SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 964

Implementacija učinkovitog algoritma za izgradnju poboljšanog sufiksnog polja

Hana Jurić Fot

Zagreb, travanj 2025.

Zahvaljujem mentorici Mirjani Domazet-Lošo na pomoći i znanju koje mi je pružila.

SADRŽAJ

1. Uvod

Problem pronalaska uzorka u tekstu (engl. *pattern matching*) jedan je od temeljnih problema kojima se bavi bioinformatika. Osim u bioinformatici, ovaj se problem javlja i u drugim područjima poput analize teksta, prepoznavanju slika i prepoznavanju govora. U bioinformatici se najčešće radi o pronalasku kratkog podniza u zadanom slijedu ili sljedovima velikih bioloških molekula poput DNA molekula ili proteina (Domazet-Lošo i Šikić, 2014). Ako taj kraći podniz ili uzorak označimo s P (engl. *pattern*), a slijed ili tekst s T (engl. *text*), onda je problem pronaći sva pojavljivanja P u T. U bioinformatici je tekst T vrlo dugačak niz. Genom bakterije sadrži i do 10^7 parova baza, a kod složenijih vrsta je on i nekoliko redova veličine veći. Upravo iz tog razloga naivna rješenja, čija je vremenska složenost O($|P| \cdot |T|$), nisu prihvatljiva. Algoritamska rješenja ovog problema mogu se podijeliti u dvije skupine. Prva su skupina rješenja u kojima se prvo indeksira podniz P, a onda se taj indeks koristi za pretraživanje teksta T u složenosti O(|T|). Najpoznatiji algoritmi koji koriste ovaj pristup su Knuth-Morris-Prattov algoritam (Knuth et al., 1977) i Boyer-Mooreov algoritam (Boyer i Moore, 1977). U drugu skupinu spadaju rješenja u kojima se prvo indeksira tekst T, a zatim se taj indeks koristi za pronalazak uzorka P. Traženje podniza P u T u ovom se slučaju obavlja u vremenu O(|P|). Zbog veličine teksta T ovaj je pristup pogodniji za primjenu u bioinformatici.

Jedna od podatkovnih struktura koja koristi taj način rada je sufiksno stablo (engl. *suffix tree*). Sufiksno stablo je stablo koje sadrži sve sufikse određenog niza. Listovi (engl. *leaf*) stabla odgovaraju početnim mjestima sufiksa u nizu, a čvorovi sadrže podnizove. Sufiksno stablo moguće je izgraditi u vremenu linearno ovisnom o duljini ulaznog niza, a problem pronalaska podniza u nizu rješava se u vremenu proporcionalnom duljini podniza. Osim rješavanja ovog problema, sufiksno stablo omogućava i rješavanje mnogih drugih složenih operacija nad znakovnim nizovima poput pronalaska najduljeg zajedničkog podniza dvaju nizova u vremenu proporcionalnom zbroju njihovih duljina (Gusfield, 1997). Prvi algoritam za izgradnju sufiksnog stabla u linearnom vremenu bio je Weinerov algoritam (Weiner, 1973), a danas se najčešće koristi

Ukkonenov *online* algoritam (Ukkonen, 1995) koji ne gradi sufiksno stablo odjednom iz cijelog niza, već ga proširuje za svaki znak u nizu. Sufiksno stablo ima široku primjenu u bioinformatici, no pati od problema velikog zauzeća memorije te se zbog toga sve manje primjenjuje. Za ulazni niz od n znakova sufiksno stablo zahtijeva 10-20n okteta (engl. *byte*) memorijskog prostora (Domazet-Lošo i Šikić, 2014). Alternativna struktura podataka, sufiksno polje (engl. *suffix array*) zauzima oko 4 okteta po ulaznom znaku.

Sufiksna polja predložili su 1990. godine Manber i Myers (Manber i Myers, 1990) kao memorijski manje zahtjevnu alternativu sufiksnom stablu. Sufiksno polje je polje nenegativnih cijelih brojeva, koji predstavljaju početne pozicije abecedno poredanih sufiksa ulaznog niza. Originalni algoritam omogućavao je izgradnju sufiksnog polja u vremenskoj složenosti O($n$log$n$), no danas postoji mnogo algoritama koji to omogućavaju u linearnom vremenu. Najpopularniji takav algoritam je Nong-Zhang-Chanov SAIS algoritam (Nong et al., 2009, 2011), koji će biti objašnjen u nastavku. Iako su sufiksna polja riješila problem prevelikog zauzeća memorije, ona ipak imaju neke nedostatke u odnosu na sufiksna stabla. Iz tog razloga postoje razne nadogradnje i proširenja sufiksnih polja nekim drugim jednostavnim strukturama. Pokazalo se da takva poboljšana sufiksna polja (engl. *enhanced suffix arrays; ESA*) mogu u potpunosti zamijeniti sufiksna stabla (Abouelhoda et al., 2004). Ipak, ni takve strukture same po sebi ne omogućuju izgradnju sufiksnog polja nad većim brojem nizova, što često želimo u bioinformatici kako bismo, recimo, našli najdulji zajednički podniz svih nizova. Godine 2017. predloženo je proširenje SAIS algoritma, gSAIS, koje omogućuje upravo to, izgradnju sufiksnog polja nad kolekcijom nizova (Louza et al., 2017).

2. Sufiksno polje

Neka je S niz znakova duljine *n*, gdje su znakovi niza elementi abecede Σ. Na kraj niza dodajemo poseban znak "$" koji ne postoji u abecedi Σ i leksikografski je manji od svakog znaka iz Σ.
Podniz niza S koji započinje na *i*-tom, a završava na *j*-tom mjestu označit ćemo s S[*i, j*], $1 \leq i \leq n$. Analogno tome, S[*i*, n] označava podniz koji započinje na *i*-tom, a završava na zadnjem mjestu, što je zapravo sufiks tog niza. Sufikse ćemo još označavati oznakom s_i.
Sufiksno polje (engl. *suffix array*) SA niz je cijelih brojeva koji označavaju početne pozicije abecedno poredanih sufiksa niza S.

Primjer: Neka je zadan ulazni niz S = "ATGA". Na kraj niza dodajemo znak "$", koji se ne nalazi u abecedi Σ i abecedno je manji od znakova iz Σ. Sufiksi niza S$ su: s_0 = "ATGA$", s_1 = "TGA$", s_2 = "GA$", s_3 = "A$" i s_4 = "$".
Tablica 2.1 prikazuje sufikse poredane abecednim redom. Sufiksno polje SA sadrži početno poredane sufikse ovog niza, odnosno SA = [4, 3, 0, 2, 1].

Tablica 2.1: Sufiksno polje niza "ATGA$"

i	SA[i]	s_i
0	4	$
1	3	A$
2	0	ATGA$
3	2	GA$
4	1	TGA$

3. Izgradnja sufiksnog polja SA-IS algoritmom

Originalni algoritam koji su osmislili Manber i Myers (1990) omogućavao je izgradnju sufiksnog polja u vremenu O($n \log n$) za ulazni niz od n znakova. Danas postoji mnogo algoritama za izgradnju sufiksnog polja (engl. *Suffix Array Construction Algorithm; SACA*) koji rade u linearnom vremenu (Kärkkäinen i Sanders, 2003; Ko i Aluru, 2003; Nong et al., 2009, 2011). Najbrži od njih je algoritam SAIS (engl. *Suffix Array Induced Sorting*) koji su osmislili Nong, Zhang i Chan 2009. godine. Ovaj algoritam koristi neke ideje iz prethodno spomenutih algoritama. Svi oni temelje se na sličnoj tehnici "podijeli pa vladaj". Početni problem se prvo reducira, odnosno podijeli u dva niza te se prvi niz rekurzivno sortira, a zatim se ostatak niza inducirano sortira korištenjem prvog. Na kraju se ta dva sufiksna polja spoje u konačno sufiksno polje SA(S). Glavna razlika u algoritmima je način redukcije početnog niza. SAIS algoritam koristi neke ideje koje su Ko i Aluru predstavili u svom algoritmu (Ko i Aluru, 2003), no Nong, Zhang i Chan uvode pojam LMS-podniza, čime je ostvarena bolja redukcija, a samim time su ostvarena i značajna poboljšanja u brzini izvođenja i memorijskom zauzeću u odnosu na druge postojeće algoritme.

3.1. Osnovni pojmovi i definicije

Neka je S niz sastavljen od znakova abecede Σ i neka S završava znakom \$ koji nije dio abecede Σ i leksikografski je manji od svih znakova abecede. Dodavanje posebnog znaka na kraj niza vrlo je bitan postupak kod izgradnje sufiksnih stabala jer osigurava da svaki sufiks završava u listu stabla. Ako je neki sufiks ujedno i prefiks nekog drugog sufiksa, on bi završio u unutarnjem čvoru stabla. Ova praksa zadržala se i kod konstrukcije sufiksnih polja. Kao što je već spomenuto, sufiks koji počinje na i-tom mjestu nizu označavat ćemo s s_i.

Ideja koja je preuzeta iz Ko-Alurovog algoritma (Ko i Aluru, 2003) jest podjela

sufiksa na S-tip i L-tip. Sufiks s_i je S-tip (engl. *S-type*) ako je $s_i < s_{i+1}$. Po definiciji sufiks \$ je S-tip. Sufiks s_i je L-tip (engl. *L-type*) ako je $s_i > s_{i+1}$. Znak S[i] je S-tip znaka ako je S[i] < S[i + 1] ili ako je S[i] = S[i + 1] i s_{i+1} je S-tip sufiksa. Analogno, znak S[i] je L-tip znaka ako je S[i] > S[i + 1] ili ako je S[i] = S[i + 1] i s_{i+1} je L-tip sufiksa. Ovakva klasifikacija znakova u nizu može se odrediti u vremenu O(n) jednim prolaskom kroz niz S zdesna na lijevo, odnosno od zadnjeg prema prvome znaku. Klasifikacija znakova pohranjuje se u pomoćnom polju t.

Primjer: Neka je zadan niz S = "AGCCTAGCCTAC\$". Tablica 3.1 prikazuje tip znaka za svaki znak niza S. U pomoćno polje t spremaju se tipovi znakova. U implementaciji se koristi polje cijelih brojeva pa broj 1 označava S-tip, a broj 0 L-tip znaka.

Tablica 3.1: Tipovi znakova niza S = "AGCCTAGCCTAC\$"

Indeksi	0	1	2	3	4	5	6	7	8	9	10	11	12
niz S	A	G	C	C	T	A	G	C	C	T	A	C	\$
tip znaka	S	L	S	S	L	S	L	S	S	L	S	L	S
polje t	1	0	1	1	0	1	0	1	1	0	1	0	1

Primijetimo da je sufiks S-tipa abecedno veći od sufiksa L-tipa ako oba počinju istim slovom. Ovo se svojstvo koristi na nekoliko mjesta u algoritmu.

Primjer: Neka je zadan niz S = "GATGTA\$". Tablica 3.2 prikazuje tipove znakova za niz S. Uzmimo kao primjer sufiks s_0 = "GATGTA\$" i sufiks s_3 = "GTA\$". Sufiks s_0 je L-tipa i abecedno je manji od sufiksa s_3 koji je S-tipa.

Tablica 3.2: Tipovi znakova niza S = "GATGCA\$"

Indeksi	0	1	2	3	4	5	6
niz S	G	A	T	G	T	A	\$
tip znaka	L	S	L	S	L	L	S

Ko i Aluru koriste S/L-tipove za podjelu niza na dva dijela i redukciju problema. Iako je taj postupak efikasan i izgradnja sufiksnog polja se odvija u linearnom vremenu, Nong, Zhang i Chan uvode novu vrstu klasifikacije podnizova koja dodatno poboljšava performanse. Da bismo definirali LMS-podniz, trebamo definirati LMS-znak. Znak S[i] je LMS-znak (engl. *leftmost S-type*) ako je S[i] S-tip i S[i - 1] L-tip (za $i \leq 1$).

Sufiks s_i je LMS sufiks ako je S[i] LMS-znak. Konačno, S[i, j] je LMS-podniz ako je S[i, j] znak na kraju niza ($i = j$) ili ako su S[i] i S[j] LMS-znakovi, a između njih nema drugih LMS-znakova.

Poredak dvaju LMS-podnizova određuje se usporedbom njihovih znakova slijeva nadesno: za svaki par znakova, uspoređujemo njihove leksikografske vrijednosti, a ako su jednake, promatramo tipove znakova, gdje S-tip sortiramo nakon L-tipa. Dva su LMS-podniza jednaka ako su jednake duljine i ako su im svi znakovi leksikografski jednaki i istog tipa. Za određivanje LMS-podnizova koristi se pomoćno polje t koje sadrži tipove znakova. Indeksi LMS-podnizova pohranjuju se u pomoćno polje P_1.

Primjer: Neka je zadan niz S = "AGCCTAGCCTAC$". Polje P_1 određuje se uz pomoć polja t koje sadrži oznake je li neki znak S-tip ili L-tip (tablica 3.1). U polju t tražimo znakove S-tipa, čiji su neposredni prethodnici znakovi L-tipa. U ovom slučaju polje P_1 = {2, 5, 7, 10, 12}. Između svaka dva susjedna LMS-znaka ne nalazi se nijedan drugi LMS-znak, što znači da oni označavaju početne i krajnje pozicije LMS-podnizova. Niz S, dakle, ima 5 LMS-podniza (prvi LMS-podniz započinje na poziciji 2 i završava na poziciji 5, drugi počinje na poziciji 5 i završava na poziciji 7 itd.). Posljednji LMS-podniz je, po definiciji, zadnji znak niza. Tablica 3.3 prikazuje LMS-podnizove niza S = "AGCCTAGCCTAC$".

Tablica 3.3: LMS-podnizovi niza S = "AGCCTAGCCTAC$"

i	j	S[i, j]
2	5	CCTA
5	7	AGC
7	10	CCTA
10	12	AC$
12	12	$

Osim pomoćnih polja t (oznake tipova znakova) i P_1 (indeksi LMS-podnizova), algoritam koristi i pomoćno polje B. Sufiksno polje SA može se podijeliti na potpolja ili pretince (engl. *bucket*) takve da svaki pretinac sadrži sve sufikse koji počinju istim slovom. Polje B je polje pokazivača na početak ili kraj pojedinog pretinca.

3.2. SAIS algoritam

U ovom poglavlju bit će pojašnjeni koraci SAIS algoritma. Algoritam 1 prikazuje glavne korake algoritma SAIS. Svaki je korak detaljnije pojašnjen u nastavku.

Algorithm 1 SAIS algoritam (S, SA)

Ulaz: S - niz znakova koji završava posebnim znakom \$

Izlaz: SA - sufiksno polje ulaznog niza

Pomoćna polja: t, P_1, B

1. Odrediti pomoćno polja t (polje tipova znakova: S-tip/L-tip).
2. Odrediti pomoćno polje P_1 (polje početnih pozicija LMS-podnizova).
3. Inducirano sortirati LMS-podnizove korištenjem P_1 i B.
4. Imenovati svaki LMS-podniz prema pripadajućem rangu (novo polje S_1).
5. Ako je svaki znak niza S_1 jedinstven, izravno odrediti SA iz S_1, inače rekurzivno pozvati SAIS(S_1, SA_1).
6. Odrediti SA iz SA_1.

3.2.1. Pomoćno polje t (polje tipova znakova: S-tip/L-tip)

Ovaj postupak opisan je u prethodnom dijelu. Jednim prolaskom kroz niz S slijeva na desno određuje se tip za svaki znak niza te se zapisuje u pomoćno polje t.

3.2.2. Pomoćno polje P_1 (polje početnih pozicija LMS-podnizova)

Ovaj postupak također je opisan u prethodnom dijelu. Za određivanje sadržaja polja P_1 koristi se polje t.

3.2.3. Inducirano sortiranje LMS-podnizova korištenjem P_1 i B

1. Postaviti sve članove sufiksnog polja SA na -1.
   Postaviti pokazivač B[k] na kraj k-tog pretinca od SA, za svaki $k \in \Sigma$.
2. Proći kroz polje S slijeva nadesno i dodavati indekse LMS-sufiksa iz S u pripadajući pretinac od SA od kraja pretinca prema početku (nakon svakog dodavanja

pomaknuti pokazivač B[k] za jedno mjesto ulijevo).

3. Postaviti pokazivač B[k] na početak k-tog pretinca od SA, za svaki $k \in \Sigma$.

4. Proći kroz polje SA slijeva nadesno. Za svaki SA[i] > 0 provjeriti je li S[SA[i] - 1] L-tip znaka. Ako je, dodati SA[i] - 1 na početak k-tog pretinca i pomaknuti B[k] za jedno mjesto udesno.

5. Postaviti pokazivač B[k] na kraj k-tog pretinca od SA, za svaki $k \in \Sigma$.

6. Proći kroz polje SA zdesna nalijevo. Za svaki SA[i] > 0 provjeriti je li S[SA[i] - 1] S-tip znaka. Ako je, dodati SA[i] - 1 na kraj k-tog pretinca i pomaknuti B[k] za jedno mjesto ulijevo.

Primijetimo da se LMS-sufiksi dodaju na krajeve pretinaca, L-tipovi sufiksa na početke, a S-tipovi ponovno na krajeve. To je posljedica upravo onog svojstva da je sufiks S-tipa abecedno veći od sufiksa L-tipa ako oba počinju istim znakom. Svi sufiksi S-tipa (LMS-sufiksi su zapravo sufiksi S-tipa (engl. *leftmost S-type*)) stavljaju se na krajeve jer su veći od sufiksa L-tipa koji se zato stavljaju na početke.

Primjer: Promotrimo ponovno niz S = "AGCCTAGCCTAC$". U prethodnom primjeru smo odredili polje početnih pozicija LMS-podnizova P_1 = {2, 5, 7, 10, 12}, koje će nam trebati u ovom koraku. Prvo trebamo podijeliti sufiksno polje na pretince. Iz niza možemo vidjeti da postoji jedan znak "$", tri znaka "A", pet znaka "C", dva znaka "G" i dva znaka "T". Dakle, sufiksno polje možemo zapisati ovako:

SA = {_}, {_,_,_}, {_,_,_,_,_}, {_,_}, {_,_}

Zatim trebamo inicijalizirati polje SA na -1 i postaviti sve pokazivače iz polja B na krajeve pretinaca.

SA = {-1}, {-1, -1, -1}, {-1, -1, -1, -1, -1}, {-1, -1}, {-1, -1}
B["$"] = 0, B["A"] = 3, B["C"] = 8, B["G"] = 10, B["T"] = 12

Sada prolazimo po nizu S slijeva nadesno i dodajemo indekse LMS-podnizova u pripadajuće pretince. Prvi indeks po redu je 2, a taj sufiks započinje slovom "C" pa dodajemo broj 2 na kraj pretinca sa sufiksima koji započinju slovom "C" i pomičemo pokazivač B["C"] za jedno mjesto ulijevo. Cijeli postupak ovog koraka prikazan je na slici 3.1. U nizu su uokvireni početni indeksi LMS-podnizova, a u sufiksnom polju su zacrvenjeni indeksi koji su dodani u tom trenutku. Strelice prikazuju pokazivače polja

0	1	2	3	4	5	6	7	8	9	10	11	12
A	G	C	C	T	A	G	C	C	T	A	C	$
S	L	S	S	L	S	L	S	S	L	S	L	S

$	A			C					G		T	
-1	-1	-1	-1	-1	-1	-1	-1	2	-1	-1	-1	-1
-1	-1	-1	5	-1	-1	-1	-1	2	-1	-1	-1	-1
-1	-1	-1	5	-1	-1	-1	-1	2	-1	-1	-1	-1
-1	-1	-1	5	-1	-1	-1	7	2	-1	-1	-1	-1
-1	-1	10	5	-1	-1	-1	7	2	-1	-1	-1	-1
12	-1	10	5	-1	-1	-1	7	2	-1	-1	-1	-1

Slika 3.1: Drugi korak induciranog sortiranja LMS-podnizova
Uokvireni su početni indeksi LMS-podnizova, u sufiksnom polju su zacrvenjeni indeksi koji su dodani u tom trenutku, a strelice prikazuju pokazivače polja B

B.

Ovime su svi LMS-podnizovi ubačeni u sufiksno polje. U sljedećem koraku trebamo postaviti sve pokazivače na početke pretinaca:

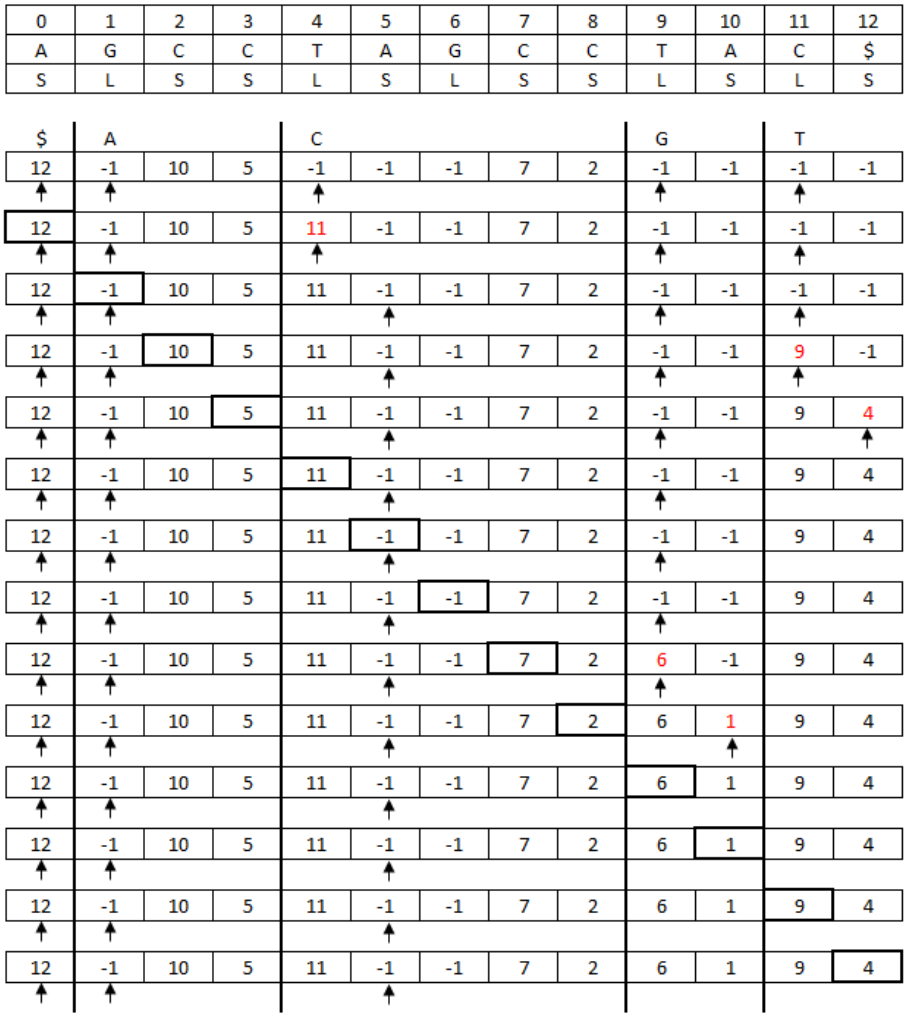
$$B["\$"] = 0, B["A"] = 1, B["C"] = 4, B["G"] = 9, B["T"] = 11$$

Nakon toga prolazimo po polju SA slijeva nadesno i tražimo članove koji su veći od 0, odnosno sve članove koji nisu -1. Za svaki takav član gledamo koji mu je prethodnik u nizu S. Ako je taj prethodnik L-tipa, dodajemo ga u pripadajući pretinac. Dakle gledamo SA polje i prvi član koji je veći od 0 je 12. Gledamo njegovog prethodnika, znači znak koji se u nizu nalazi na indeksu 11. Budući da je taj znak L-tipa i započinje slovom "C", dodajemo indeks 11 na početak pretinca u kojemu su sufiksi koji započinju slovom "C". Naravno, postupak ponavljamo dok ne dođemo do kraja polja SA, a nakon svakog dodavanja pomičemo i odgovarajući pokazivač u polju B. Ovaj je postupak prikazan na slici 3.2.

Nakon toga trebamo ponovno pomaknuti sve pokazivače polja B na krajeve svih pretinaca:

$$B["\$"] = 0, B["A"] = 3, B["C"] = 8, B["G"] = 10, B["T"] = 12$$

Sada prolazimo po polju SA zdesna nalijevo i ponovno tražimo članove koji nisu -1, no ovoga puta prethodnici tih članova trebaju biti S-tipa da bismo ih dodali u sufiksno



Slika 3.2: Četvrti korak induciranog sortiranja LMS-podnizova
Uokviren je član polja koji se upravo obrađuje, zacrvenjeni su novi, upravo dodani članovi, a strelice označavaju pokazivače polja B.

polje. Prvi indeks je 4, dakle gledamo indeks 3 u polju SA i budući da je S-tipa, dodajemo indeks 3 na kraj pretinca u kojemu se nalaze sufiksi koji počinju slovom "C". Postupak ponavljamo za sve članove polja i, naravno, ponovno pomičemo pokazivače nakon svakog dodavanja novih članova. Postupak je prikazan na slici 3.3.

Šesti korak je ujedno i zadnji pa smo ovime završili inducirano sortiranje LMS-podnizova. Trenutno naše polje izgleda ovako:

$$\text{SA} = \{12\}, \{10, 5, 0\}, \{11, 7, 2, 8, 3\}, \{6, 1\}, \{9, 4\}$$

Ovime smo sortirali sve LMS-sufikse našeg niza, no još nismo dobili konačno sufiksno polje jer indeksi sufiksa unutar pojedinih pretinaca nisu sortirani,

0	1	2	3	4	5	6	7	8	9	10	11	12
A	G	C	C	T	A	G	C	C	T	A	C	$
S	L	S	S	L	S	L	S	S	L	S	L	S

$	A			C					G		T	
12	-1	10	5	11	-1	-1	7	2	6	1	9	4
12	-1	10	5	11	-1	-1	7	3	6	1	9	4
12	-1	10	5	11	-1	-1	8	3	6	1	9	4
12	-1	10	0	11	-1	-1	8	3	6	1	9	4
12	-1	5	0	11	-1	-1	8	3	6	1	9	4
12	-1	5	0	11	-1	2	8	3	6	1	9	4
12	-1	5	0	11	7	2	8	3	6	1	9	4
12	-1	5	0	11	7	2	8	3	6	1	9	4
12	-1	5	0	11	7	2	8	3	6	1	9	4
12	10	5	0	11	7	2	8	3	6	1	9	4
12	10	5	0	11	7	2	8	3	6	1	9	4
12	10	5	0	11	7	2	8	3	6	1	9	4
12	10	5	0	11	7	2	8	3	6	1	9	4
12	10	5	0	11	7	2	8	3	6	1	9	4

Slika 3.3: Šesti korak induciranog sortiranja LMS-podnizova
Uokviren je član polja koji se upravo obrađuje, zacrvenjeni su novi, upravo dodani članovi, a strelice označavaju pokazivače polja B.

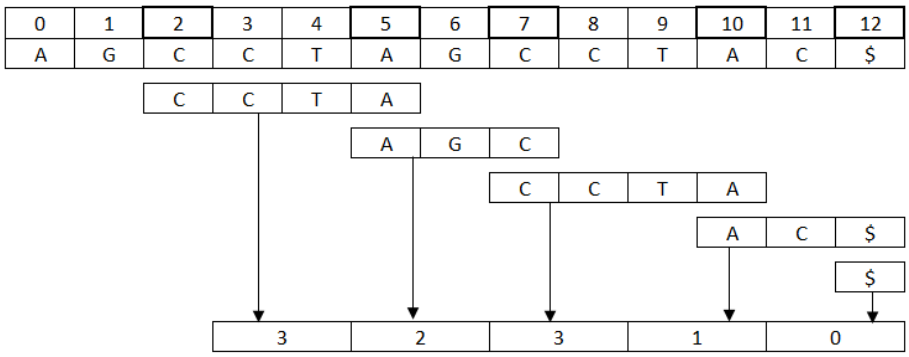
3.2.4. Imenovanje LMS-podnizova (novi niz S_1)

Cilj pridjeljivanja imena LMS-podnizovima zapravo je redukcija početnog problema, odnosno, svođenje početnog niza znakova S na novi, kraći niz S_1. Svakom LMS-podnizu pridjeljuje se novo leksikografsko ime, odnosno cijeli broj koji odgovara rangu tog podniza. Ako su LMS-podnizovi jednake duljine i na svim njihovim mjestima se nalaze jednaki znakovi jednakog tipa, podnizovi su po definiciji jednaki te će dobiti jednaka imena. Nova imena pridijeljena LMS-podnizovima članovi su novog niza S_1.

Polje SA dobiveno u prošlom koraku služi za određivanje novih imena LMS-podnizovima. Prolazeći kroz polje SA slijeva nadesno dodjeljujemo nova imena počevši od 0. Za svaki sufiks iz polja SA provjeravamo je li LMS-podniz tako da provjerimo je li njegov početni znak LMS-znak., što se odvija u O(1) vremenu. LMS-podniz

dobiva jednako ime kao i njegov prethodnik ako su oni međusobno jednaki, a inače dobiva novo (prethodno uvećano za 1).

Primjer: Za niz S = "AGCCTAGCCTAC$" u prethodnom smo koraku odredili SA = {{12}, {10, 5, 0}, {11, 7, 2, 8, 3}, {6, 1}, {9, 4}}. Sada koristimo taj rezultat kako bismo imenovali LMS-podnizove. Prolaskom kroz SA slijeva nadesno prvi LMS-podniz na koji nailazimo je onaj s indeksom 12 (znak za kraj niza "$"). Njemu dodjeljujemo ime 0. Sljedeći LMS-podniz ima indeks 10 ("AC$") i razlikuje se od prethodnog te mu dodjeljujemo ime 1. Sljedeći je podniz s indeksom 5 ("AGC") te dobiva ime 2. Zatim nailazimo na LMS-podniz s indeksom 7 ("CCTA") i dodjeljujemo mu ime 3. Zadnji LMS-podniz je onaj s indeksom 2 i jednak je prethodnom LMS-podnizu pa mu dodjeljujemo i jednako ime 3. Imenovanje je prikazano na slici 3.4.



Slika 3.4: Imenovanje LMS-podnizova za niz S = "AGCCTAGCCTAC$"
Uokvireni su indeksi početnih pozicija LMS-podnizova.

Na kraju sva nova imena LMS-podnizova spajamo u novi niz S_1 prema njihovom originalnom poretku u početnom nizu.

$$S_1 = \text{"32310"}$$

3.2.5. Određivanje SA_1 za S_1

U prethodnom koraku određen je niz S_1, odnosno niz imena LMS-podnizova. Ako je svako ime iz S_1 jedinstveno, onda se SA_1 određuje trivijalno iz S_1 jer je poredak sufiksa određen isključivo njihovim početnim znakovima, a oni su svi različiti. U suprotnom se SA_1 određuje rekurzivnim pozivom funkcije SAIS(S_1, SA_1).

Primjer: Za niz S = "AGCCTAGCCTAC$" imenovanjem LMS-podnizova dobiven je niz S_1 = ”32310”. Kako nisu sva imena LMS-podnizova jedinstvena, nije

moguće trivijalno odrediti SA_1, već moramo koristiti rekurziju. Rekurzivnim pozivom funkcije SAIS(S_1, SA_1) dobivamo SA_1 = {4, 3, 2, 1, 0}.

3.2.6. Određivanje SA iz SA_1

U ovom koraku ponovno se koristi postupak induciranog sortiranja, kao i u koraku 3.

1. Postaviti sve članove sufiksnog polja SA na -1.
   Postaviti pokazivač B[k] na kraj k-tog pretinca od SA, za svaki $k \in \Sigma$.

2. Proći kroz polje SA_1 zdesna nalijevo i staviti $P_1[SA_1[i]]$ na kraj pretinca za pripadajući sufiks (nakon svakog dodavanja pomaknuti pokazivač B[k] za jedno mjesto ulijevo).

3. Postaviti pokazivač B[k] na početak k-tog pretinca od SA, za svaki $k \in \Sigma$.

4. Proći kroz polje SA slijeva nadesno. Za svaki SA[i] > 0 provjeriti je li S[SA[i] - 1] L-tip znaka. Ako je, dodati SA[i] - 1 na početak k-tog pretinca i pomaknuti B[k] za jedno mjesto udesno.

5. Postaviti pokazivač B[k] na kraj k-tog pretinca od SA, za svaki $k \in \Sigma$.

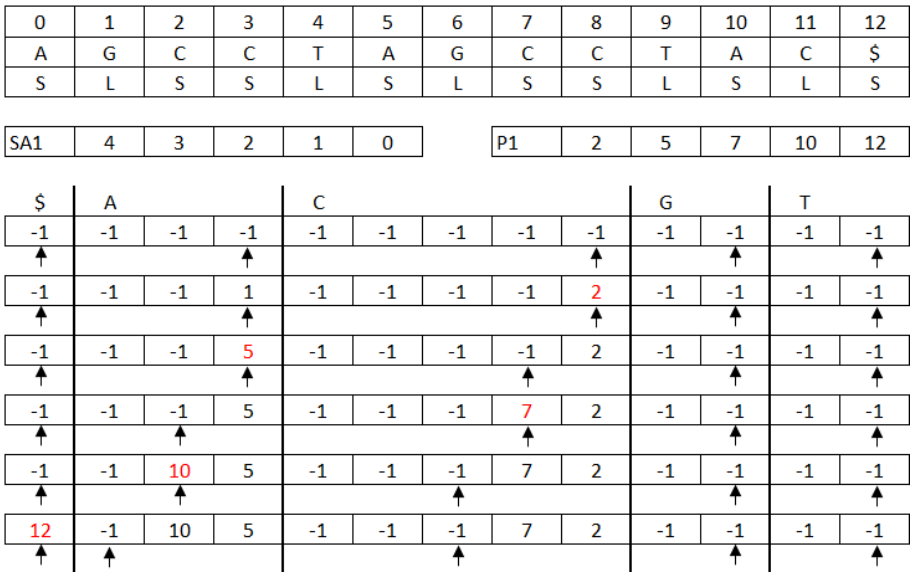
6. Proći kroz polje SA zdesna nalijevo. Za svaki SA[i] > 0 provjeriti je li S[SA[i] - 1] S-tip znaka. Ako je, dodati SA[i] - 1 na kraj k-tog pretinca i pomaknuti B[k] za jedno mjesto ulijevo.

Od 3. do 6. koraka postupak je identičan induciranom sortiranju LMS-podnizova. Svaki pretinac u podijeljen je na dio za sufikse S-tipa i L-tipa. Najprije se sortirani LMS-sufiksi dodaju u pripadajuće pretince S-tipa u SA, od kraja prema početku, a zatim se iz njih jednim prolaskom kroz niz sortiraju svi sufiksi L-tipa. Posljednji je korak sortirati sve sufikse, čime je dobiveno krajnje rješenje i sufiksno polje je do kraja sortirano.

Primjer: Imenovanjem LMS-podnizova niza S = "AGCCTAGCCTAC$" dobiven je novi niz S_1 = ”32310”, a rekurzivnim pozivom funkcije SAIS(S_1, SA_1) dobiveno je polje SA_1 = {4, 3, 2, 1, 0}. U ovom koraku koristimo dobiveno polje SA_1 kako bismo odredili konačno sufiksno polje SA. Na početku moramo postaviti sve članove polja SA na -1 i sve pokazivače polja B na krajeve pretinaca:

SA = {-1}, {-1, -1, -1}, {-1, -1, -1, -1, -1}, {-1, -1}, {-1, -1}
B["$"] = 0, B["A"] = 3, B["C"] = 8, B["G"] = 10, B["T"] = 12

Zatim prolazimo zdesna nalijevo kroz polje SA_1 i za svaki broj tražimo taj indeks polja P_1 i zatim sadržaj polja P_1 dodajemo u SA u odgovarajući pretinac. U našem polju SA_1 prvi od kraja je broj 0, znači gledamo $P_1[0] = 2$ i u SA ubacujemo indeks 2 u pretinac u kojemu se nalaze sufiksi koji počinju znakom "C". Slika 3.5 prikazuje ovaj postupak za svaki indeks polja SA_1.



Slika 3.5: drugi korak određivanja SA iz SA_1 za niz S = "AGCCTAGCCTAC$"
Zacrvenjeni su novo dodani članovi.

Nakon drugog koraka potrebno je postaviti pokazivače polja B na početke svakog pretinca i zatim dodati sufikse L-tipa na isti način kao i u koraku induciranog sortiranja LMS-podnizova. Ovime dobivamo sljedeće polje SA:

SA = {12}, {-1, 10, 5}, {11, -1, -1, 7, 2}, {6, 1}, {9, 4}

Zatim postavimo pokazivače na krajeve pretinaca i ponovimo postupak za dodavanje sufiksa S-tipa iz koraka induciranog sortiranja LMS-podnizova. Ovim korakom došli smo do kraja algoritma i dobili konačno sufiksno polje niza S = "AGCCTAGCCTAC$":

SA = {12}, {10, 5, 0}, {11, 7, 2, 8, 3}, {6, 1}, {9, 4}

4. gSAIS poboljšanje

SAIS je vrlo brz i učinkovit algoritam koji omogućuje izgradnju sufiksnog polja u linearnom vremenu. Ipak, kako je već u uvodu spomenuto, sama sufiksna polja imaju neke nedostatke u odnosu na sufiksna stabla te se zato često nadopunjuju nekim drugim jednostavnim strukturama. Osim nadopune novim strukturama, postoje i razne nadogradnje postojećih algoritama, koje, na primjer, proširuju algoritam u svrhu korištenja nad skupom ulaznih nizova. Dva takva poboljšanja algoritama predstavili su Louza, Gog i Tellers 2017. godine (Louza et al., 2017). U svome radu opisali su gSAIS i gSACA-K algoritme koji su poboljšanja ranije opisanog SAIS algoritma i algoritma SACA-K, čiji je autor također Nong (Nong, 2013). Njihova proširenja omogućuju izgradnju sufiksnog polja nad većim brojem nizova, što prijašnje varijante nisu nudile. Nadalje, nadopunili su izgradnju sufiksnog polja usporednom izgradnjom polja najdužih zajedničkih prefiksa (engl. *longest common prefix; LCP*) i dokumentirajućeg polja (engl. *document array; DA*).

4.1. Osnovni pojmovi i definicije

Neka je $\tau = T_1, T_2, ..., T_d$ kolekcija od d nizova ukupne duljine N. Budući da algoritam SAIS kao ulaz prima jedan niz, ove je nizove iz kolekcije potrebno spojiti, odnosno konkatenirati (engl. *concat*). Postoje dva glavna pristupa za spajanje nizova u jedinstveni niz T^{cat}. U prvom pristupu uvodi se d novih znakova "$\$_i$" koji služe kao separatori između nizova. Za svaki poseban znak "$\$_i$" i "$\$_j$" vrijedi sljedeće: "$\$_i$" < "$\$_j$" ako i samo ako je $i < j$. Mana ovog pristupa je što se veličina abecede za niz T^{cat} povećava za broj nizova u kolekciji d te je zato uveden drugi pristup. U drugom se pristupu kao separator za sve nizove koristi isti znak "$". Ovime veličina abecede ostaje praktički ista ($|\Sigma|$ + 1), no javlja se novi problem. Za nizove T_i i T_j, $i < j$, koji imaju jednake sufikse nije zajamčeno da će oni biti poredani s obzirom na indekse i i j. U oba pristupa na kraj niza T^{cat} stavlja se novi poseban znak "#" koji označava kraj niza i koji je manji od svih znakova.

4.2. gSAIS algoritam

Algoritam gSAIS koristi drugu varijantu konkatenacije nizova, odnosno kao separator se koristi jedan znak "$". Kako bi se spriječio problem poretka jednakih sufiksa, Louza, Gog i Telles uvode relativni poredak između simbola separatora "$" niza T^{cat} (Louza et al., 2017). Simbol "$" iz niza T_i bit će manji od simbola "$" iz niza T_j ako i samo ako je $i < j$, odnosno separator koji se u nizu T^{cat} nalazi na indeksu i' bit će manji od separatora koji se nalazi na indeksu j' ako i samo ako je $i' < j'$.

Bitno je primijetiti da će svi sufiksi koji započinju znakom "$" biti S-tipa, osim zadnjeg koji će biti L-tip jer je po definiciji "$" > "#". Prethodno je spomenuto da su sufiksi L-tipa manji od sufiksa S-tipa koji počinju istim slovom. Iz tog razloga se u koracima sortiranja algoritma SAIS sufiksi L-tipa dodaju na početak, a sufiksi S-tipa na kraj pretinca. Ovakvim sortiranjem zadnji sufiks koji započinje simbolom separatorom "$" završit će na početku pripadajućeg pretinca, što nije u skladu s maloprije opisanim relativnim poretkom. Budući da bi taj sufiks trebao završiti na zadnjem mjestu u pretincu, ono će biti rezervirano za njega. Drugim riječima, u koraku induciranog sortiranja LMS-podnizova pokazivač B["$"] neće pokazivati na zadnje mjesto u pretincu, već na predzadnje.

U koraku induciranog sortiranja LMS-podnizova, polje SA se na početku inicijalizira na -1, a svi pokazivači se postave na krajeve pretinaca, osim pokazivača pretinca B["$"] koji se postavi na predzadnje mjesto. Zatim se prolazi po nizu T^{cat} zdesna nalijevo (različito od SAIS algoritma gdje se prolazi slijeva nadesno) i traže se indeksi LMS-sufiksa. Bitna razlika u odnosu na originalan SAIS algoritam je da se u ovom koraku ne dodaju svi indeksi LMS-sufiksa u polje SA. Neka su s_i i s_j dva susjedna LMS-sufiksa, $i < j$. Ako s_i započinje znakom "$", onda se s_j ne dodaje u polje SA. Ideja je da su pozicije j upravo one koje će inducirati poredak LMS-podnizova na poziciji i, odnosno LMS-podnizova koji započinju znakom separatorom. U praksi se prolaskom po nizu svaki LMS-sufiks doda u polje SA, no ako trenutni LMS-sufiks počinje znakom "$", onda se prethodno dodani briše iz polja. Nakon dodavanja LMS-podnizova, podniz $T^{cat}[N\text{-}2,\ N]$ (zadnji podniz koji započinjem separatorom) ručno se dodaje na kraj pretinca.

Idući koraci induciranog sortiranja LMS-podnizova odvijaju se jednako kao i u SAIS algoritmu, osim što se sufiksi koji započinju znakom "$" ne dodaju u SA. Nakon što su LMS-podnizovi sortirani, još se jednom prolazi po nizu T^{cat}, ovoga puta slijeva nadesno, i dodaju se svi LMS-podnizovi koji započinju separatorom izravno u svoj pretinac, počevši od početka pretinca. Ovime se osigurava da su svi sufiksi koji počinju

separatorom ispravno sortirani.

Nakon LMS-sortiranja slijedi imenovanje LMS-podnizova. Kod imenovanja je jedino bitno da se u obzir uzima redoslijed simbola "$". Svaki LMS-podniz koji počinje separatorom dobit će različito ime ovisno o svojoj poziciji u nizu T^{cat}.

Rekurzija i svi ostali koraci algoritma jednaki su koracima algoritma SAIS.

Primjer: Neka je kolekcija nizova τ = {"banana", "anaba", "anan"} (preuzeta iz originalnog rada (Louza et al., 2017)). Kolekciju spajamo u jedinstven niz koristeći znak "$" kao separator i dodajemo znak "#" na kraj niza: T^{cat} = "banana$anaba$anan$#". U prvom i drugom koraku, kao i kod originalnog algoritma SAIS, potrebno je odrediti polja t i P_1. Tablica 4.1 prikazuje tipove znakova, odnosno polje t za niz T^{cat} = "banana$anaba$anan$#". Možemo primijetiti da je sufiks s_{17} zadnji sufiks koji počinje znakom "$" i da je on L-tipa.

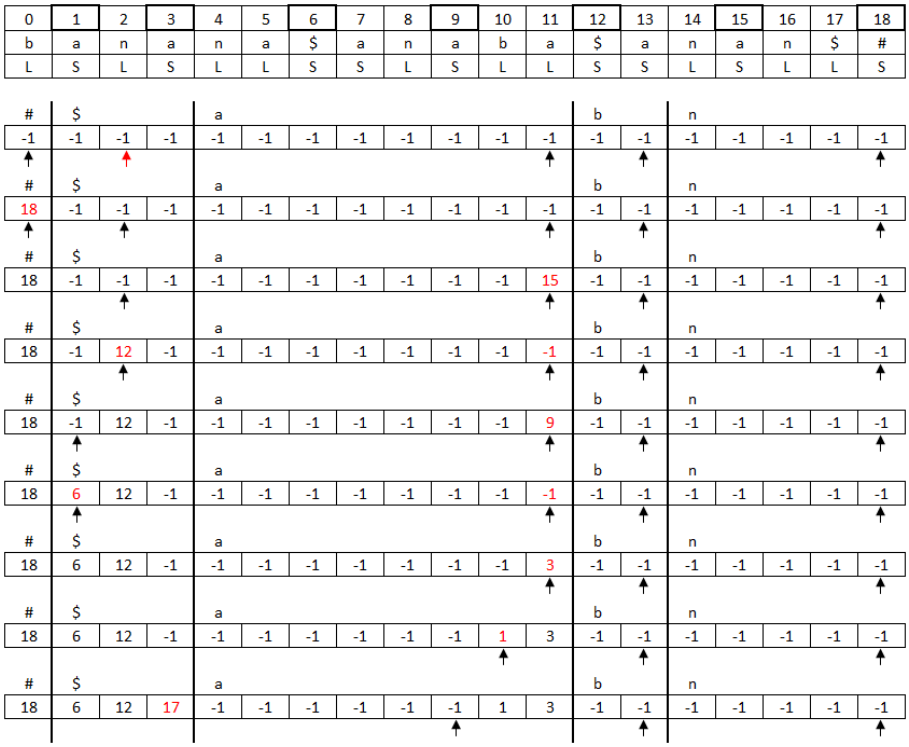
Polje P_1 lako se odredi iz polja t: P_1 = {1, 3, 6, 9, 12, 15, 18}.

Tablica 4.1: Tipovi znakova niza T^{cat} = "banana$anaba$anan$#"

Indeksi	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
niz S	b	a	n	a	n	a	$	a	n	a	b	a	$	a	n	a	n	$	#
tip znaka	L	S	L	S	L	L	S	S	L	S	L	L	S	S	L	S	L	L	S

U trećem se koraku algoritmi počinju razlikovati. Polje se prvo inicijalizira na -1, a zatim se svi pokazivači iz polja B postave na krajeve pretinaca. Kako bi sufiks s_{17} bio pravilno sortiran moramo mu rezervirati zadnje mjesto u pretincu pa se pokazivač tog pretinca postavlja na jedno polje ulijevo. Sada možemo krenuti dodavati indekse LMS-podnizova u SA. Razlika u odnosu na SAIS je što ovoga puta prolazimo po nizu zdesna nalijevo, odnosno od kraja prema početku. Također moramo paziti na sufikse koji počinju separatorom, odnosno na njihove sljedbenike koje ne dodajemo u niz. Kako je ranije napisano, svaki puta kada naiđemo na LMS-podniz dodamo ga u SA u pripadajući pretinac, no ako trenutni sufiks započinje separatorom brišemo sufiks koji smo prethodno dodali. Ovaj postupak prikazan je na slici 4.1. Zacrvenjeni su članovi koji su tek dodani ili oni koji su tek izbrisani. Možemo primijetiti da je indeks 15 u trećem redu dodan na kraj pretinca za znak "a". Idući LMS-podniz na koji nailazimo je onaj na indeksu 12. Budući da sufiks s_{12} započinje separatorom "$", indeks 12 dodajemo u pretinac sa separatorima, a uz to brišemo indeks 15 iz pretinca "a".

Sljedeći koraci induciranog sortiranja LMS-podnizova (dodavanje L-tip sufiksa na početke i S-tip sufiksa na krajeve pretinaca) ne razlikuju se od istih koraka u SAIS

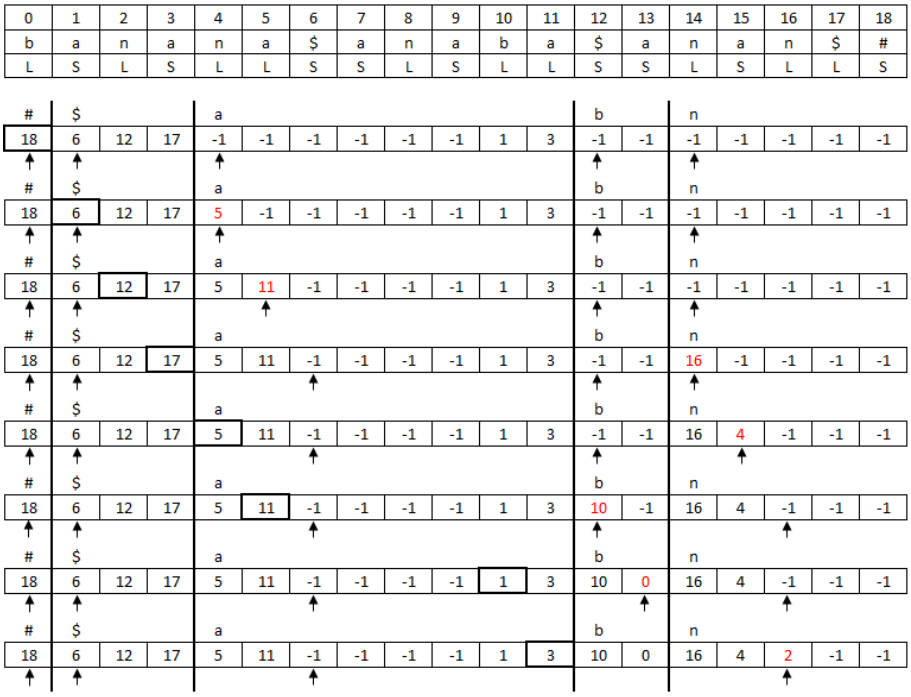


Slika 4.1: inducirano sortiranje LMS-podnizova za niz "banana$anaba$anan$#"
Uokvireni su početni indeksi LMS-podnizova. Strelicama su prikazani pokazivači polja B. U prvom koraku zacrvenjena je strelica koja pokazuje na predzadnje mjesto pretinca jer je zadnje mjesto rezervirano za sufiks koji počinje zadnjim separatorom. U svim koracima zacrvenjeni su članovi koji su dodani ili oni koji su izbrisani (crveni -1). U zadnjem koraku "ručno" se dodaje zadnji sufiks koji počinje separatorom.

algoritmu, osim što ne dodajemo sufikse koji započinju separatorom. Ovi postupci prikazani su na slikama 4.2 i 4.3. Uokvireni su članovi koji se trenutno obrađuju, a zacrvenjeni su novo dodani članovi. Koraci u kojima nema promjene nisu prikazani.

Nakon induciranog sortiranja LMS-podnizova, ponovno prolazimo po nizu slijeva nadesno i izravno dodajemo sve sufikse koji započinju separatorom u njihov pretinac od početka. U ovom primjeru, sufiksi su već bili dobro poredani, no to nije uvijek slučaj.

Nakon sortiranja LMS-podnizova slijedi imenovanje. Prolaskom po SA polju slijeva nadesno prvo nailazimo na LMS-podniz s indeksom 18 ("#") te mu dodijelimo ime 0. Idući po redu je onaj s indeksom 6 ("$ana") pa on dobiva ime 1. Ovaj postupak nastavljamo sve dok ne dođemo do kraja niza. Iako bi se na prvu moglo reći da imamo dva jednaka LMS-podniza ("$ana", na indeksima 6 i 12), oni nisu jednaki jer razlikujemo simbole "$" na različitim indeksima. Zato svi LMS-podnizovi dobivaju jedinstvena imena te nema potrebe za rekurzijom.



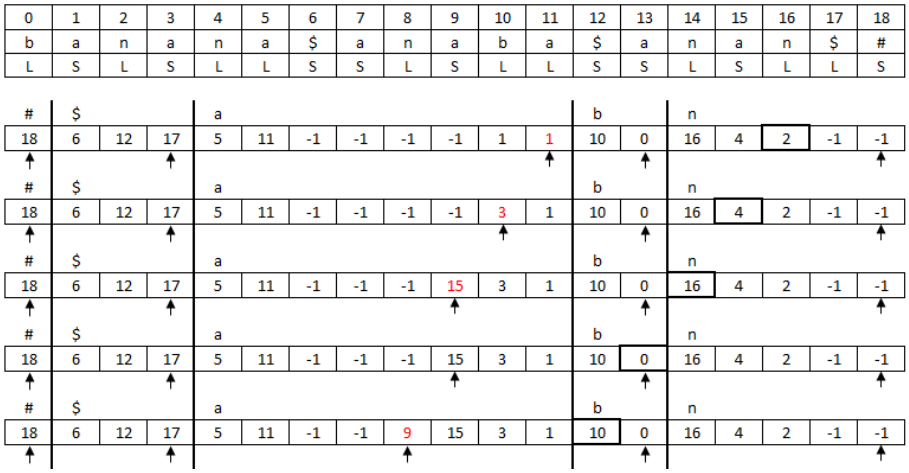
Slika 4.2: Dodavanje sufiksa L-tipa u koraku induciranog sortiranja LMS-podnizova za niz T^{cat} = "banana$anaba$anan$#"

Uokviren je član polja koji se upravo obraduje, zacrvenjeni su novi, upravo dodani članovi, a strelice označavaju pokazivače polja B.

Zadnji je korak jednak kao i kod originalnog SAIS algoritma pa neće biti opisan u ovom dijelu. Na kraju dobivamo konačno sufiksno polje SA:

$$\text{SA} = \{18\}, \{6, 12, 17\}, \{5, 11, 9, 15, 3, 7, 13, 1\}, \{10, 0\}, \{16, 4, 8, 14, 2\}$$

Sufiksno polje SA, s odgovarajućim sufiksima iz niza T^{cat} = "banana$anaba$anan$#" prikazano je u tablici 4.2.



Slika 4.3: Dodavanje sufiksa S-tipa u koraku induciranog sortiranja LMS-podnizova za niz T^{cat} = "banana$anaba$anan$#"

Uokviren je član polja koji se upravo obraduje, zacrvenjeni su novi, upravo dodani članovi, a strelice označavaju pokazivače polja B.

Tablica 4.2: sufiksno polje niza T^{cat} = "banana$anaba$anan$#"

i	SA	sufiks
0	18	#
1	6	$
2	12	$
3	17	AC$
4	5	a$
5	11	a$
6	9	aba$
7	15	an$
8	3	ana$
9	7	anaba$
10	13	anan$
11	1	anana$
12	10	ba$
13	0	banana$
14	16	n$
15	4	na$
16	8	naba$
17	14	nan$
18	2	nana$

5. Pretraživanje

Kao što je spomenuto u uvodu, sufiksna su polja struktura koja se koristi za pronalazak uzorka u tekstu, i to tako da se prvo indeksira tekst T, a zatim se taj indeks koristi za pronalazak uzorka P. Indeksiranje teksta T provodi se izgradnjom sufiksnog polja. Jednom kada je polje izgrađeno nad nizom S, ono se može koristiti za pretraživanje podnizova u tom nizu.

Sufiksno polje sadrži indekse sortiranih sufiksa početnog niza te se zato za pretraživanje može koristiti pretraživanje slično binarnom. Kod binarnog pretraživanja problem se u svakoj iteraciji prepolavlja pa je vremenska složenost algoritma O($\log n$). Kod pretraživanja podnizova u sufiksnom polju koristi se binarno pretraživanje za pronalazak prvog slova podniza u sufiksnom polju, no zatim je potrebno od tog početnog slova linearno pretražiti početni niz te je zato vremenska složenost O($m \log n$), gdje je m duljina podniza.

Kod pretraživanja niza vrlo dugim podnizovima, ova složenost može predstavljati problem. Ranije spomenuta poboljšana sufiksna polja (engl. *enhanced suffix arrays; ESA*), točnije sufiksna polja proširena poljem najduljih zajedničkih prefiksa (engl. *longest common prefix; LCP*) mogu smanjiti ovu složenost. Svaki član polja najduljih zajedničkih prefiksa, LCP[i] sadrži duljinu najduljeg zajedničkog prefiksa sufiksa SA[i] i sufiksa SA[i-1]. Ovo polje može se izgraditi paralelno sa sufiksnim poljem, u linearnom vremenu, a može smanjiti vremensku složenost pretraživanja na O($m + \log n$) (Manber i Myers, 1993). Kod takvog pretraživanja ponovno se koristi binarno pretraživanje za pronalazak prvog slova podniza u sufiksnom polju, no zatim nije potrebno svaki puta linearno pretraživati početni niz, već se koriste informacije o najduljim zajedničkim prefiksima.

U sklopu ovog rada nije implementirano LCP polje pa se za pretraživanje koristi binarno pretraživanje, čiji je pseudokod prikazan algoritmom 2.

Algorithm 2 Pretraživanje podnizova u sufiksnom polju

Ulaz: S - niz znakova, SA - sufiksno polje ulaznog niza, n - veličina sufiksnog polja, x - podniz koji tražimo

Izlaz: 1 ako se podniz nalazi u nizu, 0 inače

$left \leftarrow 0, right \leftarrow n - 1$

while $left \leq right$ **do**

  $i \leftarrow 0, middle \leftarrow \frac{left+right}{2}, index \leftarrow SA[middle]$

  if $s[index] = x[i]$ **then**

    $found \leftarrow$ **true**

    for $i \leftarrow 1$ **to** $|x|$ **do**

      $index \leftarrow index + 1$

      if $x[i] \neq s[index]$ **then**

        $found \leftarrow$ **false**

        break

      end if

    end for

    if $found$ **then**

      return true

    end if

  end if

  if $x[i] < s[index]$ **then**

    $right \leftarrow middle - 1$

  else

    $left \leftarrow middle + 1$

  end if

end while

return false

6. Implementacija i rezultati

Implementaciju gSAIS algoritma ostvarila sam u programskom jeziku C++[1]. U sklopu implementacije korištena je originalna implementacija SAIS algoritma (Nong et al., 2011), koja je zatim proširena u gSAIS algoritam. Osim samog algoritma izgradnje sufiksnog polja, implementiran je i algoritam binarnog pretraživanja opisan u prethodnom poglavlju. Testiranje je provedeno na osobnom prijenosnom računalu s procesorom Intel® Core™ i7-1255U CPU i 16 GB RAM-a na operacijskom sustavu Ubuntu 22.04.2 (64-bit).

Podaci korišteni za testiranje su rezultati očitanja dobivenih sekvenciranjem genoma preuzeti s web-stranice *National Center for Biotechnology Information*[2].

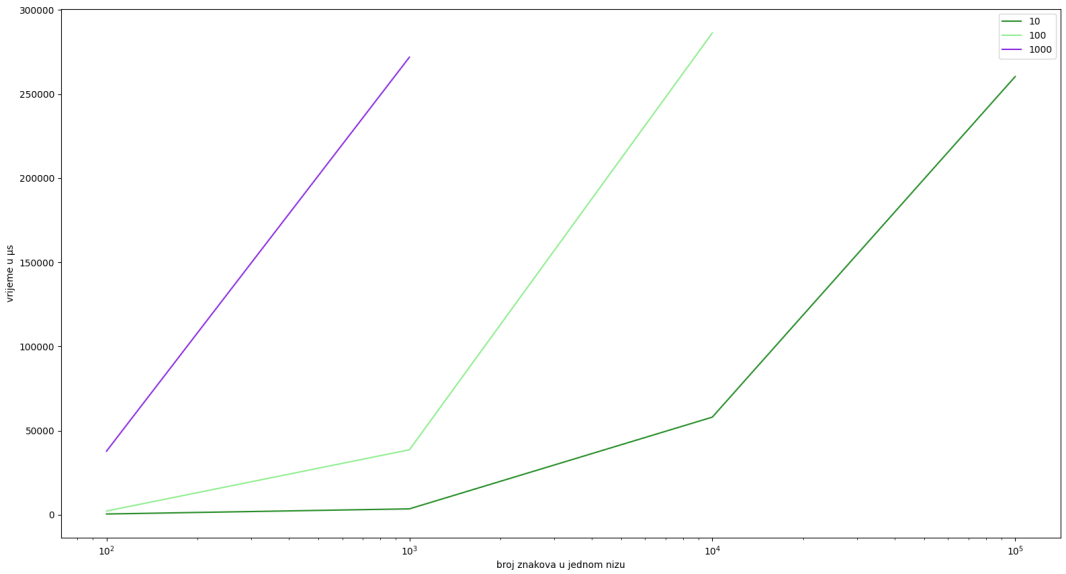
Testiranje je provedeno na sintetskim podacima duljine $10^2, 10^3, 10^4, 10^5$ i 10^6 znakova. Sintetski podaci generirani su kao nasumični podnizovi genoma E. coli. Na primjer, za stvaranje jednog podniza duljine 10^2 odabran je nasumičan indeks u datoteci u kojoj se nalazi genom E. coli te je uzeto 10^2 znakova od tog indeksa na dalje. Budući da su svi podaci stvoreni iz genoma, i to pročišćenog, abeceda se sastoji od znakova A, C, G i T. Tako generirani nizovi zatim su spojeni u kolekcije duljine 10, 100 i 1000 nizova te je nad njima izgrađeno sufiksno polje. Kao separator između nizova korišten je znak '1', a kao znak za kraj niza iskorišten je '\0', koji se u programskom jeziku C++ automatski dodaje na kraj svakog niza.

Grafički prikaz vremena izgradnje sufiksnog polja prikazan je na slici 6.1. Na x osi prikazana je duljina pojedinačnog niza u kolekciji, a na y osi prikazano je vrijeme u μs. Linije prikazuju vrijeme potrebno za izgradnju sufiksnog polja nad kolekcijama od 10, 100 i 1000 nizova. Linije na grafu nalikuju eksponencijalnoj funkciji, no x os je prikazana u logaritamskoj skali, što zapravo znači da vrijeme potrebno za izgradnju sufiksnog polja linearno raste s porastom ukupne duljine kolekcije. Bitno je na grafu uočiti da je vrijeme potrebno za izgradnju sufiksnog polja nad kolekcijom od 10 nizova duljine 10^4 znakova približno jednako vremenu za izgradnju sufiksnog polja nad

[1] https://github.com/hjf11/gSAIS

[2] https://www.ncbi.nlm.nih.gov/

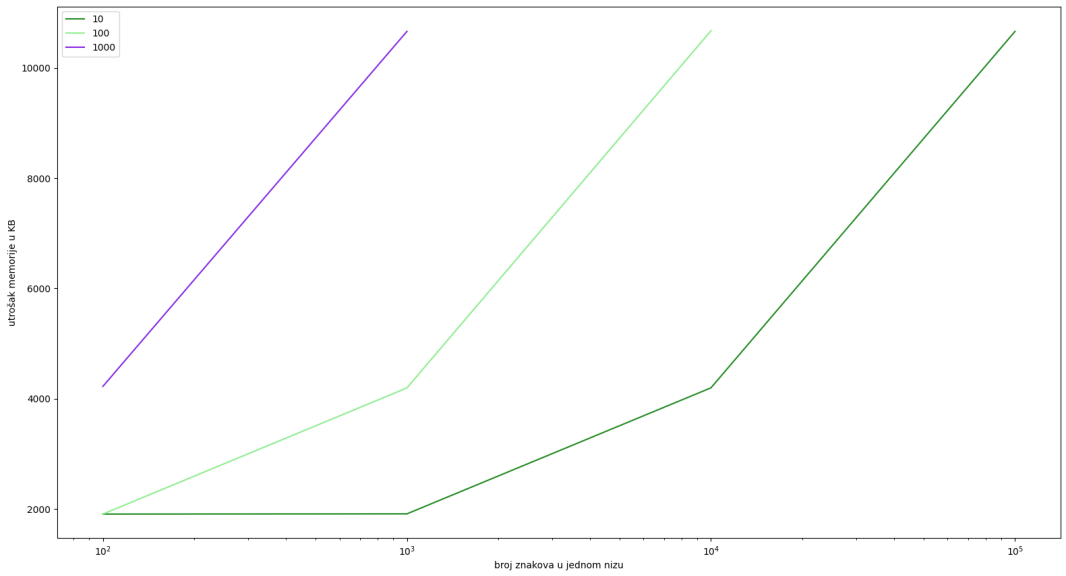
kolekcijom od 100 nizova duljine 10^3 znakova, što je približno jednako vremenu za izgradnju polja nad kolekcijom od 1000 nizova duljine 10^2 znakova. Sve spomenute kolekcije sastoje se od red veličine 10^5 znakova te im je zato vrijeme izgradnje otprilike jednako. Drugim riječima, vrijeme potrebno za izgradnju sufiksnog polja nad kolekcijom nizova ne ovisi direktno o pojedinačnim nizovima ni njihovom broju, već isključivo o ukupnoj duljini kolekcije.



Slika 6.1: Vrijeme izgradnje sufiksnog polja nad kolekcijama nizova
Na x osi prikazane su duljine početnih nizova u logaritamskoj skali, a na y osi prikazano je vrijeme u μs. Linije prikazuju vrijeme potrebno za izgradnju sufiksnog polja nad 10, 100 i 1000 nizova različitih duljina.

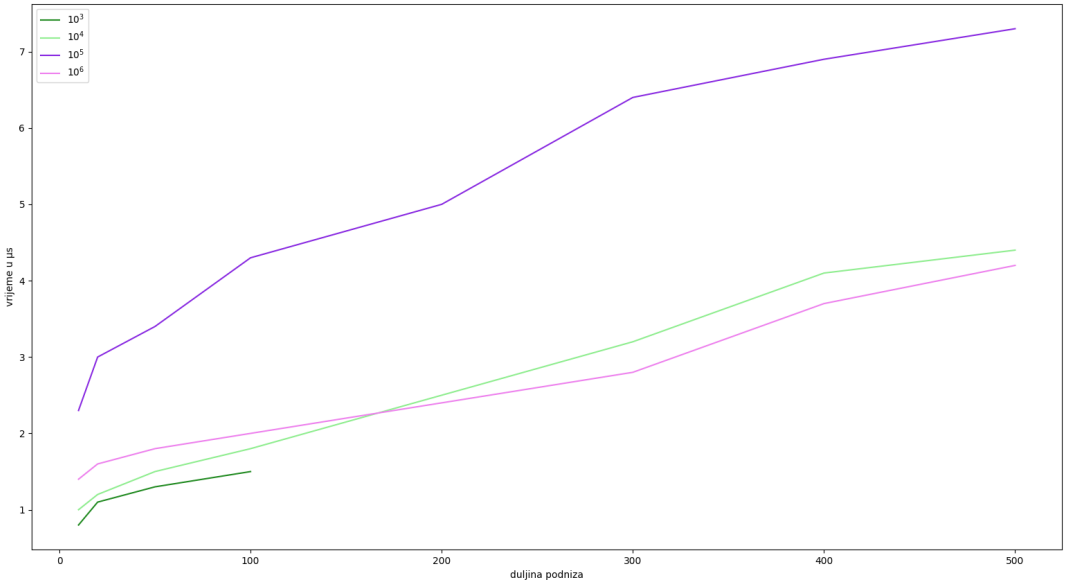
Slika 6.2 prikazuje grafički prikaz utroška memorije prilikom izgradnje sufiksnog polja. Na osi x ponovno su logaritamski prikazane duljine pojedinačnih nizova u kolekciji, a na osi y prikazana je memorija u KB. Linije prikazuju memoriju koja je utrošena prilikom izgradnje sufiksnog polja nad kolekcijama koje se sastoje od 10, 100 i 1000 nizova. Linije ponovno podsjećaju na eksponencijalnu funkciju te se zbog logaritamske skale može zaključiti da se radi o linearnom rastu. Utrošak memorije prilikom izgradnje sufiksnog polja nad kolekcijom nizova raste linearno s povećanjem ukupne veličine kolekcije.

Kao što je prethodno spomenuto, osim izgradnje sufiksnog polja implementiran je i algoritam binarnog pretraživanja. Za testiranje pretraživanja podnizova u nizovima također su korišteni sintetsko generirani podaci. Podaci su generirani na sličan način kao i za testiranje izgradnje sufiksnog polja, samo što su generirani iz prethodnih sintetskih podataka umjesto direktno iz genoma E. coli. Na primjer, ako se testira pretraživanje



Slika 6.2: Utrošak memorije izgradnje sufiksnog polja nad kolekcijama nizova
Na x osi prikazane su duljine početnih nizova u logaritamskoj skali, a na y osi prikazana je utrošena memorija u KB. Linije prikazuju memoriju koja je zauzeta prilikom izgradnje sufiksnog polja nad 10, 100 i 1000 nizova različitih duljina.

nad kolekcijom od 10 nizova duljine 10^2 znakova, onda su podnizovi generirani upravo iz tih 10 nizova kako bi se osiguralo da će podnizovi uvijek biti prisutni u kolekciji nad kojom se pretražuje. Testiranje je provedeno nad podnizovima duljine 10, 20, 50, 100, 200 i 500 znakova. Grafički prikaz rezultata prikazan je na slici 6.3. Os x prikazuje duljinu podniza koji tražimo, a os y prikazuje vrijeme u μs. Linije prikazuju vrijeme potrebno za pronalazak podniza unutar kolekcije čija je ukupna duljina $10^3, 10^4, 10^5$ i 10^6 znakova. Tamnozelena linija prestaje nakon podniza duljine 100 jer ona prikazuje vrijeme potrebno za pronalazak podnizova unutar kolekcije koja se sastoji od ukupno 10^3 znakova, a ta se kolekcija zapravo sastoji od 10 nizova duljine 10^2 znakova pa nije moguće pronaći podniz duljine 200, 300 ili više znakova u njoj. Sve linije prikazane na grafu podsjećaju na logaritamsku funkciju, što je i bilo za očekivati budući da je složenost ovakvog pretraživanja O($m \log n$).



Slika 6.3: Vrijeme potrebno za pretraživanje podnizova u kolekciji nizova
Na x osi prikazane su duljine podnizova, a na y osi prikazano je vrijeme u μs. Linije prikazuju vrijeme potrebno za pronalazak podniza određene duljine unutar kolekcije čija je ukupna duljina $10^3, 10^4, 10^5$ i 10^6 znakova.

6.1. Budući rad

Već je spomenuto da je binarno pretraživanje samo jedno od rješenja za pretraživanje sufiksnog polja i da postoje bolja. Sufiksno polje može se proširiti poljem najduljih zajedničkih prefiksa (engl. *longest common prefix; LCP*), koje omogućuje pretraživanje podniza duljine m u vremenu O($m + \log n$), za razliku od binarnog pretraživanja koje iziskuje vrijeme O($m \log n$) (Manber i Myers, 1993). Također, pretraživanje pomoću LCP polja omogućuje da se za određeni podniz odredi koliko se on puta pojavio u kolekciji.

Sufiksno polje izgrađeno nad kolekcijom nizova može se dodatno proširiti i dokumentirajućim poljem (engl. *document array*), koje za svaki sufiks pamti točno iz kojeg je niza došao. Ovakvim proširenjem moguće je, umjesto informacije da se podniz nalazi u kolekciji, doznati točno u kojem se nizu kolekcije on nalazi. Sufiksno polje prošireno ovakvim podatkovnim strukturama naziva se poboljšano sufiksno polje (engl. *enchanced suffix array; ESA*), moguće ga je izgraditi u linearnom vremenu te je ekvivalentno sufiksnom stablu, uz manje memorijsko zauzeće.

U ovom radu razmotren je problem pronalaska uzorka u tekstu, odnosno problem pronalaska određenog podniza u sljedovima velikih bioloških molekula poput DNA ili proteinskog sljedova. U bioinformatici se taj problem često javlja te je bitno

imati strukture podataka koje omogućavaju brzo i efikasno pretraživanje velikih slijedova manjim podnizovima. Ovakvo pretraživanje još se naziva i potpuno podudaranje (engl. *exact matching*) jer se u slijedu traži potpuno ista sekvenca kao ona u podnizu. Ipak, zbog prirode bioloških sekvenci ponekad bi bilo bolje dopustiti da se sekvence tu i tamo razlikuju. Jedan primjer mogao bi biti traženje DNA slijeda koji kodira određeni protein. Jednu aminokiselinu kodira više različitih kodona, koji se najčešće razlikuju u zadnjoj poziciji. Ako se traži DNA slijed u nekom genomu koji bi mogao kodirati specifičan proteinski slijed, moglo bi se dopustiti nepodudaranje sekvence na svakoj trećoj poziciji (Shrestha et al., 2014). Ovaj problem naziva se problem nepotpunog podudaranja (engl. *inexact matching*). Postoje različite vrste nepotpunog podudaranja, neke od njih dopuštaju bilo kakva nepodudaranja na unaprijed određenim pozicijama, neke dopuštaju samo određene vrste nepodudaranja na unaprijed određenim pozicijama, a neke dopuštaju nepodudaranja koja su unutar određene Hammingove udaljenosti (Shrestha et al., 2014).

7. Zaključak

Često korištene strukture podataka za indeksiranje teksta su sufiksna stabla i sufiksna polja. Obje spomenute strukture moguće je izgraditi u linearnom vremenu, no sufiksna polja zauzimaju puno manje memorije pa se zato češće koriste (Domazet-Lošo i Šikić, 2014). Ipak, ona imaju neke nedostatke u odnosu na sufiksna stabla pa se često nadograđuju i proširuju nekim drugim jednostavnim strukturama. Algoritam gSAIS (Louza et al., 2017) proširenje je najpopularnijeg algoritma za izgradnju sufiksnog polja, SAIS (Nong et al., 2009), koje omogućuje izgradnju polja nad kolekcijom nizova u linearnom vremenu. Ovako izgrađeno sufiksno polje omogućava efikasno pretraživanje cijele kolekcije nizova raznim podnizovima.

LITERATURA

Nacional center for biotechnology information. URL https://www.ncbi.nlm.nih.gov/.

Mohamed Ibrahim Abouelhoda, Stefan Kurtz, i Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

Robert S Boyer i J Strother Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

Mirjana Domazet-Lošo i Mile Šikić. *Bioinformatika - skripta*. 2014.

Juha Kärkkäinen i Peter Sanders. Simple linear work suffix array construction. 2003.

Donald E Knuth, Jr James H Morris, i Vaughan R Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

Pang Ko i Srinivas Aluru. Space efficient linear time construction of suffix arrays. U *Combinatorial Pattern Matching*, stranice 200–210, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

Felipe A. Louza, Simon Gog, i Guilherme P. Telles. Inducing enhanced suffix arrays for string collections. *Theoretical Computer Science*, 678:22–39, 2017. URL https://www.sciencedirect.com/science/article/pii/S0304397517302621.

Udi Manber i Gene Myers. Suffix arrays: A new method for on-line string searches. *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms, SODA '90*, stranice 319–327, 1990.

Udi Manber i Gene Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

Ge Nong. Practical linear-time o(1)-workspace suffix sorting for constant alphabets. 31(3):1–15, 2013.

Ge Nong, Sen Zhang, i Daricks Wai Hong Chan. Linear suffix array construction by almost pure induced-sorting. stranice 193–202, 2009.

Ge Nong, Sen Zhang, i Daricks Wai Hong Chan. Two efficient algorithms for linear time suffix array construction. *Computers, IEEE Transactions on*, 60:1471 – 1484, 2011.

Anish Man Singh Shrestha, Martin C. Frith, i Paul Horton. A bioinformatician's guide to the forefront of suffix array construction algorithms. *Briefings in Bioinformatics*, 15(2):138–154, 2014.

Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

Peter Weiner. Linear pattern matching. *IEEE*, stranice 1–11, 1973.

Implementacija učinkovitog algoritma za izgradnju poboljšanog sufiksnog polja

Sažetak

U bioinformatici se često javlja potreba za pretraživanjem velikih nizova (DNA i proteinskih sljedova) različitim podnizovima. Kako bi se taj problem riješio u razumnom vremenu, potrebno je koristiti efikasne strukture indeksiranja teksta, kao što su sufiksno stablo i sufiksno polje, koje se mogu izgraditi u linearnom vremenu. Sufiksno polje samo po sebi ima neke nedostatke u odnosu na sufiksno stablo pa se zato često proširuje i nadograđuje. Algoritam gSAIS nadogradnja je popularnog SAIS algoritma za izgradnju sufiksnog polja koja omogućuje izgradnju sufiksnog polja nad kolekcijom nizova.

Ključne riječi: bioinformatika, sufiksno polje, SAIS, gSAIS

Implementation of the efficient enhanced suffix-array construction algorithm

Abstract

In bioinformatics there is often need to search large strings (DNA and protein sequences) with different substrings. In order to solve this problem in reasonable time, it is necessary to use efficient text indexing structures, such as suffix trees and suffix arrays, which can be built in linear time. The suffix array itself has some disadvantages compared to the suffix tree and is often expanded and upgraded. The gSAIS algorithm is an upgrade of the popular SAIS suffix array construction algorithm that allows the construction of a suffix array over a collection of strings.

Keywords: bioinformatics, suffix array, SAIS, gSAIS