KONSTRUKCIJA I ANALIZA ALGORITAMA II

Algoritmi za konstrukciju sufiksnog niza linearne složenosti

Miloš Milaković, 1052/2021
mi211052@alas.matf.bg.ac.rs

profesor: dr Vesna Marinković
asistent: Jelena Marković

Beograd 2024.

Sadržaj

1 Uvod

Sufiksni niz predstavlja strukturu podataka koja može na efikasan način da reši određene probleme koje se javljaju u oblasti informatike, poput problema pronalaženja određene reči u tekstu, traženje palindroma u tekstu, upoređivanje niski, poravnanje genoma u bioinformatici. Dugi niz godina, sufiksni niz je bila inferiornija struktura podataka u odnosu na sufiksno stablo jer nije postojao efikasan algoritam, u linearnoj složenosti, za konstrukciju ove strukture a da taj algoritam nije koristio sufiksna stabla. Ova situacija se menja 2003. godine kada se pojavljuje više algoritama koji u linearnoj složenosti mogu da konstruišu sufiksni niz bez upotrebe sufiksnih stabala. Sufiksni nizovi postaju struktura koja se sve više koristi i koja dobija prednost u odnosu na sufiksna stabla pre svega zbog manjeg korišćenja memorije.

U ovom radu, biće predstavljen algoritam ***DC3***, kao i algoritam ***SA-IS***, sa određenim primerima, rezultatima i kodom koji realizuje ove algoritme.

2 Definicije i pojmovi

Definicija: Sufiksni niz niske S je niz celih brojeva koji predstavljaju početne pozicije (indekse) sufiksa niske S koji su leksikografski sortirani.

Primer: Posmatrajmo nisku $S = abcabcacab$. Ovo je niska od 10 karaktera a sufiksni niz SA koji je određen ovom niskom je:

$$\mathbf{SA} = \{\mathbf{8}, \mathbf{0}, \mathbf{3}, \mathbf{6}, \mathbf{9}, \mathbf{1}, \mathbf{4}, \mathbf{7}, \mathbf{2}, \mathbf{5}\} \tag{1}$$

U tabeli 1 mogu da se vide svi sufiksi niske S leksikografski sortirani.

Indeks niske S	**Sufiks**
8	ab
0	abcabcacab
3	abcacab
6	acab
9	b
1	bcabcacab
4	bcacab
7	cab
2	cabcacab
5	cacab

Tabela 1: Tabela sufiksa niske S leksikografski sortirani i njihovih indeksa

Definicija: Rang elementa x u nizu S je broj elemenata u nizu S koji su manji od elementa x. U slučaju sufiksnog niza, to predstavlja poziciju na kojoj se zadati element nalazi u sufiksnom nizu.

3 Algoritam DC3

DC3 algoritam je rekurzivni algoritam koji u linearnom vremenu konstruiše sufiksni niz [1, 2, 3]. Algoritam koristi *radix* sortiranje kao efikasnu tehniku za poređenje niski. Ulazna niska je konvertovana u niz brojeva tako što se svaki karakter pretvori u broj i samim tim je olakšana primena *radix* sorta. Ovaj algoritam deli sufikse u dve grupe i sortira svaku grupu posebno. Time što su sufiksi podeljeni u dve grupe, algoritam koristi jednu grupu da sortira drugu grupu i izbegava ponavljanje poređenja. Na kraju, ove dve grupe se spajaju i konstruiše se sufiksni niz.

U situacijama kada u konvertovanoj nisci, počevši od nekog indeksa koji je deljiv sa 1 ili 2, ne možemo da napravimo sekvencu od 3 broja nego nam fali još brojeva, konvertovana niska se dopunjuje onolikim brojem nula koliko nam je potrebno da bi mogle da se naprave trojke uzoraka.

3.1 Korak 0: Konvertovanje karaktera niske u brojeve

Od ulazne niske kreira se niz brojeva tako što se svaki karakter ulazne niske pretvara u broj i na kraj doda doda tri nule. Dodate nule će se iskoristiti za konstrukciju trojki uzoraka u sledećem koraku.

3.2 Korak 1: Sortiranje uzoraka sufiksa

Konstruiše se sufiksni niz za sufikse koji počinju na pozicijama i, takvim da je ostatak prilikom deljenja pozicije i sa 3 različit od 0. U ovom koraku svaki sufiks u sufiksnom nizu je dužine 3 počevši od pozicije i.

97	98	99	97	98	99	97	99	97	98	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12

Tabela 2: Konvertovana niska *abcabcacab* sa indeksima

Posmatrajući tabelu 2 dobijamo da sledeći indeksi prilikom deljenja sa 3 daju ostatak 1 ili 2, gde su prvo navedeni indeksi koji daju ostatak 1 prilikom deljenja:

$$\mathbf{SA_{12}} = \{\mathbf{1}, \mathbf{4}, \mathbf{7}, \mathbf{10}, \mathbf{2}, \mathbf{5}, \mathbf{8}\} \tag{2}$$

U ovom slučaju ne koriste se indeksi 11 i 12 jer počevši od njihove pozicije neće moći da se kreira sekvenca od 3 broja.

Nakon toga indekse u nizu SA_{12} konvertujemo u sekvence od 3 broja, tako da dobijamo sledeći niz:

$$\mathbf{SA_{12}} = \{\mathbf{98}, \mathbf{99}, \mathbf{97}\}, \{\mathbf{98}, \mathbf{99}, \mathbf{97}\}, \{\mathbf{99}, \mathbf{97}, \mathbf{98}\}, \{\mathbf{0}, \mathbf{0}, \mathbf{0}\}, \{\mathbf{99}, \mathbf{97}, \mathbf{98}\}, \{\mathbf{99}, \mathbf{97}, \mathbf{99}\}, \{\mathbf{97}, \mathbf{98}, \mathbf{0}\} \tag{3}$$

Sada se primenjuje radiks sortiranje, porede se prvi članovi svih trojki. Dobijamo sledeći rezultat:

$$\mathbf{SA_{12}} = \{\mathbf{0}, \mathbf{0}, \mathbf{0}\}, \{\mathbf{97}, \mathbf{98}, \mathbf{0}\}, \{\mathbf{98}, \mathbf{99}, \mathbf{97}\}, \{\mathbf{98}, \mathbf{99}, \mathbf{97}\}, \{\mathbf{99}, \mathbf{97}, \mathbf{98}\}, \{\mathbf{99}, \mathbf{97}, \mathbf{98}\}, \{\mathbf{99}, \mathbf{97}, \mathbf{99}\} \tag{4}$$

Radiks sortiranje se mora primeniti više puta jer među nekim trojkama su isti prvi članovi i samim tim nismo skroz sigurni koji je pravi redosled tih trojki. Prave se grupe gde je isti prvi član pa se te grupe posebno sortiraju po drugom članu trojke.

$$\mathbf{G1} = \{\mathbf{98}, \mathbf{99}, \mathbf{97}\}, \{\mathbf{98}, \mathbf{99}, \mathbf{97}\} \tag{5}$$

$$\mathbf{G2} = \{\mathbf{99}, \mathbf{97}, \mathbf{98}\}, \{\mathbf{99}, \mathbf{97}, \mathbf{98}\}, \{\mathbf{99}, \mathbf{97}, \mathbf{99}\} \tag{6}$$

Drugi članovi grupe *G1* kao i drugi članovi grupe *G2* su isti, tako da se te 2 grupe sortiraju i po trećem članu.

Konkretno u slučaju grupe *G1* neće biti promena pošto su njeni članovi isti, ali će kod grupe *G2* doći do promene pošto postoji različit treći član.

U slučaju da su svi elementi koje smo sortirali pomoću radiks sorta različiti nastavljamo dalje sa algoritmom. Ako postoje bilo koja 2 elementa koja su ista, onda ovaj algoritam pozivamo rekurzivno.

Krajnji rezultat ovog koraka je sortirani niz sufiksa koji počinju na indeksima koji nisu deljivi sa 3.

3.3 Korak 2: Sortiranje sufiksa koje nismo do sada

U ovom koraku sortiramo elemente čiji indeks prilikom deljenja sa 3 daje ostatak 0. Kreiramo niz parova gde je prvi član vrednost iz konvertovanog početnog niza a drugi član je sam indeks. Opet radimo radiks sort ali samo po prvom članu ovih parova. U slučaju da su svi prvi članovi različiti sufiksi će biti sortirani. Rezultat ovog koraka će biti niz koji sadrži druge članove ovih sortiranih parova, odnosno rezultat će biti sortirani niz indeksa na kojima počinju sufiksi.

Ako se desi da su neki prvi elementi ovih parova isti, moramo da iskoristimo već sortirani sufiksni niz sa indeksima čiji je ostatak 1 ili 2 pri deljenju sa 3. Na osnovu indeksa u parovima posmatramo koja je pozicija indeksa uvećanog za 1 u sortiranom nizu i koristimo tu vrednost kao prvi element parova koji će se dalje porediti. Ovo je ispravno zato što već imamo sortirane sufikse koji počinju na indeksima čiji je ostatak 1 ili 2 prilikom deljenje sa 3, a uvećavanjem indeksa koji je deljiv sa 3 za 1, dobićemo tačno indeks čiji je ostatak 1 pri deljenju sa 3, a njegovu poziciju već znamo.

Krajnji rezultat ovog koraka je sortirani niz indeksa koji su deljivi sa 3.

3.4 Korak 3: Spajanje sortiranih nizova

Poslednji korak ovog algoritma je da spojimo sortirane nizove. Redom idemo kroz oba niza i koristeći sortirane indekse iz tih nizova poredimo vrednosti iz originalnog niza koji je konvertovan u brojeve - *T*.

1. Ako je *T[i]* $<$ *T[j]* ili *T[i]* $>$ *T[j]*, u zavisnosti od onoga što je manje, taj indeks će se dodati u sufiksni niz. Pokazivač u nizu iz kog je uzet indeks se uvećava za 1.

2. Ako su ove dve vrednosti jednake poredimo vrednosti iz niza čiji su indeksi uvećani za 1. Ovde može da se iskoristi prednost toga što je već poznat sufiksni niz niza indeksa čiji su ostaci 1 i 2 prilikom deljenja sa 3. Ako je ostatak 1 prilikom deljenja pokazivača drugog niza sa 3, onda uvećevanjem za 1 dobijamo element koji i dalje pripada drugom nizu, a uvećavanjem pokazivača prvog niza za 1, njegov ostatak će biti 1, tako da dobijamo elemente koji pripadaju drugom nizu a sufiksni niz drugog niza već znamo. U slučaju da je ostatak 2 prilikom deljenja pokazivača drugog niza sa 3, onda uvećevanjem za 1 dobijamo element koji pripada prvom nizu, a uvećavanjem pokazivača prvog niza za 1, njegov ostatak će biti 1, i pripada drugom nizu. U ovom slučaju smo dobili elemente koji pripadaju različitim nizovima pa je njihovo poređenje potrebno.

Kada prođemo kroz sve indekse jednog niza, elemente preostalog niza samo dodamo na kraj sufiksnog niza i dobićemo traženi rezultat.

3.5 Složenost algoritma

Nulti korak predstavlja pripremu niske i transformisanje u niz brojeva. Obuhvata jedan prolazak kroz niz tako da je njegova složenost *O(n)*, gde je *n* broj karaktera.

Prvi korak obuhvata pozicije koje daju ostatak 1 ili 2 prilikom deljenja sa 3. Broj takvih pozicija je *2n/3*. Svaki od koraka zahteva linearno vreme tako da je složenost ovog koraka isto linearna.

Drugi korak obuhvata sortiranje sufiksa koje nismo do sada uz pomoć *radix* sortiranja prvi elemenata parova koji ima *n/3*. Uslučaju da ima istih elemenata potreban je još jedan prolazak kroz niz preostalih sufiksa da bi se odredili koji su sledeći indeksi koji se porede. Zbog prethodno navedenog složenost ovog koraka je *O(n)*, gde je *n* broj karaktera.

Treći korak predstavlja objedinjavanje dva sortirana dela niza tako što se prolazi kroz ta dva dela, pri čemu se ide do kraja jednog ili drugog kraja niza a ostatak elemenata se doda na kraj, tako da je složenost ovog koraka isto *O(n)*, gde je *n* broj karaktera.

S obzirom da je složenost svakog od koraka *O(n)*, može se zaključiti da će i ukupna složenost algoritma biti *O(n)*, gde je *n* broj karaktera.

Jedino u prvom koraku može da se desi da dođe do rekurzije i onda dobijamo rekurentnu jednačinu sledećeg tipa:+

$$\mathbf{T(n) = T(2n/3) + O(n)} \tag{7}$$

Rešavanjem ove rekurentne jednačine dobija se da je složenost takođe *O(n)*, gde je *n* broj karaktera.

4 Algoritam SA-IS

Algoritam *Suffix Array by Induced Sorting* na osnovu sortiranih delova sufiksnog niza pokušava da sortira i ostale delove sufiksnog niza koji već nisu sortirani [4, 5, 6].

Ako imamo nisku *S* na nju dodajemo karakter *$* koji predstavlja sentinel i služiće da predstavlja praznu nisku.

Definicija: L-tip u niski *S* je svaki onaj karakter koji je veći od karaktera koji se nalazi na sledećoj poziciji.

Definicija: S-tip u niski *S* je svaki onaj karakter koji je manji od karaktera koji se nalazi na sledećoj poziciji.

Definicija: LMS pozicija (*leftmost S*) je svaki onaj karakter koji je S-tip u niski *S* a koji se nalazi sa desne strane karaktera koji je L-tip.

Definicija: LMS interval predstavlja interval od jedne LMS pozicije do druge LMS pozicije pri čemu nema drugih LMS pozicija između njih. LMS interval određuje i LMS podnisku niske *S*.

Algoritam se sastoji od narednih koraka:

1. Generiši niz S/L tipova za nisku S
2. Nađi sve LMS pozicije
3. Sortiraj sufikse na LMS pozicijama
4. Sortiraj preostale sufikse

4.1 Korak 1: Generiši niz S/L tipova za nisku

Prolaskom kroz nisku, s desna na levo, možemo da generišemo niz S/L tipova. Karakter na poslednjoj poziciji je sentinel *$* i on je uvek S-tip, a karakter pre njega je uvek L-tip.

4.2 Korak 2: Nađi sve LMS pozicije

Prolaskom kroz niz S/L tipova, s leva na desno, proveravamo da li je trenutni tip karaktera S i da li je tip karaktera pre njega L, ako to jeste slučaj našli smo jednu od LMS pozicija.

4.3 Korak 3: Sortiraj sufikse na LMS pozicijama

U ovom koraku se sortiraju sufiksi na LMS pozicijama, odnosno LMS podniske da bi se smanjila ukupna dužina.

Za sve LMS sufikse u niski se određuju početna i krajnja pozicija na kojoj se taj sufiks može naći. To se radi tako što se za svaki sufiks odrede početni karakteri. Tako na primer ako su početni karakteri sufiksa *a* i *e*, sufiksi koju počinju karakterom *a* moraju da se nađu u sufiksnom nizu pre karaktera *e*.

Dalje na osnovu delimično sortiranih LMS pozicija se sortiraju L-tipovi a na osnovu sortiranih L-tipova se sortiraju S-tipovi. Odavde dobijamo sortirane LMS pozicije.

L-tipovi se sortiraju tako što se prolazi kroz delimično sortirani sufiksni niz s leva na desno i ako na nekoj poziciji znamo da se nalazi LMS-pozicija proveravamo koji je tip karaktera koji se nalazi pre te pozicije u originalnoj niski. Ako je taj karakter L-tip dodajemo ga u prvu slobodnu poziciju u okviru njegove kategorije karaktera. Ako je taj karakter S-tip preskačemo ga. Ovim korakom će svi L-tipovi biti korektno sortirani.

S-tipovi se sortiraju tako što se prvo iz delimično sortiranog niza izbace svi S-tipovi i krene da se iterira kroz taj niz s desna na levo. Ako se na nekoj poziciji nalazi karakter proveravamo koji je tip karaktera koji se nalazi pre te pozicije u originalnoj niski. Ako je taj karakter S-tip dodajemo ga u poslednju slobodnu poziciju u okviru njegove kategorije karaktera. Ako je taj karakter L-tip preskačemo ga. Ovim korakom će svi S-tipovi biti korektno sortirani.

Kada se LMS intervali sortiraju smanjujemo dužinu podniski tako što svaku podnisku zamenimo brojem. Na taj način smanjujemo broj elemenata u okviru podniske koji mogu da se porede u rekurzivnoj fazi. Brojeve dodeljujem redom, počevši od 0 i svakoj novoj podnisci dodelimo novi broj. Nakon ove faze ako su sve podniske različite imaćemo sortirane LMS sufikse i nastavljamo u sledećem koraku. Ako postoje neki sufiksi koji su isti, ovaj algoritam se primenjuje rekurzivno nad smanjenim brojem elemenata.

4.4 Korak 4: Sortiraj preostale sufikse

Ova faza je veoma slična prethodnoj po pitanju korišćenja kategorija kojima sufiksi pripadaju i daljeg korišćenja indukovanog sortiranja. Za sve sufikse u niski se određuju početna i krajnja pozicija na kojoj se taj sufiks može naći. To se radi tako što se za svaki sufiks odrede početni karakteri. Tako na primer ako su početni karakteri sufiksa a i e, sufiksi koju počinju karakterom a moraju da se nađu u sufiksnom nizu pre karaktera e.

U okviru svake od kategorije karaktera oni sufiksi koji počinju karakterom koji je L-tip se nalaze ispred sufiksa čiji je početni karakter S-tip.

Koristi se već sortirani niz LMS pozicija da bi se korektno sortirali L-tipovi karaktera a onda na osnovu sortiranih L-tipova se sortiraju preostali S-tipovi. Bitno je napomenuti da su ovo funkcije koje su već korišćene u prethodnoj fazi tako da su ove dve faze prilično slične.

4.5 Složenost algoritma

Prvi korak generiše niz tipova karaktera i obuhvata jedan prolazak kroz niz tako da je njegova složenost *O(n)*, gde je n broj karaktera.

Da bismo u **drugom koraku** mogli da odredimo sve LMS pozicije potrebno je da prođemo kroz ceo niz. Složenost ovog koraka *O(n)*, gde je n broj karaktera.

Treći korak obuhvata sortiranje sufiksa na LMS pozicijama. U ovom koraku prvo se prolazi kroz LMS sufikse da bi se odredile odgovarajuće početne i krajnje pozicije a zatim na osnovu toga vrši sortiranje L-tipova. Na osnovu sortiranih L-tipova vrši sortiranje S-tipova. Svaki od ovih postupaka zahteva prolazak kroz ceo niz pa je samim tim složenost ovog dela trećeg koraka *O(n)*, gde je n broj karaktera.

U ovom koraku može da se uđe u rekurziju ako su neki od LMS intervala koji su se sortirali isti. Zamenjivanjem LMS intervala brojevima koji se dalje u rekurziji porede smanjuje se dužina niske za *n/2*. Ovime se dobija rekurentna jednačina:

$$\mathbf{T(n) = c1 * n + T(n/2) + c2 * n} \tag{8}$$

Rešavanjem ove rekurentne jednačine dobija se da je složenost trećeg koraka isto *O(n)*, gde je n broj karaktera.

Četvrti korak zapravo predstavlja prvi deo trećeg koraka, tako da je složenost ovog koraka isto *O(n)*, gde je n broj karaktera.

S obzirom da je složenost svakog od koraka *O(n)*, može se zaključiti da će i ukupna složenost algoritma biti *O(n)*, gde je n broj karaktera.

5 Poređenje rezultata

Algoritmi su poređeni na različitim ulazima pri čemu se merilo vreme koje je potrebno da se algoritam izvrši.

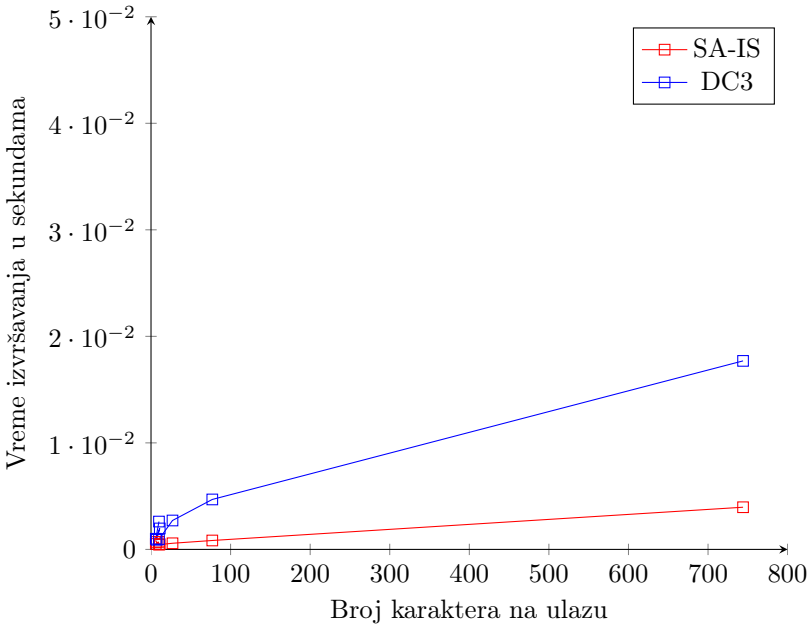
Kao što se može primetiti u tabeli 3, oba algoritma su efikasna ali je ipak SA-IS brži, što odgovara pretpostavkama da je ovo najbrži do danas poznat algoritam za konstruisanje sufiksnog niza. Na osnovu rezultata može se videti da je SA-IS brži od 1 do 100 puta od DC3. Na primeru nukleotidne sekvence se vidi da je SA-IS brži skoro 10 puta što pogotovo može biti značajno u oblasti bioinformatike za sekvenciranje genoma.

Ulaz	DC3	SA-IS
aaaaaaaaaa	0.0026137999957427382	0.0007078000053297728
banana	0.0009997000161092728	0.0005082999996375293
mississippi	0.0019668000168167055	0.0004800999886356294
racecar	0.0009052999957930297	0.0006990999972913414
abcdeedcba	0.0009528000082354993	0.0004487000114750117
abracadabraabracadabraabracadabrabanana	0.002713699999731034	0.0005761999927926809
abracadabra * 7	0.00468929999624379	0.00083710000035353
Nukleotidna sekvenca od 744 karaktera	0.017695799993816763	0.0039591999957337976

Tabela 3: Tabela sa vremenima izvršavanja

Na osnovu tabele 3 napravljen je i graf na kom se može videti da vreme izvršavanje ne raste velikom brzinom u zavisnosti od broja ulaznih karaktera:

Zavisnost vremena izvršavanja algoritma od veličine ulaza



Literatura

[1] Juha Kärkkäinen and Peter Sanders. Simple linear work suffix array construction. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Automata, Languages and Programming*, pages 943–955, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[2] Juha Kärkkäinen and Peter Sanders. Presentation: Simple linear work suffix array construction. on-line at: https://people.csail.mit.edu/jshun/6886-s19/lectures/lecture9-1.pdf.

[3] Ivan Ječmen. Sufiksno stablo i sufiksni niz. Master rad, Matematički fakultet Univerziteta u Beogradu.

[4] Ge Nong, Sen Zhang, and Wai Hong Chan. Linear suffix array construction by almost pure induced-sorting. In *2009 Data Compression Conference*, pages 193–202, 2009.

[5] Sven Rahmann. Presentation: Linear time suffix array construction. on-line at: https://www.rahmannlab.de/lehre/alsa21/02-3-sais.pdf.

[6] Slaviša Božović. Sufiksni niz. Master rad, Matematički fakultet Univerziteta u Beogradu, 2015.