Marija Bogićević

ParViewit za Unix

Konvencionalni algoritmi za pretraživanje imaju mogućnost pronalaženja samo teksta identičnog onom koji je zadao korisnik, dok sve varijacije u rečima ili njihovom redosledu ostaju nepronađene. Problem se delimično može ublažiti uvođenjem džoker znakova u obrazac za pretraživanje, ali takvo rešenje ne dozvoljava toleranciju u slučaju promene redosleda reči. Predloženo rešenje omogućava fleksibilno pretraživanje, tako da je moguće pronaći tekst koji sadrži promene i u obliku reči i u njihovom redosledu, pa čak i tekst u kome se pojavljuju proizvoljne reči, ali jedino u sklopu zadatog obrasca pretraživanja. ParViewit je rezultat nadgradnje ViewIt algoritma (istog autora).

Ključne reči: pretraživanje, paralelno procesiranje

1. Uvod

U ParViewit algoritmu reč se koristi kao osnovni oblik stringa koji se može uneti. Polazi se od pretpostavke da reč koja se traži može imati izvesne izmene ali takve da ih korisnik prihvati kao pogodak. Izmene se odnose na sam kraj reči.

Ako imamo reč softver i prihvatimo izmene krajnjih slova dobićemo: softvera, softverskog, Microsoft i sl. Ovde se da primetiti da su od polazne reči samo prva četiri slova identična sa navedenim izvedbama. Ako taj procenat istovetnosti postavimo kao konfigurabilan parametar onda sa njegovim povećanjem smanjujemo broj prihvatljivih varijacija, sve do momenta gde zahtevamo potpunu istovetnost. Procenat istovetnosti se računa od početka unete reči jer korisnik zna koju reč želi da upotrebi te koeficijentom zadaje posle koliko slova želi da dopusti promenu oblika, s tim što u nađenoj reči posle tog procenta ostatak može biti proizvoljne dužine, u obliku prefiksa i/ili sufiksa zavisno sa koje strane smo definisali poklapanje. 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.

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

Prilikom unosa parametara možemo definisati i broj reči koje dopuštamo 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 se zadaje broj izuzetaka ali tako da u ispitivanom stringu u tekstu broj nađenih izuzetaka može biti manji ili jednak zadatom limitu.

2. Implementaciona tehnologija

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 UNIX operativnim sistemom.

Međutim, program je formiran kao simulacija ponašanja algoritma u uslovima rada na višeprocesorskoj mašini. Kako svaki model realne paralelne mašine koji zadovoljava jednu od tri klase iz skupa klasa Flynnove podele, podrazumeva drugačije strukturne karakteristike, to se trebalo ograditi na teorijski model: *parallel random-access machine* odnosno PRAM model. Takođe, PRAM ignoriše i razne druge aspekte građe realne mašine, jer se time postiže da realizacija algoritma ne zavisi od modela paralelne mašine.

Arhitektura PRAM-a se zasniva na p procesora P0, P1 ... Pp-1, koji dele globalnu memoriju i pristupaju joj paralelno (dakle, u isto vreme). Procesori su takođe u stanju da paralelno obavljaju i razne aritmetičke i logičke operacije. Po tipu pristupa memoriji koristi se *combining-CRCW* model: upisuje se najveća od svih vrednosti koje procesori pokušavaju da upišu. Iz tog razloga se u ParViewit-u svaka promenljiva kojoj pristupa više procesa nalazi u okviru deljene memorije, zajedničke za sve procese te aplikacije.

3. Razvojno okruženje

Originalni dizajn UNIX sistema podrazumeva jednoprocesorku okolinu. Višeprocesorska arhitektura pod UNIX-om je omogućila i to da se jedan proces može konkurentno izvršavati na više procesora. Svaki procesor je nezavisan, ali svi oni izvršavaju svoju kopiju kernela. Procesi se ponašaju baš kao i na jednoprocesorskoj mašini, semantika za svaki sistemski poziv ostaje ista, s tim što mogu transparentno migrirati između procesora [2]. Takvo ponašanje može dovesti do problema u integritetu sistema, stoga se postavljaju različiti vidovi zaštite kao što je upotreba semafora za selektivno korišćenje resursa.

Pri projektovanju ovakvih sistema teži se da budu i *speed up-oriented* odnosno da se aplikacija deli na skup zavisnih procesa tako da ti procesi međusobno interaguju dok se izvršavaju konkurentno na više procesora.

Međutim, moguće je i eksplicitno dodeliti proces jednom procesoru. *Fork* sistemski poziv to omogućuje.

Kod *loosly coupled* sistema tipa *sattelite systems* (grupa mašina koja je povezana sa jednom centralnom mašinom. Za razliku od ostalih distribuiranih sistema, satelitski procesori nemaju autonomiju sem u retkim slučajevima), kada proces na centralnom procesoru izvrši *fork* poziv, kernel selektuje satelit procesor da izvrši proces i šalje poruku posebnom server procesu na satelit procesoru, dajući mu informacije kako da prihvati proces koji treba da izvrši. Govoreći o PRAM-u kao o *shared memory* modelu podrazumevali smo *tightly coupled* višeporcesorski sistem. 4.2 BSD Unix nudi rsh komandu kao i Sun-ov Unix komandu rex , sa *remote invocation* sistemom koji predstavlja mogućnost da se inicijalizuje novi proces na drugoj mašini, ali ne i sposobnost da se to uradi ako je proces već pokrenut [4].

Sličan sitem samo predstavljen u obliku mogućnosti migracije procesa sa jedne na drugu mašinu koja je u *idle* stanju obezbeđen je putem kernel poziva Proc_Migrate na Sprite Network operativnom sistemu Unix like koncepcije [4].

U ParViewit-u fork je prihvaćen upravo kao naznaka da izdvojeno parče koda treba izvršavati kao proces na zasebnom porcesoru, što spada (pod drugim nazivima) u obe klase sistema.

4. Pretraživanje

Osnovno pretraživanje: možemo da formalizujemo tako što ćemo tekst da predstavimo kao polje *Text* 0...*n*-1 veličine *n* a uzorak koji se traži kao polje *Pattern* 0...*m*-1 veličine *m*. To znači da su polja *Text* i *Pattern* stringovi čiji karakteri pripadaju konačnom alfabetu.

Kažemo da se uzorak *Pattern* pojavljuje u stringu *Text* na poziciji *s* (tj. uzorak počinje od s-tog karaktera stringa *Text*) ako 0 s *n-m* i *Text* s+0 ... *s+m*-1 = *Pattern* 0... m-1 tj. *Text s+j =Pattern j* za 0 *j* *m*-1. Svrha pretraživanja je nalaženje svih pozicija na kojima se uzorak nalazi u tekstu. String je definisan tako da između reči moraju stajati blanko znaci da bi reči kao takve postojale.

Podela algoritma: se može izvršiti po funkcionalnim celinama te ih tako i možemo ispitivati.

Pronalaženje najduže reči: (zvaćemo je ključnom) unetog stringa svodi se na upotrebu Boyer-Moor-Pratt algoritma za traženje fiksnog stringa u tekstu i dat je u obliku funkcija u algoritmu. Boyer-Moor-Praat

(BMP) algoritam je ovde iskorišćen u potpunosti pa 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, ispitati da li je karakter neposredno ispred pozicije vraćene BMP algoritmom blanko znak tj. znak novog reda ako je upoređivanje reči definisano kao dozvoljavanje sufiksa, ili neki drugi znak za prefiksnu formu. Ako je string nađen na samom početku teksta onda takvi znaci i ne mogu postojati pa se i ovakav slučaj prihvata ali se pri tom izuzima pretraživanje sa leve strane od ključne reči. U sistem pretrage ubačen je i novi pojam. U pitanju je pojam izuzetaka. Pogledajmo u kom odnosu mogu biti izuzeci i reči iz teksta.

Ako nađeni string obeležimo kao *Find* i posmatramo ga kao skup čiji su elementi reči uzorka (dakle, unetog stringa *Pattern*) i reči izuzeci, onda se može napisati *Pattern* *Find* pošto reči uzorka pripadaju nađenom stringu a da pritome mogu biti u proizvoljnom rasporedu. Ako sada *Find* posmatramo ne kao skup već kao konačan string, prilikom definisanja broja izuzetaka većih od 0, to svojstvo ostaje s tim što *y* *Find* i *y* 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: predstavlja traženje preostalih reči uzorka nakon nalaženja ključne reči gde vrednost pozicije odakle se učitavaju znaci teži ka uvećanju.

Celokupan rad funkcije se obavlja u okviru petlje sve dok se ne ispuni uslova da je broj nađenih reči u osnovi jednak broju zadatih reči ili se dođe do samog kraja stringa *Text* u kome se vrši pretraga. Ako se naišlo na nepoznatu reč prekinuće se dalji proces ako je broj izuzetaka definisan od strane korisnika jednak nuli ili ako je broj pronađenih izuzetaka jednak tom unetom broju izuzetaka. Reči se učitavaju direktno iz stringa *Text* u zasebnu matricu na čijem se kraju vrši setovanje kontrolnog (zadnjeg) polja na (-1).

Kontrolna polja mogu imati vrednosti:

value	purpuse
-1	exeption or not yet checked
-2	word
-3	word copy

- kontrolna polja za reči nadjenog stringa

value	purpuse
-1	exeption
0, 1 ... n	nb. of similar word

- kontrolna polja za reči traženog stringa

Kontrolna polja traženih stringova (reči uzorka) su setovana na (-1) jer još nisu nađeni odgovarajući stringovi u tekstu. Od pozicije dobijene BMP algoritmom 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 ne znamo da li će se poklopiti bar sa jednom 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 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 kažemo da su se poklopila ako zadovoljavaju relaciju: R Q R S => Q S gde bi R predstavljao traženi string 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 Righttext k će biti setovano na (-2) a kontrolno polje Pattern m na vrednost *k* (*k* je pozicija reči u okviru matrice Righttext a *m* pozicija reči u okviru matrice Pattern).

Pošto je učitan string iz teksta potrebno je izvršiti njegovo upoređivanje sa svakim stringom iz matrice Pattern čije kontrolno polje ima vrednost (-1) to jest koji nije dosad nađen. U slučaju da odgovara, njeno kontrolno polje je setovano na vrednost (-2), čime se ta reč označava kao nađena.

To znači da će sledeći učitan string biti upoređivan sa jednom reči manje. Može se desiti da string ne zadovolji relaciju. Za broj izuzetaka jednak nuli, takav string bi izazvao prekidanje daljeg traženja i izlazak iz funkcije, gde bi povratna vrednost bila jednaka broju nađenih reči pre nailaska na izuzetak.

Ispitivanje se ponavlja ponavlja sve dok je broj pronađenih reči manji od broja zadatih, što predstavlja broj reči uzorka. Međutim, ako se u nađenom stringu nailazi i na izuzetke (dopušten broj izuzetaka veći od nule) pretraživanje treba nastaviti. Tako se granica ispitivanja pomera za jedan kad god se ne naiđe na dopušteni izuzetak.

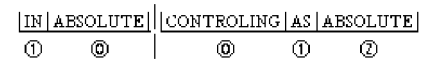
Pretraživanje ulevo: predstavlja traženje preostalih reči uzorka nakon nalaženja ključne reči i prolaska kroz funkciju pretračivanja udesno gde vrednost pozicije odakle se učitavaju znaci teži ka umanjenju.

Već smo uočili kako se dolazi do ključne reči i šta predstavlja njenu poziciju. Funkcija dobija upravo tu poziciju kao jedan od argumenata ali korigovanu za broj znakova ispred ako je pretračivanje zadato kao sufiksni oblik traženja ili istu ako je definisan kao prefiksni oblik (tada se ne dopušta postojanje drugog sem blanko ili znaka novog reda ispred te pozicije). Do bočnog traženje neće doći ukoliko ključni string ne zadovoljava

dati kriterijum. Reči nađene sa leve strane ključne reči učitavaju se u matricu. Sam način unošenja reči je identičan sa unošenjem u funkciji pretraživanja sa desne strane i već je razmatran. Kako se učitavanje vrši od poslednjeg karaktera stringa to ga je potrebno invertovati i setovati kontrolna polja na vrednost (-1).

Ovde je vrednost (-3) preuzela ulogu vrednosti (-2) iz funkcije za pretraživanje sa desne strane, to jest, ulogu odgovarajuće reči dok je (-1) i dalje vrednost za izuzetke. Njihov uticaj ima mnogo dublju pozadinu i biće odvojeno razmatrani u delu rasprave o modifikaciji prilikom dupliranja stringa. Izuzeci su i ovde tretirani na sličan način.

Modifikacije pri dupliranju stringa: Postavimo da traženi tekst bude: "controling in apsolute".



Algoritam se standardno ponaša setujući kontrolna polja na vrednost (-2) za reči i (-1) za izuzetke. Počinjući pretraživanjem sa desne strane, nailazimo na dve regularne reči i jedan izuzetak (za broj izuzetaka različit od nule), dakle ostalo je da se pronađe još samo jedna reč iz unetog stringa pa da pretraga bude uspešna. Ipak, sa leve strane nalazi se po drugi put reč "absolute" koju moramo da prihvatimo da bismo ispunili uslov o nalaženju svih reči zadatog stringa. Da li ćemo te reči prihvatiti kao reči ili izuzetke je sve jedno, bitno je da se nađu u konačnom stringu. Ta sporna reč (u slučaju da je broj izuzetaka ograničen na jedan, inače bismo je prihvatili kao izuzetak te ne bi bilo potrebe za daljim izmenama) je kao nađena sa desne strane za kontrolno polje imala vrednost (-2), kao ponovo nađena, tom polju, kao i polju u zadatom stringu za tu reč se menja vrednost u (-3), kako se ne bi ponovo kasnije ispitivala, te je potrebno sa desne strane odstraniti zajedno sa izuzecima sve do prve sledeće regularno prihvaćene reči.

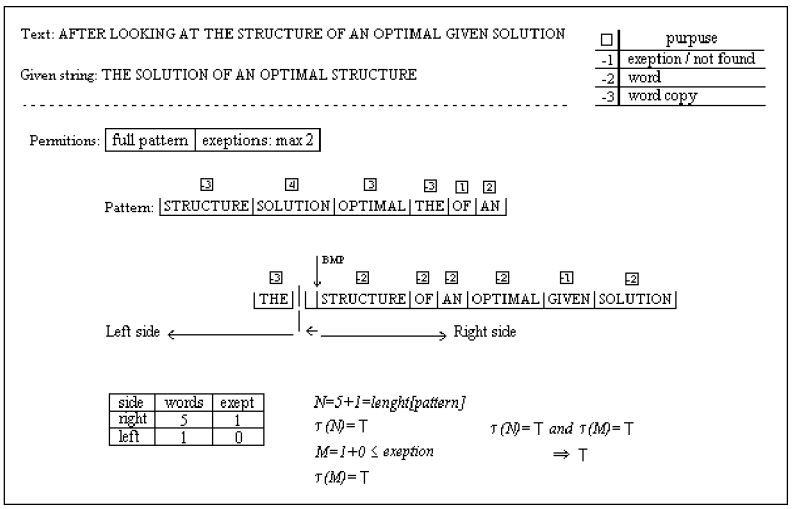
Dat je i primer pretraživanja koji treba da pokaže kako se ParViewit algoritam ponaša u praksi. Zadat je string čije se reči u tekstu nalaze u obliku kakvom su zadate ali nešto drugačijeg rasporeda. Dat je stoga što pokazuje da je raspored reči pri unosu za fiksan string, sasvim nebitan jer ga algoritam uvek preuređuje na isti način.

Paralelno pretraživanje: sa leve i pretraživanje sa desne strane od ključne reči mogu se razmatrati kao dva nezavisna procesa.

U osnovi, algoritam se izvršavao tako što se počne prvo sa pretraživanjem sa desne strane gde se iz daljeg toka traženja izbacuju reči iz zadatog stringa koje su pronađene, te je nakon toga pretraživanje sa leve strane vršeno sa preostalim rečima iz zadatog stringa. Reči koje su u

suštini bile kopije već pronađene reči, nisu bile drugačije detektovane nego kao izuzeci jer je odgovarajuća reč iz *pattern*-a bila označena tako da se zanemarivala u daljoj pretrazi čim bi jednom našla svoje mesto u ispitivanom stringu. Sistem modifikacije pri dupliranju uneo je promenu u setovanju kontrolnih polja reči.

Ako se reč pojavila negde u stringu a kopija se nalazi sa još par izuzetaka na samom kraju sa desne stane i predstavlja desnu barijeru, upravo taj broj izuzetaka može dovesti do odbacivanja celog stringa jer je korisnik dopustio manji broj, dok bi se upotrebom modifikacije, kopija i izuzeci (sve do prve regularno nađen reči) izbacili i string bi bio prihvaćen. Bitno nam je da se tražena reč pojavi u stringu koji prosleđujemo kao nađeni i da njene kopije eliminišemo kad god je to moguće (dakle, kad predstavlja zadnju, graničnu, reč).



Primer 1. Strelice pokazuju smer kretanja kroz tekst prilikom učitavanja karaktera. Kratka promena smera pri ispitivanju sa desne strane predstavlja ispitivanje prvog karaktera ispred pozicije vraćene od strane BMP algoritma. Kontrolno polje ključne reči u *pattern*-u je setovano na (-2) jer se ne sme dozvoliti da ako postoji takva reč i sa leve strane, ključna zbog toga izostavi (u skladu sa modifikacijom pri dupliranju reči), jer smo u prošlom procesu pretraživanja tu reč već jednom razmatrali kao ključnu. Sad je možemo prihvatiti samo kao izuzetak, što važi i za reč the pošto je nađena samo sa leve strane (da je nađena i sa desne, kontrolno polje kopije sa desne strane bi isto bilo setovano na (-3) i time se indicira da tu kopiju treba izostaviti ako je zadnja reč sa desne strane, odnosno razmatrati je kao izuzetak ako nije).

Ovaj sistem je omogućio da se uz neke minimalne izmene ova pretraživanja vrše u više procesa (pod fork-om) bez gubljenja validnosti rezul-

tata. Sada sistem mora da odreaguje pravilno jer se procesi mogu izvršavati po, za nas, sasvim slučajnom redosledu, dakle, sa pretraživanjem se može početi i sa leve strane od nađene reči. Mi smo u prvoj postavci algoritma na levoj strani nalazili reči čije smo kopije označavali na desnoj strani, a sada za svaku stranu (jer ne možemo znati raspored izvršavanja) možemo detektovati kopije u odnosu na suprotnu.

Izmene u sistemu se svode na sposobnost algoritma da pravilno reaguje na odnos kopija za obe strane, te da na vrednosti (uvedene zarad sinhronizacije) nađene u kontrolnim poljima, odgovori zabranom da se neke reči dalje traže (pošto se sada pretraživanje vrši sa punim sastavom reči iz *pattern*-a, za oba smera pretrage).

5. Analiza

Analiza ponekad predstavlja ispitivanje zahteva algoritma za određenim resursima. Katkada, resursi kao što su memorija, veličina programa i slično su od primarnog značaja ali ono što u našem slučaju želimo da merimo je vreme. To je i proces koji određuje koliko je algoritam brz i koliko efikasan u iskorišćavanju postojećih resursa.

I prilikom ispitivanja paralelnih algoritama je bitna prethodna analiza istog algoritam ali u sekvencijalnom obliku, pošto rezultujuća verzija predstavlja ustvari implementaciju za neku realnu višeprocesorsku mašinu.

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 neku konkretnu mašinu.

Kod višeprocesorske mašine razlikujemo dve vrste koraka:

- programski korak koji predstavlja aritmetičku ili logičku operaciju koju izvrši procesor nad određenim podacima

- korak rutiranja gde se podatak kreće od jednog procesora do drugog preko deljene memorije ili preko mreže (dobar primer je *broadcast,* proces razašiljanja jednog podatka svim procesorima, i zahteva *log N* paralelnih koraka za N procesora) .

U ParViewit-u zanemarujemo postojanje koraka rutiranja, taj problem je rešen deljenom memorijom gde svaki proces može pristupati zajedničkim promenljivim, te proces razašiljanja podataka nije potreban.

Određeno vreme je potrebno za izvršenje svake linije koda. Jedna linija može zahtevati duže izvršavanje od neke druge, ali pretpostavimo da svako izvršenje *i*-te linije zahteva vreme *c*, gde je *c* konstanto (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). Jedan od zahteva je da sprecifikacija algoritma mora da sadrži i precizan opis funkcija koje se ispituju.

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.

Pošto se ponašanje algorima menja u odnosu na moguće unose, moramo kompletno to ponašanje svesti na jednostavnu i lako razumljivu formulu.

Boyer-Moor-Praat (BMP) algoritam: je u celosti upotrebljen za traženje ključne reči i pretstavlja bitan faktor u brzini ParViewit algoritma.

Ako je uzorak P relativno dugačak a alfabet pristojne veličine, algoritam čiji su tvorci Robert S. Boyer, J. Strother Moor i M. Praat, može se predstaviti kao jedan od najefikasnijih algoritama za pretraživanje sa zadatim stringom. Ovde je iskorišćena verzija data u pseudokodu koja je modifikovana za potrebe ParViewit algoritma. BMP algoritam koristi manje od *m+n* koraka (*n* - veličina traženog stringa, *m* - dužina teksta koji se pretražuje) za poređenja karaktera, i oko *n/m* koraka za ne previše dugačak podstring i dovoljno bogat alfabet.

Preciznije, *worst-case* vreme izvršavanja u asimptotskoj notaciji je:

$$O\left((m - n + 1) \cdot n + |\Sigma|\right)$$

Opaska o alfabetu proizilazi iz činjenice da ovaj algoritam, recimo, ne bi mnogo pomogao za binarne stringove, gde postoje samo dva karaktera.

Algoritam u obliku funkcije vraća vrednost (-1) ako string uopšte nije nađen što prekida dalje traženje.

Analiza ParViewit algoritma: može se predstaviti uz korišćenje T( ) notacije koja predstavlja *worst-case* vreme izvršavanja, gde su parametri celobrojnog tipa .

$$T(n, m, l, w, e) = \frac{m}{1024^2} \cdot \left(\frac{2 \cdot (W + 2) \cdot O(l)}{(w + e - 1)/(w + e)} + O(l^2) + 2 \cdot l \cdot w + O((m - n + 1) \cdot m + |\Sigma|) \right)$$

gde je: m – dužina teksta
l – broj reči teksta

n – dužina zadatog stringa
w – broj reči zadatog stringa
e – broj dozvoljenih izuzetaka
Σ – alfabet

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.

6. Zaključak

ParViewit algoritam je kreiran bez tendencije za pravljenjem korisničke aplikacije. Unos parametara je iz komandne linije premda bi funkcija algoritma došla do izražaja tek u okviru nekog editora ili tekst procesora gde bi rezultati bili vidljivi na samom tekstu. Ovakav oblik ParViewita je određen iz težnje da se stvori kod koji je sposoban da se sa minimalnim izmenama izvršava na raznim tipovima realnih višeprocesorskih mašina (otud generalizacija u obliku PRAM-a). Veliki deo istraživanja je usmeren ka izolovanju i rešavanju konkretnih problema paralelizacije koda.

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] Thomas, Cormen H. et al. 1991. *Introduction to algorithms*. MIT Press

[2] Maurice, Bach J. 1986. *The design of the Unix operating system*. Prentice-Hall

[3] Selim, Akl G. 1989. *Parallel algorithms*. Prentice-Hall

[4] John, Ousterhout K. et al. 1988. The Sprite Network Operating System. *IEEE Computer*, February 1988: 23-36

[5] Brian, Karnighan W., Dennis, Ritchie M. 1988. *The C programming language*. Prentice Hall

[6] Donald Knuth E. 1973. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesly
[7] Werner, Feibel. 1988. *Using ANSI C in Unix*. McGraw-Hill
[8] Bogićević Marija 1996. *Postavka i testiranje ViewIt algoritma, Petničke sveske 41, Valjevo ISP, str. 57-83*

Marija Bogićević

ParViewit for UNIX

Possibilities of conventional text searching algorithms are limited only to finding the text identical to pattern given by user, while all other variations in words or word order in sentence are undetected. Using wild-chars in search pattern can partially solve this problem, but problem with word order still remains unsolved. Proposed solution, described in this paper, provides more flexible text searching, including finding the text containing variations of word syntax and order, even the text with random words, but only within the given search form.

Key words: searching, parallel proccessing