UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

MASTER RAD

SUFIKSNI NIZ

Mentor:
Prof. dr Miodrag Živković

Student:
Slaviša Božović 1014/2011.

Beograd, 2015.

UVOD

Proučavanje genoma živih organizama predstavlja jedan od bitnih projekata u bologiji. Genom se može definisati kao skup gena koje sadrži jedna haploidna ćelija. Geni se nalaze na hromozomskoj DNK i predstavljaju linerani raspored nukleotida [1]. S obzirom da biolozi nukleotide DNK obeležavaju velikim latiničnim slovima (A, T, G, C), gen kao linearni raspored nukleotida, mogao bi da se zapiše kao niska karaktera – string, a samim tim, genomi kao skup gena (stringova), mogli bi da se posmatraju kao jedan tekst. Ovakva reprezentacija genoma i gena omogućava njihovu obradu različitim algoritmima pretrage stringova. Do trenutka uvođenja računara u proces istraživanja, ova oblast je bila praktično neistražena, a rezultati su se poboljšavali uporedo sa razvojem bioinformatike kao posebne grane informatike. Jedan od osnovnih zadataka bioinformatike predstavlja razvoj algoritama za što efikasniju pretragu stringova.

Postoji veliki broj algoritama za rad sa stringovima čiji je glavni cilj pronalaženje broja pojavljivanja određene niske karaktera (uzorka) P dužine m u tekstu S dužine n ($m \leq n$). Algoritmi koji rade u navedenom vremenu izvode određeno predprocesiranje uzorka P i na taj način omogućuju isključivanje određenih delova teksta iz dalje pretrage. Međutim, dokle god je $m = O(1)$, ovi algoritmi ne mogu da dostignu vremensku složenost $O(n)$. Da bi se postigla linearna vremenska složenost, potrebno je prethodno obraditi tekst S. Prethodna obrada teksta je korisna u situacijama kada je tekst konstantan tokom vremena (npr. genom) i kada će pretraga da se radi za više različitih uzoraka.

U ovom radu biće opisana struktura podataka koja se dobija predobradom teksta, i koja se pokazala kao izuzetno efikasna u pronalaženju broja pojavljivanja određenog uzorka u nekom tekstu. Ta struktura podataka se naziva sufiksni niz i usko je povezana sa strukturom podataka sufiksno stablo, a prednost sufiksnog niza u odnosu na stablo je u tome što je potreban memorijski prostor manji za konstantni faktor.

U prvom poglavlju rada predstavljeno je pojmovno određenje sufiksnih nizova i sufiksnog stabla, osnovne definicije i teorijske tvrdnje. U drugom poglavlju predstavljeni su algoritmi za direktno konstruisanje sufiksnog niza sa svim svojim osobenostima, algoritam Ko-Aluru i algoritam SA-IS. U trećem poglavlju je opisana programska realizacija algoritama za konstrukciju sufiksnog niza u vremenu $O(n)$, izloženih u drugom poglavlju.

1. OSNOVNI POJMOVI I DEFINICIJE

1.1. Sufiksni niz

Sufiksni niz se koristi za širok spektar problema, kao što su indeksiranje, kompresija, skladištenje i obrada tekstualnih podataka. S obzirom na manji memorijski prostor koji je potreban za skladištenje sufiksnog niza u odnosu na sufiksno stablo, i sa početkom masovne upotrebe web-a i razvojem bioinformatike, radilo se na pronalaženju algoritama koji će koristeći sufiksni niz pri rešavaju problema u radu sa stingovima raditi asimptotski isto efikasno, kao što su do tada radili algoritmi koji koriste sufiksno stablo. Uvođenjem dodatne strukture podataka, niza LCP, došlo se do algoritma koji je iste asimptomske složenosti kao algoritmi koji koriste sufiksno stablo. Ova struktura podataka je prvi put predstavljena od strane Menbera i Majersa [7] kao prostorno efikasnija alternativa za sufiksno stablo.

Neka je $S = S_1S_2\ldots S_n$ string dužine n.
Podstring stringa S označava se sa $S[i, j] = S_i\, S_{i+1}\ldots S_j$, gde je $1 \leq i \leq j \leq n$.

Definicija 1.1: Neka je $S=S_1S_2\ldots S_n$ string dužine n. *i*-ti sufiks stringa S predstavlja njegov specijalni podstring S [*i*, *n*], gde važi da je $1 \leq i \leq n$. Podstring S [*i*, *n*] je *i*-ti sufiks stringa S.

Definicija 1.2: Neka je $S = S_1S_2\ldots S_n$ string dužine n. Sufiksni niz A stringa S predstavlja niz celobrojnih indeksa svih sufiksa stringa S koji su leksikografski sortirani u rastućem poretku.

Iz definicije sufiksnog niza sledi da elemenat A[*i*] sufiksnog niza A sadrži startnu poziciju *i*-tog najmanjeg sufiksa u S, i za svako *i* ($1 < i \leq n$) važi $S[A[i-1], n] \prec S[A[i], n]$, gde znak „$\prec$“ označava relaciju „leksikografski manji“. Neka je S niz znakova dužine n, tj. $|S| = n$, gde su znakovi niza S elementi abecede Σ. Neka σ označava broj znakova abecede, tj. $|\Sigma| = \sigma$. Znak na *i*-tom mestu u nizu S označava se sa S[*i*] ili kraće S_i, gde je $1 \leq i \leq n$.

Uvodi se i sledeća oznaka :

- prefiks niza S je podniz $S[1, j]$, $1 \leq j \leq n$

Primer 1.1: Razmotrimo string S = "banana", dužine 6 znakova.

String S je indeksiran na sledeći nacin:

i	**1**	**2**	**3**	**4**	**5**	**6**
S[*i*]	b	a	n	a	n	a

Sufiksi stringa S su:

S[*i, n*]	***Sufiks***
S[*1, 6*]	banana
S[*2, 6*]	anana
S[*3, 6*]	nana
S[*4, 6*]	ana
S[*5, 6*]	na
S[*6, 6*]	a

Sortirani sufiksi stringa S su:

S[*i, n*]	***Sufiks***
S[*6, 6*]	a
S[*4, 6*]	ana
S[*2, 6*]	anana
S[*1, 6*]	banana
S[*5, 6*]	na
S[*3, 6*]	nana

Prema tome, sufiksni niz A stringa S je:

i	**1**	**2**	**3**	**4**	**5**	**6**
A[*i*]	6	4	2	1	5	3

Na primer, A [*2*] = 4, znači da je sufiks S [*4, 6*] = “ana” drugi najmanji sufiks stringa S.

1.2. Sufiksno stablo

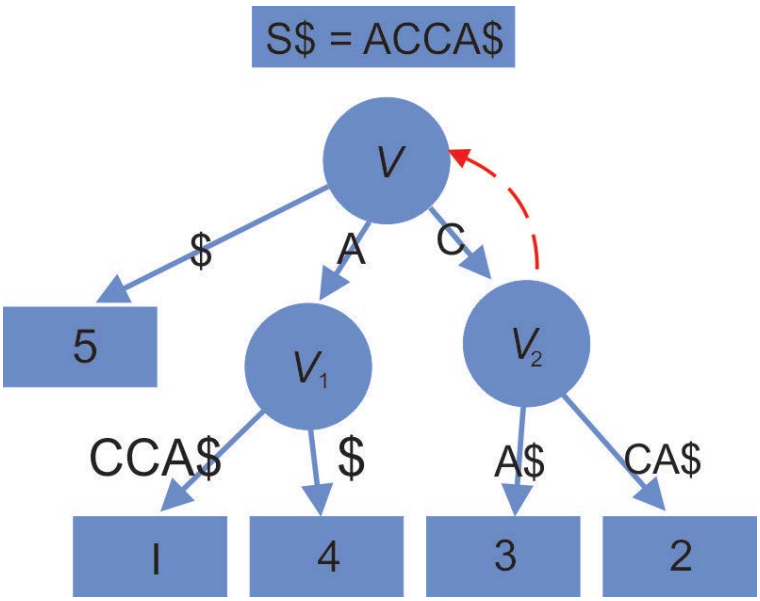
Sufiksni niz je usko povezan sa drugom strukturom podataka koja se naziva sufiksno stablo. Sufiksno stablo (eng. suffix tree) predstavlja strukturu podataka koja opisuje internu strukturu niske karaktera na vrlo iscrpan način.

Da bi bilo moguće definisati sufiksno stablo, string S dužine *n* se proširuje dodatnim završnim karakterom koji ne pripada alfabetu stringa, obično $, koji je različit od svih drugih karaktera u stringu, pa je S dužine *n+1* i oblika S = string$.
Sufiksno stablo je indeksna podatkovna struktura namenjena efikasnom obavljanju različitih operacija nad znakovnim nizovima i često se koristi u bioinformatici [6].

Definicija 1.3: Sufiksno stablo za string S dužine *m* znakova je korensko orjentisano stablo sa tačno *m* listova numerisanih od 1 do *m*. Svi unutrašnji čvorovi koji su različiti od korena imaju najmanje dva sina i svaka grana je označena nepraznim podstringom stringa S. Nikoje dve grane koje izlaze iz čvora ne mogu da imaju oznake grane koje počinju istim znakom. Najvažnija osobina kod sufiksnih stabala je ta da za svaki list *i* konkatenacija oznaka grana na putu od korena do lista *i* je jednaka sufiksu S koji počinje na poziciji *i*.

Primer 1.2: Zadat je string S= ACCA.
Na slici 1. prikazano je sufiksno stablo za string S$ =ACCA$.
Znak $ je dodat na kraj stringa S, čime je svaki sufiks S[*i*, *n*], $1 \leq i \leq 5$ i $n=5$, jednoznačno određen pripadajućim listom *i*.

Kada znak $ ne bi bio dodat na kraj niza S, onda bi sufiks, koji je prefiks nekog drugog sufiksa, završavao u unutrašnjem čvoru stabla (eng. inner node; branch node), a ne u listu stabla (eng. leaf; terminal node).



Slika 1. *Sufiksno stablo za niz S$ = ACCA$.*

Sufiksi sadržani u prikazanom sufiksnom stablu su:

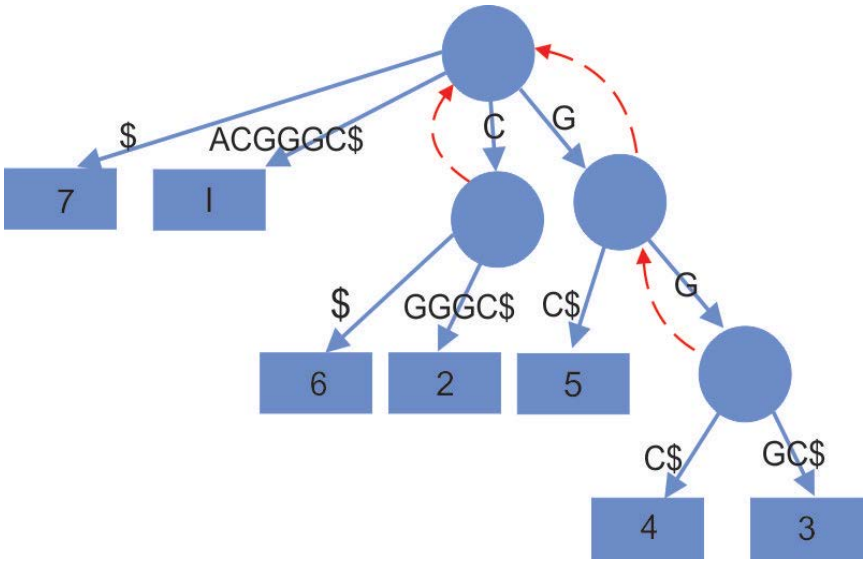
S1 = S[1, n + 1] = S[1, 5] = ACCA$
S2 = S[2, n + 1] = S[2, 5] = CCA$
S3 = S[3, n + 1] = S[3, 5] = CA$
S4 = S[4, n + 1] = S[4, 5] = A$
S5 = S[n + 1, n + 1] = S[5, 5] = $

Na primer, može se proveriti za sufiks *S2*= CCA$ postoji li odgovarajući list u stablu. Provera se obavlja tako što se sledi put od korena (čvor *v*) prema listovima stabla (Slika 1). Kreće se od korena i upoređuje prvi znak sufiksa S*2* (C) sa svim prvim znakovima na oznakama grana koje kreću od korena prema čvorovima - sinovima v_1 i v_2 - iz korena kreću tri grane čije su oznake: "$", "A" i "C". Krajnje desna grana na slici 1 označena je sa "C", pa se put nastavlja tom granom do čvora v_2.

Iz v_2 vode dve grane: leva grana označena je sa "A$", a desna sa "CA$". Upoređuje se drugi znak iz *S2* (C) sa prvim znakom u granama koje vode iz čvora v_2. S obzirom da oznaka desne grane počinje sa znakom C, nastavlja se sa poređenjem znakova iz *S2* i znakova na oznaci te grane: upoređuje se dalje treći znak iz *S2* s drugim znakom na oznaci desne grane. S obzirom da se znakovi podudaraju, onda se upoređuje i sledeći par znakova: četvrti znak u *S2* i treći znak oznake grane. S obzirom da se i taj par znakova podudara, nastavlja se put tom granom.

Kako više nije ostao ni jedan znak u *S2*, a put istovremeno završava u listu s oznakom *2*, potvrđeno je da sufiks *S2* ima pripadajući list u stablu *T*.

Na Slici 2 prikazane su sufiksne veze kao crvene isprekidane strelice. Sufiksne veze povezuju unutrašnji čvor koji odgovara sufiksu *x*α sa unutrašnjim čvorom koji odgovara sufiksu α, gde je *x* neki znak, a α (moguće prazan) niz znakova. Koriste se i kod izgradnje sufiksnog stabla u linearnom vremenu.



Slika 2. *Sufiksno stablo za niz S$ = ACGGGC$ ($ je leksikografski manji od znakova iz S). Sufiksne veze označene su crvenim crtkanim strelicama.*

Uvođenjem znaka $ obezbeđuje se da nijedan sufiks stringa S ne predstavlja prefiks nekog drugog sufiksa, i da stablo ima *n* listova, za svaki sufiks po jedan list.

Sufiksno stablo na elegantan način rešava problem traženja podstringa u stringu. Naime, traženje nekog podstringa u stringu odgovara traženju nekog pojma u knjizi. Ako knjiga sadrži indeks pojmova, vrlo je jednostavno pronaći traženi pojam u knjizi: umesto da se pretražuje cela knjiga, pojam se traži pretraživanjem indeksa. Pretraživanje je dodatno ubrzano, s obzirom da su pojmovi abecedno poređani. Ako je pojam zabeležen u indeksu, onda je uz pojam navedena i stranica gde se pojam spominje u knjizi, pa se pronalaskom pojma u indeksu, jednostavno dolazi i do mesta gde se pojam spominje u knjizi.

Indeksna struktura „sufiksno stablo“ omogućava brz pronalazak podstringa u posmatranom tekstu, preciznije, podstring je moguće pronaći u vremenu proporcionalnom dužini podstringa. Takođe, korišćenjem sufiksnog stabla moguće je rešiti i mnoštvo drugih složenih operacija nad stringovima. Klasičan primer je pronalazak najdužeg zajedničkog podstringa dva stringa u vremenu proporcionalnom zbiru njihovih dužina [6].

Sufiksno stablo može biti izgrađeno u vremenu linearno zavisnom od dužine ulaznog stringa.
Prvi algoritam koji je to omogućio bio je Vajnerov algoritam (Weiner, 1973), a zatim i MekKrejtov algoritam (McCreight, 1976).

Današnji je standard Ukonenov algoritam iz 1995. godine (Ukkonen, 1995). To je bio prvi algoritam linearne složenosti za izgradnju sufiksnog stabla: umesto da se sufiksno stablo gradi odjednom iz celog ulaznog stringa (pri čemu celi ulazni string mora biti poznat na samom početku izgradnje), sufiksno stablo se proširuje za svaki znak u stringu.

Neka je zadati string S dužine *n*, gde su znakovi stringa iz abecede Σ. Sufiksno stablo izgrađeno nad S je korensko stablo T (eng. rooted tree) koje sadrži sve sufikse stringa S$. (Slika 1).

Listovi stabla T označeni su brojevima od 1 do *n*, a odgovaraju početnim mestima sufiksa u stringu S.
Iz korena sufiksnog stabla izlazi tačno onoliko grana koliko je znakova abecede, i još jedna grana za znak $.
Svaki unutrašnji čvor ima dva ili više čvorova – sinova. Oznaka grane sadrži jedan ili više znakova koji odgovaraju nekom podstringu stringa S.
Oznake grana koje izlaze iz istog čvora moraju započinjati različitim znakovima. Znakovna dubina (eng. string depth) nekog čvora je zbir dužina svih oznaka grana na putu od korena do tog čvora.

1.3. Sufiksni niz i sufiksno stablo

Kako je već napomenuto, stablo zahteva više prostora od niza, zato se umesto sufiksnog stabla ponekad pamte samo neke potrebne informacije koje se mogu smestiti u niz.

Polazeći od sufiksnog stabla stringa T može se konstruisati njegov sufiksni niz.
Konstruiše se sufiksno stablo za *T*. Leksikografski se obilazi stablo po dubini i zapisuju indeksi listova u redosledu u kome su bili posećeni. Taj niz indeksa je sufiksni niz. Vremenska složenost je linearna ali je potreban dodatni prostor za konstrukciju stabla. Stablo može biti izbrisano kada se konstruiše sufiksni niz.

Kako je sufiksno stablo podatkovna struktura pogodna za rad sa znakovnim nizovima često se koristi u radu s DNK sekvencama, ali i u radu s drugim biološkim sekvencama [3].

Kao nedostatak sufiksnog stabla može se navesti njegova velika memorijska zahtevnost te je, zavisno od implementacije, memorijski prostor potreban za pohranu sufiksnog stabla od 10 n do 20 n okteta za znakovni niz od *n* znakova. Taj problem posebno dolazi do izražaja u radu sa vrlo dugačkim znakovnim nizovima, kao što su celi genomi sisara čija je dužina reda veličine 10^9 znakova. Zbog toga se danas u radu sa biološkim sekvencama koristi, kao zamena za sufiksno stablo, memorijski manje zahtevan sufiksni niz, koji u svojoj unapređenoj verziji, tzv. poboljšani sufiksni niz (eng. enhanced suffix array), omogućuje sprovođenje svih operacija nad znakovnim nizovima jednako kao i sufiksno stablo [2].

Definicija 1.4: Neka je A sufiksni niz stringa S dužine *n*, i neka je lcp*(u,v)* dužina najdužeg zajedničkog prefiksa stringova *u* i *v*. Niz *H* dužine *n* je niz LCP stringa S ako je *H*[1] nedefinisano i *H*[*i*] = lcp*(S[A[*$_{i-1}$*], n], S[A[i], n])* za svako $1 < i \leq n$.

Tako, *H*[*i*] sadrži dužinu najdužeg zajedničkog prefiksa od leksikografski *i*-tog najmanjeg prefiksa i njegovog prethodnika u sufiksnom nizu.

Niz LCP, ili niz najdužih zajedničkih prefiksa, predstavlja pomoćnu strukturu podataka sufiksnom nizu. Ovaj niz sadrži dužine najdužih zajedničkih prefiksa između uzastopnih parova leksikografski uređenih sufiksa stringa. Drugim rečima, to je niz dužina zajedničkog prefiksa između dva uzastopna sufiksa u leksikografski uređenom sufiksnom nizu.

Primer 1.3: Razmotrimo string S = "banana", dužine 6 znakova. Konstruisaćemo niz LCP za dati string S.

Polazi se od sufiksnog niza A određenog u primeru 1.1:

i	**1**	**2**	**3**	**4**	**5**	**6**
A[*i*]	6	4	2	1	5	3

Na osnovu definicije dobijaju se vrednosti članova niza LCP:

i	**1**	**2**	**3**	**4**	**5**	**6**
H[*i*]	⊥	1	3	0	0	2

Tako je na primer H[*3*] = 3 dužina najdužeg zajedničkog prefiksa “ana” koji dele sufiksi **A[*2*] = S[*4, 6*] = “ana” i A[*3*] = S[*2, 6*] = “anana”.**

Primećuje se da u tabeli stoji da je *H*[1] = ⊥, tj. nedefinisana vrednost, s obzirom da je ***A*[1] = S[**6, 6] = **”a”** leksikografski najmanji sufiks u sortiranom sufiksnom nizu, odnosno sufiks koji u sortiranom redosledu nema prethodnika.

Postoje dve vrste algoritama koje se bave izgradnjom niza LCP:

Prva vrsta algoritama su oni koji konstruišu niz LCP kao sporedni proizvod sufiksnom nizu a **druga vrsta** su oni koji koriste već napravljene sufiksne nizove sa ciljem da od njih konstruišu niz LCP.

Algoritmi koji koriste sufiksni niz u kombinaciji sa nizom LCP imaju istu vremensku složenost rešavanja problema kao algoritmi koji koriste sufiksno stablo [2].

2. ALGORITMI ZA DIREKTNU KONSTRUKCIJU SUFIKSNOG NIZA

Algoritmi za direktnu konstrukciju sufiksnog niza objavljeni 2003. godine omogućili su da se pomoću sufiksnih nizova rešavaju isti problemi za čije su se rešavanje do tada koristila sufiksna stabla. Zbog eksplicitnije strukture i postojanja linearnog algoritma za konstrukciju, teoretičari su uvek davali prednost sufiksnim stablima u odnosu na sufiksne nizove. Sa druge strane, ljudi iz prakse često koriste sufiksne nizove, jer su efikasniji u pogledu prostora i jednostavniji za implementaciju.

Pojavom prvog direktnog linearnog algoritma [5] za konstrukciju sufiksnog niza, sufiksni nizovi se podižu na isti nivo sa sufiksnim stablima.

2.1. Naivni algoritam

Naivni algoritam je jednostavan za razumevanje i implementaciju. Osnovna ideja naivnog algoritma se sastoji u tome da se pronađe dobra funkcija za sortiranje stringova koja se zasniva na poređenjima. Kvik sort algoritam odrađuje ovaj zadatak jako dobro.

Ako je ulaz u algoritam string S koji je duzine *n,* onda će on imati *n* sufiksa S[*i, n*] 1<=*i*<=*n*, uključujući i ceo string S, koje je potrebno međusobno porediti i sortirati. Poznato je da je Kvik sort algoritam do sada najbrži i najpoznatiji algoritam za sortiranje i da je njegova vremenska složenost O(*n* log *n*).

Pošto naivni algoritam koristi Kvik sort i za poređenje *n* sufiksa potrebno je O(*n*) upoređivanja sufiksa, dolazi se do zaključka da je ukupna vremenska složenost naivnog algoritma jednaka O(n^2 log *n*).

S obzirom da sufiksni niz predstavlja niz indeksa tako sortiranih sufiksa, da bi nakon sortiranja bilo moguće izgraditi sufiksni niz, potrebno je uvesti informaciju uz svaki sufiks koji je njegov indeks u stringu S. U programskim jezicima se to lako postiže korišćenjem struktura podataka koje se zovu “mape” ili “rečnici”, u kojima se kao ključ čuva sufiks, a kao vrednost se čuva njegov indeks u stringu S. Prilikom sortiranja, elementi te strukture podataka

se sortiraju po ključu, a povratnu vrednost predstavlja niz vrednosti tako sortirane strukture podataka.

2.2. Modifikovani naivni algoritam

Modifikovani naivni algoritam smanjuje vremensku složenost osnovnog algoritma, sa $O(n^2 \log n)$ na $O(n \log^2 n)$, korišćenjem činjenice da su svi stringovi koji se sortiraju, podstringovi iste niske karaktera. Navedeno zapažanje omogućava sortiranje stringova u iteracijama, pri čemu se u svakoj iteraciji stingovi sortiraju samo po slogovima koji su maksimalne duzine 2^i karaktera, gde je i redni broj iteracije.

Na taj način, stringovi se prvo sortiraju po svoja prva dva karaktera (ukoliko imaju dva ili vise karaktera), zatim po svoja prva četiri karaktera, itd.
Algoritam prestaje da deli stringove na slogove i da ih sortira, onog trenutka kada dužina sloga bude veća od dužine originalnog stringa od koga se pravi sufiksni niz. U tom slučaju, značiće da su sifiksi sortirani.

Primer 2.1: Korišćenjem modifikovanog naivnog algoritma, konstruisaćemo sufiksni niz stringa S = “mississippi”:

Indeksirani sufiksi stringa S su:

Sufiksni indeks (i)	S[i, 11]
1	mississippi
2	ississippi
3	ssissippi
4	sissippi
5	issippi
6	ssippi
7	sippi
8	ippi
9	ppi
10	pi
11	i

Prva iteracija: Sortiraju se stringovi po svoja prva $2^1 = 2$ karaktera. Ukoliko stringovi imaju identična prva dva karaktera, onda dobijaju isti indeks sortiranja.

Indeks sortiranja	Sufiksni indeks (i)	S[i, 11]
1	11	i
2	8	ippi
3	2	ississippi
3	5	issippi
4	1	mississippi
5	10	pi
6	9	ppi
7	4	sissippi
7	7	sippi
8	3	ssissippi
8	6	ssippi

Druga iteracija: Sada je potrebno iskoristiti rezultate prethodne iteracije i sortirati stringove prema njihova prva $2^2 = 4$ karaktera. Da bi se to postiglo, prvi korak je da se svakom stringu iz prethodne tabele dodele dve vrednosti $(_{j, k})$.

Prva vrednost predstavlja indeks sortiranja tekućeg stringa u prethodnoj iteraciji, a druga vrednost je sufiksni indeks iz prethodne iteracije, sufiksa tekućeg stringa koji počinje na poziciji 2^{i-1}. Ako je dužina tekućeg stringa manja od 2 karaktera, onda se za drugu vrednost u paru (*j*, *k*), stavlja -1. To možemo videti u tabeli koja sledi:

Sufiksni indeks (i)	S [i, 11]	Sufiksni indeks iz prethodne iteracije, sufiksa tekuceg stringa koji pocinje na poziciji 2^{i-1}	(j, k)
11	i	-1	(1, -1)
8	ippi	10	(2, 5)
2	ississippi	4	(3, 7)
5	issippi	7	(3, 7)
1	mississippi	3	(4, 8)
10	pi	-1	(5, -1)
9	ppi	11	(6, 1)
4	sissippi	6	(7, 8)
7	sippi	9	(7, 6)
3	ssissippi	5	(8, 3)
6	ssippi	8	(8, 2)

Drugi korak je poziv funkcije Kvik sort koja treba da sortira sufikse koristeći parove (j, k) koji su konstruisani iznad. Rezultat druge iteracije je:

Indeks sortiranja	Sufiksni indeks (i)	S[i, 11]
1	11	i
2	8	ippi
3	2	ississippi
3	5	issippi
4	1	mississippi
5	10	pi
6	9	ppi
7	7	sippi
8	4	sissippi
9	6	ssippi
10	3	ssissippi

Navedeni postupak će se ponavljati *i* puta ($i = 1,2,\ldots$), dok se ne ispuni uslov da je $2^i >= n$.

Rezultujući sufiksni niz A biće:

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
A[i]	**11**	**8**	**5**	**2**	**1**	**10**	**9**	**4**	**7**	**3**	**6**

Kao što je iznad napomenuto, modifikacija naivnog algoritma za konstrukciju sufiksnog niza smanjuje vremensku složenost sa $O(n^2 \log n)$ na $O(n \log^2 n)$. Međutim, i modifikovani naivni algoritam će da radi jako sporo za većinu ulaznih stringova. Zbog toga se jako brzo javila potreba za pronalaženjem algoritama koji će da konstruišu sufiksni niz u dosta kraćem vremenskom roku od modifikovanog naivnog algoritma, ne bi li mogli da se obrađuju jako veliki ulazi.

2.3. Algoritmi za formiranje sufiksnog niza u linearnom vremenu

U nastavku su predstavljene osnovne karakteristike dva algoritma za formiranje sufiksnog niza u linearnom vremenu, algoritma Ko-Aluru (KA) i algoritma SA-IS [5] [8].

Oba algoritma imaju po tri koraka:

1. Razdvajanje,
2. Rekurzivno sortiranje podskupa sufiksa i
3. Indukovano sortiranje svih sufiksa.

Neka je $T = t_1 t_2 ... t_n$ string nad alfabetom $\Sigma = \{1 ... n\}$. Bez gubitka opštosti, pretpostavimo da se poslednji karakter stringa T ne pojavljuje nigde drugde u T, i da predstavlja leksikografski najmanji karakter. Ovaj karakter ćemo označiti sa „$“.

Neka $T_i = t_i t_{i+1} ... t_n$ označava sufiks stringa T čiji je početni karakter t_i. Za čuvanje sufiksa T_i, potrebno je čuvati samo indeks i njegove startne pozicije u stringu T.

Za stringove α i β, koristimo $\alpha \prec \beta$ da bismo označili da je α leksikografski manja od β. U daljem tekstu, termin „sortirani redosled“ označava leksikografski sortirani redosled u rastućem poretku.

Algoritam Ko-Aluru klasifikuje sufikse stringa na dva tipa, S i L. Sufiks T_i je tipa S ukoliko je $T_i \prec T_{i+1}$, a tipa L ukoliko je $T_{i+1} \prec T_i$. Poslednji sufiks T_n nema naredni sufiks, pa se klasifikuje i kao tip S i kao tip L.

Naredna lema pokazuje da se identifikacija tipova sufiksa stringa može efikasno izvršiti.

Lema 2.1: Svi sufiksi stringa T mogu biti klasifikovani kao tip S ili tip L u vremenu O (n).

Dokaz: Razmatramo sufiks T_i ($i < n$).

T	M	I	S	S	I	S	S	I	P	P	I	$
Type	L	S	L	L	S	L	L	S	L	L	L	L/S
Pos	1	2	3	4	5	6	7	8	9	10	11	12

Sika broj 3. *String “MISSISSIPPI$ i tipovi njegovih sufiksa [5].*

Slučaj 1: Ako je $t_i \neq t_{i+1}$, potrebno je samo uporediti t_i i t_{i+1} i utvrditi da li je T_i sufiks tipa *S* ili sufiks tipa *L*.

Slučaj 2: Ako je $t_i = t_{i+1}$, pronaći najmanji $j > i$ takav da je $t_j \neq t_i$.

Ako je $t_j > t_i$, onda su sufiksi T_i, T_{i+1}, ... T_{j-1} tipa *S*.

Ako je $t_i > t_j$, onda su sufiksi T_i, T_{i+1}, ... T_{j-1} tipa *L*.

Prema tome, tipovi svih sufiksa T mogu se odrediti prolaskom kroz string s leva na desno, u vremenu O (*n*).

Klasifikacija sufiksa na tipove *S* i *L* stringa MISSISSIPPI$ prikazana je na slici 3.

Naredna lema opisuje leksikografski poredak sufiksa koji imaju isti početni karakter ali pripadaju različitim tipovima karaktera.

Lema 2.2: Sufiks tipa *S* je leksikografski veći od sufiksa tipa *L* koji ima isti početni karakter.

Dokaz: Pretpostavimo da sufiks T_i tipa *S* i sufiks T_j tipa L imaju isti početni karakter *c*. Možemo da napišemo $T_i = c^k c_1 \alpha$ i $T_j = c^l c_2 \beta$, gde c^k i c^l označavaju karakter *c* koji se ponavlja *k*, *l* > 0 puta; $c_1 > c$, $c_2 < c$; α i β su (moguće prazni) stringovi.

Slučaj 1: Ako je $k < l$ onda se c_1 poredi sa karakterom *c* u c^l. Tada važi $c_1 > c \Rightarrow T_j \prec T_i$.

Slučaj 2: Ako je $k > l$ onda se c_2 poredi sa karakterom *c* u c^k. Tada važi $c > c_2 \Rightarrow T_j \prec T_i$.

Slučaj 3: Ako je $k = l$ onda se c_1 poredi sa c_2. Pošto je $c_1 > c$ i $c > c_2$, tada je $c_1 > c_2 \Rightarrow T_j \prec T_i$.

Tako je sufiks tipa *S* leksikografski veći od sufiksa tipa *L* koji ima isti početni karakter.

Posledica 2.1: U sufiksom nizu stringa T među svim sufiksima koji počinju istim karakterom, sufiksi tipa *S* se pojavljuju nakon sufiksa tipa *L*.

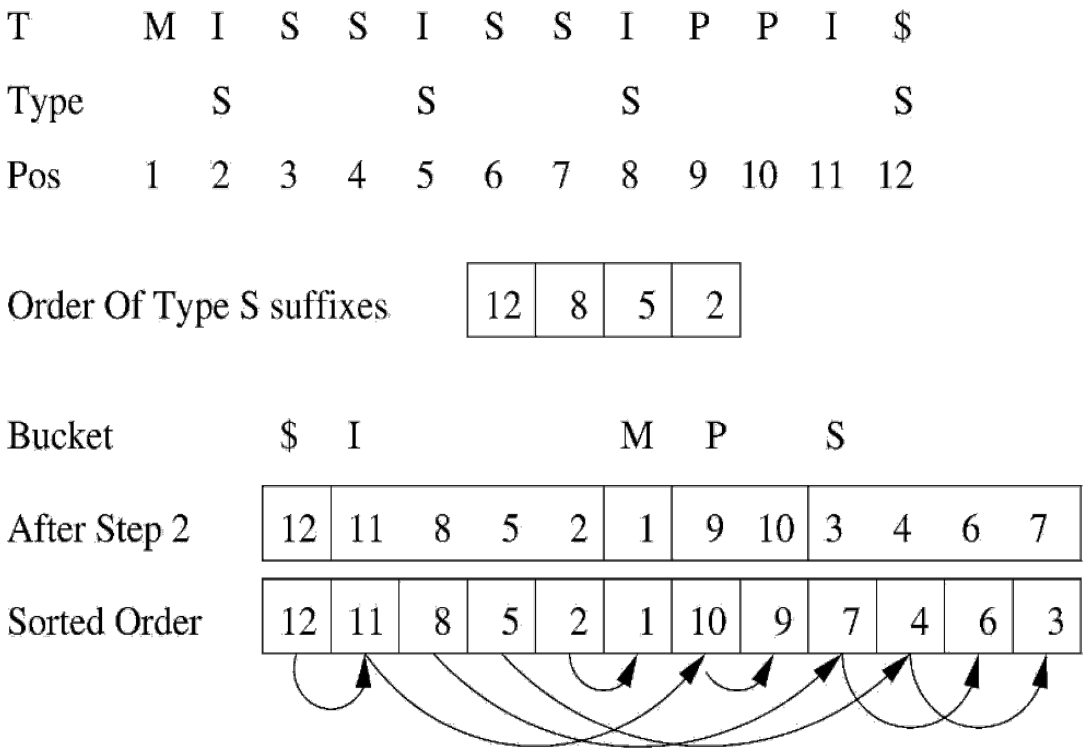
Dokaz: Sledi direktno iz leme 2.2.

2.3.1. Algoritam Ko-Aluru

Ko-Aluru algoritam su 2003. godine objavili Pang Ko i Srinivas Aluru [5] kao algoritam koji sortira sufikse stringa u linearnom vremenu. Algoritam koristi samo 9,25 n bita za alfabete

fiksne dužine i zasnovan je na jedinstvenoj rekurzivnoj formulaciji gde veličina podproblema nije fiksirana već zavisi od karakteristika ulaznog stringa.

Neka je *A* niz koji sadrži sve sufikse stringa *T*, koji ne moraju biti u leksikografski sortiranom poretku. Neka je B niz svih sufiksa tipa S, sortiranih leksikografski u rastućem poretku. Koristeći B, može se odrediti leksikografski poredak svih sufiksa *T* na sledeći način:



Slika br. 4. *Ilustracija konstrukcije sufiksnog niza koristeći sortirani niz sufiksa tipa S stringa MISSISSIPPI$ [5].*

(1) Razvrstaju se u grupe svi sufiksi stringa *T* na osnovu njihovog prvog karaktera. Svaka grupa sadrži sve sufikse stringa *T* koji počinju istim karakterom. Grupe sufiksa se smeštaju u niz *A* u leksikografskom poretku. Ovaj korak zahteva O (*n*) vremena.

(2) Prolazi se kroz *B* s desna na levo. Svaki sufiks na koji se naiđe u prolazu pomeriti na trenutni kraj svoje grupe u nizu *A*, a zatim kraj grupe pomeriti za jednu poziciju levo. Pomeranje sufiksa u nizu *A* na novu poziciju može se posmatrati kao zamena mesta sa sufiksom koji trenutno zauzima novu poziciju. Nakon prolaza kroz sve sufikse u nizu *B*, na osnovu posledice 2.1, svi sufiksi tipa *S* se nalaze na ispravnim pozicijama u nizu *A*. Vreme potrebno za ovaj korak iznosi O (|*B*|), koje je ograničeno sa O (*n*).

(3) Prolazi se kroz *A* s leva na desno. Za svaki element A[*i*], ako je $T_{A[i]-1}$ sufiks tipa *L*, A[*i*] se pomera na početak svoje grupe u nizu *A*. Ovaj korak se naziva *indukovano sortiranje* i zahteva O (*n*) vremena. Na kraju ovog koraka, niz *A* sadrži sve sufikse stringa *T* u leksikografski sortiranom poretku.

Pseudo kod Ko-Aluru algoritma se nalazi u nastavku:

Ko-Aluru(D,T,n,Alf,m,B,k)

```
/* Alf - alfabet, ukupno m slova, zajedno sa Alf[0]='$';
* pretpostavlja se da Alf definise poredak znakova, odnosno Alf[i]<Alf[j] ako je i<j
 * D - string duzine n za koji se trazi sufiksni niz; pretpostavlja se da je D[n-1]=$
 * B - (zadata lista leksikografski sortiranih sufiksa S) sortirana lista (duzine k) indeksa sufiksa tipa S;
pretpostavlja se da je zadata
 * T - niz sufiksa zadatog stringa duzine n
  * pretpostavlja se da je za svaki indeks i u T ustanovljeno da li je T[i] tipa L ili S
   * A - tekuca lista indeksa svih sufiksa; na kraju je to sufiksni niz */
 begin
   for all c in Alf do L[c] = prazna lista  /* inicijalizacija lista */
   for i = 1 to n do  /* razvrstavanje sufiksa prema prvom slovu */
            umetnuti sufiks T[i] u listu L[c], gde je c prvi znak D[i]
   A = konkatenacija svih lista L[c] leksikografskim redosledom
   Neka su granice grupa indeksa sufiksa sa istim pocetnim slovom c, poc[c], kraj[c]
    /* grupe ostaju na svojim mestima do kraja algoritma */
   for i = k downto 1 do
        neka je c = D[B[i]]
        indeks B[i] sufiksa T[B[i]] zameniti sa poslednjim u (svojoj) grupi c u nizu A
        kraj[c] = kraj[c] - 1
   /* nakon prolaska kroz ovu petlju svi sufiksi tipa S  su na svojim tacnim pozicijama u A
   for i =1 to n do                    /* prolaz kroz A sa leva na desno */
      if sufiks T[A[i]-1] je tipa L then
          neka je c = D[A[i]-1] prvi znak tog sufiksa T[A[i]-1]
          if sufiks T[A[i]-1] je tipa L then
             pomeriti indeks A[i]-1 tog sufiksa na pocetak svoje grupe c
              /* ostali indeksi u grupi se pri tome pomeraju, zadrzavajuci svoj poredak */
             poc[c] <- poc[c] + 1
   /* nakon prolaska kroz ovu petlju svi sufiksi su na svojim tacnim pozicijama u A */
 end
```

Naredna lema dokazuje korektnost procedure koraka (3).

Lema 2.3: U koraku (3), kada pretraga dostigne A[*i*], tada se sufiks $T_{A[i]}$ već nalazi na svojoj sortiranoj poziciji u nizu A.

Dokaz: Indukcijom po *i*. Najmanji sufiks u *T* mora biti tipa *S* i samim tim se nalazi na poziciji A[1]. Neka induktivna hipoteza glasi da su A[1], A[2],… A[*i*] prvih *i* sufiksa u sortiranom redosledu. Potrebno je pokazati da kada pretraga dostigne sufiks A[*i+1*] tada je $T_{A[i+1]}$ već na svojoj sortiranoj poziciji u nizu A. Pretpostavimo da nije. Tada postoji sufiks A[*k*] (*k* > *i* + *1*) koji bi trebao da bude na mestu A[*i+1*] u sortiranom redosledu, tj. $T_{A[k]} \prec T_{A[i+1]}$. Kako su svi sufiksi tipa *S* već na svojim ispravnim pozicijama, oba $T_{A[k]}$ i $T_{[i+1]}$ moraju biti tipa *L*. Pošto su

sufiksi u A raspoređeni u grupe na osnovu svog početnog karaktera, a sufiksi se ne premeštaju u druge grupe, to znači da $T_{A[k]}$ i $T_{A[i+1]}$ počinju istim karakterom, recimo c. Neka je $T_{A[i+1]} = c\alpha$ i $T_{A[k]} = c\beta$, $\beta \prec \alpha$. Pošto je $T_{A[k]}$ tipa L, $\beta \prec T_{A[k]}$, a ispravna pozicija sufiksa $T_{A[k]}$ u nizu A je $A[i+1]$, β mora da se pojavi u A[1], A[2],… A[i] . Pošto je $\beta \prec \alpha$, $T_{A[k]}$ je trebalo pomeriti na trenutni kraj grupe pre $T_{A[i+1]}$. Tako, $T_{A[k]}$ ne može da se pojavi u nizu A nakon $T_{A[i+1]}$ i dolazimo do kontradikcije.

Do sada je pokazano da ako su svi sufiksi tipa S sortirani, onda se sortirane pozicije svih sufiksa stringa T mogu odrediti u vremenu O(n). Na sličan način, sortirane pozicije svih sufiksa stringa T mogu se odrediti koristeći sortiran niz sufiksa tipa L. Da bi ovo bilo moguće izvesti, svi sufiksi stringa T grupišu se na osnovu njihovog početnog karaktera u niz A. Zatim se prolazi kroz sortirani niz sufiksa tipa L s leva na desno i utvrđuje njihova tačna pozicija u nizu A pomerajući ih na trenutni početak njihove grupe. Na kraju se kroz niz A prolazi s desna na levo i za svaki $A[i]$, ako je sufiks $T_{A[i]-1}$ tipa S, onda se on pomera na trenutni kraj svoje grupe.

Pošto se za svaki sufiks stringa T ustanovi da li je tipa S ili L, za sortiranje se bira onaj tip koji ima manji broj sufiksa. Bez gubitka opštosti, pretpostavićemo da sufiksa tipa S ima manje. U nastavku sledi objašnjnje samog načina rekurzivnog sortiranja sufiksa tipa S.

Pozicija i u stringu T definiše se kao S pozicija ako je sufiks T_i tipa S, i slično, ako je sufiks T_i tipa L, pozicija i se definiše kao L pozicija. Podstring $t_i...t_j$ se naziva S podstring ako su i i i j S pozicije, a svaka pozicija između i i j je L pozicija.

Cilj je sortirati sve sufikse tipa S u stringu T. Da bi se ovo postiglo, potrebno je prvo sortirati sve S podstringove. Sortiranje generiše grupe, gde se u jednoj grupi nalaze svi podstringovi koji su identični. Grupe se numerišu koristeći uzastopne celobrojne vrednosti počevši od 1. Zatim se generiše novi string T' na sledeći način: prolazi se kroz T s leva na desno i za svaku S poziciju u T, zabeležiti broj grupe S podstringa koji počinje na toj poziciji. Taj niz brojeva grupa formira string T'. U lemi 2.4 se dokazuje da je sortiranje svih sufiksa tipa S u stringu T ekvivalentno sortiranju svih sufiksa stringa T. String T' se sortira rekurzivno.

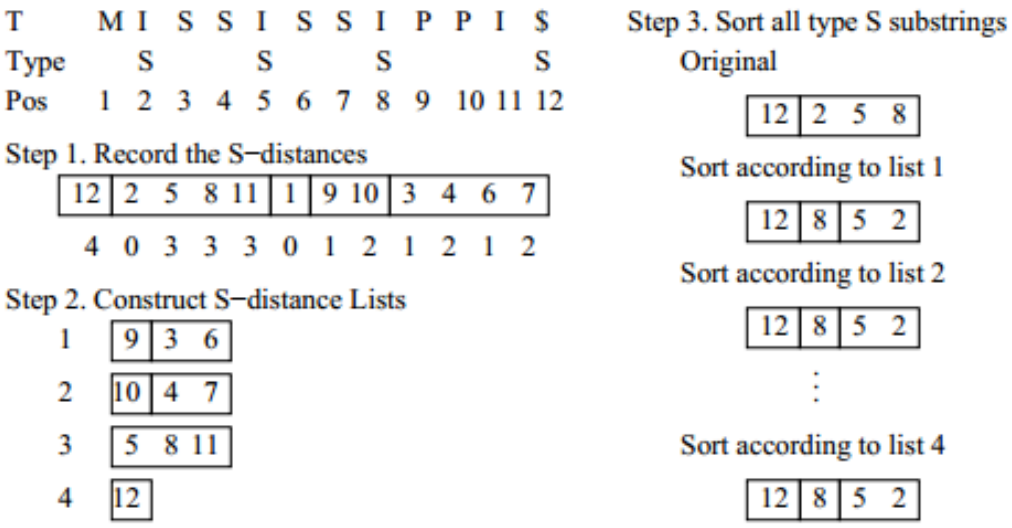
Prvo se pokazuje kako sortirati sve S podstringove u vremenu O(n), uzevši u obzir niz A koji sadrži sve sufikse stringa T grupisane na osnovu njihovog početnog karaktera. Za svaki sufiks T_i treba odrediti njegovo *S-rastojanje*, koje predstavlja udaljenost od startne pozicije i-tog sufiksa do njegove najbliže S pozicije sa leve strane (ne uključujući poziciju i). Ako ni jedna S pozicija ne postoji sa leve strane sufiksa T_i, njegovo *S-rastojanje* se definiše kao 0. Tako

svaki sufiks koji počinje na prvoj *S* poziciji ili pre nje, ima *S-rastojanje* jednako 0. *S* podstringovi se sortiraju na sledeći način (videti primer na slici 5):

(1) Za svaki sufiks u nizu *A* odrediti njegovo *S-rastojanje.* Ovo je moguće uraditi prolazom kroz string *T* s leva na desno beležeći rastojanje od trenutne pozicije do najbliže *S* pozicije sa leve strane. Na poziciji *i*, *S-rastojanje* sufiksa T_i je poznato i beleži se u niz *Dist. S-rastojanje* sufiksa T_i je smešteno u elementu *Dist[i]*. Dakle, *S-rastojanje* za sve sufikse može biti određeno u linearnom vremenu.

(2) Neka je *m* najveće S-rastojanje. Kreirati *m* lista, tako da lista *j (1 ≤ j ≤ m)* sadrži sve sufikse čije je *S-rastojanje* jednako *j* u onom redosledu u kome se pojavljuju u nizu *A.*To može biti odrađeno u linearnom vremenu, prolazom kroz niz A koristeći *Dist[A[i]]* da bi se sufiks $T_{A[i]}$ smestio u ispravnu listu.

(3) Nakon toga se sortiraju *S* podstringovi koristeći liste kreirane u koraku (2). Sortiranje se radi ponavljanjem grupisanja na osnovu novog (narednog) karaktera u svakom koraku. Grupisanje bazirano na prvom karakteru je određeno redosledom kojim se sufiksi tipa *S* pojavljuju u nizu *A*. Pretpostavimo da su *S* podstringovi grupisani na osnovu svojih *j-1* karaktera. Da bi se grupisanje proširilo na osnovu *j* karaktera, prolazi se kroz listu *j*. Za svaki sufiks T_i na koji se naiđe u prolazu kroz grupu liste *j*, pomeriti *S* podstring koji počinje na poziciji t_{i-j} na trenutni početak svoje grupe, a zatim početak grupe pomeriti za jedno mesto u desno. Po završetku prolaza kroz grupu liste *j*, nove granice grupa moraju biti iscrtane između svih *S* podstringova koji su bili pomereni i *S* podstringova koji nisu pomereni. S obzirom da je ukupna veličina svih listi O(*n*), sortiranje *S* podstringova zahteva O(*n*) vremena.



Slika 5. *Ilustracija sortiranja podstringova tipa S stringa MISSISSIPPI$ [5].*

Sortiranje *S* podstringova koristeći opisani algoritam poštuje njihov leksikografski redosled, sa jednim bitnim izuzetkom: ako je *S* podstring prefiks drugog *S* podstringa, broj grupe koji

je dodeljen kraćem podstringu biće veći od broja grupe koji je dodeljen većem podstringu.Ova nepravilnost će biti korišćena u lemi 2.4.

Kao što je pomenuto ranije, sada se konstruiše novi string *T'*, koji odgovara svim *S* podstringovima u nizu *T*. Svaki *S* podstring se zamenjuje brojem grupe u kojoj se nalazi, pa *T'* predstavlja sekvencu brojeva grupa u onom redosledu u kome se *S* podstringovi pojavljuju u stringu *T*. Pošto svaki sufiks tipa *S* u stringu *T* počinje *S* podstringom, postoji uzajamno jednoznačna korespodencija između S sufiksa u stringu *T* i svih sufiksa u *T'* . Neka je T_i sufiks stringa *T* i $T'_{i'}$ njegov odgovarajući sufiks u stringu *T'*. Primećuje se da $T'_{i'}$ može da se dobije iz T_i ako se svaki *S* podstring zameni brojem grupe u kojoj se nalazi u *T*. Slično, T_i može da se dobije iz $T'_{i'}$ zamenom svakog broja grupe njegovim odgovarajućim *S* podstringom i uklanjanjem zajedničkog karaktera koji dele dva uzastopna *S* podstringa. Uklanjanje zajedničkog karaktera se radi zato što je poslednji karakter *S* podstringa takođe prvi karakter narednog *S* podstringa u nizu *T*.

Lema 2.4: Neka su T_i i T_j dva sufiksa stringa *T* i $T'_{i'}$ i $T'_{j'}$ njihovi odgovarajući sufiksi u stringu *T'*. Tada su nejednakosti $T_i < T_j$ i $T'_{i'} < T'_{j'}$.

Dokaz: Prvo se dokazuje da iz nejednakosti $T'_{i'} < T'_{j'}$ sledi da je $T_i < T_j$. Prefiksi sufiksa T_i i T_j koji odgovaraju najvećem zajedničkom prefiksu $T'_{i'}$ i $T'_{j'}$ moraju biti identični. Razlog zbog koga je to tako jeste to što ako su brojevi dve grupe isti, onda odgovarajući podstringovi moraju biti isti. Razmotrićemo „najlevlju“ poziciju na kojoj se $T'_{i'}$ i $T'_{j'}$ razlikuju. Takva pozicija postoji i karakteri (brojevi grupa) stringova $T'_{i'}$ i $T'_{j'}$ na toj poziciji određuju koji od $T'_{i'}$ i $T'_{j'}$ je leksikografski manji. Neka je *k* broj grupe u $T'_{i'}$ a *l* u $T'_{j'}$ na toj poziciji. Neka je α podstring koji odgovara *k*, a β podstring koji odgovara *l*. Primećuje se da α i β mogu imati različite dužine. Bez gubitka opštosti, pretpostavićemo da je $|\alpha| \leq |\beta|$.

Slučaj 1: α nije prefiks β. U ovom slučaju, relativni poredak *k* i *l* određuje leksikografski poredak α i β, koji je isti kao leksikografski poredak T_i i T_j.

Slučaj 2: α jeste prefiks β. Neka je *c* poslednji karakter podstringa α i njegova pozicija u stringu *T* je *S* pozicija. Pozicija karaktera *c* u podstringu β mora biti *L* pozicija.
Pošto dva sufiksa koja počinju na ovim pozicijama imaju isti početni karakter, na osnovu leme 2.2, sufiks tipa *L* mora biti leksikografski manji od sufiksa tipa *S*. Dakle, $T_j \prec T_i$. Na osnovu toga kako su *S* podstringovi sortirani, sledi da je $l < k$. Dakle relativni poredak brojeva grupa i u ovom slučaju tačno određuje leksikografski poredak originalnih sufiksa.

Iz uzajamno jednoznačne korespodencije između sufiksa stringa *T'* i sufiksa tipa *S* stringa *T*, sledi ako je $T_i \prec T_j$ onda je $T'_{i'} \prec T'_{j'}$.

Posledica 2.2: Sortirani poredak sufiksa stringa *T'* određuje sortirani poredak sufiksa tipa *S* stringa *T*.

Dokaz: Neka je $T'_{i1'}$ $T'_{i2'}$ $T'_{i3'}$… sortirani redosled sufiksa stringa *T'*. Neka T_{i1} T_{i2} T_{i3}… predstavlja niz dobijen zamenom svakog sufiksa $T'_{ik'}$ odgovarajućim sufiksom tipa *S* T_{ik}. Tada T_{i1} T_{i2} T_{i3}… predstavlja sortirani redosled sufiksa tipa *S* stringa *T*. Dokaz sledi direktno iz leme 2.4.

Otuda se problem sortiranja sufiksa tipa *S* stringa *T* svodi na problem sortiranja svih sufiksa stringa *T'*. Pošto se string *T'* sastoji od uzastopnih celobrojnih vrednosti počevši od 1, algoritam sufiksnog sortiranja može rekurzivno biti primenjen na string *T'*.

Ako string *T* ima manji broj sufiksa tipa *L* od sufiksa tipa *S*, onda se sufiksi tipa *L* sortiraju koristeći sličnu proceduru. Podstring $t_i...t_j$ se naziva *L* podstring ukoliko su *i* i *j* *L* pozicije, a svaka pozicija u stringu *T* između njih je *S* pozicija. Potrebno je sve sufikse tipa *L* sortirati i na osnovu dobijenog rezultata izgraditi odgovarajući string *T'* zamenom svakog *L* podstringa njegovim brojem grupe. Sortiranjem stringa *T'* dobija se sortirani poredak sufiksa tipa *L*.

Dakle, problem sortiranja sufiksa stringa *T* dužine *n* može da se svede na problem sortiranja sufiksa stringa *T'* dužine najviše $\left\lceil \frac{n}{2} \right\rceil$ i dopunskih operacija složenosti najviše O(n), tj. važi rekurentna relacija:

$$T(n) = T(\left\lceil \frac{n}{2} \right\rceil) + O(n)$$

Posledica ove rekurentne relacije je naredna teorema.

Teorema 2.1: Prostorna i vremenska složenost algoritma Ko-Aluru je O(*n*) [5].

2.3.2. Algoritam SA-IS

Algoritam SA-IS su objavili Nong i drugi autori 2008. godine [8], i predstavlja trenutno najpoznatiji algoritam linearne vremenske složenosti za sortiranje sufiksih nizova.

Do tog trenutka, najpoznatiji algoritam za sortiranje sufiksnih nizova je bio Ko-Aluru algoritam. Elegancija Ko-Aluru algoritma leži u njegovom trećem koraku, gde je uveden novi koncept nazvan indukovano sortiranje, koji efikasno sortira neredukovane probleme na svim nivoima rekurzivnih poziva. Medjutim, mana Ko-Aluru algoritma je prilično komplikovan prvi korak u kome je potrebno sortirati sve *S* podstringove različitih dužina i zameniti ih njihovim sortiranim indeksima. Za ovaj zadatak, Ko i Aluru su predložili korišćenje listi u koje se smeštaju svi karakteri koji imaju isto *S-rastojanje*. Ovakvo rešenje ne samo da zahteva dodatni prostor, već ima izuzetno komplikovanu programsku implementaciju.

SA-IS algoritam se razlikuje od Ko-Aluru algoritma u dva aspekta. Prvo, SA-IS koristi samo najlevije *S* (*LMS*) podstringove (koji će biti formalno definisani u daljem tekstu), umesto *S* podstringova koje koristi Ko-Aluru algoritam. Pošto su *LMS* podstringovi najčešće duži od *S* podstringova, SA-IS može da redukuje problem brže od Ko-Aluru algoritma. Drugo, SA-IS ne koristi indukovano sortiranje samo u trećem koraku da indukuje redosled neredukovanog problema kako je to slučaj kod Ko-Aluru algoritma, već i u prvom koraku da indukuje redosled redukovanog problema.

Pseudo kod SA-IS algoritma, koji će biti opisan u daljem tekstu, nalazi se u nastavku:

SA-IS (S, SA)

/* *T* je ulazni string, *A* je izlaz koji predstavlja sufiksni niz stringa *T* */
/* *t:* niz logičkih vrednosti *[1,… n]* koji čuva informaciju o tipu sufiksa (S/L)
**n*$_1$: broj LMS podstringova
* *S*$_1$: niz celobrojnih vrednosti *[1,… n*$_1$*]* koji sadrži indekse LMS podstringova u sortiranom redosledu
**P*$_1$*:* niz celobrojnih vrednosti *[1,… n*$_1$*]* u kome se čuvaju indeksi početaka LMS podstringova stringa T */

Begin

/* Redukcija originalnog problema */

1. Proći kroz string *T* s leva na desno, i svaki karakter klasifikovati na tip *S* ili tip *L*. Rezultat smestiti u niz *t*
2. Proći kroz niz *t* s desna na levo, i pronaći sve *LMS* podstringove. Rezultat smestiti u niz *P*$_1$
3. Primeniti indukovano sortiranje na niz *P*$_1$
4. LMS podstringove iz *T* razvrstati u grupe prema prvom znaku, a zatim redne brojeve tih grupa smestiti u niz S_1

/* Rekurzivno rešavanje problema */

5. **if** svaki karakter u stringu *S*$_1$ jedinstven **then**
6. Direktno izračunati *A*$_1$ iz *S*$_1$;
7. **else**
8. SA-IS(S_1, A_1); *// rekurzivni poziv*

/* Formiranje sufiksnog niza */

9. Formirati *A* na osnovu sufiksnog niza *A*$_1$ niza *S*$_1$
10. **return**

End

Rezultat prve četiri linije pseudo koda predstavlja redukovan originalni problem, koji se zatim rekurzivno rešava (linije 5 – 8). Na kraju se iz rešenja redukovanog problema formira rešenje originalnog problema (linija 9).

Redukcija problema

Definicija 2.1 *(LMS karakter):* Karakter *T[i]* stringa *T*, *i* ∈ [2, n], predstavlja najleviji karakter tipa *S (LMS)*, ako je *T[i]* karakter tipa *S*, i *T[i-1]* karakter tipa *L*.

Definicija 2.2 *(LMS podstring):* Podstring *T[i, j]* naziva se *LMS* podstring ako:

1) *i≠j*, *T[i]* i *T[j]* su *LMS* karakteri, i ni jedan drugi karakter između njih nije *LMS* karakter
2) *i=j*, sastoji se samo od poslednjeg karaktera stringa *T* ($)

Intuitivno, ako se *LMS* podstringovi posmatraju kao osnovni blokovi stringa i ako se mogu efikasno sortirati, onda je moguće svaki *LMS* podstring zameniti njegovim indeksom u sortiranom redosledu i tako formirati kraći string S_1. Na taj način se redukuje veličina ulaznog stringa i olakšava rešavanje problema.

Da bi se odredio sortirani redosled bilo koja dva *LMS* podstringa, potrebno je leksikografski uporediti odgovarajuće parove karaktera, prolazom kroz podstringove s leva na desno. Ukoliko su neka dva karaktera identična, radi se poređenje po tipu, gde karakter tipa *S* ima viši prioritet od karaktera tipa *L*. Odavde sledi da dva *LMS* podstringa mogu imati isti indeks sortiranja ukoliko su iste dužine, imaju identične odgovarajuće karaktere i tipovi odgovarajućih karaktera se poklapaju.

Da bi se sortirali svi *LMS* podstringovi stringa *T*, nije potrebno rezervisati dodatni fizički prostor. Dovoljno je održavati niz indeksa P_1, koji sadrži indekse početaka *LMS* podstringova stringa *T*. Ovaj korak može biti odrađen jednim prolazom kroz string *T* s desna na levo u vremenu O(*n*).

Definicija 2.3 *(Niz pokazivača):* P_1 predstavlja niz indeksa na sve *LMS* podstringove stringa *T*, u onom poretku u kome se pojavljuju u stringu *T*.

Pretpostavimo da su svi *LMS* podstringovi stringa *T* leksikografski sortirani u grupe, gde su svi podstringovi u jednoj grupi identični. Formira se novi niz S_1 tako da je $S_1[i]$ redni broj grupe kojoj pripada *LMS* podstring sa indeksom $P_1[i]$, *i* = [1,..*n*]. Odavde, dva podstringa iste

dužine $T[i.\ j]$ i $T[i', j']$ su identična ako i samo ako $T[i + k] = T[i' + k]$ i $t[i + k] = t[i' + k]$, za $k \in [1, j - i]$.

Lema 2.5: Dužina stringa S_1 (n_1) je najviše polovina dužine stringa $T(n)$, tj. $n_1 \leq \left\lfloor \frac{n}{2} \right\rfloor$

Dokaz*:* Iz definicije *LMS* karaktera sledi da prvi karakter stringa *T* ne može biti *LMS* karakter, kao ni bilo koja dva uzastopna karaktera stringa *T* ne mogu oba biti *LMS* karakteri.

Lema 2.6: Poslednji karakter stringa S_1 je jedinstven i leksikografski najmanji karakter stringa.

Dokaz*:* Iz definicije *LMS* karaktera, *T[n-1]* mora biti *LMS* karakter, i *LMS* podstring koji počinje na poziciji *T[n-1]* mora biti najmanji među svim *LMS* podstringovima na koje pokazuje niz P_1.

Lema 2.7*:* Ako je $S_1[i] = S_1[j]$, tada je $P_1[i+1] - P_1[i] = P_1[j+1] - P_1[j]$.

Dokaz: Ako je $S_1[i] = S_1[j]$, tada iz definicije stringa S_1 mora da važi:

(1) $T[P_1[i], P_1[i+1]] = T[P_1[j], P_1[j+1]]$
(2) $t[P_1[i], P_1[i+1]] = t[P_1[j], P_1[j+1]]$

Dakle, dva *LMS* podstringa stringa *T* koji počinju na pozicijama $T[P_1[i]]$ i $T[P_1[j]]$, moraju biti iste dužine.

Lema 2.8: Relativni poredak bilo koja dva sufiksa $S_1[i]$ i $S_1[j]$ u stringu S_1, isti je kao poredak sufiksa $T[P_1[i]]$ i $T[P_1[j]]$ u stringu *T*.

Dokaz*:* Razmatraju se sledeća dva slučaja:

(1) $S_1[i] \neq S_1[j]$. Tada postoji par karaktera u dva podstringa stringa *T* koji imaju različite leksikografske vrednosti ili pripadaju različitim tipovima karaktera. Ako dva karaktera imaju različite leksikografske vrednosti, jasno je da je tvrđenje tačno. Ako dva karaktera pripadaju različitim tipovima, tada karakter tipa *S* ima viši prioritet, pa je početno tvrđenje tačno.
(2) $S_1[i] = S_1[j]$. U ovom slučaju, poredak sufiksa $S_1[i]$ i $S_1[j]$ je određen poretkom sufiksa $S_1[i+1]$ i $S_1[j+1]$. Isti argument se može rekurzivno primeniti na $S_1[i+1] = S_1[j+1] \ldots S_1[i+k-1] = S_1[j+k-1]$, dokle god $S_1[i+k] \neq S_1[j+k]$. Pošto je $S_1[i, i+k-1] = S_1[j, j+k-1]$, iz leme 2.7 sledi $P_1[i+k] - P_1[i] = P_1[j+k] - P_1[j]$, tj. dužine podstringova $T[P_1[i], P_1[i+k]]$ i $T[P_1[j], P_1[j+k]]$ su iste. Tako, sortiranje $S_1[i, i+k]$ i $S_1[j, j+k]$ je jednako sortiranju $T[P_1[i], P_1[i+k]]$ i $T[P_1[j], P1[j+k]]$. Otuda je tvrđenje tačno.

Iz leme 2.8 se zaključuje da se sortiranje *LMS* sufiksa stringa *T* može svesti na sortiranje stringa S_1. Pošto je dužina stringa S_1 najviše $\frac{1}{2}$ dužine stringa *T*, sortiranje stringa S_1 ima složenost najviše $\frac{1}{2}$ složenosti sortiranja stringa *T*.

Neka *A* predstavlja oznaku za sufiksni niz stringa *T*, a A_1 oznaku za sufiksni niz stringa S_1. Pretpostavimo da je A_1 određeno. U nastavku će biti opisano određivanje sufiksnog niza *A* na osnovu sufiksnog niza A_1 u linearnom vremenu i prostoru.

Određivanje A iz A_1

Kao što je definisano ranije, niz *A* sadrži indekse svih sufiksa stringa *T* u njihovom leksikografskom poretku. Trivijalno, indeksi sufiksa stringa *T* koji imaju isti početni karakter, uzastopno su poređani u sufiksnom nizu *A*. U skladu sa tim, niz *A* se deli na podnizove (grupe), gde se u jednoj grupi nalaze svi sufiksi koji imaju isti početni karakter. U jednoj grupi, svi sufiksi istog tipa su grupisani na jednom mestu, tj. svi sufiksi tipa *S* nalaze se sa desne strane sufiksa tipa *L*. Odatle, svaka grupa se dalje može podeliti na dve podgrupe, gde prva sadrži sve sufikse tipa *L* koji imaju isti početni karakter, a druga sve sufikse tipa *S* koji imaju isti početni karakter.

Algoritam za indukovanje niza *A* iz A_1:

(1) Pronaći kraj svake grupe tipa *S*, i svaki član niza A_1 smestiti u odgovarajuću grupu tipa *S* niza *A*, čuvajući poredak u kome se članovi pojavljuju u nizu A_1.
(2) Pronaći početak svake grupe tipa *L*. Proći kroz niz *A* s leva na desno, i za svaki član *A[i]*, ako je $T[A[i]-1]$ karakter tipa *L*, smestiti *A[i]-1* na trenutni početak grupe tipa *L* za *T[A[i]-1]*, i trenutni početak pomeriti za jedno mesto u desno.
(3) Pronaći kraj svake grupe tipa *S* u nizu *A*. Proći kroz niz *A* s desna na levo, i za svaki član *A[i]*, ako je $T[A[i]-1]$ karakter tipa *S*, smestiti *A[i]-1* na trenutni kraj grupe tipa S za *T[A[i]-1]*, i trenutni kraj grupe pomeriti za jedno mesto u desno.

Očigledno, složenost svakaog koraka algoritma je O(n). Da bi se dokazala korektnost navedenog algoritma, potrebno je razmotriti svaki od navedena tri koraka. Koraci će biti razmatrani u obrnutom redosledu.

Korektnost trećeg koraka, u kome se opisuje sortiranje svih sufiksa stringa *S* na osnovu induktivno sortiranih sufiksa tipa *L*, ustanovljena je lemom 3 u [5] koja u ovom radu neće biti dokazivana.

Lema 2.9*:* Ako su dati sufiksi tipa *L* (ili tipa *S*) stringa *T* u sortiranom redosledu, onda se svi sufiksi stringa *T* mogu sortirati u vremenu O(*n*).

U kontekstu opisanog algoritma, lema 2.9 može da se preformuliše na sledeći način:

Lema 2.10*:* Ako su dati svi sufiksi tipa *L* stringa *T* u sortiranom redosledu, onda se svi sufiksi stringa *T,* u trećem koraku algoritma, mogu sortirati u vremenu O(*n*).

Naredna lema dokazuje korektnost drugog koraka.

Lema 2.11*:* Ako su dati svi *LMS* sufiksi stringa *T* u sortiranom redosledu, onda se svi sufiksi tipa *L* stringa *T,* u drugom koraku algoritma, mogu sortirati u vremenu O(*n*).

Dokaz*:* Na osnovu leme 2.9, ako su svi sufiksi tipa *S* dati u sortiranom redosledu, tada se svi sufiksi (tipa *S* i tipa *L*) mogu sortirati kroz indukciju jednim prolaskom kroz niz *A* s leva na desno. Međutim, ne može svaki sufiks tipa *S* da se iskoristi za indukovano sortiranje sufiksa tipa *L*. Odnosno, sufiks tipa *S* je koristan za sortiranje sufiksa tipa *L*, samo ako je ujedno i *LMS* sufiks. Drugim rečima, ispravan redosled svih *LMS* sufiksa je dovoljan da indukuje sortirani redosled svih sufiksa tipa *L* u vremenu i prostoru O(*n*).

U prvom koraku algoritma, svi *LMS* sufiksi se smeštaju u odgovarajuće grupe tipa *S* u nizu *A*, od kraja ka početku. Dakle, na osnovu leme 2.11, u drugom koraku algoritma, svi sufiksi tipa *L* će biti ispravno sortirani, a na osnovu leme 2.10, u trećem koraku algoritma, svi sufiksi stringa *T* će biti ispravno sortirani na osnovu sortiranih sufiksa tipa *L*.

Indukovano sortiranje podstringova

Ideja je da se indukovano sortiranje koje se koristi u indukovanju *A* iz A_1 proširi i na problem sortiranja podstringova različitih dužina. Jedina razlika u odnosu na algoritam za indukovanje niza *A* iz A_1 je u prvom koraku algoritma:

(1) Pronaći kraj svake grupe tipa *S*, i svaki *LMS* sufiks stringa *T* smestiti u odgovarajuću grupu tipa *S* niza *A* u skladu sa njegovim početnim karakterom.

Da bi se olakšala diskusija koja sledi u nastavku, definiše se pojam *LMS* prefiksa.

Definicija 2.4 *(LMS prefiks)*: *LMS* prefiks *pre(T, i)* predstavlja:

(1) jedan *LMS* karakter
(2) prefiks *T*[*i, k*] u sufiksu T_i gde je *T[k]* prvi *LMS* karakter nakon *T[i]*

LMS prefiks *pre(T, i)* može biti tipa *S* ili tipa *L*, u zavisnosti od toga da li je T_i sufiks tipa *S* ili tipa *L*.

Iz definicije *LMS* prefiksa se zaključuje da bilo koji *LMS* prefiks tipa *L* ima najmanje dva karaktera. Naredna teorema potvrđuje korektnost modifikovanog algoritma za indukovano sortiranje svih *LMS* prefiksa dužine veće od 1.

Teorema 2.2*:* Modifikovani algoritam indukovanog sortiranja ispravno sortira sve *LMS* prefikse dužine veće od 1 i leksikografski najmanji karakter $.

Dokaz*:* Inicijalno, u prvom koraku algoritma, svi *LMS* prefiksi tipa *S* dužine 1 su raspoređeni u odgovarajuće grupe u nizu *A*. Nakon prvog koraka, svi *LMS* prefiksi koji se nalaze u nizu *A*, biće sortirani u odgovarajućem poretku.

Sada je potrebno dokazati korektost drugog koraka algoritma, tj. drugi korak algoritma će ispravno sortirati sve *LMS* prefikse tipa *L*. Dokazuje se indukcijom.

Kada se prvi *LMS* prefiks tipa *L* smesti u odgovarajuću grupu u nizu *A*, on je ispravno sortiran zajedno sa svim *LMS* prefiksima tipa *S* koji se nalaze u njegovoj grupi. Ako se pretpostavi da drugi korak algoritma ispravno sortira *k LMS* podstringova tipa *L*, gde je $k \geq 1$, potrebno je dokazati da kada se *(k+1)-i LMS* prefiks tipa *L* smesti u odgovarajuću grupu u nizu *A*, da će se nalaziti na ispravnoj sortiranoj poziciji. Ako se pretpostavi suprotno, tj. kada se *(k+1)-i LMS* prefiks *pre(T, i)* tipa *L* doda na trenutni početak svoje grupe u nizu *A*, da se tada u njegovoj grupi nalazi neki drugi *LMS* prefiks *pre(T, j)* tipa *L* koji je leksikografski veći od njega, i nalazi se sa leve strane *pre(T, i)*. U tom slučaju mora da važi *T[i] = T[j], pre(T, j+1) > pre(T, i+1)* i *pre(T, j+1) se* nalazi ispred *pre(T, i+1)* u nizu *A*. To znači da se prilikom prolaska kroz niz *A* s leva na desno, pre ubacivanja *pre(T, i)* u odgovarajuću grupu, morao uočiti *LMS* prefiks koji nije ispravno sortiran. To je u kontradikciji sa pretpostavkom. Kao rezultat, *LMS* prefiksi tipa *S* dužine 1, i svi *LMS* prefiksi tipa *L*, biće sortirani u ispravnom poretku nakon drugog koraka algoritma.

Na kraju je potrebno dokazati korektnost trećeg koraka algoritma, čiji je rezultat sortiran niz svih *LMS* prefiksa dužine veće od 1. Kada se prvi *LMS* prefiks tipa *S* smesti u odgovarajuću grupu, on je ispravno sortiran zajedno sa svim *LMS* prefiksima tipa *L* koji se nalaze u njegovoj grupi. U prvom koraku algoritma, svi *LMS* prefiksi dužine 1 su smešteni na krajeve odgovarajućih grupa u nizu *A*. Prema tome, u ovom koraku, kada se doda *LMS* prefiks tipa *S* dužine veće od 1 u odgovarajuću grupu, on će da „pregazi“ *LMS* prefiks tipa *S* dužine 1 ako je bio na trenutnom kraju grupe. Ako se pretpostavi da treći korak algoritma ispravno sortira *k*

LMS prefiksa tipa *S*, gde je $k \geq 1$, potrebno je dokazati da kada se *(k+1)-i LMS* prefiks tipa *S* smesti u odgovarajuću grupu u nizu *A*, da će se nalaziti na ispravnoj sortiranoj poziciji. Ako se pretpostavi suprotno, tj. kada se *(k+1)-i LMS* prefiks *pre(T, i)* tipa *S* doda na trenutni kraj svoje grupe u nizu *A*, da se tada u njegovoj grupi nalazi neki drugi *LMS* prefiks *pre(T, j)* tipa *S* koji je leksikografski manji od njega, i nalazi se sa desne strane *pre(T, i)*. U tom slučaju mora da važi $T[i] = T[j]$, $pre(T, j+1) < pre(T, i+1)$ i *pre(T, j+1) se* nalazi iza *pre(T, i+1)* u nizu *A*. To znači da se prilikom prolaska kroz niz *A* s desna na levo, pre ubacivanja *pre(T, i)* u odgovarajuću grupu, morao uočiti *LMS* prefiks dužine veće od 1 koji nije ispravno sortiran. To je u kontradikciji sa pretpostavkom. Kao rezultat trećeg koraka algoritma, svi *LMS* prefiksi dužine veće od 1 i leksikografski najmanji karakter $ (koji je raspoređen u svoju grupu u prvom koraku) će biti sortirani u ispravnom poretku.

Iz teoreme 2.2 se mogu izvesti sledeći zaključci:

1) Bilo koji *LMS* podstring je *LMS* prefiks dužine veće od 1 ili leksikografski najmanji karakter $; ako su svi *LMS* prefiksi dužine veće od 1 i leksikografski najmanji karakter $ sortirani, onda su i svi *LMS* podstringovi sortirani,
2) Svaki *S* podstring je prefiks nekog *LMS* prefiksa dužine veće od 1 ili leksikografski najmanjieg karaktera $; ako su svi *LMS* prefiksi dužine veće od 1 i leksikografski najmanji karakter $ sortirani, onda su i svi *S* podstringovi sortirani takođe.

Otuda, algoritam indukovanog sortiranja *LMS* prefiksa dužine veće od 1 i leksikografski najmanjeg karaktera $ može da se iskoristi za sortiranje *LMS* podstringova u SA-IS algoritmu, kao i za sortiranje *S* ili *L* podstringova u Ko-Aluru algoritmu.

Složenost algoritma

Teorema 2.3 *(vremenska/prostorna složenost):* Ako je dat string *T* nad konstantnim ili celobrojnim alfabetom, vremenska i prostorna složenost SA-IS algoritma za računanje sufiksnog niza *A(T)* su redom O (*n*) i O(*n log n*) bita.

Dokaz: S obzirom da je u svakom rekurzivnom pozivu algoritma koeficijent redukcije najmanje $\frac{1}{2}$, vremenska složenost algoritma regulisana je jednačinom ispod, gde je veličina redukovanog problema najviše $\left\lfloor \frac{n}{2} \right\rfloor$.

$$T(n) = T(\left\lfloor \frac{n}{2} \right\rfloor) + O(n) = O(n)$$

Prvo O(n) u rekurzivnoj jednačini se odnosi na redukovanje problema i indukovanje finalnog rešenja iz redukovanog problema.

Kada je u pitanju prostorna složenost, najviše memorije je neophodno za skladištenje sufiksnog niza za redukovani problem u svakoj iteraciji. Pošto je veličina sufiksnog niza u prvoj iteraciji najveća i ograničena sa $n \lceil \log n \rceil$ bita, a u svakoj narednoj iteraciji se smanjuje bar za polovinu, jasno je da je prostorna složenost algoritma O($n \log n$) bita.

Kao što je ranije pomenuto, sortiranje S ili L podstringova različitih dužina predstavlja problematičan korak Ko-Aluru algoritma, čije rešavanje zahteva korišćenje listi sa S-rastojanjima i odgovarajući algoritam koji obrađuje ove strukture podataka. Međutim, ovaj naizgled težak problem, jednostavno može biti rešen ako se koristi idukovano sortiranje.

3. PROGRAMSKA REALIZACIJA I REZULTATI

U dodatku rada nalazi se modul *LinearSuffixSort.c* koji implementira rad dva algoritma za konstrukciju sufiksnog niza, algoritma Ko-Aluru i algoritma SA-IS. Aplikacija je konzolna, pisana u programskom jeziku C, i služi za nalaženje sufiksnog niza A zadatog stringa T. String se zadaje neposredno, preko standardnog ulaza.

U narednom segmentu sledi uputstvo za pravilno pokretanje i korišćenje programa.

3.1. Uputstvo za pokretanje i korišćenje programa

Za kompajliranje programa koristi se GNU GCC kompajler. Komanda kojom se kompajlira je: *gcc -o LinearSuffixSort LinearSuffixSort.c*. Program je moguće pokrenuti komandom *./LinearSuffixSort*.

Nakon pokretanja programa, od korisnika se zahteva unos dužine stringa za koji je potrebno konstruisati sufiksni niz, unos stringa i izbor jednog od dva ponuđena algoritma.

Primer 3.1:
Da bi se dobio sufiksni niz stringa T = "*acbcacab*" dužine 8 karaktera, posle pokretanja programa na standardnom ulazu je potrebno uneti:
Unesite duzinu stringa:
8
Unesite string T:
acbcacab

Unesite broj 1 ukoliko zelite da konstruisete sufiksni niz koristeci algoritam Ko-Aluru ili broj 2 koristeci SA-IS algoritam:
1

3.2. Funkcije koje se koriste u programu:

Imena i namena funkcija koje se koriste u algoritmu Ko-Aluru navedene su u sledećem spisku:

- raspodelaSL – određuje tip svakog karaktera prosleđenog stringa
- brojSL – vraća broj S ili L tipova karaktera u prosleđenom stringu
- rastojanjeS – za svaki karakter stringa određuje njegovo S-rastojanje i vraća maksimalno S-rastojanje
- rastojanjeL – za svaki karakter stringa određuje njegovo L-rastojanje i vraća maksimalno L-rastojanje
- vratiNizPodstringovaS – vraća niz indeksa početnih pozicija S podstringova, u redosledu u kome se nalaze nakon sortiranja po početnom karakteru
- vratiNizPodstringovaL – vraća niz indeksa početnih pozicija L podstringova, u redosledu u kome se nalaze nakon sortiranja po početnom karakteru
- vratiGranicePodstringovaS – vraća gornje granice grupa niza koji sadrži S podstringove
- vratiGranicePodstringovaL – vraća gornje granice grupa niza koji sadrži L podstringove
- konstruisiSAtipaL – konstruiše sufiksni niz koristeći sortiran niz sufiksa tipa L
- konstruisiSAtipaS – konstruiše sufiksni niz koristeći sortiran niz sufiksa tipa S
- rekonstruisiBtipS – rekonstruiše niz sortiranih sufiksa tipa S iz sufiksnog niza stringa T’
- rekonstruisiBtipL – rekonstruiše niz sortiranih sufiksa tipa L iz sufiksnog niza stringa T’

- **KoAluru – rekurzivna funkcija koja vraća sufiksni niz zadatog stringa po algoritmu**

Imena i namena funkcija koje se koriste u algoritmu SA-IS navedene su u sledećem spisku:

- saisRaspodelaSL - određuje tip svakog karaktera prosleđenog stringa
- saisBrojSL - vraća broj S ili L tipova karaktera u prosleđenom stringu
- saisBrojLMS – vraća broj LMS karaktera u prosleđenom stringu
- saisLMS – vraća niz indeksa LMS karaktera prosleđenog stringa
- saisIzracunajSAizS – odrađuje prvi deo SA-IS algoritma (konstruisanje sufiksnog niza na osnovu prosleđenog stringa)

- saisIndukujSAizSA1 – indukuje sufiksni niz originalnog stringa na osnovu sufiksnog niza redukovanog stringa

- **SAIS - rekurzivna funkcija koja vraća sufiksni niz zadatog stringa po algoritmu**

Imena i namena zajedničkih funkcija za oba algoritma:

- Main – glavna funkcija koja upravlja programom
- vratiBrojBucketa – vraća broj različitih grupa u zavisnosti od početnih karaktera sufiksa prosleđenog stringa
- pocetnoSortiranje – radi sortiranje sufiksnog niza na osnovu početnih karaktera sufiksa, koristeći „bucket sort“
- vratiGraniceGrupa – vraća niz sa indeksima gornjih granica grupa

3.3. Izlaz programa

Korisnik mora imati na umu uštedu vremena i prostora prilikom ispisa rezultata na standardnom izlazu, pa shodno tome, samo za stringove koji imaju dužinu manju od 1000 karaktera, izlazne parametre programa predstavljaju svi koraci koji su karakteristični za izabran algoritam za konstrukciju sufiksnog niza i konačan sufiksni niz A. Ukoliko je dužina veća od 1000 karaktera, na standardnom izlazu se se ne ispisuje dobijeni sufiksni niz sa karakterističnim koracima za izabrani algoritam.

Primer 3.2: Konstrukcija sufiksnog niza dužine 8 karaktera, koristeći algoritam Ko-Aluru

Ulaz: String T = acbcacab, dužine 8 karaktera, izabrani algoritam Ko-Aluru

Izlaz: A = (6, 4, 0, 7, 2, 5, 3, 1)

Prikaz rezultata na ekranu za dati string:

Raspodela karaktera stringa T=acbcacab$ na tip S i tip L:

1 0 1 0 1 0 1 0 0

Pocetno sortiranje (bucket sort-om) indeksa stringa T=acbcacab$:

8 0 4 6 2 7 1 3 5

LISTE SA L-UDALJENOSTIMA I NIZ L-PODSTRINGOVA ZA STRING T=acbcacab$ NAKON PROLASKA KROZ LISTU i:

L-Udaljenost: 1

Lista indeksa: 8 4 6 2

Niz L-podstringova nakon prolaska kroz listu: 8 7 3 5

L-Udaljenost: 2

Lista indeksa: 7 3 5

Niz L-podstringova nakon prolaska kroz listu: 8 7 5 3

SORTIRAN NIZ L-SUFIKSA ZA STRING T=acbcacab$: 8 7 5 3 1

SORTIRANJE SUFIKSNOG NIZA A ZA STRING T=acbcacab$ NA OSNOVU SORTIRANOG NIZA L-SUFIKSA:

Sortiranje sufiksnog niza A stringa T=acbcacab$ na osnovu L - sufiksa 8:

8 0 4 6 2 7 1 3 5

Sortiranje sufiksnog niza A stringa T=acbcacab$ na osnovu L - sufiksa 7:

8 0 4 6 7 2 1 3 5

Sortiranje sufiksnog niza A stringa T=acbcacab$ na osnovu L - sufiksa 5:

8 0 4 6 7 2 5 3 1

Sortiranje sufiksnog niza A stringa T=acbcacab$ na osnovu L - sufiksa 3:

8 0 4 6 7 2 5 3 1

Sortiranje sufiksnog niza A stringa T=acbcacab$ na osnovu L - sufiksa 1:

8 0 4 6 7 2 5 3 1

SORTIRANJE SUFIKSNOG NIZA A STRINGA T = acbcacab$ NA OSNOVU S-SUFIKSA:

Sortiranje sufiksnog niza A stringa T=acbcacab$ na osnovu S - sufiksa 1:

8 6 4 0 7 2 5 3 1

Sortiranje sufiksnog niza A stringa T=acbcacab$ na osnovu S - sufiksa 5:

8 6 4 0 7 2 5 3 1

KONACNI SORTIRANI NIZ SUFIKSA STRINGA T=acbcacab JE:

A = 6 4 0 7 2 5 3 1

Primer 3.3: Konstrukcija sufiksnog niza dužine 8 karaktera, koristeći algoritam SA-IS

Ulaz: String T = acbcacab, dužine 8 karaktera, izabrani algoritam SA-IS

Izlaz: A = (6, 4, 0, 7, 2, 5, 3, 1)

Prikaz rezultata na ekranu za dati string:

Raspodela sufiksa stringa T=acbcacab$ na S i L tip:

1 0 1 0 1 0 1 0 1

Niz indeksa LMS sufiksa stringa T=acbcacab$: 2 4 6 8

ODREĐIVANJE SUFIKSNOG NIZA A IZ STRINGA T=acbcacab$:

Prvi korak: inicijalizacija sufiksnog niza A stringa T=acbcacab$ na -1 i raspoređivanje LMS sufiksa na krajeve odgovarajućih grupa:

A = 8 -1 6 4 -1 2 -1 -1 -1

Drugi korak: Prolazak kroz sufiksni niz A stringa T=acbcacab$ s leva na desno, i raspoređivanje sufiksa A[i] - 1 tipa L na početke odgovarajućih grupa:

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa L A[i]-1=7:

A = 8 -1 6 4 7 2 -1 -1 -1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa L A[i]-1=5:

A = 8 -1 6 4 7 2 5 -1 -1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa L A[i]-1=3:

A = 8 -1 6 4 7 2 5 3 -1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa L A[i]-1=1:

A = 8 -1 6 4 7 2 5 3 1

Treći korak: Prolazak kroz sufiksni niz A stringa T=acbcacab$ s desna na levo, i raspoređivanje sufiksa A[i] - 1 tipa S na krajeve odgovarajućih grupa:

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa S A[i]-1=0:

A = 8 -1 6 0 7 2 5 3 1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa S A[i]-1=2:

A = 8 -1 6 0 7 2 5 3 1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa S A[i]-1=4:

A = 8 -1 4 0 7 2 5 3 1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa S A[i]-1=6:

A = 8 6 4 0 7 2 5 3 1

Niz TPrime stringa T=acbcacab$ je TPrime=3210

ODREĐIVANJE SUFIKSNOG NIZA A IZ STRINGA T=3210$:

Prvi korak: inicijalizacija sufiksnog niza A stringa T=3210$ na -1 i raspoređivanje LMS sufiksa na krajeve odgovarajucih grupa:

A = 4 -1 -1 -1 -1

Drugi korak: Prolazak kroz sufiksni niz A stringa T=3210$ s leva na desno, i raspoređivanje sufiksa A[i] - 1 tipa L na početke odgovarajućih grupa:

Sufiksni niz A stringa T=3210$ nakon smeštanja sufiksa tipa L A[i]-1=3:

A = 4 3 -1 -1 -1

Sufiksni niz A stringa T=3210$ nakon smeštanja sufiksa tipa L A[i]-1=2:

A = 4 3 2 -1 -1

Sufiksni niz A stringa T=3210$ nakon smeštanja sufiksa tipa L A[i]-1=1:

A = 4 3 2 1 -1

Sufiksni niz A stringa T=3210$ nakon smeštanja sufiksa tipa L A[i]-1=0:

A = 4 3 2 1 0

Niz A1 stringa TPrime=3210 je: A1 = 3 2 1 0

INDUKOVANJE STRINGA A IZ STRINGA A1:

Prvi korak: Inicijalizacija elemenata niza A na -1. Prolazak kroz niz A1 s desna na levo i smestanje LMS sufiksa P1[A1[i]] na odgovarajuće krajeve grupa u nizu A:

A = 8 -1 6 4 -1 2 -1 -1 -1

Drugi korak: Prolazak kroz sufiksni niz A stringa T=acbcacab$ s leva na desno, i raspoređivanje sufiksa A[i] - 1 tipa L na početke odgovarajućih grupa:

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa L A[i]-1=7:

A = 8 -1 6 4 7 2 -1 -1 -1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa L A[i]-1=5:

A = 8 -1 6 4 7 2 5 -1 -1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa L A[i]-1=3:

A = 8 -1 6 4 7 2 5 3 -1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa L A[i]-1=1:

A = 8 -1 6 4 7 2 5 3 1

Treći korak: Prolazak kroz sufiksni niz A stringa T=acbcacab$ s desna na levo, i raspoređivanje sufiksa A[i] - 1 tipa S na krajeve odgovarajućih grupa:

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa S A[i]-1=0:

A = 8 -1 6 0 7 2 5 3 1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa S A[i]-1=2:

A = 8 -1 6 0 7 2 5 3 1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa S A[i]-1=4:

A = 8 -1 4 0 7 2 5 3 1

Sufiksni niz A stringa T=acbcacab$ nakon smeštanja sufiksa tipa S A[i]-1=6:

A = 8 6 4 0 7 2 5 3 1

KONAČNI SORTIRANI NIZ SUFIKSA STRINGA T=acbcacab JE:

A = 6 4 0 7 2 5 3 1

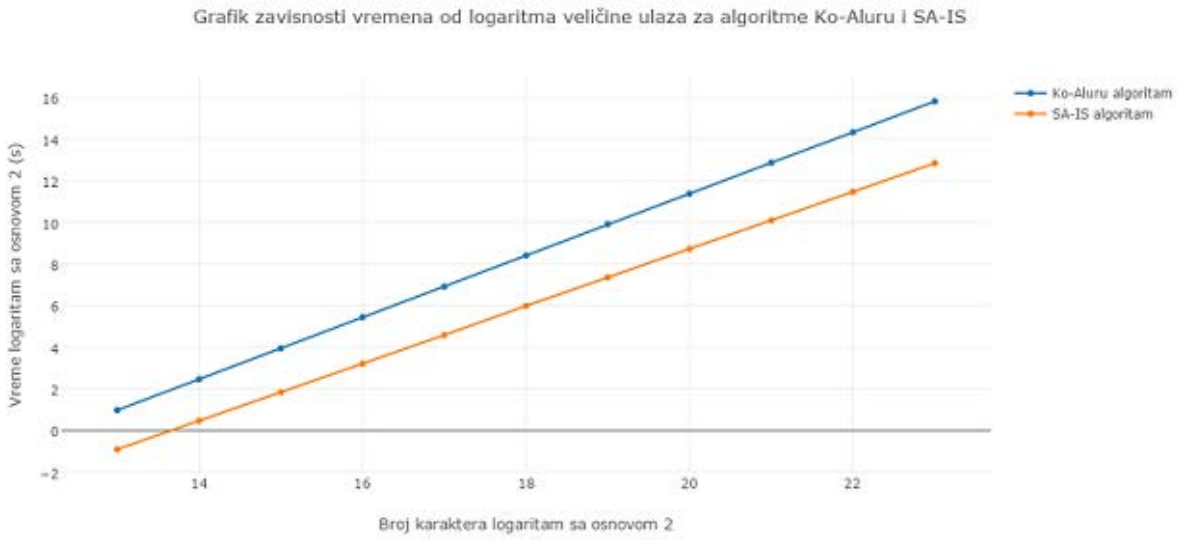
3.4. Konstrukcija sufiksnog niza za stringove velikih dužina

Program je testiran i za ulaze čija dužina prelazi 1000 karaktera. U delu 3.3 je napomenuto da se zbog uštede vremena i prostora za ovakve ulaze na standardnom izlazu ne štampa rezultujući sufiksni niz.

Za potrebe testiranja napravljen je jedan veliki string T dužine n = 2^{23} nasumično raspoređenih karaktera nad alfabetom ∑ = {A, G, T, C, U}. Za ulaze je uzeto 11 prefiksa stringa T dužine 2^i, $13 \leq i \leq 23$. U tabeli 3.1 je prikazano vreme potrebno za konstrukciju sufiksnog niza koristeći algoritam Ko-Aluru i algoritam SA-IS, a na slici 7 je prikazan grafik zavisnosti $\log_2 x$ (veličine u rasponu od 13 do 23) od $\log_2 y1$ i $\log_2 y2$, gde je x veličina ulaznog stringa, a $y1$ i $y2$ vremena konstrukcije sufiksnog niza za algoritme Ko-Aluru i SA-IS redom. Program je testiran na računaru sa sledećim karakteristikama:
Procesor: Intel(R) Core(TM) i3 – 3217U CPU @ 1.8GHz
RAM memorija: 4.00 GB

Veličina ulaza (x)	**$\log_2 x$**	**Ko-Aluru**		**SA-IS**		**Odnos vremena izvršavanja y1 / y2**
		Vreme izvršavanja (y1)	$\log_2 y1$	Vreme izvršavanja (y2)	$\log_2 y2$	
8192	13	1.99	0.96	0.53	-0.92	3.75
16384	14	5.48	2.45	1.36	0.45	4.03
32768	15	15.28	3.93	3.54	1.82	4.32
65536	16	43.19	5.43	9.17	3.19	4.71
131072	17	119.70	6.90	23.84	4.57	5.02
262144	18	337.82	8.40	62.90	5.97	5.37
524288	19	962.27	9.91	162.99	7.35	5.90
1048576	20	2646.92	11.37	420.28	8.71	6.30
2097152	21	7435.94	12.86	1086.75	10.09	6.84
4194304	22	20566.38	14.32	2828.46	11.47	7.27
8388608	23	57604.74	15.81	7313.77	12.84	7.88

Tabela 3.1: Prikaz dobijenih rezultata testiranja



Slika 7: *Prikaz rezultata merenja vremena potrebnog za konstrukciju sufiksnog niza A koristeći algoritme Ko-Aluru i SA-IS u zavisnosti od dužine ulaznog niza*

Na osnovu dobijenih rezultata testiranja algoritama Ko-Aluru i SA-IS, iz tabele 1 i sa slike 7 može se uočiti da je vreme izvršavanja algoritma SA-IS, za ulaze iste veličine, kraće od vremena izvršavanja algoritma Ko-Aluru. Odnos ova dva vremena (y1 / y2) kreće se u rasponu od 3.75 do 7.88.

Ocena koeficijenta pravca pravih

Neka su *p1* i *p2* prave koje oslikavaju zavisnost vremena od logaritma veličine ulaza za algoritme Ko-Aluru i SA-IS redom. Sa dijagrama se mogu oceniti koeficijenti pravca *c1* i *c2* pravih *p1* i *p2* kroz prvu i poslednju tačku:

$\log_2 y = \log_2 y_0 + k\ (\log_2 x - \log_2 x_0)$
$\log_2 y - \log_2 y_0 = k\ (\log_2 x - \log_2 x_0)$
$k = (\log_2 y - \log_2 y_0) / (\log_2 x - \log_2 x_0)$
$k = c1 = (15.81 - 0.99) / (23 - 13) = 14.82 / 10 = 1.48$
$k = c2 = (12.84 + 0.92) / (23 - 13) = 13.76 / 10 = 1.38$

Iz toga sledi procena da je za algoritam Ko-Aluru $y1 = (y1_0 / x_0^{1.48}) x^{1.48}$, a za algoritam SA-IS $y2 = (y2_0 / x_0^{1.38}) x^{1.38}$, što može predstavljati potvrdu da algoritam SA-IS zahteva manje vremena za konstrukciju sufiksnog niza od algoritma Ko-Aluru. Na osnovu dobijenih rezultata testiranja se zaključuje da postoje mala odstupanja od linearne složenosti, a to može da bude posledica rada sa virtuelnom memorijom za veće dužine stringa. Dalji rad bi mogao da bude da se pronađu problematični delovi koda i interpretiraju na neki drugi način koji će dovesti do toga da se složenost maksimalno približi linearnoj.

ZAKLJUČAK

U radu su izložena dva algoritma za konstrukciju sufiksnog niza zadatog stringa u vremenu O(n), algoritam Ko-Aluru i algoritam SA-IS. Prvi algoritmi linearne složenosti za konstrukciju sufiksnog niza (direktno, a ne polazeći od sufiksnog stabla) pojavili su se tek 2003. godine. Pojava ovih algoritama je podstakla upotrebu sufiksnih nizova pre svega kao jednostavnije strukture od sufiksnih stabala, ali isto tako i zbog manjih zahteva za memorijom. Zahvaljujući efikasnosti ovih algoritama u današnje vreme za pretraživanje relativno velikih genoma dovoljan je personalni računar.

Priloženi program može se iskoristiti za konstrukciju sufiksnog niza stringa koristeći neki od dva implementirana algoritma, a dobijeni rezultat bi mogao da bude iskorišćen u bilo kojoj oblasti koja se bavi pronalaženjem podudarnosti stringova (analiza genoma, kompresija podataka,..). Mogući pravac daljeg rada bi bio pokušaj smanjenja složenosti programa za velike ulaze, kao i traženje novih primena sufiksnih nizova.

Dalji logičan pravac razvoja bio bi istraživanje naprednih modela izračunavanja i korišćenje paralelnih računara za realizaciju ovih algoritama.

LITERATURA

1. Marinković D, Tucić N, Kekić V: *Genetika*, Naučna knjiga, Beograd, 1991.
2. M.I. Abouelhoda, S. Kurtz, E. Ohlebusch: *Replacing suffix trees with enhanced suffix arrays,* Faculty of Computer Science, University of Ulm Germany, Center for Bioinformatics, University of Hamburg Germany, 2004.
3. T. Kasai, G. Lee, H. Arimura, S. Arikawa, K. Park, Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and its Applications, In Proceedings of the 12th annual Symposium on Combinatorial Pattern Matching, 596-604, IEEE Computer Society, New York, 1999.
4. J. Kärkkäinen, P. Sanders, Simple linear work suffix array construction, In Proc. 30th International Conference on Automata, Languages and Programming, vol. 2719 serije LNCS, 943-955, Springer, Berlin, 2003.
5. P. Ko and S. Aluru. Space efficient linear time construction of suffix arrays. In *Proceedings 14th CPM, LNCS 2676, Springer-Verlag*, pages 200–210, 2003.
6. S. Kurtz, a Jomuna V. Choudhuri, E. Ohlebusch, C. Schleiermacher,1 J. Stoye, 2 and R. Giegerich, REPuter: the manifold applications of repeat analysis on a genomic scale, Nucleic Acids Res. 2001 Nov 15; 29(22): 4633–4642.
7. Manber, Udi*;* Myers, Gene (1990). *Suffix arrays: a new method for on-line string searches*. First Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 319–327.
8. Ge Nong, Sen Zhang, Wai Hong Chan. Linear Suffix Array Construction by Almost Pure Induced-Sorting