Marija Bogićević

Postavka i testiranje ViewIt algoritma

ViewIt algoritam pripada grupi algoritama za pretraživanje. Pri korišćenju standardnih formi se uočava da ne postoji mogućnost izbora između potencijalnih rešenja, (stringova koji se razlikuju od unetog po nekom parametru) već se prihvata isključivo identičan oblik. ViewIt algoritam je nastao iz potrebe za stvaranjem fleksibilnijeg sistema.

1.0 Uvod

Verzija koja je ovde predstavljena, koristi reč kao osnovni oblik stringa koji se može uneti. Program polazi od pretpostavke da reč koja se traži može imati izvesne izmene ali takve da ih korisnik prihvati kao pogodak. Izmene se svode na sam kraj reči.

Ako imamo reč „softver", i prihvatimo izmene krajnjih slova dobićemo: softvera, software, softverskog i sl. Ovde se da primetiti da su od polazne reči samo prva četiri slova (60 %) identična sa navedenim izvedbama. Ako taj procenat istovetnosti postavimo kao konfigurabilan parametar onda sa njegovim povećanjem smanjujemo broj prihvatljivih varijacija, sve do 100 % gde zahtevamo potpunu istovetnost. Procenat istovetnosti se računa od početka reči jer korisnik zna koju reč želi da upotrebi te koeficijentima zadaje posle koliko slova želi da dopusti promenu oblika, s tim što u nađenoj reči posle tog procenta ostatak može biti proizovljne dužine (parametar promenljiv /f prekidačem pri koeficijentu 100). Da bi se pokrio što veći opseg mogućnosti, zadate reči mogu imati proizvoljan raspored u stringu nađenom u tekstu, a i uveden je pojam izuzetaka.

Prilikom unosa parametara korisnik može definisati i broj reči koje dopušta da se prihvate u stringu koji se ispituje, a koje se ne nalaze u zadatom stringu, znači potpuno nedefinisane reči tj. izuzeci. U pitanju je mogućnost a ne uslov. To znači da korisnik zadaje broj izuzetaka ali tako

Marija Bogićević (1978), Beograd, Stanka Paunovića 66/12, učebica 3. razreda Elektrotehničke škole „Nikola Tesla" u Beogradu

da u ispitivanom stringu u tekstu broj nađenih izuzetaka može biti manji ali ne i veći od zadatog limita.

1.1 Startovanje programa

Program se startuje iz komandne linije i ne zahteva nikakvo dodatno okruženje. Navodi se samo ime programa, datoteka koja se pretražuje i parametri, na primer:

viewit a10.doc /k100 /i0 /f

Korisničke opcije:

** /k koeficijent (0 do 100)
procenat svake reči unetog stringa koji mora biti identičan onom u tekstu, da bi se data reč uzela za dalje ispitivanje.

** /i izuzeci (0 do 10)
broj proizvoljnih reči koje su dopuštene u pronađenom stringu. Prilikom unosa, broj prijavljenih izuzetaka može biti manji ili jednak unetom, dok se broj veći od unetog ne toleriše i takav string proglašava neprihvatljivim. Maksimalno je moguće postaviti deset izuzetaka.

** /f
određuje da li će se prihvatiti reč koja u sebi sadrži zadatu reč ali tako da ostatak slova bude proizvoljan ili će se zahtevati da broj slova zadate i nađene reči mora biti isti. Prilikom navođenja opcije koeficijent se automatski menja u vrednosti 100 dok broj izuzetaka ostaje nepromenjen.

Program se može startovati bez navođenja parametara što onda podrazumeva vrednosti /k100 /i0 kao i kada se navede samo /f opcija. Parametri /k i /i se navode uvek u paru bilo /f opcije ili ne, inače program prijavljuje nedostatak parametara. Za pretraživanje se uzima maksimalno deset reči od broja unetih.

2.0 Analiza algoritma

Analiza ponekad predstavlja ispitivanje zahteva algoritma za određenim resursima. Katkada, resurski kao što su memorija, veličina programa i sl. su od primarnog značaja ali ono što u našem slučaju želimo da merimo je vreme.

Pre no što pređemo na analizu, moramo imati model implementacione tehnologije koja će se koristiti, uključujući model za resurse te tehnologije

i njihovu cenu. U našem slučaju koristimo jednoprocesorsku mašinu sa random-access machine modelom izračunavanja kao implementacionom tehnologijom, tako da će algoritam biti postavljen kao kompjuterski program koji radi pod DOS operativnim sistemom.

Za postavku i analiziranje algoritma potreban je i matematički aparat koji obično uključuje kombinatoriku, poznavanje algebre i sposobnost prepoznavanja najvažnijih odnosa u skupovima. Pošto se ponašanje algorima menja u odnosu na moguće unose, moramo kompletno to ponašanje svesti na jednostavnu i lako razumljivu formulu.

Pravi cilj je da se algoritam predstavi u obliku matematičkih izraza koji su jednostavni za manipulisanje i takvi da pokazuju najbitnije karakteristike algoritma, izostavljajući pri tome nepotrebne detalje.

Obično, vreme potrebno za izvršenje nekog alogirtma, raste sa uvećanjem ulazne vrednosti, pa potrebno vreme možemo opisati kao funkciju od ulazne vrednosti. U opštem slučaju ulaznu vrednost možemo opisati kao konačni broj bitova potrebnih da se ona predstavi u binarnoj notaciji.

Potrebno vreme za dati ulaz predstavlja broj „koraka" koje je potrebno izvršiti. Obično se korak definiše sa što manje vezivanja za samu mašinu. Određeno vreme je potrebno za izvršenje svake linije našeg pseudokoda. Jedna linija može zahtevati duže izvršavanje od neke druge, ali prepostavimo da svako izvršenje i-te linije zahteva vreme c(i), gde je i konstanto (*1). Jedan od zagteva je da sprecifikacija algoritma mora da sadrži i precizan opis funkcija koje se izpituju. Uobičajeno je da se algorimi opisuju kao programi pisani u pseudokodu (*2) koji sadrži sistem petlji i ispitivanja koji je sličan u nekim višim programskim jezicima (C, Paskal, itd.) a time ćemo se i ovde služiti. Ali ponekad je najbolji metod čist govorni jezik, tako da se mogu sresti i jezičke fraze smeštene između delova čistog koda.

(*1)
Evo razlike: linija koja poziva funkciju zahteva konstantno vreme, ali funkcija kada se pozove traži više vremena. Tako da treba odvojiti pozivanja funkcije, prosleđivanja paramtetara i sl. od procesa izvršavanja funkcije.

(*2)
Principi pisanja pseudokoda pruzeti iz: Introduction to algorithms [2]

Principi pseudokoda:

1. Delovi koda su grupisani u zavisnosti od toga kom bloku pripadaju. Sami blokovi nisu posebno obeleženi da se nebi vezivalo za konvencije ma kog programskog jezika.
2. Simbol predstavlja liniju komentara
3. Sve promenljive su lokalnog tipa, globalne variable su posebno izdvojene
4. Polja u vektorima počinju od nultog broja.
5. Da bismo predstavili broj elemenata nekog niza ili broj polja vektora koristimo oblik *length* $[P]$, a za označavanje adrese, $f[P]$.

2.1 ViewIt algoritam

Prvo formalizujmo problem pretraživanja. Predstavimo tekst kao polje $Text\,[0 \ldots n-1]$ veličine n a uzorak koji se traži kao polje $R\,[0 \ldots n-1]$ veličine m. To znači da su polja Text i R stringovi čiji karakteri pripadaju konačnom alfabetu.

Kažemo da se uzorak R pojavljuje u stringu Text na poziciji s (tj. uzorak počinje od s-tog karaktera stringa Text) ako $0 \leq s \leq n - m$ i $Text[s + 0 \ldots s + m - 1] = R\,[0 \ldots m-1]$, tj. $Text\,[s-j] = R\,[j]$, za $0 \leq \leq j \leq m-1$. Svrha pretraživanja je nalaženje svih pozicija na kojima se uzorak nalazi u tekstu.

Terminologija:

Veličina stringa x je predstavljena kao $[x]$. Objedinjenost dva stringa x i y, znači xy uma dužinu $|x+y|$ i sadrži karaktere iz x praćene karakterima iz y. Kada se kaže da je string prefiks stringa x, to predstavljamo u obliku ω[x ako $x = \omega y$ za neki stringa $y \in \Sigma$, a to znači da $|\omega| \leq |x|$. Na sličan način se definiše i sufiks samo se tada upotrebljava znak]. Takođe treba naglasiti da za svaki string x i y i neki karakter a, važi x[y ako i samo ako $a\,x\,[\,a\,y$. Relacije [i] imaju i svojstvo tranzitivnosti. Koristićemo se ovim da bi smo uočili osnovne relacije.

ViewIt algoritam predstavljamo u obliku pseudokoda:

```
globalne promenljive: brrec, lizuz, rizuz, brcpyall
dimenzije vektora: R[MAXW][MAXC+1] za
pre[MAXW][MAXC+1]
pos[MAXW-1][MAXC+1]
maxw[MAXW]
Find_Right_Words( text, pretpopoz, priswit)
Find_Left_Words (text, pronrec, pretpopoz, priswit )
Init_Search (string)
Str_Search (text, retpos)
CpyStrToStr (str1, str2)
ReverseStr (str)
CmpTwoStr( str1, str2)
CmpNTwoStr(str1,str2,brchar)

Viewit ( text, string, KOEF, IZUZ, priswit )

1.brrec= maxword =k =0
2.do repeat
3.        brslova= 0
4.            while c=string[k]not ' '
5.                te[brrec][brslova] =c
```

```
6.                brslova =brslova+1
7.                k= k+1
8.     maxw[brrec] brslova
9.     konbrr= integer[( brslova/100)KOEF+0.5]
10.        if konbrr<1
11.            konbrr= 1
12.    te[brrec][konbrr] '\0'
13.    te[brrec][MAXC] 0
14.until c'not \0' and brrecAXW
15.    for a =0 to brrec-1
16.        max =0
17.            for k= 0 to brrec-1
18.                if maxw[k]>max
19.                    max= maxw[k]
20.                    maxword= k
21.                CpyStrToStr(R[a], te[maxword])
22.                maxw[maxword] 0
23.     Find_Right_Words(R[0])
24.  do repeat
25.     retpos (text, 0)
26.     c=text[retpos-1]
27.     if (c==' ' or c=='\n' or retpos==0) and retpnos not (-1)
28.         rbrpro= (text, retpos, priswit)
29.         if retpos= 0
30.             lbrpro (text, rbrpro, retpos, priswit)
31.         else lbrpor= 0
32.         p= rbrpro+rizuz
33.         while p>=0 and prek1=1
34.             if pre[p][MAXC==(-3) and brcpyall not 0
35.                 rbrpro= rbrpro-1
36.                 brcpyall= brcpyall-1
37.             else
38.                 if pre[p][MAXC]=(-2)
39.                   rizuz =rizuz+brcpyall
40.                 rbrpro= rbrpro-1
41.                 prek1=0
42.               else rizuz= rizuz-1
43.             p= p-1
44.      if brrec=rbrpro+lbrpro and rizuz+lizuzZUZ
45.          print "Found string"
46.             if lizuz+lbrprno not 0
47.                 for p= lizuz+lbrpro-1 to 0
```

```
48.                     print pos[p]
49.               if rizuz+rbrpro= 0
50.                   for p= 0 to rizuz+rbrpro-1
51.                     print pre[p]
52.           if retpos(-1)
53.              retpos=retpos+1
54.  until retpos(-1)
```

Ako odložimo zasad analiziranje funkcija α, β, ω i ϖ, uočiće se da se kôd svodi na deo za obradu uzorka, deo za pronalaženje najduže reči uzorka i deo za traženje ostalih reči i njihovu prezentaciju.

Definišimo najpre odnos uzorka i teksta u kom se on traži. Ako za alfabet uzmemo ASCII set znakova, uzorak možemo svesti na:

$a = ' ' \in \Sigma$
string $\in \Sigma$
string $= y_0\, a\, y_1\, a \ldots y_{brrec}$
te $= y_0\, y_1 \ldots y_{brrec}$ te$[\ k\ ] = y_{k,} \quad k \in \{0, 1, \ldots brrec\}$

String je definisan tako da između reči yk, moraju stajati blanko znaci da bi reči kao takve postojale. U slučaju da je blanko znak u reči prisutan na početku i pored obaveznog blanko znaka onda je takav znak tretiran kao reč koja je sama blanko znak bez zahteva da bude ograničena novim blanko znakom sto znači

$$\exists\, y : a\, [\ y \wedge a \neq y \quad y = a\, y' \Rightarrow a\, y'' \wedge y = y''\, y'$$

Prilikom navođenja korisničkih parametara objašnjena je upotreba koeficijenata istovetnosti. Upoređujući dve reči tretirajući ih kao stringove možemo naći određen broj istih karaktera koji čine prefiks svake od njih dok se u ostalim karakterima mogu razlikovati. Dužinu tog prefiksa definiše korisnik u obliku koeficijenta. Dakle potrebno je ograničiti dužinu reči u prerađenom uzorku na onu dozvoljenu koeficijentom. Broj karaktera koji je dodeljen koeficijentom izračunava se po formuli (te $[k]/100) \cdot$ KOEF + 0.5 čiji celobrojni deo predstavlja. Ako je dobijeni broj jednak nuli vrši se zaokruživanje na 1 što je realna situacija ako za reč imamo jedno slovo a veličina koeficijenta je ispod 100. Vrednost 0.5 koja se dodaje proizvodu u formuli data je stoga što se želeo postići što veći celobrojni deo. Ta vrednost proširuje opseg koeficijenata za koji će uzorak imati istu dužinu pri fiksnom unetom stringu. To znači da za unetu reč softver dobiće se uzorak soft za opseg koeficijenata do 50 do 64% tj. (lenght $[''\text{soft}'']/100) \cdot$ KOEF $\pm$ $(0.5/100) \cdot$ KOEF, odnosno 57 $\pm$ 7% (kliko iznosi 0.5 u odnosu na dužinu unetog stringa). Nakon toga osamde-

seto-kontrolno polje stringa setuje se na (–1). Međutim ako pogledamo primer:

Dati string: a to assembling
Nađeni string: assembling to a

i ako koristimo standardnu grupu funkcija za upoređivanje stringova (funkcija CmpNTwoStr upoređuje dva stringa za zadati broj znakova u ovom slučaju dužinu zadatog stringa), primetićemo da se a poklapa sa assembling , to sa to ali da neće doći do poklapanja assembling i a pa nastaje problem utapanja" stringova. Zbog toga se reči moraju složiti u matrici po dužini, kako bi se uvek upoređivalo prvo najdužom. Slaganje reči se vrši tako što se pri prvom prolazu kroz petlju, u stringu u kom su izdvojene samo dužine pronađe najveći broj (kako su slagane reči, tako su slagane i njihove dužine) pa se reč sa tim poljem postavi kao prva, broj u stringu sa dužinama se za to polje setuje na nulu i tako kroz onoliko iteracija koliko ima reči u zadatom stringu.

Pronalaženje najduže reči (zvaćemo je ključnom) stringa svodi se na upotrebu Boyer-Moor-Pratt algoritma za traženje fiksnog stringa koji je dat u obliku funkcija Init_Search i Str_Search. Pošto je string sa rečima uzorka već sortiran, najduža je nulta reč. BMP algoritam će kasnije biti detaljno opisan iz razloga što je ovde iskorišćen u potpunosti te čini bitan faktor u brzini Viewit algoritam. BMP algoritam prosleđuje poziciju čak i ako uzorak predstavlja sufiks (skup krajnjih znakova) neke reči pa je potrebno pre pokretanja seta funkcija za bočno traženje, u tridesetoj liniji ispitati da li je karakter neposredno ispred pozicije vraćene BMP algoritmom blanko znak tj. znak novog reda. Ako je string nađen na samom početku $s = 0$ $R[0] \in Text$, onda takvi znaci i nemogu postojati pa se i ovakav slučaj prihvata ali se pri tom izuzima pretraživanje sa leve strane. Linije 32-43 pripadaju grupi za modifikaciju prilikom dupliranja delova nađenog stringa i biće kasnije potpuno obrađene, ne pripadaju samo jednoj funkciji pošto se njihovo delovanje prostire i na Init_Search i Str_Search finkcije. Poslednjih sedam linija služe za ispisivanje pronađenih reči onim rasporedom koje imaju u tekstu.

Već na sam pogled, u petljama koje ispisuju reči nađene sa bočnih strana uočavaju se nove promenljive, *lizuz* i *rizuz*. U pitanju su izuzeci. Pogledajmo u kom odnosu mogu biti izuzuci i reči iz uzorka.

Ako nađeni string obeležimo kao Find i posmatramo ga kao skup čiji su elementi reči uzorka i reči izuzeci za IZUZ ≤ MAXIZUZ, onda se može napisati $R \subset Find$ pošto reči uzorka pripadaju nađenom stringu a da pritom mogu biti u proizvoljnom rasporedu. Ako sada posmatramo ne kao skup već kao konačan string, prilikom definisanja izuzetaka IZUZ ≠ 0, to svojstvo ostaje s tim što $y \in Find$ i $y' \in Find$ gde su y i y' dve različite reči

iz uzorka jer se izuzetak sme tolerisati samo u prostoru između tih reči. Pošto se odvojeno vrši pretraživanje sa desne odnosno leve strane ključne reči, broj izuzetaka je izdvojen u dve promenljive zavisno od strane na kojoj su nađeni.

Pretraživanje udesno

Pretraživanje udesno predstavlja traženje preostalih reči uzorka nakon nalaženja ključne reči gde vrednost pozicije odakle se učitavaju znaci teži ka uvećanju. Funkcija Find_Right_Words se može predstaviti pseudokodom:

```
Find_Right_Words( text, pretpopoz, priswit )

1.  prek1= prek2 1
2.  brojrec =brrec
3.  brpronr =prebrr =trap =rizuz =rizuz= 0
4.  do repeat
5.     brslova= 0
6.     while c= text[pretpopnoz] not ' ' and c not'\n' and c not'\0'
7.        pre[prebrr][brslova]= c
8.        brslova= brslova+1
9.        pretpopoz= pretpopoz+1
10.   pre[prebrr][brslova]='\0'
11.   pre[prebrr][MAXC] =(-1)
12.   pretpopoz =pretpopoz+1
13.   brtest= 0
14.      while brtest< brrec and prek1==1
15.         if R[brtest][MAXC]==(-1)
16.            if KOEF=100 and priswit=1
17.               if CmpTwoStr(R[brtest], pre[prebrr])==0
18.                     if brtest==0
19.                        R[brtes][MAXC]=(-3)
20.                     else
21.                        R[brtes][MAXC] =prebrr
22.                     pre[prebrr][MAXC]=(-2)
23.                     brpronr= brpronr+1
24.                     prek1= 0
25.            else
26.if CmpTwoNStr(R[brtest], pre[prebrr], length[R[brtest]])==0
27.                  if brtest==0
28.                     R[brtes][MAXC]=(-3)
29.                  else
```

```
30.                         R[brtes][MAXC]= prebrr
31.                     pre[prebrr][MAXC]=(-2)
32.                     brpronr= brpronr+1
33.                     prek1= 0
34.         brtest =brtest+1
35.     prek1= 1
36.         if pre[prebrr][MAXC]==(-1) and IZUZ not 0
37.             prek2= 0
38.         else
39.             if pre[prebrr][MAXC]==(-1) and IZUZ not 0
40.                 if rizuz<= IZUZ
41.                     if pre[prebrr-1][MAXC]==(-1)
42.                         if pre[prebrr-2][MAXC] not (-1)
43.                             if rizuz not 1
44.                                 trap= riziz-1
45.                             else
46.                                 trap =rizuz
47.                          if rizuz==IZUZ
48.                             rizuz =trap
49.                             prek2 =0
50.                          else
51.                             rizuz =rizuz+1
52.                             brojrec =brojrec+1
53.                     else
54.                         if rizuz=IZUZ
55.                             rizuz =rizuz+1
56.                              brojrec =brojrec+1
57.     prebrr =prebrr+1
58.  until prebrr <brojrec and prek2==1 and c not '\0'
59.     return(brpronr)
```

Celokupan red funkcije se obavlja u okviru do repeat-until petlje sve dok se ne ispuni uslova da je broj nađenih reči prebrr u osnovi jednak broju zadatih reči ili se dođe do samog kraja stringa *Text* u kome se vrši pretraga. Promenljiva *prek2* koja se nalazi u tom uslovu dobiće vrednost 0 tj. prekinuće dalji proces ako se naišlo na nepoznatu reč a broj izuzetaka definisan od strane korisnika je jednak nuli ili ako je broj pronađenih izuzetaka jednak tom unetom broju izuzetaka. Linije 6-10 služe za učitavanje reči iz stringa *Text* u matricu pre i funkcionišu na isti način kao i u glavnom programu. Ustvari takav način izdvajanja reči biće isti i za glavni program i za funkcije bočnog pretraživanja, pa stringovi i ovde imaju istu karakteristiku. Potrebno je samo voditi računa da se prekine

unos vrednosti kada se dođe na kraj stringa. Nakon terminisanja reči nulom u desetoj liniji nailazimo na setovanje kontrolnog polja na (−1). Kontrolna polja mogu imati vrednosti:

value	purpose
-1	exeption
-2	word
-3	word copy

za nadjeni string [pre[MAXW][MAXC]]

value	purpose
-1	exeption
0,...brrec	nb. of similar word

za traženi string [R [MAXW][MAXC]]

Kontrolna polja traženih stringova (reči uzorka) su setovana na (−1) jer još nisu nađeni odgovarajući stringovi u tekstu. Promenljivom pretpopoz dobijamo vrednost pozicije na kojoj se nalazi ključna reč. Od te pozicije se počinje sa učitavanjem znakova. Nakon izdvajanja i terminisanja, reč je spremna za upoređivanje. Njeno kontrolno polje je takođe postavljeno na vrednost (−1) jer još uvek neznamo da li će se poklopiti sa ijednom reči uzorka. Kada kažemo da je došlo do poklapanja reči?

Definišimo prvo relaciju Q ~ P za stringove Q i P tako da Q[P ili P[Q. Pri takvom odnosu dva stringa mi ih možemo izravnati po nekom broju karaktera na samim njihovim počecima tako da imamo parove istih karaktera iz svakog stringa tj. stingovi će se za dati broj karaktera poklopiti. Relacija je simetrična dakle Q~P ⇔ P~Q.

Dakle za dva stringa (R[ ] − traženi i *pre*[ ] − nađeni) kažemo da su se poklopila ako zadovoljavaju relaciju: R[Q ∧ R[S ⇒ QS gde bi R predstavljao string R odnosno traženih K posto slova reči unetog stringa, Q reč iz unetog stringa a S odgovarajuću reč iz nađenog stringa. Tek ako je zadovoljena ova relacija kontrolno polje stringa *pre*[*k*] će biti setovano na (−2) a kontrolno polje *R*[*m*] na vrednost *k* (*k* je pozicija reči u okviru matrice pre a m pozicija reči u okviru matrice R).

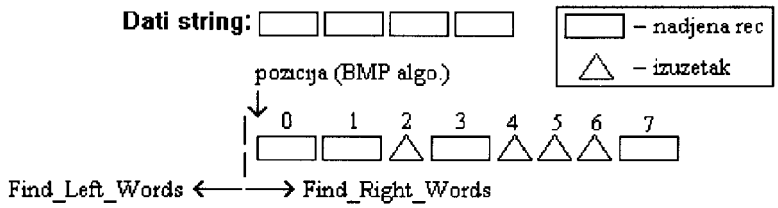
Pošto je učitan string (*pre*[ ]) iz *Text* matrice potrebno je izvršiti njegovo upoređivanje sa svakim stringom iz matrice R čije kontrolno polje ima vrednost (−1) to jest koji nije dosad nađen. Ako je došlo do preklapanja promenljiva prek1 menja vrednost u nula i prekida se dalje upoređivanje. To znači da će sledeća učitana reč biti upoređivana sa jednim manje stringom. U šesnestoj liniji nalazi se uslov da je vrednost koeficijenata 100 i da je navedena /f opcija (*priswit* = 1). Za koeficijent 100 znamo da predstavlja nedirnutu reč, onakvu kakva je data pri unosu. Međutim, nađena reč može uneti sadržati u potpunosti ili joj biti jednaka da bi se prihvatilo preklapanje. Tu dakle imamo dva slučaja:

− KOEF = 100 *priswit* = 0 (bez /f)
  važi relacija R [Q ∧ R [*S* ⇒ Q ~ S

– KOEF = 100 *priswit* ≠ 0 (sa /f) ulazni string i obrađeni su jednaki R = Q i važi R = Q ∧ R = S ⇒ Q = S tj. ulazni i izlazni string moraju biti jednaki

Izuzmimo za trenutak linije 19-21 i 28-31 primetićemo da se dalje jednako tretira string za oba uslova bitno je da, zavisno od priswit vrednosti, zadovolji jednu od relacija. Za tako nađeni string broj pronađenih reči uvećaće se za jedan.

Može se desiti da string ne zadovolji relaciju. Za broj izuzetaka IZUZ = 0 takav string bi izazvao prekidanje daljeg traženja i izlazak iz funkcije, gde bi povratna vrednost bila jednaka broju nađenih reči pre nailaska na izuzetak. Za broj izuzetaka: 0 < IZUZ ≤ MAXIZUZ dolazi do ispitivanja položaja stringa kroz linije 36-56. Predstavimo grafički jedan odgovarajući string:



Algoritam će se prema stringu odnositi zavisno od broja dozvoljenih izuzetaka. Za date vrednosti promenljive IZUZ možemo pokazati kako će se algoritam ponašati za bilo koji unos i raspored.

• IZUZ = 0. Ako se naiđe na izuzetak prekida se petlja i vraća se vrednost reči nađenih pre pojave izuzetka

• IZUZ = 1. Pri nailasku na izuzetak u drugom polju, uslov *rizuz* ≤ ≤ IZUZ je zadovoljen za *rizuz* = 0. Pošto polje pre ovog nije izuzetak i da sledeće polje može biti ili izuzetak ili reč te se broj izuzetaka može bezbedno uvećati dok se to ne utvrdi, ali samo pod uslovom da je rizuz manji od IZUZ jer bi se moglo preći preko granice IZUZ vrednosti. Prilikom nailaska na izuzetak u četvrtom polju i dalje važi relacija rizuz IZUZ, pa pošto prethodna reč nije izuzetak a *rizuz* = 1 onda se promenljiva trap postavlja na vrednost rizuz odnosno 1. Odmah u sledećoj liniji zadovoljava se uzlov da je *rizuz* = IZUZ, rizuz zadržava vrednost i izlazi se iz petlje. Broj pronađenih reči je 3 a izuzetaka 1.

• IZUZ = 2. Do izuzetka u petom polju važi isti postupak kao i kod slučaja za

• IZUZ = 1. Za peto polje uslovi su zadovoljeni do linije 42. Polje pre petog je i samo izuzetak pa je *trap* = *rizuz* – 1 = 1 koja pokazuje koliko izuzetaka ima do zadnje nađene reči tako da u slučaju da je broj

zadatih izuzetaka veći od ovde datog odbrojavali bismo izuzetke sve dok se na kraju, kao i u našem slučaju, ne zadovolji uslov *rizuz* = IZUZ (ili ne naiđe na reč) broj izuzetaka postavi na vrednost promenljive *trep*. Potom se izlazi iz petlje. Broj nađenih reči je 3 a izuzetaka 1.

Na samom početku analize ove funkcije rečeno je da se do repeat-until petlja ponavlja sve dok je broj pronađenih reči (umanjen je za jedan jer se računa od nule) manji od promenljive brojrec, te da je promenljiva brojrec u osnovi jednaka broju reči uzorka. Međutim, ako se u nađenom stringu nailazi i na izuzetke potrebno je tu promenljivu uvećati. Tako se granica pomera za jedan kad god se naiđe na izuzetak.

Pretraživanje ulevo

Pretraživanje ulevo predstavlja traženje preostalih reči uzorka nakon nalaženja ključne reči i prolaska kroz funkciju Find_Right_Words gde vrednost pozicije odakle se učitavaju znaci teži ka umanjenju. Funkcija Find_Left_Words se može predstaviti pseudokodom:

```
Find_Lefr_Words( text, pronrec, pretpopoz, priswit )

1.    prek1= prek2 =1
2.    brcpyall =brcpytmp= prebrr= trap =izuz= 0
3.    restr =(brrec-pronrec)2
4.        if restr==0
5.            return (0)
6.    pretpopoz =pretpopoz-2
7.  do repeat
8.    brslova= 0
9.     while c= text[pretpopoz] not ' ' and c not'\n' and c not'\0'
10.           pos[prebrr][brslova]= c
11.           brslova =brslova+1
12.           pretpopoz =pretpopoz-1
13.    pos[prebrr][brslova] ='\0'
14.    ReverseStr(pos[prebrr])
15.    pos[prebrr][MAXC] =(-1)
16.    pretpopoz =pretpopoz-1
17.    brtest= 0
18        .while brtest<brrec and prek1==1
19.           if R[brtest][MAXC]not (-3)
20.               if KOEF==100 and priswit==1
21.                   if CmpTwoStr(R[brtest], pos[prebrr])==0
22.                       if R[brtest][MAXC] not (-1)
23.                           brcpyall brcpyall+1
```

```
24.                             brcpytmp =brcpytmp+1
25.                              pre[R[brtest][MAX]][MAXC]= (-3)
26.                           else
27.                             brcpytmp =0
28.                             brpronr =brpronr+1
29.                           R[brtest][MAXC]= (-3)
30.                           pos[prebrr][MAXC] =(-3)
31.                           prek1= 0
32.                   else
33.if CmpNTwoStr(R[brtest], pos[prebrr], length[R[brtest]])==0
34.                     if R[brtest][MAXC] not (-1)
35.                         brcpyall= brcpyall+1
36.                         brcpytmp= brcpytmp+1
37.                         pre[R[brtest][MAXC]][MAXC]= (-3)
38.                      else
39.                          brcpytmp =0
40.                          brpronr= brpronr+1
41.                       R[brtest][MAXC]= (-3)
42.                       pos[prebrr][MAXC] =(-3)
43.            prek1= 0
44.     brtest =brtest+1
45.  prek1= 1
46.             if pos[prebrr][MAXC]==(-1) and (IZUZ=0 or
                                  brrec<=pronrec+brpronr)
47.            prek2= 0
48.         else
49.            if pos[prebrr][MAXC]==(-1) and IZUZ= 0
50                 if lizuz=IZUZ
51.                    if prebrr not 0
52.                       if pos[prebrr-1][MAXC]==(-1)
53.                          if pos[prebrr-2][MAXC]==(-2)
54.                             trap= lizuz-1
55.                          if lizuz==IZUZ
56.                             lizuz= trap
57.                             prek2=0
58.                          else
59.                             lizuz=lizuz+1
60.                             restr=restr+1
61.                       if lizuz<IZUZ
62.                          lizuz=lizuz+1
63.                          restr= restr+1
64.                else
```

```
65.                 if brrec==pronrec
66.                     prek2= 0
67.                 else
68.                     lizuz= lizuz+1
69.                     restr=restr+1
70.       prebrr =prebrr+1
71.  until (prebrr<restr) and prek2==1 and c not '\0'
72.       return( brpronr+brcpyall )
```

U glavnom programu uočili smo kako se dolazi do ključne reči i šta predstavlja njenu poziciju. Funkcija dobija upravo tu poziciju kao jedan od argumenata. Kao što je već pomenuto do startovanja funkcija za bočno traženje neće doći ukoliko ključni string ne predstavlja sam po sebi reč a ne deo neke druge. To svojstvo ključnog stringa utvrđivali smo ispitivanjem karaktera ispred. Dakle, vrednost promenljive *pretpopoz* predstavlja poziciju prvog karaktera ključne reči, pretpopoz − 1 karakter iz kontrolne grupe (′ ′ i ′\n′) dok *pretpopoz* − 2 poslednji karakter prve reči sa leve strane. Kao što se vidi potrebno je promenljivu *pretpopoz* već na početku umanjiti za dva.

Reči nađene sa leve strane ključne reči učitavaju se u dvodimenzionalnu matricu *pos* [*MAXW*] [*MAXC*] gde su MAXW i MAXC njene dimenzije. Sam način unošenja reči je identičan sa unošenjem u funkciji Find_Right_Word i već je razmatran. Kako se učitavanje vrši od poslednjeg karaktera stringa to ga je potrebno invertovati što se obavlja u četrnestoj liniji neposredno posle učitavanja. Nakon toga kontrolna polja stringa se setuju na vrednost (−1). Kontrolna polja mogu imati vrednosti:

value	purpose	
-1	exeption	za na eni string (pos([MAXW]MAXC])
-3	word/ word copy	i tra eni string (pos([MAXW]MAXC])

Zanemarimo trenutno uticaj promenljivih *brcpytmp* i *brcpyall* kao i pitanje novog vida setovanja kontrolnih polja. Njihov uticaj ima mnogo dublju pozadinu i biće odvojeno razmatrani u delu rasprave o modifikaciji prilikom dupliranja stringa što će baciti novo svetlo na ceo algoritam. Da bismo se uopšte upustili u to bilo je potrebno da razjasnimo kako rade delovi funkcija koji se ne oslanjaju na te promenljive. Zasad je dovoljno da znamo da je vrednost (−3) preuzela ulogu vrednosti (−2) iz Find_Right_Words funkcije to jest, ulogu odgovarajuće reči dok je (−1) i dalje vrednost za izuzetke. Izuzeci su i ovde tretirani na sličan način. Sada možemo i grafički izložiti kroz obe funkcije proces traženja uzorka:

Text: AFTER LOOKING AT THE STRUCTURE OF AN OPTIMAL SOLUTION

□	purpose
-1	exeption
-2	word
-3	word / word

1) Given string: THE SOLUTION OF AN OPTIMAL STRUCTURE

/k100 | /i(0...MAXIZUZ)

Pattern: | STRUCTURE [-3] | SOLUTION [4] | OPTIMAL [3] | THE [-3] | OF [1] | AN [2] |

THE [-3]	STRUCTURE [-2]	OF [-2]	AN [-2]	OPTIMAL [-2]	SOLUTION [-2]
⓪	⓪	①	②	③	④

Find_Left_Words ← | → Find_Right_Words

side	words	exept
right	5	0
left	1	0

$N=5+1=lenght[pattern]$

$\tau(N)=\top$

Text: AFTER LOOKING AT THE STRUCTURE OF AN OPTIMAL SOLUTION

□	purpose
-1	exeption
-2	word
-3	word / word

2) Given string: OPTIMAL LOOKING AFTER STRUCTURE

/k100 | /i (0, 1, 2, 3)

Pattern: | STRUCTURE [-3] | OPTIMAL [-1] | LOOKING [-1] | AFTER [-1] |

STRUCTURE [-2]
⓪

Find_Left_Words ← | → Find_Right_Words

side	words	exept
right	1	0
left	0	0

$N=1+0=lenght[pattern]$

$\tau(N)=\bot$

/k100 | /i(4,... MAXIZUZ)

Pattern: | STRUCTURE [-3] | OPTIMAL [3] | LOOKING [-3] | AFTER [-3] |

AFTER [-3]	LOOKING [-3]	AT [-1]	THE [-1]	STRUCTURE [-2]	OF [-1]	AN [-1]	OPTIMAL [-2]
③	②	①	⓪	⓪	①	②	③

Find_Left_Words ← | → Find_Right_Words

side	words	exept
right	2	2
left	2	2

$N=2+2=lenght[pattern]$

$\tau(N)=\top$

$M=2+2 \leq IZUZ$

$\tau(M)=\top$

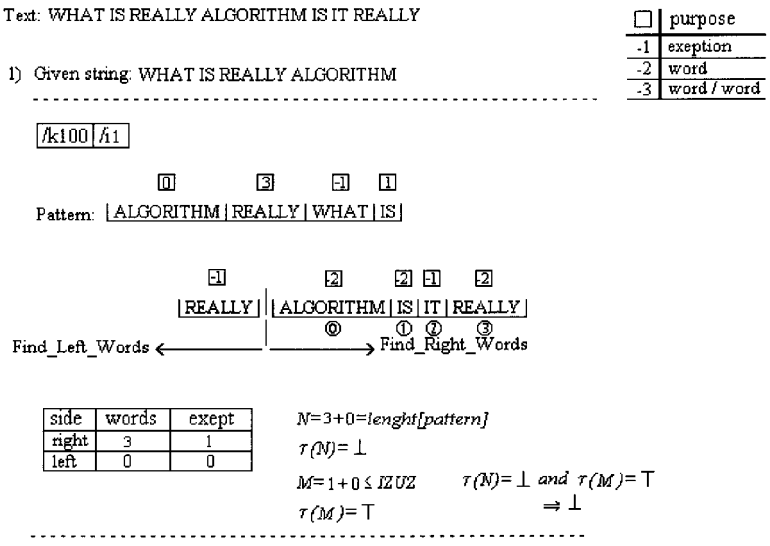
$\tau(N)=\top$ *and* $\tau(M)=\top$

$\Rightarrow \top$

Ovde su data dva primera pretraživanja koja trebaju da pokažu kako ViewIt algoritam izgleda u praksi. Ubrzo ćemo videti i da postoje određene situacije kada je potrebno dodatno preurediti konačan broj nađenih reči i izuzetaka te da sistem, u osnovi dobar, ipak mora da pretrpi izvesne izmene. Pod brojem jedan, zadat je string čije se reči u tekstu nalaze u obliku kakvom su zadate ali nešto drugačijeg rasporeda. Dat je stoga što pokazuje da je raspored reči pri unosu za fiksan string, sasvim nebitan jer ga algoritam uvek preuređuje na isti način. Pod brojem dva, zadat je drugačiji string ali su prikazani rezultati za dve različite vrednosti izuzetaka. Uočava se koliko drastično drugačiji rezultat dobijamo za samo jedan više definisan izuzetak.

Modifikacije pri dupliranju stringa

Pogledajmo kakav bi rezultat davao algoritam ako bismo iz pseudokoda izuzeli line u kojima se pojavljuju promenljive *brcpyall* i *brcpytmp* tj. kakav bi rezultat bio ako bi se ponašao samo u skladu sa onim što smo do sada rekli. Za tekst je dat string čiji sadržaj ma koliko nelogično delovao ipak predstavlja realnu kombinaciju.



Vidi se da zadati string postoji u tekstu. Zašto ga onda algoritam nije registrovao? Znamo da se pretraživanje vrši prvo funkcijom Find_Right_Words znači od ključne reči na desno. Algoritam se standardno ponaša setujući kontrolna polja na vrednost (–2) za reči i (–1) za izuzetke. Tako da su stringovi is i really prihvaćene kao nađene reči, međutim kada se krene sa pretraživanjem na levoj strani već pri nailasku

na string really , pošto je jedan izuzetak već nađen na desnoj strani i time dostignut limit, prekida se sa daljim traženjem. Ptimetićemo da nam sa leve strane ostaje i string what . Da je algoritam pronašao sve zadate reči sa desne strane, on nebi učitao string really na levoj strani a time ni pravio problem kopija. Ali pošto kao krajnju reč imamo while iz zadatog stringa, to znači da ta reč predstavlja granicu na levoj strani i sve reči pre nje moraju pripadati konačnom stringu. Da li ćemo te reči prihvatiti kao reči ili izuzetke je sve jedno, bitno je da se nđu u konačnom stringu. Međutim još uvek nam ostaje par kopija sa desne strane.

Pogledaimo:

da je regularna reč a ne kopija na krajnjem desnom položaju onda bi ta regularna reč predstavljala granicu i sa desne strane bi bili prinuđeni da reči is i really sa leve strane ostavimo kao regularne reči a sa desne straneproglasimo za izuzetke, te bi kasnija ispitivanja ustanovila da li je taj novi broj izuzetaka prihvatljiv ili ne. Međutim to ovde nije slučaj dakle potrebno je nekako označiti reči koje su kopije na desnoj strani (kopije na levoj strani se tretiraju kao regularne reči jer su ispred granične (regularne reči) inače se nibi prigvatile tj. bile bi nepoznate reči koje se normalno odbacuju) i te kopije na desnoj istrani ispitati da li su krajnje te ih je moguće ukloniti ili ne te ih moramo prihvatiti kao izuzetke.

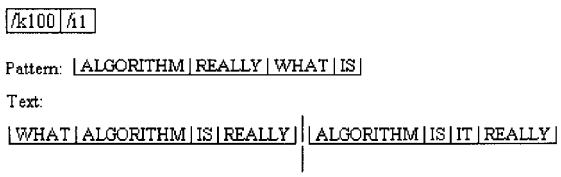
Postoji jednostavan sistem da se to uradi. Od linije 21 u pseudokode funkcije Find_Left_Word imamo ovakav deo koda:

```
...
if CmpTwoStr(R[brtest], pos[prebrr])==0
    if R[brtest][MAXC] not  (-1)
        brcpyall=brcpyall+1
        brcpytmp= brcpytmp+1
        pre[R[brtest][MAXC]][MAXC]= (-3)
    else
        brcpytmp= 0
        brpronr =brpronr+1
R[brtest][MAXC]= (-3)
pos[prebrr][MAXC]= (-3)
...
```

U pitanju je deo gde upoređujemo reč uzorka sa stringom iz matrice pos. Ako je kontrolno polje reči uzorka *R* [*brtest*] različito od (–1) što znači da je ta reč već nađena sa desne strane, onda se promenljiva *brcpyall* (ukupan broj kopija) uvećava za jedan kao i promenljiva *brcpytmp* (broj kopija do zadnje nađene regularne reči). Vrednost kontrolnog polja

R [*brtest*] [*MAXC*] pokazuje poziciju (0, ... *brrec*) reči koja je nađena sa desne strane (ta pozicija je označena u primerima kao brojevi u krugovima neposredno ispod reči). Setovanje kontrolnog polja zadate reči vršeno je još u liniji 21 funkcije Find_Right_Words. Tako dobijena pozicija će poslučiti da se kontrolno polje na toj pozciji sa desne strane, setuje na vrednost (–3) što će tu reč označiti kao kopiju. I kontrolno polja reči uzorka (*R*[ ]) sa kojom je upoređivanje izvršeno promeniće vrednost na (–3) jer nemožemo dopustiti triplikate. Linija 19 funkcije Find_Left_Words će se pobirinuti da ne dođe do upoređivanja sa rečima čije kontrolno polje ima tu vrednost. Iz tog razloga se i polje adekvatne reči (*pos*[ ])sa leve strane setuje na (–3) vrednost.

Postavlja se pitanje kakao će se tretirati izuzeci u ovakvoj situaciji. U Find_Right_Words funkciji neće doći do promene što se tiče izuzetaka. Međutim postavimo ovakav slučaj:



Pri traženju za prvu reč algorithm pošto je ona najduža odatle se kreće u pretragu na desnoj strani. Pri nailasku na drugu reč algorithm prestaje se sa traženjem na desnoj strani pošto funkcija upoređuje reči samo sa stringovima iz uzorka koji dotada nisu nađeni tj. vrednost kontrolnog polja im je (–1).

Ali kada se krene od pozicije vraćene BMP algoritmom za drugu reč algorithm ; kao najdužu, pri pokretanju funkcije za traženje sa leve strane prva reč algorithm će postaviti drugu za kopiju što se nikako ne sme dozvoliti jer će se onda rezultat, kao što ćemo uskoro razjasniti, svesti na rezultat za prvu reč algorithm . Stoga se još u okviru funkcije Find_Right_Words mesto da se za tu ključnu u string sa uzorkom (*R*[ ]) upiše 0 kao vrednost njene pozicije, upisuje (–3) (u kontrolno polje nađene reči se normalno upisuje vrednost (–2)), čime se u startu onemogućuje stvaranje kopije.

U funkciji Find_Left_Words se izuzeci i dalje tretiraju na isti način, pošto su broj pronađenih reči, izuzetaka i broj kopija pretvorenih u reči potpuno nezavisne promenljive. Ipak, prilikom vraćanja vrednosti ova funkcija vraća zbir pronađenih reči sa konačnim brojem kopija (*brcpyall*) pretvorenih u reči.

Tako preuređen pre string potrebno je propustiti kroz kod koji će izbaciti kopije ako se nalaze na kraju string. Tu funkciju obavlja glavni program pre nego što će se stringovi štampati.

2.2 Boyer-Moor-Praat algoritam

Ako je uzorak P relativno dugačak a alfabet Σ pristojene veličine, algoritam čiji su tvorci Robert S. Boyer, J. Strother Moor i M. Praat, može se predstaviti kao jedan od najefikasnijih algoritama za pretraživanjem sa zadatim stringom. Ovde ćemo koristiti verziju u pseudokodu koja je modifikovana zarad potreba ViewIt algoritma.

```
Boyer_Moor_Praat ( P, T, position )

1.len=length[P ]
2.limit=length[T ]
3.pos=len+position-1
4.findme=P
5.shigt
        ** Vektor table[ ] ima  za dimenziju
5.i=0
6.while i<=255
7.      table[i ]=len
8.      i=i+1
9.i=0
10.   while i<len
11.       table[P [i ]]=len -i-1
12.    while pos<limit
13.       while pos<limit and shift=table[T [pos ]]>0
14.           pos=pos+shift
15.        if shift=0
16.           if.CmpNTwoStr( findme, [T[pos-len+1]], len )==0
17.              return (pos-len+1)
18.           else pos=pos+1
19. return (-1)
```

Algoritam kao što se vidi upoređuje uzorak kroz tekst pomerajući ga sa desna u levo. Kroz petlju u sedmoj liniji, popunjavaju se sa dužinom uzorka, polja vektora table čija je dimenzija veličine alfabeta Σ. U pitanju je veličina od 256 polja, koliko ima ASCII znakova. U jedanestoj liniji nailazimo na petlju u kojoj se manipuliše sa vrednostima slova uzorka tako što za svako slova pojedinačno određuje kod u Σ tabeli i tom polju u vektoru table menja vrednost u $len - i - 1$ za $i = 0, 1, ... len - 1$ i tako pravi tabelu vrednosti koja u suštini predstavlja broj mesta koji razdvaja ispitivani karakter od poslednjeg karaktera uzorka. U slučaju da u uzorku ima istih znakova, vrednosti polja pod tim kodom biće jednaka vrednosti

poslednjeg takvog znaka. Ako u kodu pod promenljivom *position* podrazumevamo mesto u tekstu odakle želimo da se počne sa pretraživanjem, onda je promenljiva *pos* još na početku setovana na mesto za *len* − 1 pomereno od mesta početka pretraživanja. To znači da se promenljiva *pos* sadrži brojnu vrednost pozicije poslednjeg karaktera uzorka ako bi se on nalazio na samom požetku teksta.

Kada se pojavi slaganje, algoritam koristi informaciju gde se karakter iz teksta pojavio u uzorku (ako se uopšte pojavljuje) da bi promenljivoj shift dodelio novu vrednost. U najboljem slučaju do neslaganja dolazi već na početku $P\,[pos] \neq T\,[pos]$ tako da nepostoje zajednički znakovi. U ovom slučaju mi možemo uvećati pos za vrednost len, sve doj nema preklapanja i dok je vrednost pos manja od veličine teksta. Ako se takva situacija ponavlja uzastopno, algoritam ispituje samo deo od 1/*len* svih karaktera iz teksta, i sve dok postoji neslaganje uvećavamo vrednost pos za veličinu uzorka. Ovo ponavljanje kod najboljeg služaja ilustruje snagu sistema pretraživanja sa desne ka levoj strani.

Pretpostavimo da smo upravo naišli na ispitivanje za neko *pos*, gde je $0 \leq pos < limit$. U tom slučaju postavljamo da shift bude indeks u opsegu $0 < shift \leq len$ u zavisnosti od sledeća tri slučaja:

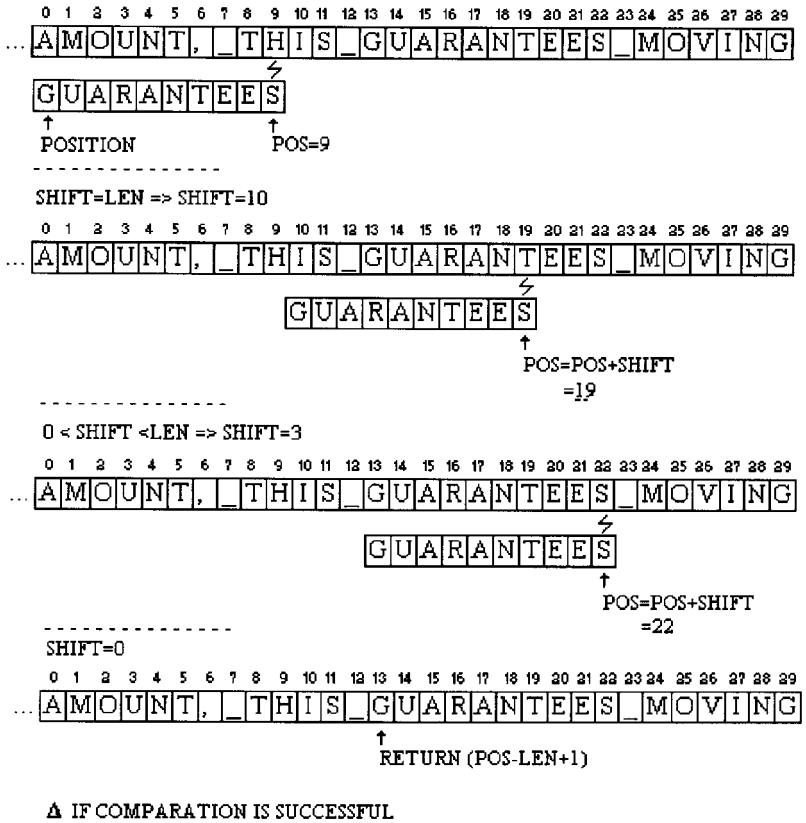
- $shift = len$: nije došlo do slaganja $P\,[pos] \neq T\,[pos]$ na toj poziciji, tako da *pos* možemo sa sigunrošću uvećati za veličinu uzorka bez ikakvog gubitka.
- $0 < shift < len$: došlo je do preklapanja $P\,[pos] = T\,[pos]$ sa nekim od prvih *len* − 1 karaktera, međutim kako uvek nastojimo da dođe do preklapanja zadnjeg karaktera sa istim u tekstu, onda moramo shift uvećati za vrednost koja predstavlja razliku dužine uzorka i broja mesta u uzorku na kom se našao karakter, koji se poklopio sa adekvatnim u tekstu. Ta vrednost se nalazi u vektoru table na poziciji čiji kod odgovara kodu ispitivanog karaktera tj. *table* [*P* [*pos*]].
- $shift = 0$: došlo je do preklapanja poslednjeg karaktera uzorka sa karakterom iz teksta, pa se shift setuje na 0 što predstavlja vrednost $shift = table\,[P\,[pos]\,]$ i dalje shodno tome pokreće ispitivanje u šesnestoj liniji koda. Funkcija CmpNTwoStr vrši komparaciju dva stringa za zadati broj karaktera, i predstavlja funkciju prisutnu u standardnim bibliotekama nekih viših programskih jezika. Tom funkcijom u sedamnestoj liniji vršimo upoređivanje uzorka i dela teksta koji počinje od pozicije pos-len+1 što smatramo za početak reči za koju pretpostavljamo da odgovara uzorku. Ako je pretpostavka tačna, tj. funkcija vrati nulu kao znak da je komparacija uspela, algoritam vraća naznačenu poziciju kao mesto početka pronađenog uzorka, inače inkrementira promenljivu pos i nastavlja traženje uzorka u tekstu sve dok pos ne postane jednaka dužini teksta kada se vraća vrednost (−1) kao znak da uzorak nije nađen u tekstu.

Da se primetiti da se skokovi vrše samo u desno (kretanje algoritma je s leva na desno) i nema vraćanja na već pređene karaktere, te da sa povećanjem dužine uzorka raste i veličina skoka. To znači da algoritam uvek radi progresivno.

Ilustracija algoritma:

pattern: guarantees
text: amount,_this_ guarantees_moving

	code	value
G	71	9
U	85	8
R	82	6
A	65	5
N	78	4
T	84	3
E	69	1
S	83	0



3.0 Realizacija

U pseudokodu je predstavljen algoritam u formi najpogodnijoj za analizu ali specifičnosti jezika koji se koristi pri realizaciji zahtevaju izmene i dopune pseudokoda. One će biti predstavljene u ovom delu.

Funkcije koje se sreću u programu su pisane u C-u i prema nameni se mogu podeliti na:

– priprema unetog stringa
  void trasstr (*FILE *stream*);
– osnovno pretraživanje
  void init_search (*const char *string*);
  int strsearch (*const char *string,size_t retpos*);
– ispitivanje okoline
  int rightword (*FILE *fp, char *buffer, char *rdodbuff, long pretpopoz, int priswit*);
  int leftword (*char *buffer, char *ldodbuff, int pronrec, long pretpopoz, int priswit*);

U pseudokodu date su funkcije čiji su ekvivalenti u C-u:

Pseudocode (user defined)	equivalent in C
Find_Right_Words	rightword
Find_Left_Words	leftword
Init_Search	init_Search
Str_Search	strsearch
Pseudocode (standard)	–
CpyStrToStr	strcpy
ReverseStr	strrev
CmpTwoStr	strcmp
CmpNTwoStr	strncmp.

3.1 Priprema unetog stringa

Priprema unetog stringa vrši se funkcijom čija deklaracija glasi:

void trasstr (*FILE *stream*)

Potrebno je uneti string razdvojiti na reči koje se svrstavaju u dvodimenzionalnu matricu te, gde prva dimenzija predstavlja broj reči a druga broj slova. Broj slova se izdvaja u posebnu matricu maxw. Prilikom unosa znakova u matricu sem o blanko znakovima ovog puta vodi se računa i o znaku novog reda pošto se unos vrši iz komandne linije dok se u pseudokodu pošlo od već zadatog niza.

Jednačina: celobrojni deo od $konbr = (brslova/100) \cdot KOEF + 0.5$ određuje koji broj znakova predstavlja uneti koeficijent i posle kojih se reč terminiše nulom.

Argumenti: pointer na fajl koji sadrzi rezultate. U fajl sa izveštajem se beleži string koji je unet od strane korisnika.

Povratne vrednosti: nema

3.2 Osnovno pretraživanje

Boyer-Moor-Praat algoritam je realizovan u obliku dve funkcije:
void init_search (*const char *string*)
init strsearch (*const char *string, size_t retpos*)

U praksi, programi za pretraživanje se najčešće projektuju tako da pri nailasku na odgovarajući string obaveštavaju korisnika o njegovom postojanju (markiranjem, ispisivanjem stinga i slično) i prestaju sa daljim pretraživanjem. Ovde to nije slučaju jer nam je cilj da prikažemo sve oblike unetog stringa u skladu sa parametrima (pogledati paralelu sa pretraživanjem sa džoker znacima u zaključku).

Funkcijom *init_search* inicijalizujemo tabelu vrednosti što je potrebno uraditi samo jednom sobzirom da se uneti string više neće menjati. Od celog unetog stringa funkciji se predaje samo sadržaj nultog polja matrice *R* pošto su reči u stringu već prethodno gradirane po dužini. Tako da će se u tekstu tražiti samo najduža reč iz zadatog stringa.

Strsearch funkcija kao svoje argumente uzima bafer za string koji će pretraživati i poziciju odakle se počinje sa traženjem. Još prilikom učitavanja, dužina datoteke je podeljena sa brojem 32768 (32 KB) i tako dobijena cifra fflen koja predstavlja broj delova na koje je rastavljena datoteka. Proces pretraživanja u osnovi se izvodi fflen puta. Međutim pošto se u okviru jednog bafera može naći više odgovarajućih stringova, proces traženja se i dalje nastavlja dogod se ne dođe do kraja bafera koji se zatim prazni a potom puni novom porcijom i tako fflen puta. Pre same alokacije bafera modifikuje se njegova veličina u slučaju da je datoteka ili poslednja porcija manja od 32 KB.

Drugi parametar predstavlja mesto u baferu odakle se počinje sa traženjem. Nakon učitavanja sadržaja u bafer ta vrednost je jednaka nuli (samom početku bafera) a potom za jedan većem broju od vrednosti pozicije koja je vraćena pri prošlom traženju jer se odatle nastavlja dalje sa pretragom. Funkcija vraća vrednost (−1) ako string uopšte nije nađen što prekida dalje traženje u datom baferu.

3.3 Ispitivanje okoline

Pretraživanje udesno

Traženje ostalih reči iz zadatog stringa nakon nalaženja najduže reči obavlja se funkcijom čija deklaracija glasi:

int rightword (*FILE *fp, char *buffer, char *rdodbuff, long pretpopoz, int priswit*)

Pošto se prilikom razmatranja reči nađenih sa desne strane, menjaju kontrolna polja reči zadatog stringa, potrebno je iste setovati na vrednost (−1) svaki put kada se funkcija pozove. Funkcije strsearch i rightword će se pozivati

$$\sum_{k=1}^{fflen} p_k = p_1 + p_2 + \ldots p_{fflen}$$

gde je *fflen* broj punjenja glavnog bafera, *p* koliko puta je nađena najveća reč zadatog stringa u glavnom baferu.

Međutim, tu se javlja problem pri učitavanju znakova iz bafera kada pozicija odakle se učitava znak postane jednaka veličini bafera. Kako onda nastaviti učitavanje sledećeg znaka ? To se postiže tako što se pri nailasku na poziciju kraja bafera algocira prostor veličine (*IZUZ* + *MAXW*) ·*MAXC* gde je *IZUZ* broj dozvoljenih izuzetaka, *MAXW* maksimalan broj reči a *MAXC* maksimalan broj slova, u dodatni rdodbuff bafer. Potom se pozicija postavlja na nulu i nastavlja sa učitavanjem znakova. Ako je datoteka ili porcija u baferu manja od 32 KB, onda se alokacija dodatnog bafera ne vrši. Dodatna alokacija se ne vrši i ako pozicija učitavanja uopšte i ne dostigne kraj bafera. Inače u ostatku funkcije se radi sa trbuff nizom koji sadrži adresu glavnog ili sporednog bafera zavisno od situacije jer praktično ne postoji razlika u njihovom korišćenju.

Početna pozicija je jedan od argumenata funkcije. To je broj koji vraća funkcija strsearch iz seta funkcija za osnovno pretrazivanje. Daljim kauntovanjem se uvećava njena vrednost. Drugi argument predstavlja priswit vrednosti 1 ili 0 u zavisnosti da li je upotrebljen /f parametar ili ne.

Funkcija kao vrednost vraća broj pronađenih reči sa desne strane. Promenljiva rizuz koja sadrži broj nađenih izuzetaka je globalnog tipa kao i niz u koji se upisuju nađene reči, pošto će se koristiti i u drugim funkcijama.

Pretraživanje ulevo

Traženje preostalih reči iz zadatog stringa posle ispitivanja rightword funkcijom obavlja se funkcijom čija deklaracija glasi:

int leftword (*char *buffer, char *ldodbuff, int pronrec, long pretpopoz, int priswit*)

Funkcija će se pozivati $\sum_{k=1}^{fflen} p_k - i, \quad i \in \{0, 1\}$

puta, gde prvi član predstavlja koliko je puta funkcija rightword pozivana a i koliko puta je vrednost pozicije vraćene od strane funkcije strsearch jednaka nuli tj. koliko se puta najveća reč zadatog stringa našla na nultoj poziciji pri prvom punjenju bafera. Jasno je da vrednost i može biti samo 0 ili 1.

I ovde se javlja problem pri učitavanju znakova iz bafera, kada pozicija sa koje se učitava postane jednaka nuli a i dalje se teži učitavanju znakova ovog puta iz prethodne porcije bafera. Prilikom novog punjenja bafera stari sadržaj se gubi, dakle pre novog punjenja bafera potrebno je izdvojiti maksimalno 32766 − (*IZUZ* + *MAXW* − 1) *MAXC* znakova. Zbir *IZUZ* + *MAXW* predstavlja makimalan broj reči a oduzima mu se 1 stoga što ta jedna reč predstavlja najveću reč zadatog stringa i vrednost njene pozicije je uvek veća ili jednaka nuli. *Ldodbuff*, jedan od parametara funkcije, upravo predstavlja spremište za taj broj znakova, parametar je stoga što je kao i glavni bafer definisan van same funkcije. U funkciji je definisan niz koji sadrži adresu glavnog ili sporednog bafera zavisno od potreba. Kao parametar funkcija uzima *pronrec*, vrednost koja predstavlja broj dotada nađenih reči tj. vrednost vraćenu funkcijom rightword.

Funkcija kao vrednost vraća broj pronađenih reči sa leve strane. Promenljiva *lizuz* koja sadrži broj nađenih izuzetaka je globalnog tipa kao i niz u koji se upisuju nađene reči, pošto će se koristiti i u drugim funkcijama.

Prilikom analize rightword i leftword funkcija, u pseudokodu su naznačene promenljive *prek*1 i *prek*2 i pritom je objašnjena njihova prekidačka funkcija. U C-u, jeziku u kom je vršena implementacija, nema potrebe za tim jer postoji kao naredba rezervisana reč break. Ona služi za momentalni izlazak iz petlje i predstavlja adekvatnu zamenu za date promenljive štedeći pri tome vreme potrebno za inicijalizaciju i pozivanje.

3.4 Problemi pri realizaciji

Standardne funkcije koje se koriste su gotovo po pravilu rađene tako da se međusobno dopunjuju, te prave lanac gotov za korišćenje od strane programera. Taj lanac bi po pravilu trebao da radi bez greške. Jasno je da ako se u nekoj od tih funkcija pojave greške takvog nivoa da je tu funkciju nemoguće primeniti za dati problem, javlja se značajna poteškoća napraviti

spregu između ostalih funkcija lanca, zaobići problematičnu, i kao takve ih koristiti u programu.

Problem se pojavio kod rada sa funkcijom fseek kako u TopSpidu tako i u Borlandovom paketu. Naime, ta funkcija služi za pomeranje pointera u fajlu za zadati broj karaktera. Prilikom nailaska na veći broj novih linija fseek odkauntuje više nego što bi trebalo. Greške je teško lokalizovati tako da se ne može garantovati da samo tu pravi problem. Pošto se program intenzivno oslanja na tekst fajl ali pri tome radi i sa baferom dolazilo je do toga da se pozicija koja je vraćena od strane funkcije strsearch u baferu ne poklapa sa pozicijom u fajlu. Samim tim pošlo se od toga da je greška u kodu, ali intenzivnim ispitivanjen funkcije fseek na većem broju datoteka i još dve verzije Boyer-Moor ovog algoritma dalo se ustanoviti da je greška ipak do funkcije fseek. Datoteke su pregledane i heksadecimalno pa se prilikom izlistavanja delova direktno iz datoteke i iz bafera jasno videlo da je fseek davao pogrešne rezultate. Tako da je sve rađeno preko bafera i funkcije fread koja nije pokazala problematično ponašanje. Iz tog razloga se u funkciji rightword koristi dodatni bafer kako bi se izbeglo direktno čitanje iz datoteke. Kod funkcije leftword je mesto bafera upotrebljen statički deklarisan niz iz tog razloga sto bi se učitavanje znakova u bafer vršilo za zadato mesto iz datoteke jer se pozicioniranje pointera može vršiti samo fseek funkcijom tako da nam to ne bi garantovalo da će znakovi odatle biti i pročitani.

3.5 Zaključak

ViewIt algoritam kakav je ovde prikazan predstavlja „the engine" i nedostaje mu samo korisnički interfejs do prave aplikacije. Zaista, ovde postoji unos parametara iz komandne linije ali bi funkcija došla do izražaja tek u okviru nekakavog editora ili tekst procesora gde bi rezultati bili vidljivi na samom tekstu.

Ideja o koeficijentima nije potpuno nova. Kada se pogleda pretraživanje sa džoker znacima vidi se da za zadanu reč, broj slova pre džoker znaka predstavlja nekakav koeficijent istovetnosti. Novi momenat u pretraživanju predstavlja definisanje izuzetaka kao i činjenica da se prihvata proizvoljan raspored reči u rečenici.

To daje veliku paletu tolerancija, od minimalne (/k100 /i0 /f) pa do maksimalne (/k1 /i10). Takav mehanizam razdvaja pojedine faktore pretrage dajući mogućnost da pojedinačno postavimo prag istovetnosti (istovetnost je obrnuto srazmerna toleranciji), što i jeste cilj pravljenja algoritma.

Literatura

[1] Cormen T. H., Leiserson C. E., Rivest R. L. 198*. *Introdustion to algorithms*. Bostin: MIT Press

[2] Karnighan B. W., Ritchie D. M. 1990. *The C programming language*. Prentice Hall

[3] Knuth D. E. 1991. *Sorting and Searching, volume 3 of The Art of Computer Programming*. Addison-Wesly

[4] Feibel W. 1982. *Using ANSI C in UNIX*. McGraw Hill

Marija Bogićević

ViewIt algorithm

ViewIt algorithm belong to group of searching algorithms. During the using standard forms we could see that there is no possibilities to choose between potential solutions, (strings that vary from input string depends of parameter) because only identical form is in use. ViewIt algorithm result from needing to create flexible system.