni pokušaji.

3 Knut-Moris-Pratov algoritam

Osnovna ideja Knut-Moris-Pratovog (KMP) algoritma sastoji se u sledećem [8]: kada se naiđe na neslaganje tekućeg karaktera teksta i tekućeg karaktera obrasca, „pogrešni pokušaj“ se sastoji od karaktera koje unapred poznajemo, jer čine prefiks obrasca. Tu informaciju bi trebalo iskoristiti i ne vraćati pokazivač i preko već poznatih karaktera.

Razmotrimo ponovo primer iz prethodnog odeljka u kome se obrazac $p = abcabcacab$ sravnjivao sa tekstom $x = babcbabcabcaabcabcabcacabc$. Šema 2. prikazuje rad algoritma, koji poznavajući obrazac koji sravnjuje, posle svakog neslaganja preskače preko onih pozicija u tekstu koje ne mogu dovesti do slaganja.

Iz opisa rada ovog algoritma se vidi da algoritam izbegava sve pokušaje za koje je, iz delimično sravnjenog obrasca, unapred jasno da moraju biti neuspešni. Vidi

		neuspeh	nastavak
	babcbabcabcaabcabcabcacabc		
1.	*abc*	$i = 5,\ j = 4$	$i = 5,\ j = 0$
2.	*abcabca*	$i = 13,\ j = 8$	$i = 13,\ j = 5$
3.	*abca*	$i = 13,\ j = 5$	$i = 13,\ j = 1$
4.	*abcabca*	$i = 20,\ j = 8$	$i = 20,\ j = 5$
5.	*abcabcacab*		

Šema 2. Knut-Moris-Pratov algoritam

se, takođe, da se pokazivač teksta i nikada ne vraća unazad. Funkcija *KPMmetod* koristi ovakav algoritam da pronađe prvo pojavljivanje obrasca u tekstu.

```
function KMPmetod : integer;
  var i, j: integer;
  begin
  i := 1; j := 1;
  repeat
   if (j = 0) or (x[i] = p[j])
     then begin i := i + 1; j := j + 1 end
     else begin j := nast[j] end;
  until (j > M) or (i > N);
  if j > M then KMPmetod:= i - M
           else KMPmetod:= i;
  end;
```

Ovaj funkcijski potprogram formalizuje gornje ideje: kada se naiđe na neslaganje teksta sa j-tim karakterom obrasca, $nast[j]$ određuje sa kojim karakterom obrasca treba nastaviti sravnjivanje, ne pomerajući pokazivač teksta. Postavljanje pokazivača j na 0 znači da početak obrasca treba pomeriti iza tekućeg karakatera teksta a pokazivač teksta i uvećati. Dakle, u svakom prolasku kroz **repeat** petlju pomera se ili pokazivač teksta ili obrazac.

Da bi se dokazalo da je ovaj program korektan može se koristiti sledeća invarijanta: Neka je $k = i - j$, tj. neka je k pozicija u tekstu koja prethodi prvom karakteru tekuće pozicije obrasca [7]. Tada je

$$\forall l\colon\ 1 \le l < j\ x[k+l] = p[l] \wedge$$
$$\forall t\colon\ 0 \le t < k\ \exists s : 1 \le s \le M,\ x[t+s] \neq p[s].$$

Gornji program će, naravno, biti korektan samo ako se funkcija *nast* izračuna tako da gornja invarijanta važi kada se izvršava operacija $j := nast[j]$. Kada program izvršava operaciju $j := nast[j]$ znamo da je $j > 0$ i da je poslednjih j karaktera teksta, uključujući $x[i]$

$$p[1] \ldots p[j-1]y$$

gde je $y \neq p[j]$. Želimo da odredimo najmanji pomak obrasca za koji bi se ovi karakteri teksta mogli sravniti sa obrascem. Drugim rečima, $nast[j]$ treba da bude

najveće l manje od j takvo da je poslednjih l karaktera teksta

$$p[1]\ldots p[l-1]y$$

i $p[l] \neq p[j]$. Ako takvo l ne postoji, $nast[j]$ se postavlja na 0. Lako se dokazuje da, ako je $nast[j]$ definisano na ovaj način, gornja invarijanta programa važi. Prvi deo invarijante sledi iz definicije $nast[j]$. Treba još dokazati

$$\forall t: i-j \leq t < i-nast[j]$$
$$x[t+1]\ldots x[i-1] \neq p[1]\ldots p[i-t-1]$$

Ako bi postojalo t iz $[i-j, i-nast[j])$ takvo da je $x[t+1]\ldots x[i-1] = p[1]\ldots p[i-t-1]$, onda bi bilo $nast[j] < i-t$, što je u suprotnosti sa definicijom $nast[j]$.

Ostaje još da se izračuna funkcija $nast[j]$. Problem bi se lakše rešio kada se u definiciji $nast[j]$ ne bi zahtevalo da bude $p[l] \neq p[j]$. Pogledajmo zato prvo kako se rešava lakši zadatak. Neka je $f[j]$ najveće l manje od j takvo da je $p[1]\ldots p[l-1] = p[j-l+1]\ldots p[j-1]$. Kako je uslov uvek ispunjen za $l = 1$, imamo da je za $j > 1$ uvek $f[j] \geq 1$. Neka je $f[1] = 0$.

Da bi se izračunala funkcija $f[j]$, možemo zamisliti da kopiju prvih $j-1$ karaktera obrasca pomeramo iznad same sebe s leva nadesno, počev od prvog karaktera kopije postavljene iznad drugog karaktera obrasca. Zaustavljamo se kada se podudare svi karakteri koji se preklapaju, ili kada ustanovimo da takvog podudaranja nema. Broj karaktera koji se preklapaju uvećan za jedan daje vrednost $f[j]$. Na primeru obrasca $p = abcabcacab$ dobijaju se sledeće vrednosti:

j	=	1	2	3	4	5	6	7	8	9	10
$p[j]$	=	a	b	c	a	b	c	a	c	a	b
$f[j]$	=	0	1	1	1	2	3	4	5	1	2

Lako se uočava da ako je $p[j] = p[f[j]]$, onda je $f[j+1] = f[j]+1$. Zanimljivo je da se, ako je $p[j] \neq p[f[j]]$, za računanje $f[j+1]$ može primeniti potpuno isti algoritam kao i za sravnjivanje obrasca i teksta. Uočava se sličnost problema računanja $f[j]$ i invarijante algoritma sravnjivanja: algoritam računa najveće j manje od k za koje je $p[1]\ldots p[j-1] = x[k-j+1]\ldots x[k-j]$. S toga je i program za izračunavanje niza $f[j]$ u osnovi isti kao i algoritam sravnjivanja, samo što se obrazac p sravnjuje sa samim sobom.

```
procedure InitNast;
  var i, j: integer;
  begin
  i := 1; j := 0; f[1] := 0;
  repeat
    if (j = 0) or (p[i] = p[j])
      then begin
             i := i + 1; j := j + 1; f[i] := j
```

```
        end
    else begin j := nast[j] end;
until i > M;
end;
```

Procedura *InitNast* pretpostavlja da je funkcija *nast* već poznata. Ako se uporede definicije funkcija f i *nast* vidi se da je za $j > 1$

$$nast[j] = \begin{cases} f[j], & \text{ako je } p[j] \neq p[f[j]]; \\ nast[f[j]], & \text{ako je } p[j] = p[f[j]]. \end{cases}$$

Da bi se izračunala funkcija *nast*, proceduru *InitNast* treba preraditi, izostavljajući sasvim računanje funkcije f tako što iskaz $f[i] := j$ treba zameniti sa

```
if p[j] <> p[i] then nast[i] := j
                else nast[i] := nast[j];
```

3.1 Efikasnost

Efikasnost KMP algoritma, kao i ostalih algoritama za sravnjivanje niski utvrđuje se, s jedne strane, u odnosu na broj karaktera teksta koji se ispituju i, s druge strane, u odnosu na broj izvršenih naredbi programa (elementarnih koraka algoritma). KMP algoritam ispituje svaki karakter teksta tačno jednom dok ne sravni obrazac, odnosno utvrdi da obrazac u tekstu ne postoji. Procedura *KMPmetod* ima osobinu da se operacija $j := nast[j]$ ne izvršava češće od operacije $i := i+1$; šta više, ona se obično izvršava ređe jer se, u opštem slučaju, obrazac pomera udesno ređe od pokazivača teksta. Slično važi i za proceduru *InitNast*. Operacija $j := nast[j]$ uvek pomera gornju kopiju obrasca udesno pa se može izvršiti najviše M puta. Iz ovoga sledi da KMP algoritam pronalazi obrazac dužine M u tekstu dužine N u $O(N + M)$ jedinica vremena.

Osim toga, Knut, Moris i Prat su dokazali [7] da je broj izvršavanja operacije $j := nast[j]$, pre nego što se pokazivač i pomeri, ograničen aproksimacijom funkcije $log_\phi M$, gde je ϕ zlatan presek $((1 + \sqrt{5})/2 \approx 1.618)$. Ova funkcija je i najbolja gornja granica. Da bi ovo dokazali koristili su Fibonačijeve niske koje se definišu na sledeći način:

$$\begin{aligned} \phi_1 &= b, \\ \phi_2 &= a, \\ \phi_n &= \phi_{n-1}\phi_{n-2}, \quad \text{za } n \geq 3 \end{aligned}$$

a koje su izgledale kao patološki obrazac za njihov algoritam. Oni su pokazali da se za Fibonačijeve niske, skanirajući jedan karakter teksta, operacija $j := nast[j]$ najviše izvršava približno $log_\phi M$ puta. Pokazali su, takođe, da su Fibonačijeve niske najgori slučaj, što znači da je $log_\phi M$ i gornja granica. Iz ovoga sledi zaključak

da ako se tekst učitava iz spoljašnje teke, između učitavanja dva susedna karaktera protekne samo $O(logM)$ jedinica vremena.

Sami autori Knut-Moris-Pratovog algoritma su uočili da njihov algoritam neće biti značajno efikasniji od algoritma „grube sile" u većini realnih aplikacija. U takvim aplikacijama se, uglavnom, ne sravnjuje samoponavljajući obrazac sa veoma samoponavljajućim tekstom. Međutim, sa praktične strane, algoritam ima veoma korisnu osobinu. Algoritam sekvencijalno prolazi kroz ulaz i nikad se ne vraća unazad, što ga čini veoma pogodnim za korišćenje na tekstovima koji se učitavaju iz spoljašnjih teka. U takvim slučajevima, u algoritme koji ne isključuju vraćanje unazad mora da bude ugrađena složena tamponaža. Osim toga, postoje načini za značajno ubrzanje ovog algoritma kao i mogućnost njegovog proširenja na složenije obrasce.

3.2 Poboljšanja i proširenja

U uređivačima teksta obrasci su obično kratki, tako da je najefikasnije da se obrazac direktno prevede u mašinski kod koji implicitno sadrži funkciju *nast*. Na primer, sledeći program je ekvivalentan programu *KMPmetod* za obrazac $p = abcabcacab$, ali je mnogo efikasniji:

```
      i := 0;
0 :   i := i + 1;
1 :   if x[i] <>'a' then goto 0;   i := i + 1;
2 :   if x[i] <>'b' then goto 1;   i := i + 1;
3 :   if x[i] <>'c' then goto 1;   i := i + 1;
4 :   if x[i] <>'a' then goto 0;   i := i + 1;
5 :   if x[i] <>'b' then goto 1;   i := i + 1;
6 :   if x[i] <>'c' then goto 1:   i := i + 1;
7 :   if x[i] <>'a' then goto 0;   i := i + 1;
8 :   if x[i] <>'c' then goto 5;   i := i + 1;
9 :   if x[i] <>'a' then goto 0;   i := i + 1;
10 :  if x[i] <>'b' then goto 1;   i := i + 1;
      Metod := i - 8;
```

Obeležja u goto iskazima podudaraju se sa funkcijom *nast*. Šta više, procedura *InitNast* koja računa funkciju *nast* može se modifikovati tako proizvede ovakav program. Da bi se izbegla provera $i > N$ svaki put kada se pokazivač i pomeri, sam obrazac može se dopisati na kraj teksta, u $x[N+1..N+M]$. (Ovo poboljšanje može se primeniti i u standardnoj proceduri *KMPmetod*).

Ovakav program može se opisati pomoću veoma jednostavnog modela, konačnog automata. Ovaj konačni automat se sastoji od stanja, koja su karakteri obrasca, i prelaza iz stanja u stanje. U svakom stanju moguća su dva prelaza: prelaz u slučaju uspeha sravnjivanja (kada su tekući karakteri obrasca i teksta jednaki) i prelaz u slučaju neuspeha sravnjivanja (kada oni nisu jednaki). U slučaju kada su ovi karakteri jednaki, pokazivač teksta se pomera.

Ovakvo tumačenje KMP algoritma omogućava njegovo uopštavanje na slučaj kada je obrazac konačan skup više ključnih reči, tj. kada se više ključnih reči paralelno sravnjuje: sravnjivanje je uspešno kada se u tekstu pronađe prvo pojavljivanje jedne od ključnih reči obrasca. Jedno moguće rešenje je da se za svaku ključnu reč formira funkcija *nast* i da se za njega održava pokazivač j. Ovakvo rešenje, međutim, za sravnjivanje obrasca od k ključnih reči čija je ukupna dužina M zahteva $O(kN + M)$ jedinica vremena. Korišćenjem istih ideja koje su u osnovi KMP algoritma, može se konstruisati algoritam koji sravnjuje obrazac od k ključnih reči u $O(N + M)$ jedinica vremena. U $O(M)$ jedinica vremena se prvo konstruiše automat za sravnjivanje obrasca koji se sastoji od sledećih komponenti [1]:

1. S, konačan skup stanja,
2. Σ, konačna ulazna azbuka,
3. $napr: S \times \Sigma \to S \cup \{otkaz\}$, funkcija prelaza napred,
4. $nast: S - \{s_0\} \to S$, funkcija prelaza nazad,
5. s_0, početno stanje,
6. F, skup završnih stanja.

Da bi se konstruisala funkcija prelaza napred, ključne reči se kombinuju u trai (engl. *trie*), čiji čvorovi predstavljaju prefikse jedne ili više ključnih reči, a grane specifikuju funkciju *napr* kao funkciju tekućeg stanja i tekućeg karaktera teksta. Funkcija prelaza nazad se konstruiše tako da zadovolji sledeći uslov: Ako stanja s i t predstavljaju prefikse u i v nekih ključnih reči, tada je $f[s] = t$ ako i samo ako je v najduži sufiks od u koji je ujedno i prefiks neke ključne reči. Da bi se izbegla nepotrebna vraćanja unazad, uvođenjem dodatnog uslova,

$$S_{f[s]} = \{s \in S \mid g(f[s], a) \neq \text{otkaz}\}$$
$$S_s = \{s \in S \mid g(s, a) \neq \text{otkaz}\}$$

$$nast[s] = \begin{cases} nast[s] = nast[f[s]], & S_{f[s]} \subset S_s \\ nast[s] = f[s], & \text{inače} \end{cases}$$

iz funkcije f može se konstruisati funkcija *nast*. Analogija sa funkcijama f i *nast* iz KMP algoritma je očigledna. Važno je da se, kao i u slučaju KMP algoritma, funkcije f i *nast* mogu konstruisati u $O(M)$ jedinica vremena. U slučaju obrasca $p = \{abcab, ababc, bcac, bbc\}$, automat bi se mogao predstaviti Tabelom 1.

Ako se automat, odnosno funkcije g i *nast* definišu iz obrasca na ovaj način, program za prepoznavanje obrasca je vrlo jednostavan. Celobrojni niz g ima vrednost -1 u slučaju otkaza, logički niz *kraj* ima vrednost **true** ako je stanje završno a vrednost niza *duž* je dubina stanja. Ovi nizovi se, kao i niz *nast*, formiraju u fazi konstruisanja automata.

		g				
čvor	prefiks	a	b	c	f	$nast$
0		1	9	0		
1	a	ot	2	ot	0	0
2	ab	3	ot	6	9	9
3	aba	ot	4	ot	1	0
4	$abab$	ot	ot	5	2	2
5	$ababc$	ot	ot	ot	6	6
6	abc	7	ot	ot	12	0
7	$abca$	ot	8	ot	13	13
8	$abcab$	ot	ot	ot	2	2
9	b	ot	10	12	0	0
10	bb	ot	ot	11	9	9
11	bbc	ot	ot	ot	12	12
12	bc	13	ot	ot	0	0
13	bca	ot	ot	14	1	1
14	$bcac$	ot	ot	ot	0	0

Tabela 1. Rad automata sa obrascem p.

```
function Automat: integer;
  var i, stanje: integer;
  begin
  i := 1; stanje := 0;
  repeat
    if g[stanje, x[i]] <> -1
      then begin
             i := i + 1; stanje := g[stanje, x[i]]
           end
      else begin stanje :=nast[stanje] end
  until (not kraj[stanje]) or (i > N);
  if kraj[stanje] then Automat:=i-duž[stanje]
                  else Automat:=i
  end;
```

Sličnost ove procedure sa funkcijom *KMPmetod* je očigledna: *stanje* zamenjuje pokazivač j, pomeranje pokazivača se zamenjuje funkcijom g a uslov $x[i] = p[j]$ uslovom da funkcija g za tekući karakter teksta i tekuće stanje ne otkazuje.

Ovaj algoritam za sravnjivanje obrasca je korišćen u sistemima za pretraživanje bibliografije u kojima su korisnici sistema zadavali ključne reči (i njihove bulovske kombinacije) po kojima su zahtevali pretraživanje [2]. Varijanta ovog algoritma je korišćena za identifikovanje funkcionalnih reči, prefiksa, sufiksa itd. u korpusu pisanih tekstova [9].

4 Boaje-Morov algoritam

R. S. Boyer i J. S. Moore su u [3] opisali algoritam za sravnjivanje niski, koji je, ako vraćanje unazad po tekstu nije problem, u velikom broju slučajeva značajno brži od do sada opisanih algoritama i obrazac sravnjuje u sublinearnom vremenu.

Osnovna ideja ovog algoritma je da se više informacija može dobiti ako se obrazac sravnjuje sa tekstom s desna ulevo, umesto s leva udesno. Zamislimo da je obrazac p dužine M postavljen iznad teksta x dužine N tako da je početak obrasca iznad prvog karaktera teksta. Prvo se upoređuju karakteri $p[M]$ i $x[M]$. Moguće su sledeće situacije:

a) Karakteri $p[M]$ i $x[M]$ nisu jednaki i karakter $kr = x[M]$ se ne pojavljuje nigde u obrascu. Tada nema potrebe da se početak obrasca postavlja na pozicije $2, 3, \ldots, M$, jer se ni na jednoj od tih pozicija obrazac ne može sravniti sa tekstom.

b) Karakteri $p[M]$ i $x[M]$ nisu jednaki a krajnje desno pojavljivanje karaktera $kr = x[M]$ u obrascu je udaljeno *skok* karaktera od kraja obrasca. Tada znamo da obrazac možemo odmah da pomerimo za *skok* mesta udesno.

Prema tome, ako tekući karakter teksta $kr = x[i]$ nije jednak poslednjem karakteru obrasca, može se preskočiti *skok* karaktera teksta, ne ispitujući uopšte preskočene karaktere. Veličina *skok* je funkcija karaktera kr i definiše se na sledeći način: Ako se karakter kr ne pojavljuje u obrascu, $skok[kr] = M$; inače je $skok[kr]$ razlika između dužine obrasca M i krajnje desne pozicije karaktera kr u obrascu.

c) Karakteri $p[M]$ i $x[M]$ su jednaki. Tada moraju da se uporede pretposlednji karakter obrasca i prethodni karakter teksta. Ako su i oni jednaki nastavlja se sa pomeranjem i pokazivača teksta i pokazivača obrasca sve dok se ili ne stigne do početka obrasca, čime je obrazac sravnjen sa tekstom, ili se naiđe na neslaganje karaktera $kr = x[M-j]$ i $p[M-j]$. U tom slučaju mogu se opet koristiti iste ideje kao u slučajevima a) i b): obrazac se pomera udesno za k pozicija da bi se izravnala dva pojavljivanja karaktera kr, pokazivač teksta se pomera za $k+j$ karaktera a pokazivač j se ponovo postavlja na kraj obrasca. Rastojanje k za koje treba pomeriti obrazac udesno zavisi od pozicije karaktera kr u obrascu. Ako je $skok[kr] < M-j$, tj. ako je tekući karakter obrasca ispred pozicije krajnje desnog pojavljivanja karaktera kr u obrascu, obrazac bi se morao pomeriti unazad da bi se izravnali karakteri $x[M-j]$ i $p[M-skok]$. To, naravno nije poželjno jer može predstavljati ulazak u beskonačnu petlju, te u ovom slučaju funkcija *skok* nije od koristi, pa se obrazac pomera za $k = 1$ karakter a pokazivač teksta za $M+1$ karaktera. Ako je, pak, $skok[kr] > M-j$, tj. ako je krajnje desno pojavljivanje karaktera kr u obrascu ispred tekućeg karaktera obrasca, obrazac se može pomeriti unapred za $k = skok[kr]-j$ karaktera a pokazivač teksta za $skok[kr]-j+j$ karaktera.

Na Šemi 3. je prikazano kako bismo, koristeći ovako opisan algoritam, pronašli nisku PRAKSI u tekstu. Iz ovog primera vidimo da je za pronalaženje obrasca dužine 6 karaktera u tekstu dužine 55 karaktera bilo potrebno ispitati samo 9 karaktera teksta (plus još šest da bi se ustanovilo sravnjivanje).

U funkciji *BMmetod* su neposredno primenjene ove neformalno izložene ideje. Niz *skok*, koji određuje pomeranje pokazivača teksta u slučaju razlikovanja tekućeg

```
JEDAN PRIMER KOJI POTVRDJUJE LINEARNOST METODE U PRAKSI
PRAKSI
      PRAKSI
          PRAKSI
               PRAKSI
                    PRAKSI
                         PRAKSI
                              PRAKSI
                                   PRAKSI
                                        PRAKSI
                                             PRAKSI
                                                  PRAKSI
```

Šema 3. Sravnjivanje niske PRAKSI Boaje-Morovim algoritmom

karaktera obrasca i teksta, dobija se preprocesiranjem obrasca p. On je veličine ulazne azbuke i određuje ga procedura *InitSkok*.

```
function BMmetod: integer;
  var i, j, ind: integer;
  begin
  i := M; j := M;
  repeat
    if x[i] = p[j]
      then begin i := i - 1; j := j - 1 end
      else begin
             i := i + max(M - j + 1, skok[ord(x[i])]);
             j := M
           end
  until (j < 1) or (i > N);
  BMmetod:= i + 1
  end;

procedure InitSkok;
  var j: integer;
  begin
    for j := 0 to 127 do skok[j] := M;
    for j := 1 to M do skok[ord(p[j])] := M - j
  end;
```

BMmetod i *InitSkok* se zasnivaju na originalnoj ideji Boajea i Mora da se obrazac sravnjuje sa tekstom s desna ulevo. Oni su, međutim, predložili dalje poboljšanje svog algoritma koje se zasniva na sličnim idejama koje su ugrađene i u Knut-Moris-Pratov algoritam.

Opišimo ove ideje prvo neformalno. Neka je $M-j$ karaktera teksta x sravnjeno sa završnih $M-j$ karaktera obrasca p. Označimo ovu podnisku sa *subp*. Neka, takođe, ovom pojavljivanju *subp* u tekstu prethodi karakter *kr* koji se razlikuje od

karaktera koji u obrascu p prethodi sufiksu *subp*. Tada želimo, grubo govoreći, da pomerimo obrazac p udesno za onoliko karaktera koliko je potrebno da bi se poravnao najduži sufiks otkrivene podniske *subp* sa njegovim krajnjim desnim pojavljivanjem u obrascu, kome ne prethodi isti karakter kao i njegovom završnom pojavljivanju.

U tom slučaju, dakle, želimo da pomerimo obrazac za k karaktera udesno, pri čemu k zavisi od pozicije u obrasu p krajnje desnog ponovnog pojavljivanja sufiksa obrasca p dužine $l \leq M - j$, kome ne prethodi isti karakter kao i sufiksu. Tada želimo da počnemo sa ispitivanjem karaktera teksta koji je poravnat sa poslednjim karakterom obrasca, što znači da se pokazivač teksta pomera za $k+M-j$ karaktera. Nazovimo ovo rastojanje *sskok* a definišemo ga kao funkciju pozicije j u obrascu na kojoj je došlo do neslaganja.

U novoj verziji algoritma, pošto je $M - j$ karaktera obrasca p sravnjeno sa tekstom pre nego što je došlo do neslaganja, pomeramo pokazivač teksta za $1+M-j$ ili $skok[kr]$ ili $sskok[j]$ karaktera, u zavisnosti od toga koji od njih ga dalje pomera. Kako je $sskok[j]$ po definiciji $k+M-j$, a uvek je $k \geq 0$, sledi da je uvek $sskok[j] \geq 1+M-j$. Stoga se pokazivač teksta pomera za $max(skok[kr], sskok[j])$ karaktera.

Knut, Moris i Prat su dali preciznu definiciju finkcija *skok* i *sskok* i definisali su efikasan algoritam za izračunavanje funkcije *sskok* [7].

Za svaki karakter kr iz azbuke je

$$skok[kr] = min\{k \mid k = M \vee (0 \leq k < M \wedge p[M-k] = kr)\},$$

a za svako j, $1 <= j < M$ je

$$sskok[j] = min\{k + M - j \mid k \geq 1 \wedge (k \geq j \vee p[j-k] \neq p[j]) \wedge ((k \geq i \vee p[i-k] = p[i]) \text{ za } j < i \leq M)\}$$

Da bismo primenili novu funkciju *sskok*, u proceduri *BMmetod* treba samo promeniti iskaz kojim se pomera pokazivač teksta:

$$i := i + max(skok[ord(kr)], sskok[j]);$$

Procedura *InitSskok* izračunava funkciju *sskok* u $O(M)$ koraka.

```
procedure InitSskok;
  var i, j, k, l: integer;
  begin
  for k := 1 to M do sskok[k] := 2 * M - k;
  j := M; l := M + 1; f[M] := M + 1;
  repeat
    if (l > M) or (p[j] = p[l])
     then begin
            l := l - 1; j := j - 1
            f[j] := l
```

```
        end
    else begin
           sskok[l] := min(sskok[l], M − j);
           l := f[l]
         end
until (j <= 0);
for k := 1 to l do
  sskok[k] := min(sskok[k], M + l − k);
end;
```

Na primer, za obrazac $p = badbacbacba$ dobijaju se sledeće vrednosti funkcija f i *sskok*:

j	=	1	2	3	4	5	6	7	8	9	10	11
$p[j]$	=	b	a	d	b	a	c	b	a	c	b	a
$f[j]$	=	10	11	6	7	8	9	10	11	11	11	12
$sskok[j]$	=	19	18	17	16	15	8	13	12	8	12	1

4.1 Efikasnost

Boaje i Mor su iscrpno testirali svoju implementaciju algoritma na računaru PDP-10 na tri razne vrste „tekstova" dužine 10000 karaktera: prvi je slučajna sekvencija 0 i 1, drugi slučajno izabrani tekst na Engleskom jeziku a treći slučajna sekvencija karaktera iz azbuke od 100 karaktera. Iz svakog od ovih tekstova je programski izabrano 300 obrazaca dužine M, za svako M od 1 do 14. Algoritam je zatim primenjen na sravnjivanje ovih obrazaca u tekstu, počev od neke slučajno izabrane pozicije u tekstu.

Za utvrđivanje efikasnosti algoritma korišćena su dva pokazatelja: jedan je odnos broja ispitanih karaktera teksta prema broju pređenih karaktera teksta dok se obrazac ne sravni (ili se ne stigne do kraja teksta) a drugi odnos broja izvršenih mašinskih instrukcija prema broju pređenih karaktera teksta.

Prvi pokazatelj, koji ne zavisi od konkretne implementacije algoritma, je pokazao da je broj ispitanih karaktera teksta po karakteru manji od 1, i smanjuje se sa povećanjem dužine obrasca. Tako je, na primer, za engleski obrazac dužine 5 prosečno ispitano 0,24 karaktera po karakteru, što znači da algoritam za svaki ispitani karakter teksta prelazi preko (približno) 4 karaktera. Za engleski obrazac dužine 14 ispituje se prosečno 0,11 karaktera. Ovi empirijski podaci su potvrdili da je algoritam sublinearan u odnosu na broj ispitanih karaktera teksta. Ovaj pokazatelj je korisno uporediti sa istim pokazateljem za KMP algoritam i za algoritam „grube sile": KPM algoritam ispituje tačno jedan karakter po pređenom karakteru, dok empirijski podaci za tekst na prirodnom jeziku pokazuju da algoritam „grube sile" ispituje približno 1,1 karakter po pređenom karakteru.

Drugi pokazatelj, koji zavisi od konkretne implementacije algoritma, je pokazao da je ukupan broj izvršenih instrukcija po pređenom karakteru manji ukoliko je dužina obrasca veća. Osim toga, ukoliko je azbuka dovoljno velika a obrazac dovoljno

dugačak, algoritam izvršava manje od jedne instrukcije po pređenom karakteru. Tako, na primer, za engleski tekst algoritam izvršava manje od jedne instrukcije za obrasce duže od 5 karaktera. Ovo praktično znači da ni jedan algoritam koji ispituje bar jedan karakter po pređenom karakteru u ovakvim slučajevima ne može biti brži od BM algoritma. KMP algoritam, implementiran na efikasan način opisan u tački 3.2, uzima svaki karakter teksta tačno jednom, ali se svaki karakter teksta može porediti sa više karaktera obrasca. Svako ovakvo poređenje zahteva bar tri instrukcije a broj ovakvih poređenja je, kao što je rečeno u tački 3.1, ograničen sa $log_{\phi}M$. Prema tome, za KMP algoritam, broj izvršenih instrukcija po pređenom karakteru je najmanje $3-p$, gde je p verovatnoća da je karakter teksta jednak tekućem karakteru obrasca. Empirijski rezultati pokazuju da efikasno kodiran algoritam „grube sile" zahteva prosečno 3,3 instrukcije po pređenom karakteru.

Ovi empirijski rezultati, kao i probabilistički model koji su Boaje i Mor izradili da bi analizirali prosečno ponašanje algoritma, su pokazali da je algoritam sublinearan u tipičnim slučajevima. Ponašanje algoritma u najgorem slučaju nije lako utvrditi: to potiče otuda što BM algoritam kad god pomeri obrazac udesno „zaboravlja" sve što je saznao o do tada sravnjenim karakterima. U [7] je pokazano da je, u slučaju kada se obrazac ne nalazi u tekstu, BM algoritam linearan i u najgorem slučaju i da je gornja granica ukupnog broj poređenja karaktera teksta sa karakterima obrasca $7N$. U [6] je pokazano da se ova gornja granica može smanjiti na $4N$.

LITERATURA

[1] Aho, A.V.: *Pattern Matching in Strings,* Formal Language Theory, Academic Press, 1980, pp. 325–347.

[2] Aho, A.V., Corasick, M.J.: *Efficient String Matching: An Aid to Bibliographic Search,* Communications of the ACM 18(6), June 1975, pp. 333–340.

[3] Boyer, R.S., Moore, J.S.: *A Fast String Searching Algorithm,* Communications of the ACM 20(10), October 1977, pp. 762–772.

[4] Davies, D.J.M.: *String Searching in Text Editors,* Software— Practice and Experience 12, 1982, pp. 709–717.

[5] Farber, D.J., Griswold, R.E., Polonsky, I.P.: *SNOBOL, A String Manipulation Language,* Journal of the ACM 11(2), January 1964, pp. 21–30.

[6] Guibas, L.J., Odlyzko, A.M.: *A new Proof of the Linearity of the Boyer-Moore String Searching Algorithm,* SIAM J. Computing 9(4), November 1980, pp. 672–682.

[7] Knuth, D.E., Morris, J.H., Pratt, V.R.:*Fast Pattern Matching in Strings,* SIAM J. Computing 6(2), June 1977, pp. 323–350.

[8] Sedgewick, R.: *Algorithms,* Addison-Wesley Publishing Company, 1983.

[9] Vitas, D.: *Prikaz jednog sistema za automatsku obradu teksta,* Zbornik radova sa Symp. Informatica 79, Bled, oktobar 1979.

Matematički fakultet
Studentski trg 16
11000 Beograd