SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4186

Poboljšano sufiksno polje

Antea Hadviger

Zagreb, lipanj 2015.

Zahvaljujem mentoru Mili Šikiću na pomoći i brzom odgovaranju na mailove.

Sadržaj

1. Uvod

Rješavanje problema pronalaženja pojavljivanja nekog uzorka u zadanom tekstu, odnosno dugačkom nizu znakova, jedan je od osnovnih zadataka u bioinformatici. U većini primjena u bioinformatici, radi se o veoma velikoj količini podataka. Genom bakterije sadrži i do 10^7 parova baza, a drugi organizmi i do nekoliko redova veličine više. Nad takvim tekstovima provodi se velik broj upita. U tim slučajevima, potrebno je efikasnim strukturama izgrađenim nad zadanim podacima podržati što brže pretraživanje jer su naivna rješenja neprihvatljiva zbog svoje prevelike složenosti. Algoritamska rješenja problema pronalaska uzorka u tekstu možemo podijeliti na dvije skupine. Prva su skupina rješenja u kojima se prvo indeksira uzorak P (engl. *pattern*), a onda se indeks izgrađen nad uzorkom koristi za pretraživanje teksta T u složenosti $O(|T|)$. Najpoznatiji algoritmi koji koriste takav pristup su Knuth-Morris-Prattov algoritam (Knuth et al., 1977) i Boyer-Mooreov algoritam (Boyer i Moore, 1977). U drugu skupinu možemo svrstati rješenja u kojima se prvo indeksira tekst, a zatim se indeks izgrađen nad tekstom koristi za pronalazak uzorka. Takva rješenja rade u složenosti $O(|P|)$ i kao takva su pogodnija za primjenu u bioinformatici gdje je tekst T često vrlo dugačak i pretražuje se mnogo različitih uzoraka.

Jedna takva efikasna i učinkovita podatkovna struktura koja se koristi za indeksiranje dugačkih tekstova jest sufiksno polje. Ono u suštini predstavlja niz početnih pozicija sufiksa u tekstu, sortiranih leksikografskim redom. Sufiksna polja jednostavna su za razumijevanje i implementaciju i predstavljaju bazu za druge, sofisticiranije tehnike i strukture za indeksiranje.

Sufiksna polja povezana su s drugom, nešto kompleksnijom strukturom podataka, poznatom kao sufiksno stablo. I sufiksna polja i sufiksna stabla nude vremenski učinkovite načine pretraživanja podnizova unutar teksta, kao i rješenja drugih, sličnih problema. Iz tih razloga te su strukture veoma popularne u bioinformatici jer imaju široku paletu primjena, kao što su poravnanje DNA očitanja ili određivanje sufiks-prefiks preklapanja između DNA sekvenci. Sufiksna

stabla pate od problema velikog zauzeća memorije i zbog toga se sve manje primjenjuju. Optimalna implementacija sufiksnog stabla zahtijeva 20 okteta memorije po ulaznom znaku, dok u praksi obrada DNA očitanja zahtijeva oko 12.55 okteta po ulaznom znaku. Sufiksna polja, s druge strane, u najjednostavnijem obliku zauzimaju samo 4 okteta po ulaznom znaku. Činjenica jest da su sufiksna stabla mnogo funkcionalnija i moćnija od običnog sufiksnog polja pa prethodna usporedba nije posve opravdana. Nasreću, pokazalo se da sufiksna polja nadograđena drugim jednostavnim strukturama, kao što je polje najduljih zajedničkih podniza, mogu u potpunosti zamijeniti sufiksna stabla, čak i njihove najkompleksnije funkcije (Abouelhoda et al., 2004). Štoviše, takva se poboljšana sufiksna polja (engl. *enhanced suffix array; ESA*) mogu konstruirati u linearnom vremenu, kako je kasnije i pokazano.

2. Osnovne sastavnice poboljšanog sufiksnog polja

2.1. Sufiksno polje

Neka je S niz znakova duljine n čiji su znakovi elementi abecede Σ. Također, neka niz S završava posebnim znakom \$ koji je leksikografski manji od svih ostalih znakova abecede Σ i ne pojavljuje se nigdje drugdje unutar S.

Definirajmo $S[i, j]$ kao podniz niza S koji započinje na i-tom, a završava na j-tom indeksu. Sufiks niza S je svaki njegov podniz koji završava posljednjim znakom, tj. $S[i, n], 0 \leq i < n$. Sufiks koji počinje na i-tom indeksu označavat ćemo sa s_i ili $suf(S, i)$.

Sufiksno polje (engl. *suffix array*) SA je niz cijelih brojeva koji označavaju početne pozicije abecedno poredanih sufiksa niza S. Dakle, vrijedi da $SA[i]$ označava početnu poziciju i-tog najmanjeg sufiksa niza S.

Primjer: Neka je zadan ulazni niz znakova S = ACTTA. Na kraj niza dodajemo posebni znak \$, jedinstven i abecedno veći od svih ostalih znakova abecede. Sufiksi niza S\$ su sljedeći: s_0 = ACTTA\$, s_1 = CTTA\$, s_2 = TTA\$, s_3 = TA\$, s_4 = A\$, s_5 = \$.

Tablica 2.1: Sufiksno polje niza ACTTA\$

i	$SA[i]$	s_i
0	0	ACTTA\$
1	4	A\$
2	1	CTTA\$
3	3	TA\$
4	2	TTA\$
5	5	\$

Tablica 2.1 prikazuje sufikse poredane abecednim redom. Dakle, sufiksno polje SA sadrži početne pozicije abecedno poredanih sufiksa, odnosno, $SA = 0, 4, 1, 3, 2, 5$.

2.2. Inverzno sufiksno polje

Inverzno sufiksno polje (engl. *inverse suffix array; ISA*) koristi se u određivanju polja najduljih zajedničkih prefiksa. Definirano je tako da vrijedi:

$$ISA[i] = j \leftrightarrow SA[j] = i$$

2.3. Polje najduljih zajedničkih prefiksa

Polje najduljih zajedničkih prefiksa (engl. *longest common prefix array; LCP array*) jedan je od dodataka sufiksnom polju s kojim čini poboljšano sufiksno polje. LCP niz (ili LCP tablica) koja pripada nizu S sadrži jednak broj cjelobrojnih članova kao i S. Svaki član niza $LCP[i]$ sadrži duljinu najduljeg zajedničkog prefiksa sufiksa $suf(S, SA[i])$ i $suf(S, SA[i-1])$, $1 \leq i \leq n$. Potrebno je definirati i $LCP[0] = 0$, kao i napomenuti da uvijek vrijedi $LCP[n] = 0$ jer niz završava jedinstvenim posebnim znakom \$.

Polje najduljih zajedničkih prefiksa može se računati paralelno s izgradnjom sufiksnog polja ili naknadno.

2.3.1. Intervali najduljih zajedničkih prefiksa

Intervali najduljih zajedničkih prefiksa ili, kraće, LCP-intervali (engl. *longest common prefix intervals; LCP-intervals*) koriste se u algoritmima nad poboljšanim sufiksnim poljem koji su ekvivalentni algoritmima nad sufiksnim stablom, no mnogo su jednostavniji.

Definicija 1. *Interval [i..j], $0 \leq i < j \leq n$ je LCP-interval s LCP-vrijednošću l ako:*

1. $LCP[i] < l$
2. $LCP[k] \geq l$ *za svaki* k, $i + 1 \leq k \leq j$
3. $LCP[k] = l$ *za barem jedan* k, $i + 1 \leq k \leq j$

4. $LCP[j+1] < l$

Dakle, LCP-vrijednost intervala duljina je najduljeg zajedničkog prefiksa svih nizova unutar LCP-intervala. Koristit će se i oznaka *l-interval* (ili *l*-$[i..j]$) za LCP-interval LCP-vrijednosti l. Svaki indeks k, $i+1 \leq k \leq j$ za koji je $LCP[k] = l$ naziva se *l-indeksom*.

Primjer: Neka je zadan niz znakova S = CAAACATAT$.

Tablica 2.2: Polja niza CAAACATAT$

i	$SA[i]$	$LCP[i]$	s_i
0	2	0	AAACATAT$
1	3	2	AACATAT$
2	0	1	ACAAACATAT$
3	4	3	ACATAT$
4	6	1	ATAT$
5	8	2	AT$
6	1	0	CAAACATAT$
7	5	2	CATAT$
8	7	0	TAT$
9	9	1	T$
10	10	0	$

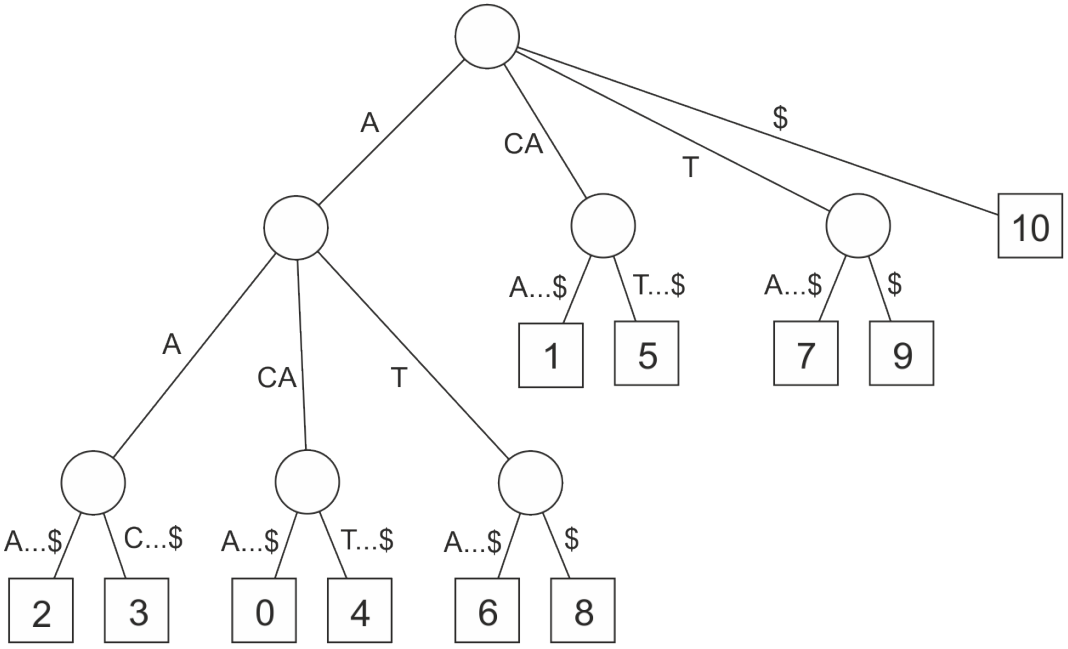
Tablica 2.2 prikazuje polja SA i LCP za niz S. Primijetimo da je $LCP[i] = 0$ za svaki i za koji vrijedi da je s_i prvi sufiks u sufiksnom polju s nekim početnim znakom.

Primjerice, interval [0..5] je LCP-interval s LCP-vrijednošću 1 jer je $LCP[0] = 0 < 1$, $LCP[6] = 0 < 1$, $LCP[i] \geq 1, 1 \leq i \leq 5$ i $LCP[2] = LCP[4] = 1$. Dakle, [0..5] je *1-interval*, a 2 i 4 su njegovi *1-indeksi*. Slično, [2..3] je *3-interval*, a 3 je tada *3-indeks*.

2.3.2. Stablo intervala najduljih zajedničkih prefiksa

Poboljšano sufiksno polje u potpunosti može zamijeniti sufiksno stablo (engl. *suffix tree*), popularnu i efikasnu strukturu za indeksiranje teksta (Abouelhoda et al., 2004). Sufiksno stablo izgrađeno nad nizom S ukorijenjeno je stablo koje sadrži sve sufikse tog niza. U listovima stabla zapisana su početna mjesta sufiksa

u nizu S indeksima od 0 do n. Iz korijena stabla izlazi točno onoliko bridova (grana) koliko ima znakova abecede kojom je izgrađen niz S, uključujući i završni znak \$. Svaki unutarnji čvor ima barem dvoje djece. Svaka oznaka brida sadržava neki podniz niza S, a oznake grana koje izlaze iz istog čvora moraju započinjati različitim znakovima abecede.



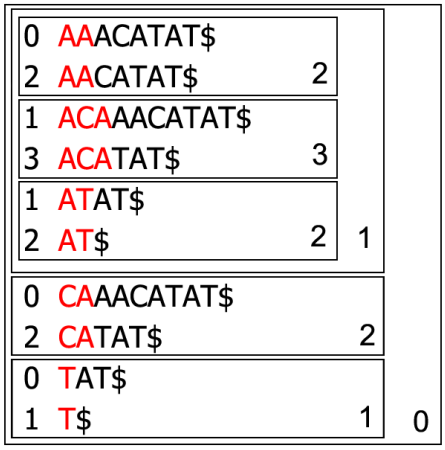
Slika 2.1: Sufiksno stablo niza ACAAACATAT\$. Listovi u stablu odgovaraju članovima sufiksnog polja.

Primjer: Promotrimo sufiksno stablo niza ACAAACATAT\$ prikazano na slici 2.1. Jedan od sufiksa tog niza jest s_6 = ACATAT\$. Pronađimo njegov list u sufiksnom stablu. Slijedimo put od korijena stabla tako da usporedimo prvi znak sufiksa sa svim prvim znakovima oznaka grana koje izlaze iz korijena. Oznaka "A" krajnje lijeve grane odgovara prvom znakom sufiksa pa stoga prelazimo u čvor u koji ona vodi. Nadalje, uspoređujemo drugi znak sufiksa s početnim znakom grana koje izlaze iz tog čvora i slijedimo put grane označene s "CA" jer njen prvi znak odgovara drugom znaku sufiksa, a također i drugi znak oznake odgovara trećem znaku sufiksa. Preostaje nam odlučiti hoćemo li slijediti lijevu ili desnu granu trenutnog čvora te odlazimo putem lijeve grane jer njen početni znak "A" odgovara četvrtom sufiksa. Tako dolazimo do lista označenog sa 6, što je upravo početni indeks sufiksa s_6 u nizu S

Ostaje pitanje kako postići analogiju između poboljšanog sufiksnog polja i sufiksnog stabla, za što je potrebno definirati stablo intervala najduljih zajedničkih

prefiksa.

Definicija 2. *Kažemo da je m-interval* $[l..r]$ *sadržan unutar l-intervala* $[i..j]$ *ako je on njegov podinterval* $(i \leq l < r \leq j)$ *te ako je* $m > l$*. Tada kažemo da interval* $[i..j]$ *obuhvaća interval* $[l..r]$*. Ako* $[i..j]$ *obuhvaća* $[l..r]$ *i ne postoji nijedan drugi interval sadržan unutar* $[i..j]$ *koji obuhvaća* $[l..r]$*, tada je* $[l..r]$ *dijete intervala* $[i..j]$ *unutar stabla LCP-intervala.*



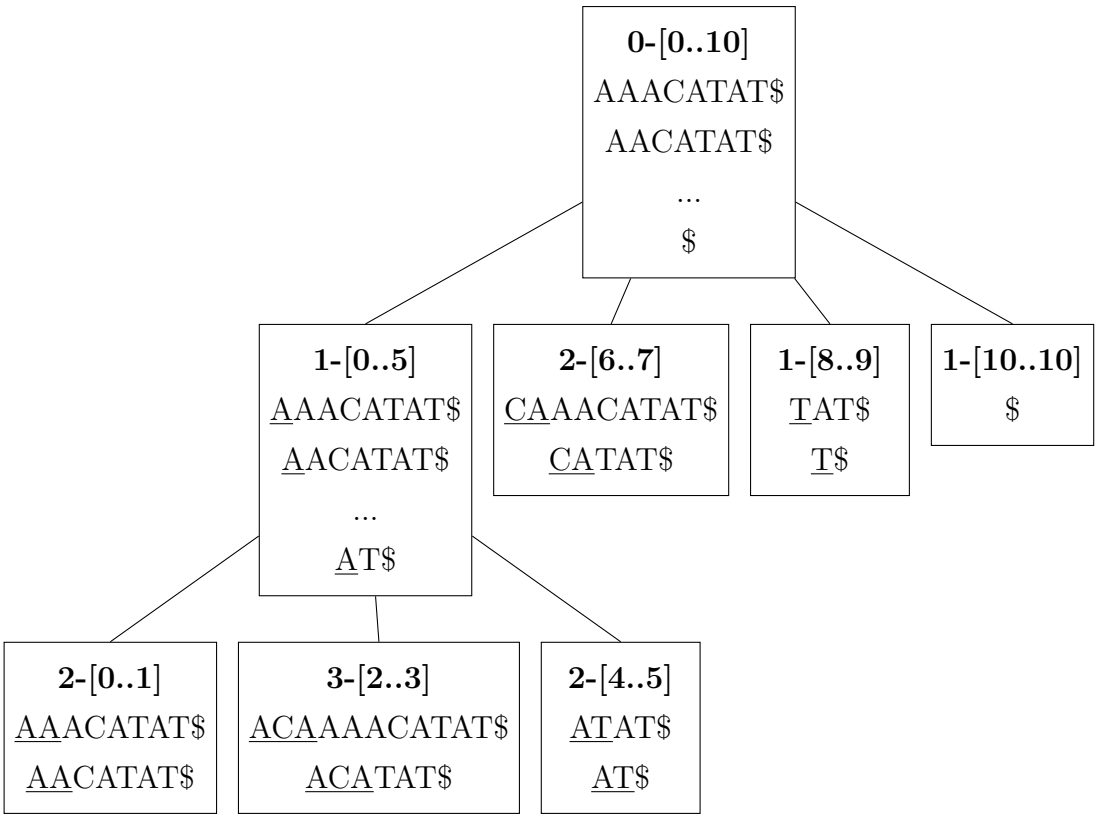
Slika 2.2: LCP-intervali niza ACAAACATAT$. Lijevo su navedene vrijednosti polja najduljih zajedničkih prefiksa, a desno LCP-vrijednosti intervala ograničenih pravokutnicima.

Svu djecu nekog LCP-intervala možemo odrediti pomoću njegovih *l-indeksa*. Ako je $[i..j]$ *l-interval* čiji su *l-indeksi* $i_1 < i_2 < ... < i_k$, tada su njegova djeca intervali $[i..i_1 - 1], [i_1..i_2 - 1], ..., [i_k..i_j]$. Neki od njih mogu biti sastavljeni samo od jednog člana.

Dokažimo da je to zaista tako. Neka je $[l..r]$ jedan od intervala $[i..i_1 - 1], [i_1..i_2 - 1], ..., [i_k..i_j]$. Ako $[l..r]$ ima samo jedan član, tada je sigurno dijete intervala $[i..j]$. Pretpostavimo da je $[l..r]$ *m-interval*. Kako $[l..r]$ ne sadržava nijedan *l-indeks*, slijedi da je $[l..r]$ sadržan unutar $[i..j]$. Zbog $LCP[i_1] = LCP[i_2] = ... = LCP[i_k] = l$, ne postoji nijedan interval sadržan unutar $[i..j]$ koji obuhvaća $[l..r]$. Dakle, interval $[l..r]$ je dijete intervala $[i..j]$. Također, osim navedenih, ne može postojati nijedno drugo dijete intervala $[i..j]$.

Temeljem prethodno navedenih činjenica i definicije možemo izgraditi stablo intervala najduljih zajedničkih prefiksa, ili *stablo LCP-intervala* (engl. *lcp-interval tree*) prikazano na slici 2.3. Korijen stabla jest 0-interval $[0..n]$. Postoji izravna veza između LCP-stabla intervala i sufiksnog stabla. Naime, stablo

LCP-intervala zapravo je sufiksno stablo bez listova. Preciznije, čvorovi stabla LCP-intervala odgovaraju unutarnjim čvorovima sufiksnog stabla, što možemo vidjeti usporedimo li sliku 2.3 sa slikom 2.1.



Slika 2.3: Stablo LCP-intervala niza ACAAACATAT\$. *l*-[*i..j*] označava LCP interval čije su granice i i j, a njegova je LCP-vrijednost l. Ispod oznake intervala navedeni su sufiksi koji se nalaze u tom intervalu.

2.3.3. Tablica djece

Kako bismo mogli ostvariti obilazak LCP-stabla, poželjno je za svaki *l-interval* [*i..j*] moći što brže odrediti njegovu djecu u stablu. Zato koristimo tablicu djece (engl. *child table*) koja ima jednak broj redaka kao i ulazni niz, a svaki redak sastoji se od 3 vrijednosti: *up* (gore), *down* (dolje) i *nextlIndex* (sljedeći *l-indeks*).

Za *l-interval*[*i..j*] čiji su l-indeksi $i_1 < i_2 < ... < i_k$, vrijednost *childtab*[*i*].*down* ili *childtab*[$j+1$].*up* (neke su vrijednosti redundantne) koristi se za određivanje najmanjeg *l-indeksa*, i_1. Drugi *l-indeksi*, $i_2, i_3, ..., i_k$, mogu se odrediti iz vrijednosti *childtab*[i_1].*nextlIndex*,...,*childtab*[i_{k-1}].*nextlIndex*, respektivno, jer je *childtab*[i_p].*nextlIndex* $= i_{p+1} (p = 1, 2, ..., k-1)$.

Svaku od triju vrijednosti moguće je odrediti na sljedeći način:

– $childtab[i].up = min\{q \in [0..i-1] \mid LCP[q] > LCP[i] \text{ i } \forall k \in [q+1..i-1] : LCP[k] \geq LCP[q]\}$

– $childtab[i].down = max\{q \in [i+1..n] \mid LCP[q] > LCP[i] \text{ i } \forall k \in [i+1..q-1] : LCP[k] > LCP[q]\}$

– $childtab[i].nextlIndex = min\{q \in [i+1..n] \mid LCP[q] = LCP[i] \text{ i } \forall k \in [i+1..q-1] : LCP[k] > LCP[q]\}$

Neke vrijednosti mogu ostati nedefinirane.

Možemo reći da tablica djece implicitno sadrži veze između roditelja i djece u stablu LCP-intervala pa tako pomoću nje možemo ostvariti obilazak stabla. Naime, kako je ranije pokazano, intervali $[i..i_1-1], [i_1..i_2-1], ..., [i_k..i_j]$ su djeca intervala $[i..j]$ pa nam tablica djece zaista omogućuje obilazak stabla. Kasnije je pokazano kako je pomoću tablice djece moguće odrediti djecu nekog intervala u konstantnom vremenu.

Tablica 2.3: Tablica djece niza CAAACATAT$

i	up	$down$	$nextlIndex$	s_i
0	-	2	6	AAACATAT$
1	-	-	-	AACATAT$
2	1	3	4	ACAAACATAT$
3	-	-	-	ACATAT$
4	3	5	-	ATAT$
5	-	-	-	AT$
6	2	7	-	CAAACATAT$
7	-	-	8	CATAT$
8	7	9	-	TAT$
9	-	-	10	T$
10	9	-	-	$

Na slici 2.3 vidljivo je da *l-interval* [0..5] ima *l-vrijednost* 1 i ima 3 djece u stablu. Njegovi su *l-indeksi* 2 i 4. Prvi *l-indeks*, 2, spremljen je u $childtab[0].down$ i $childtab[6].up$, kako prikazuje tablica 2.3. Sljedeći *l-indeks* spremljen je u $childtab[2].nextlIndex$. Iz toga zaključujemo da su djeca intervala [0..5] upravo [0..1], [2..3] i [4..5], kako je i prikazano na slici.

3. Metode

3.1. Izgradnja sufiksnog polja SA-IS algoritmom

Algoritam Manbera i Myersa (1990.) koji su prvi uveli pojam sufiksnog polja omogućavao je njegovu konstrukciju u složenosti $O(n \log n)$. Danas postoji mnogo algoritama za izgradnju sufiksnog polja (engl. *Suffix Array Construction Algorithm; SACA*), od kojih neki rade u linearnom vremenu (Kärkkäinen i Sanders, 2003; Ko i Aluru, 2005; Nong, 2013). Trenutno najbrži i najpopularniji od njih je algoritam SA-IS koji su osmislili Nong, Zhang i Chan 2011. Poboljšanja u brzini izvođenja i memorijskom zauzeću u odnosu na druge postojeće algoritme linearne vremenske složenosti ostvarena su prvenstveno korištenjem LMS-podnizova i boljom iskoristivošću induciranog sortiranja (kako mu i ime kaže), elegantnog postupka koji su ranije u svoj algoritam uveli Ko i Aluru. Prednosti SA-IS algoritma nad ostalima pažljivom je implementacijom i eksperimentima pokazao Yuta Mori[1].

[1] https://sites.google.com/site/yuta256/sais

3.1.1. Koraci algoritma

Algoritam 1 SA-IS(S, SA)

Ulaz: S – niz znakova

Izlaz: SA – sufiksno polje ulaznog niza

Pomoćna polja: t, P_1, B

1. Odrediti sadržaj pomoćnog polja t jednim prolaskom kroz niz S.
2. Odrediti sadržaj pomoćnog polja P_1 jednim prolaskom kroz niz t.
3. Inducirano sortirati LMS-podnizove koristeći P_1 i B.
4. Imenovati svaki LMS-podniz prema pripadajućem indeksu (dobiti novo skraćeno polje $S1$).
5. Ako je svaki član niza S_1 jedinstven, izravno odrediti SA_1, inače rekurzivno pozvati SA-IS(S_1, SA_1).
6. Odrediti SA iz SA_1.

Spomenuti pojmovi i pojedini koraci objašnjeni su u daljnjem tekstu.

3.1.2. Pojmovi i defincije

Neka je S niz znakova duljine n sastavljen od znakova abecede Σ koji završava jedinstvenim leksikografski najmanjim znakom \$. Dodavanje posebnog znaka na kraj niza nužno je za sufiksna stabla jer osigurava da nijedan sufiks nije prefiks nekog drugog sufiksa. Kod sufiksnih polja nije uvijek nužan, no neki algoritmi za konstrukciju sufiksnog polja ga zahtijevaju. Neka je niz S indeksiran od 0 do $n-1$. Sufiks koji počinje na i-tom indeksu označavamo sa s_i, kao što je i ranije navedeno.

Neka je s_i S-tip sufiksa (engl. *S-type*) ako vrijedi $s_i < s_{i+1}$ ili ako je $s_i = \$$. Suprotno, neka je s_i L-tip sufiksa (engl. *L-type*) ako vrijedi $s_i > s_{i+1}$. Analogno tomu, možemo reći da je svaki znak niza $S[i]$ S-tip ili L-tip znaka ako je s_i S-tip ili L-tip sufiksa, respektivno. Iz toga slijedi da je $S[i]$ 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. Također, $S[i]$ 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. Na

ovaj način možemo odrediti tip svakog znaka u složenosti $O(1)$ jednim prolaskom kroz niz S, čime je ukupna složenost određivanja tipova znakova $O(n)$.

Primjer: Neka je zadan niz S = TGTGTGTGCACCG\$. Tablica 2.1 prikazuje oznake je li pojedini znak S-tip ili L-tip pohranjene u polju bitova t (bit 0 označava L-tip, a bit 1 S-tip znaka). Oznake određujemo jednim prolaskom kroz niz S zdesna nalijevo jer je važan tip sljedećeg, a ne prethodnog znaka.

Tablica 3.1: Tipovi znakova niza TGTGTGTGCACCG\$

indeks	0	1	2	3	4	5	6	7	8	9	10	11	12	13
niz S	T	G	T	G	T	G	T	G	C	A	C	C	G	\$
niz t	L	S	L	S	L	S	L	L	L	S	S	S	L	S

3.1.3. Određivanje LMS-podnizova

Definirajmo i pojmove LMS-znaka i LMS-podniza koje Nong, Zhang i Chan prvi spominju u svom radu.

Definicija 3. *Znak $S[i]$, $i \in [1, n-1]$, je LMS-znak (engl. leftmost S-type) ako je $S[i]$ S-tip i $S[i-1]$ L-tip znaka.*

Definicija 4. *Podniz $S[i, j]$ je LMS-podniz ako je $S[i, j]$ znak za kraj niza, tj. $i = j = n$ ili ako su $S[i]$ i $S[j]$ LMS-znakovi i $\neq j$ te između njih nema drugih LMS-znakova.*

Kada bismo koristili LMS-podnizove kao jedinice ulaznog niza te ako bismo ih mogli efikasno sortirati, tada bi svaki LMS-podniz mogao dobiti ime prema svojoj poziciji u sortiranom nizu. Niz S u tom bi se slučaju mogao predstaviti kao kraći niz $S1$, odnosno, niz imena LMS-podnizova. Time bi početni problem bio reduciran. Da bi to bilo moguće, moramo definirati poredak LMS-podnizova.

Definicija 5. *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.*

Prilikom uspoređivanja, ako su parovi znakova leksikografski jednaki, veći prioritet dajemo S-tipu jer je $s_i > s_j$ ako je s_i S-tipa, a s_j L-tipa. Iz definicije

slijedi da su dva LMS-podniza jednaka ako su jednake duljine i ako su im svi parovi znakova leksikografski jednaki i jednakog tipa.

Za sortiranje LMS-podnizova koristit ćemo pomoćni niz P_1 koji će sadržavati indekse svih LMS-podnizova, pri čemu je sačuvan njihov prvotni poredak unutar niza S. Niz P_1 dobivamo prolaskom kroz S ili t zdesna nalijevo, u složenosti $O(n)$.

Primjer: Neka je zadan niz S = TGTGTGTGCACCG\$. Odredimo njegove LMS-podnizove, tj. niz P_1. Koristit ćemo ranije dobiven niz t (Tablica 3.1) koji sadrži oznake je li pojedini znak niza S ili L-tip. Prateći definiciju LMS-znaka, u nizu t promatramo znakove S-tipa neposredno prije kojih se nalazi znak L-tipa. Na taj način dobivamo $P_1 = \{1, 3, 5, 9, 13\}$. Između svaka dva susjedna LMS-znaka ne nalazi se nijedan drugi LMS-znak, što znači da označavaju početne i krajnje pozicije LMS-podniza. Vidljivo je da u nizu S nalazimo 4 LMS-podniza (prvi LMS-podniz počinje na poziciji 1 i završava na poziciji 3, sljedeći počinje na poziciji 3 i završava na poziciji 5, itd.). Posljednji LMS-znak i LMS-podniz je, prema definiciji, znak za kraj niza.

Tablica 3.2: LMS-podnizovi niza TGTGTGTGCACCG\$

i	j	$S[i, j]$
1	3	GTG\$
3	5	GTG\$
5	9	GTGCA\$
9	13	ACCG\$
13	13	\$

Kako je ranije rečeno, dva su LMS-podniza jednaka ako su jednake duljine i svi su im odgovarajući parovi znakova leksikografski jednaki i istoga tipa. Primijetimo da su u ovom primjeru dva LMS-podniza jednaka, $S[1, 3]$ i $S[3, 5]$ (GTG\$).

3.1.4. Inducirano sortiranje LMS-podnizova

U algoritmu koji su osmislili Ko i Aluru inducirano sortiranje korišteno je u koraku koji određuje SA iz SA_1, a SA-IS uvodi vrlo jednostavnu promjenu koja omogućuje inducirano sortiranje LMS-podnizova. Upravo je to ono što čini SA-IS nadmoćnijim u odnosu na druge algoritme.

Algoritam, osim već spomenutih pomoćnih polja t (oznake S ili L-tipa znakova) i P_1 (indeksi LMS-podnizova) koristi i polje B. Sufiksno polje SA možemo

promatrati kao da je podijeljeno na potpolja gdje svako od njih sadrži indekse sufiksa koji počinju istim početnim znakom iz abecede Σ. B je polje pokazivača na početak ili kraj takvih potpolja ili pretinaca (engl. *bucket*).

U ovom koraku algoritma potrebno je definirati pojmove LMS-sufiksa i LMS-prefiksa:

Definicija 6. *Sufiks s_i je LMS-sufiks ako je $S[i]$ LMS-znak.*

Definicija 7. *LMS-prefiks je LMS-znak ili prefiks $S[i, k]$ sufiksa s_i takav da je $S[k]$ prvi LMS-znak nakon $S[i]$. Označavat ćemo ga s $pre(S, i)$. LMS-prefiks je S-tipa ili L-tipa ako je sufiks s_i S-tipa ili L-tipa, respektivno.*

Primijetimo da vrijedi da svaki LMS-prefiks L-tipa ima najmanje 2 znaka.

Inducirano sortiranje LMS-podnizova zapravo se svodi na sortiranje LMS-prefiksa. Koraci sortiranja su sljedeći:

1. Inicijalizirati članove polja SA na -1.
   Svaki član polja $B[i]$ postaviti na kraj i-tog pretinca (potpolja) SA, $i \in \Sigma$.

2. Svaki indeks LMS-sufiksa dodati u odgovarajuće potpolje od SA, tako da im originalan redoslijed ostane nepromijenjen. Taj rezultat postići prolaskom kroz polje S slijeva nadesno, dodati indeks LMS-sufiksa na kraj pripadajućeg potpolja te pomaknuti pokazivač potpolja za jedno mjesto ulijevo.

3. Svaki član polja $B[i]$ postaviti na početak i-tog pretinca (potpolja) SA, $i \in \Sigma$.

4. Proći kroz polje SA slijeva nadesno. Za svaki $SA[i] > 0$ provjeriti je li $S[SA[i] - 1]$ L-tip znaka. Ako jest, dodati $SA[i] - 1$ na početak potpolja znaka $S[SA[i] - 1]$ te pomaknuti pokazivač na kraj tog potpolja za jedno mjesto udesno.

5. Svaki član polja $B[i]$ postaviti na kraj i-tog pretinca (potpolja) SA, $i \in \Sigma$.

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

Dokažimo da će ovaj postupak doista sortirati LMS-podnizove. Najprije se svi LMS-prefiksi S-tipa duljine 1 postavljaju u svoja potpolja te su time svi LMS-podnizovi koji su trenutno u SA sortirani.

3. i 4. korak algoritma će sortirati sve LMS-prefikse L-tipa. Dokažimo tu činjenicu indukcijom. Kada prvi LMS-prefiks L-tipa dodamo u odgovarajuće potpolje, on mora biti točno sortiran u odnosu na već postojeće LMS-prefikse S-tipa. Pretpostavimo da je ovaj korak točno sortirao k LMS-prefiksa L-tipa ($k \geq 1$). Nadalje, pretpostavimo da nakon što dodamo sljedeći LMS-prefiks L-tipa $pre(S, i)$ na trenutni početak njegova potpolja, ispred (lijevo od) njega možemo pronaći barem jedan veći LMS-prefiks L-tipa $pre(S, j)$. U tom slučaju, moralo bi vrijediti $S[i] = S[j]$, $pre(S, j+1) > pre(S, i+1)$ te bi $pre(S, j+1)$ morao biti ispred $pre(S, i+1)$ u SA. To znači da prije dodavanja $pre(S, i)$ LMS-prefiksi nisu bili točno sortirani, što je u suprotnosti s ranije navedenom pretpostavkom. Time je dokazano da su nakon provođenja ovih koraka svi LMS-prefiksi L-tipa i LMS-prefiksi S-tipa duljine 1 točno sortirani.

Dokažimo da će nakon provođenja 5. i 6. koraka doista biti sortirani svi LMS-prefiksi duljine veće od 1. Kada u odgovarajuće potpolje dodajemo prvi LMS-prefiks S-tipa, on mora biti sortiran točno u odnosu na postojeće LMS-prefikse L-tipa. Prisjetimo se da smo ranije LMS-prefikse L-tipa dodavali na kraj potpolja. Dakle, dodavanjem novih LMS-prefiksa S-tipa neki od indeksa postojećih LMS-prefiksa biti će prepisani. Pretpostavimo da je ovaj korak točno sortirao k LMS-prefiksa S-tipa ($k \geq 1$). Nadalje, pretpostavimo da nakon što dodamo sljedeći LMS-prefiks S-tipa $pre(S, i)$ na trenutni kraj njegova potpolja, iza (desno od) njega možemo pronaći barem jedan manji LMS-prefiks S-tipa $pre(S, j)$. U tom slučaju, moralo bi vrijediti $S[i] = S[j]$, $pre(S, j+1) < pre(S, i+1)$ te bi $pre(S, j+1)$ morao biti iza $pre(S, i+1)$ u SA. To znači da prije dodavanja $pre(S, i)$ LMS-prefiksi nisu bili točno sortirani, što je u suprotnosti s ranije navedenom pretpostavkom. Time je dokazano da su nakon provođenja svih koraka algoritma svi LMS-prefiksi duljine veće od 1 i završni znak niza (jedini u svom potpolju) točno sortirani u SA.

Kako vrijedi da je bilo koji LMS-podniz zapravo LMS-prefiks duljine veće od 1 ili završni znak niza, slijedi da su svi LMS-podnizovi točno sortirani.

Primjer: Promotrimo ponovno niz S = TGTGTGTGCACCG\$. Ranije smo odredili tipove znakova i indekse LMS-podnizova (1, 3, 5, 9, 13) što će nam biti potrebno i u ovom koraku. Najprije treba izgraditi polje B, odnosno postaviti

pokazivače na kraj potpolja pojedinih znakova. Jasno je da postoji 1 sufiks koji započinje znakom '$', 1 koji započinje s 'A', 3 koja započinju s 'C', 5 koji započinju s 'G' te 4 koja započinju s 'T'. Polje B i podjela SA na potpolja (poredana abecednim redom) izgledaju ovako:

$$SA = \{\_\}, \{\_\}, \{\_, \_, \_\}, \{\_, \_, \_, \_, \_\}, \{\_, \_, \_, \_\}$$

$$B['\$'] = 0, B['A'] = 1, B['C'] = 4, B['G'] = 9, B['T'] = 13$$

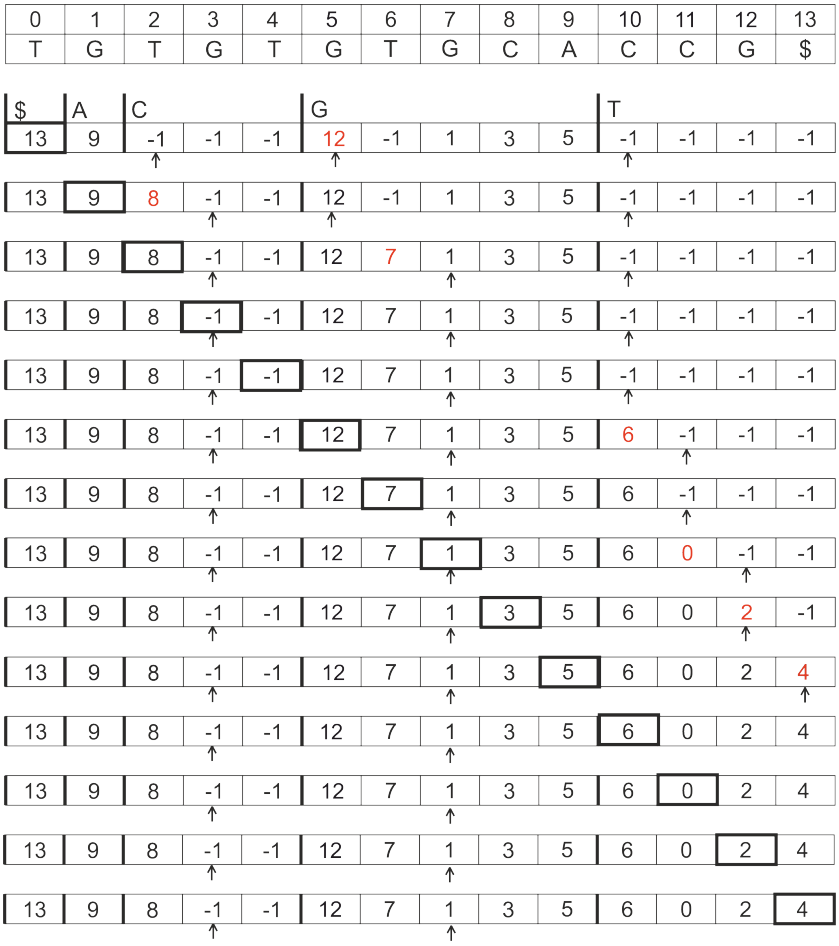
Polje SA inicijalizirano je na -1. Sada provodimo 2. korak algoritma za inducirano sortiranje LMS-podnizova te dobivamo sljedeći rezultat (razdvojeno po potpoljima radi bolje preglednosti, od $ do T):

$$SA = \{13\}, \{9\}, \{-1, -1, -1\}, \{-1, -1, 1, 3, 5\}, \{-1, -1, -1, -1\}$$

$$B['\$'] = -1, B['A'] = 0, B['C'] = 4, B['G'] = 6, B['T'] = 13$$

Sljedeći korak algoritma zahtijeva pomicanje pokazivača na početak potpolja:

$$B['\$'] = 0, B['A'] = 1, B['C'] = 2, B['G'] = 5, B['T'] = 10$$



Slika 3.1: 4. korak induciranog sortiranja LMS-podnizova (uokviren je član polja koji se obrađuje, strelicama su označeni pokazivači na početke potpolja, crvenom bojom su označeni novi članovi polja).

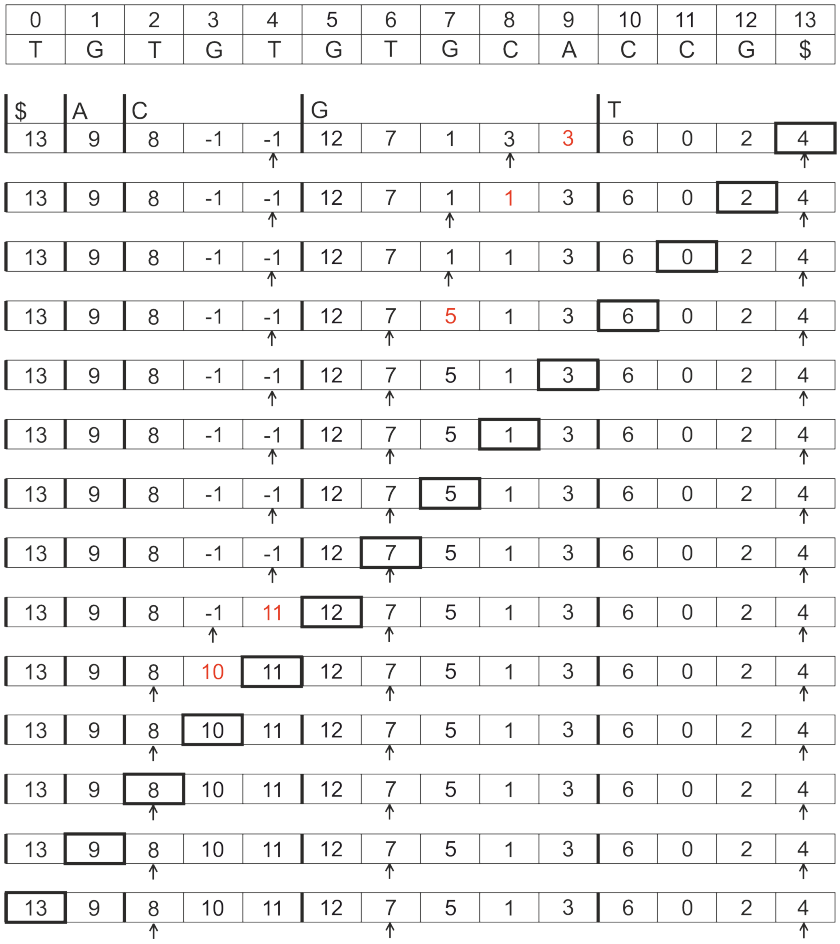
Nadalje, prolazimo poljem SA slijeva nadesno i provodimo 4. korak induciranog sortiranja (slika 3.1):

$$SA = \{13\}, \{9\}, \{8, -1, -1\}, \{12, 7, 1, 3, 5\}, \{6, 0, 2, 4\}$$

$$B['\$'] = 0, B['A'] = 1, B['C'] = 3, B['G'] = 7, B['T'] = 13$$

Pokazivače ponovno pomičemo na krajeve potpolja:

$$B['\$'] = 0, B['A'] = 1, B['C'] = 4, B['G'] = 9, B['T'] = 13$$



Slika 3.2: 6. korak induciranog sortiranja LMS-podnizova (uokviren je član polja koji se obrađuje, strelicama su označeni pokazivači na krajeve potpolja, crvenom bojom su označeni novi članovi polja).

Provodimo i posljednji korak induciranog sortiranja prolaskom kroz SA zdesna nalijevo i dobivamo krajnji rezultat (slika 3.2):

$$SA = \{13\}, \{9\}, \{8, 10, 11\}, \{7, 5, 1, 3, 12\}, \{6, 0, 2, 4\}$$

Ipak, ovo još nije krajnje rješenje za SA jer indeksi sufiksa unutar pojedinih potpolja još nisu pravilno sortirani.

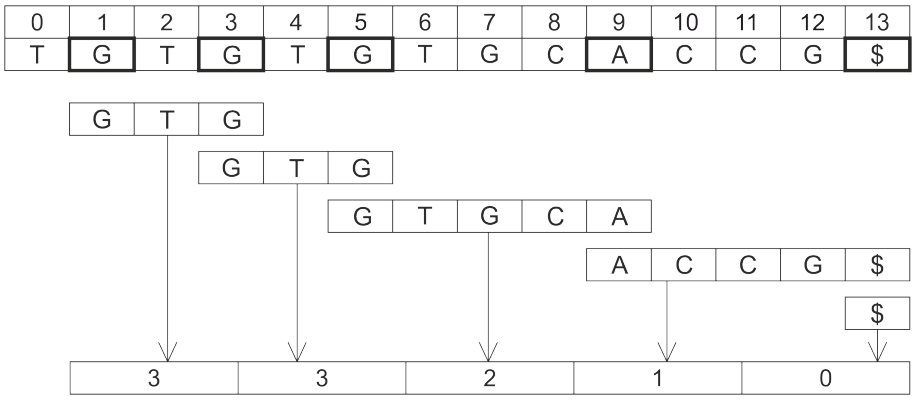
3.1.5. Imenovanje LMS-podnizova

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 ime, tj. cijeli broj, na temelju njegovog indeksa u polju SA. Ako su LMS-podnizovi jednaki, odnosno, ako su jednake duljine i za

svaki odgovarajući par znakova vrijedi da su jednaki i istog tipa, tada ti podnizovi imaju ista imena. Upravo ta imena LMS-podnizova članovi su novog niza S_1.

U prethodnom koraku dobiveno polje SA služi nam za imenovanje LMS-podnizova. Prolazimo kroz SA slijeva nadesno i započinjemo dodjeljivanje imena od 0. Za svaki sufiks iz polja SA provjeravamo je li LMS-podniz tako da provjerimo je li njegov početni znak LMS-znak. LMS-podniz dobiva jednako ime kao i njegov prethodnik ako su oni međusobno jednaki, a inače dobiva novo ime (prethodno se uvećava za 1).

Primjer: Za niz $S = \text{TGTGTGTGCACCG\$}$ u prethodnom smo koraku odredili $SA = \{\{13\}, \{9\}, \{8, 10, 11\}, \{12, 7, 5, 3, 1\}, \{6, 4, 2, 0\}\}$. Sada na temelju tog rezultata imenujemo LMS-podnizove niza S, odnosno dobivamo novi niz S_1. Prolaskom kroz SA slijeva nadesno prvi LMS-podniz na koji nailazimo je onaj s indeksom 13, znak za kraj niza. Njemu dodjeljujemo ime 0. Sljedeći LMS-podniz ima indeks 9 (ACCG$) te on dobiva ime 1 jer se razlikuje od prethodnika. Nadalje, LMS-podniz GTGCA$ dobiva ime 2, a njegov sljedbenik s početnim indeksom 3 (GTG$), imenujemo brojem 3. Na kraju, uočavamo da je posljednji LMS-podniz s početnim indeksom 5 jednak prethodnom (GTG$) pa ga imenujemo također brojem 3.



Slika 3.3: Imenovanje LMS-podnizova niza TGTGTGTGCACCG$. Uokvireni su LMS-znakovi.

Naposljetku sva imena LMS-podnizova spajamo u novi niz S_1, prema njihovu originalnom poretku u početnom nizu S:

$$S_1 = \text{"33210"}$$

3.1.6. Određivanje SA_1 iz S_1

U prethodnom koraku određen je niz S_1, niz imena LMS-podnizova. Ako svaki LMS-podniz ima jedinstveno ime, trivijalno se određuje SA_1, tj. sufiksno polje, za taj niz jer je poredak sufiksa određen isključivo njihovim početnim znakovima, a svi su različiti. U tom slučaju vrijedi: $SA_1[S_1[i]] = i, i \in S_1$. U suprotnom, ako postoji više LMS-podnizova s istim imenom, SA_1 određuje se rekurzivnim pozivom funkcije SA-IS(S_1, SA_1).

Uzevši u obzir činjenicu da prvi član niza S sigurno nije LMS-znak te da sigurno ne postoje dva susjedna LMS-znaka, slijedi da je duljina niza S_1 najviše pola duljine niza S, čime se složenost početnog problema smanjuje na pola.

Dokažimo da je sortiranje LMS-sufiksa niza S doista ekvivalentno sortiranju niza S_1 rekurzivnim pozivom funkcije SA-IS(S_1, SA_1). Dakle, potrebno je dokazati da je međusobni poredak svakog para sufiksa $suf(S_1, i)$ i $suf(S_1, j)$ jednak međusobnom poretku sufiksa $suf(S, P_1[i])$ i $suf(S, P_1[j])$.

Promotrimo prvo slučaj u kojem je $S_1[i] \neq S_1[j]$. Tada u odgovarajućim podnizovima mora postojati ili par leksikografski različitih znakova, kada tvrdnja očito vrijedi, ili postoji par znakova različitih po tipu. Definirano da je znak S-tipa većeg prioriteta od znaka L-tipa pa tvrdnja tada također vrijedi.

U drugom slučaju jest $S_1[i] = S_1[j]$. Tada je poredak sufiksa $suf(S_1, i)$ i $suf(S_1, j)$ određen poretkom $suf(S_1, i+1)$ i $suf(S_1, j+1)$. Ako je $S_1[i+1] = S_1[j+1]...S_1[i+k-1] = S_1[j+k-1]$ tada na isti način dolazimo do usporedbe $suf(S_1, i+k)$ i $suf(S_1, j+k)$ za prvi par $S_1[i+k] \neq S_1[j+k]$. Kako za svaki par $S_1[i] = S_1[j]$ mora vrijediti $P_1[i+1] - P_1[i] = P_1[j+1] - P_1[j]$ (LMS-podnizovi su jednake duljine), vrijedi i da su $S[P_1[i]..P_1[i+k]]$ i $S[P_1[j]..P_1[j+k]]$ jednake duljine. Dakle, vrijedi da je sortiranje $S_1[i..i+k]$ i $S_1[j..j+k]$ ekvivalentno sortiranju $S[P_1[i]..P_1[i+k]]$ i $S[P_1[j]..P_1[j+k]]$, čime je početna tvrdnja dokazana.

Primjer: Pretpostavimo da smo za neki niz S dobili S_1 = "41203". U S_1 svi su članovi jedinstveni pa je SA_1 moguće izravno odrediti. Slijedi da je SA_1 = {3, 1, 2, 4, 0}.

Primjer: Za niz S = TGTGTGTGCACCG$ imenovanjem LMS-podnizova dobiven je niz S_1 = ”33210”. Kako nisu sva imena LMS-podnizova jedinstvena, nije moguće trivijalno odrediti SA_1. Rekurzivnim pozivom funkcije SA-IS(S_1, SA_1) dobivamo SA_1 = {4, 3, 2, 1, 0}.

3.1.7. Određivanje SA iz SA_1

U ovom koraku ponovno se koristi postupak induciranog sortiranja, kao i u sortiranu LMS-podnizova. Postupak je sljedeći:

1. Inicijalizirati članove polja SA na -1.
   Svaki član polja $B[i]$ postaviti na kraj i-tog pretinca (potpolja) SA, $i \in \Sigma$.

2. Prolaskom kroz polje SA_1 zdesna nalijevo, dodati $P_1[SA_1[i]]$ na kraj pripadajućeg potpolja te pomaknuti pokazivač na kraj potpolja za jedno mjesto ulijevo.

3. Svaki član polja $B[i]$ postaviti na početak i-tog pretinca (potpolja) SA, $i \in \Sigma$.

4. Proći kroz polje SA slijeva nadesno. Za svaki $SA[i] > 0$ provjeriti je li $S[SA[i] - 1]$ L-tip znaka. Ako jest, dodati $SA[i] - 1$ na početak potpolja znaka $S[SA[i] - 1]$ te pomaknuti pokazivač na kraj tog potpolja za jedno mjesto udesno.

5. Svaki član polja $B[i]$ postaviti na kraj i-tog pretinca (potpolja) SA, $i \in \Sigma$.

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

Svaki od navedenih koraka može se provesti u linearnom vremenu. Možemo primijetiti da je od 3. do 6. koraka postupak identičan induciranom sortiranju LMS-podnizova. Svako potpolje u SA dodatno je podijeljeno na dio za sufikse S-tipa i L-tipa. Najprije se sortirani LMS-sufiksi dodaju u pripadajuća potpolja S-tipa u SA, od kraja prema početku. 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 konačno.

Primjer: Imenovanjem LMS-podnizova niza S = TGTGTGTGCACCG$ određen je novi niz S_1 = ”33210”, a rekurzivnim pozivom funkcije SA-IS(S_1, SA_1) dobivamo $SA_1 = \{4, 3, 2, 1, 0\}$. Kako bi se odredilo sufiksno polje SA niza S, potrebno je iskoristiti upravo SA_1. Svi članovi polja B na početku su postavljeni na krajeve pretinaca:

$$B['\$'] = 0, B['A'] = 1, B['C'] = 4, B['G'] = 9, B['T'] = 13$$

Polje SA inicijalizirano je na -1. Postavljanjem indeksa LMS-sufiksa na kraj pripadajućih potpolja prolaskom kroz niz SA_1 zdesna nalijevo dobivamo:

$$SA = \{13\}, \{9\}, \{-1, -1, -1\}, \{12, -1, 5, 3, 1\}, \{-1, -1, -1, -1\}$$

$$B['\$'] = -1, B['A'] = 0, B['C'] = 4, B['G'] = 6, B['T'] = 13$$

Pokazivače pomičemo na početak potpolja:

$$B['\$'] = 0, B['A'] = 1, B['C'] = 2, B['G'] = 5, B['T'] = 10$$

Nadalje, prolazimo poljem SA slijeva nadesno i sortiramo sufikse L-tipa:

$$SA = \{13\}, \{9\}, \{8, -1, -1\}, \{12, 7, 5, 3, 1\}, \{6, 4, 2, 0\}$$

$$B['\$'] = 0, B['A'] = 1, B['C'] = 2, B['G'] = 6, B['T'] = 13$$

Pokazivače ponovno pomičemo na krajeve potpolja:

$$B['\$'] = 0, B['A'] = 1, B['C'] = 4, B['G'] = 9, B['T'] = 13$$

Provodimo i posljednji korak algoritma prolaskom kroz SA zdesna nalijevo i dobivamo konačno rješenje, sufiksno polje niza S:

$$SA = \{13\}, \{9\}, \{8, 10, 11\}, \{12, 7, 5, 3, 1\}, \{6, 4, 2, 0\}$$

3.1.8. Složenost algoritma SA-IS

Uzmemo li u obzir da se ulazni niz sastoji od elemenata konstantne ili cjelobrojne abecede, vremenska složenost izgradnje sufiksnog polja jest linearna i ovisi o broju ulaznih znakova. Problem se svakim rekurzivnim pozivom reducira barem na pola i svaka razina ima linearnu složenost.

Prostornu složenost određuje količina memorije potrebna za spremanje sufiksnog polja reduciranog problema u svakoj iteraciji. U prvoj iteraciji sufiksno polje zauzima najviše $n\lceil \log n \rceil$ bita pa je tako i prostorna složenost upravo $O(n\lceil \log n \rceil)$.

3.2. Određivanje polja najduljih zajedničkih prefiksa

Kao što je već spomenuto, polje najduljih zajedničkih prefiksa može se odrediti paralelno s određivanjem sufiksnog polja ili naknadno. Trenutno najbrži algoritam koji stvara polje LCP paralelno sa sufiksnim poljem (Fischer, 2011) temelji

Tablica 3.3: Sufiksno polje niza S = TGTGTGTGCACCG$

i	$SA[i]$	s_i
0	13	$
1	9	ACCG$
2	8	CACCG$
3	10	CACCG$
4	11	CCG$
5	12	CG$
6	7	GCACCG$
7	5	GTGCACCG$
8	3	GTGTGCACCG$
9	1	GTGTGTGCACCG$
10	6	TGCACCG$
11	4	TGTGCACCG$
12	2	TGTGTGCACCG$
13	0	TGTGTGTGCACCG$

se upravo na ranije obrađenom algoritmu za izgradnju sufiksnog polja SA-IS. Sljedeći algoritam (Kasai et al., 2001) gradi polje najduljih zajedničkih prefiksa koristeći već određeno sufiksno polje te njemu pripadajuće inverzno sufiksno polje.

Navedeni algoritam izgradnje LCP polja temelji se na činjenici da je

$$lcp(s_{SA[i]}, s_{SA[i+1]}) \geq lcp(s_{SA[i-1]}, s_{SA[i]}) - 1$$

Dakle, da bismo izračunali $lcp(s_{SA[i-1]}, s_{SA[i]})$, ne moramo uspoređivati njihove znakove od početka, već počevši s h-tim znakom, osim ako je $h \leq 1$. Detaljniji dokaz točnosti algoritma izložen je u originalnom radu (Kasai et al., 2001).

Vremenska složenost algoritma je linearna. Naime, složenost je određena brojem ponavljanja 6. linije algoritma, odnosno brojem uvećavanja varijable h. Varijabla h nikada neće postati veća od n, tj. broja znakova niza. U 10. liniji varijabla h umanjit će se najviše n puta pa je ukupan broj uvećavanja varijable h najviše $2n$, što znači da je ukupna složenost algoritma $O(n)$.

Algoritam 2 Određivanje LCP polja

Ulaz: SA – sufiksno polje, ISA – inverzno sufiksno polje

Izlaz: LCP – polje najduljih zajedničkih prefiksa

```
1:  h := 0
2:  for i := 0; i < n; i := i + 1 do
3:      k := ISA[i]
4:      j := SA[k − 1]
5:      while (i + h < n) and (j + h < n) and (S[i + l] = S[j + l]) do
6:          h := h + 1
7:      end while
8:      LCP[k] := h
9:      if h > 0 then
10:         h := h − 1
11:     end if
12: end for
```

3.3. Izgradnja tablice djece stabla LCP-intervala

Tablica djece stabla LCP-intervala može se izgraditi u linearnom vremenu. Radi jasnoće, razdvojeni su algoritmi za računanje *up* i *down* vrijednosti te drugi za računanje $nextlIndex$ vrijednosti tablice. Jedini podatak koji je potreban je LCP polje, a koristi se i stog kao pomoćna struktura.

Algoritam 3 Određivanje *up* i *down* vrijednosti tablice djece

Ulaz: LCP – polje najduljih zajedničkih prefiksa

Izlaz: *childtab* – tablica djece (*up* i *down* vrijednosti)

```
1:  lastIndex := −1
2:  push(0)
3:  for i := 1; i ≤ n; i := i + 1 do
4:     while LCP[i] < LCP[top] do
5:        lastIndex := pop()
6:        if LCP[i] ≤ LCP[top] and LCP[top] ≠ LCP[lastIndex] then
7:           childtab[top].down := lastIndex
8:        end if
9:        if lastIndex ≠ −1 then
10:          childtab[i].up := lastIndex
11:          lastIndex := −1
12:       end if
13:    end while
14:    push(i)
15: end for
```

Jasno je da se naredbe push() i pop() te *top* odnose na korištenje stoga kao pomoćne strukture. Algoritam 3 se oslanja na održavanje određenih invarijantnosti tijekom prolaska kroz petlju. Ako su $i_1, i_2, ..., i_p$ indeksi na stogu, pri čemu je i_p posljednji element stavljen na stog, tada je $i_1 < i_2 < ... < i_p$ i $LCP[i_1] \leq LCP[i_2] \leq ... \leq LCP[i_p]$. Nadalje, ako je $LCP[i_j] < LCP[i_{j+1}]$, tada za sve k, $i_j < k < i_{j+1}$ vrijedi da je $LCP[k] > LCP[i_{j+1}]$.

Algoritam 4 Određivanje *nextlIndex* vrijednosti tablice djece

Ulaz: LCP – polje najduljih zajedničkih prefiksa

Izlaz: *childtab* – tablica djece (*nextlIndex* vrijednost)

```
1: push(0)
2: for i := 1; i ≤ n; i := i + 1 do
3:    while LCP[i] < LCP[top] do
4:       pop()
5:    end while
6:    if LCP[i] = LCP[top] then
7:       lastIndex := pop()
8:       childtab[i].nextlIndex := i
9:    end if
10:   push(i)
11: end for
```

Određivanja *nextlIndex* vrijednosti algoritmom 4 jednostavnije je od određivanja *up* i *down* vrijednosti koje ne moraju od ranije biti poznate. Također se koristi stog. Postupak se svodi na provjeru nalazi li se na stogu indeks j za koji vrijedi $LCP[i] = LCP[j]$. Ako se pronađe, tada je $childtab[j].nextlIndex = i$.

3.4. Određivanje odnosa u stablu LCP-intervala

Određivanje odnosa u stablu LCP-intervala, odnosno djece pojedinih čvorova stabla, ključno je za mogućnost obilaska stabla. Da bi se u konstantnom vremenu pronašla djeca LCP-intervala $[i..j]$, potrebno je pronaći njegov prvi *l-indeks* pomoću *up* i *down* vrijednosti zapisanih u tablici djece. Naime, za svaki *l-interval* $[i..j]$ vrijedi sljedeće:

1. $i < childtab[j+1].up \leq j$ ili $i < childtab[i].down \leq j$
2. $childtab[j+1].up$ sadrži prvi (najmanji) *l-indeks* intervala $[i..j]$ ako je $i < childtab[j+1].up \leq j$
3. $childtab[i].down$ sadrži prvi (najmanji) *l-indeks* intervala $[i..j]$ ako je $i < childtab[i].down \leq j$

Koristeći ove činjenice, lako je implementirati funkciju koja određuje LCP-vrijednost nekog LCP-intervala, kako prikazuje algoritam 5.

Algoritam 5 Određivanje LCP-vrijednosti LCP-intervala

Ulaz: i, j – granice LCP-intervala

Izlaz: LCP-vrijednost

1: **if** $i < childtab[j+1].up$ and $j \geq childtab[j+1].up$ **then**
2: return $LCP[childtab[j+1].up]$
3: **else**
4: return $LCP[childtab[i].down]$
5: **end if**

Algoritam 6 Određivanje djece LCP-intervala

Ulaz: i, j – granice LCP-intervala

Izlaz: lista granica djece ulaznog LCP-intervala

1: $intervalList := []$
2: **if** $i < childtab[j+1].up$ **then**
3: $i_1 := childtab[j+1].up$
4: **else**
5: $i_1 := childtab[i].down$
6: **end if**
7: add$(intervalList, (i, i_1 - 1))$
8: **while** $childtab[i].nextlIndex \neq \perp$ **do**
9: $i_2 := childtab[i_1].nextlIndex$
10: add$(intervalList, (i_1, i_2 - 1))$
11: $i_1 := i_2$
12: **end while**
13: add$(intervalList, (i_1, j))$
14: return $intervalList$

Algoritam 6 prikazuje kako odrediti djecu nekog LCP-intervala unutar stabla LCP-intervala. Funkcija ima konstantnu vremensku složenost, točnije $O(|\Sigma|)$, pod pretpostavkom da je abeceda konstantna. Pomoću nje, moguće je simulirati bilo koji obilazak sufiksnog stabla predstavljenog poboljšanim sufiksnim poljem koji posjećuje djecu prije roditeljskog čvora. Također, jednostavnim modifikacijama moguće je postići da funkcija za LCP-interval $[i..j]$ traži dijete $[l..r]$ čija je LCP-vrijednost l čiji sufiksi imaju $a \in \Sigma$ (zadan kao parametar) na poziciji l, ako takav postoji. Svi sufiksi intervala $[l..r]$ u tom slučaju imaju zajednički prefiks

duljine l jer je $[l..r]$ podinterval od $[i..j]$. Takva funkcija također ima konstantnu vremensku složenost.

3.5. Pretraživanje poboljšanog sufiksnog polja

Jedna od upotreba poboljšanog sufiksnog polja jest odgovaranje na upite tipa "*Je li P podniz od S?*". Obično sufiksno polje na takve i slične upite može odgovoriti u složenosti $O(m \log n)$, gdje je m duljina niza P. Logaritamski dio složenosti odnosi se na binarno pretraživanje pomoću kojeg se nalazi pozicija niza P unutar S koristeći sufiksno polje. Poboljšano sufiksno polje u najgorem slučaju na upite tog tipa odgovara u optimalnoj vremenskoj složenosti $O(m)$. Također, nadograđeni upit oblika "*Gdje se nalazi svih z ponavljanja niza P unutar S?*" obrađuje se u vremenu $O(m + z)$, potpuno neovisno o duljini niza S, tj. n.

Algoritam 7 Pretraživanje niza znakova

Ulaz: S – niz znakova, P - niz znakova koji se traži

Izlaz: indeksi na kojima se u S pojavljuje P

1: $c := 0$
2: $queryFound := True$
3: $(i, j) :=$ getInterval$(0, n, P[c]$
4: **while** $(i, j) \neq \perp$ and $c < m$ and $queryFound = True$ **do**
5: **if** $i \neq j$ **then**
6: $l :=$ getlcp(i, j)
7: $min := min\{l, m\}$
8: $queryFound := S[SA[i] + c..SA[i] + min - 1] = P[c..min - 1]$
9: $c := min$
10: $(i, j) :=$ getInterval$(i, j, P[c])$
11: **else**
12: $queryFound := S[SA[i] + c..SA[i] + m - 1] = P[c..m - 1]$
13: **end if**
14: **end while**
15: **if** $queryFound$ **then**
16: report(i, j)
17: **end if**

Algoritam 7 započinje detekcijom intervala $[i..j]$ čiji sufiksi počinju prvim

znakom niza P. Ako je $[i..j]$ interval od jednog člana, tada se P pojavljuje unutar S ako i samo ako je $S[SA[i]..SA[i]+m-1] = P$. Inače, ako je $[i..j]$ LCP-interval, tada određujemo njegovu LCP-vrijednost l na način opisan ranije, algoritmom 5. Neka je $\omega = S[SA[i]..SA[i]+l-1]$ najdulji zajednički prefiks sufiksa $s_{SA[i]}, s_{SA[i+1]}, ..., s_{SA[j]}$. Ako je $l \geq m$, tada se P pojavljuje unutar S ako i samo ako je $\omega[0..m-1] = P$. Inače, ako je $l < m$, tada provjeravamo je li $\omega = P[0..l-1]$. Ako nije, tada se P ne pojavljuje unutar S. Ako jest, tada tražimo interval $[i'..j']$ čiji sufiksi počinju prefiksom $P[0..l]$ (svi sufiksi unutar intervala $[i'..j']$ imaju $P[0..l-1]$ kao zajednički prefiks jer je $[i'..j']$ podinterval od $[i..j]$). Ako je $[i'..j']$ sastavljen od jednog člana, tada se P nalazi unutar S ako i samo ako je $S[SA[i']+l..SA[i']+m-1] = P[l..m-1]$. Inače, ako je $[i'..j']$ LCP-interval s LCP-vrijednošću l', proglasimo $\omega' = S[SA[i']..SA[i']+l'-1]$ najduljim zajedničkim prefiksom sufiksa $s_{SA[i']}, s_{SA[i'+1]}, ..., s_{SA[j']}$. Ako je $l' \geq m$, tada se P pojavljuje unutar S ako i samo ako je $\omega'[l..m-1] = P[l..m-1]$ (ili, ekvivalentno, $\omega'[0..m-1] = P[0..m-1]$). U suprotnom, ako je $l' < m$, tada provjeravamo je li $\omega[l..l'-1] = P[l..l'-1]$. Ako nije, P se ne nalazi nigdje unutar S. Ako jest, postupak ponavljamo sa sljedećim intervalom.

Za pronalazak svih z ponavljanja niza P unutar S koristimo ranije opisan algoritam vremenske složenosti $O(m)$. Dodatno, početni indeks svakog pojavljivanja određuje se u $O(z)$ pa je ukupna složenost postupka $O(m+z)$.

3.5.1. Detekcija sufiks-prefiks preklapanja nizova

Uz manje modifikacije, moguće je prikazani algoritam za pretragu poboljšanog sufiksnog polja prilagoditi tako da obavlja i neke slične funkcije, npr. traženje sufiks-prefiks preklapanja nizova (engl. *suffix-prefix overlap*) što je vrlo važan problem u bioinformatici pri analizi genoma.

Sufiks-prefiks preklapanje definiramo kao niz znakova koji je sufiks jednog niza, a ujedno i prefiks drugog niza.

Primjer: Pronađimo sufiks-prefiks preklapanja nizova S_1 = ACCAGTCAGTC i S_2 = GTCAGTCTG. Sufiks niza S_1 GTC prefiks je niza S_2. Također je i sufiks niza S_1 GTCAGTC prefiks niza S_2. Dakle, postoje 2 sufiks-prefiks preklapanja nizova i njihove su duljine 3, odnosno 6 znakova pa nizove možemo "preklopiti" na 2 načina: AACCAGTCAGTCAGTCTG ili AACCAGTCAGTCTG.

Algoritam 7 modificiramo na način da svaki put nakon usporedbe odgovaraju-

ćih podnizova nizova S i P (linije 8 i 12) dodajemo provjeru jesmo li našli sufiks-prefiks preklapanje. Ako je usporedbom utvrđeno da su podnizovi jednaki, provjerimo je li duljina posljednjeg sufiksa iz trenutno promatranog LCP-intervala, tj. s_j, jednaka upravo duljini podnizova koje smo u ovom koraku uspoređivali (manjoj od vrijednosti između LCP-vrijednosti intervala i duljine niza P). Ako jest, znači da je s_j sufiks koji je jednak prefiksu niza P pa smo pronašli sufiks-prefiks preklapanje duljine tog sufiksa ($n - SA[j]$). Svako sljedeće preklapanje koje pronađemo bit će dulje od svih ranije pronađenih. Kako ćemo modifikacijom algoritma za pretraživanje pronaći preklapanja takva da je sufiks niza nad kojim je izgrađeno polje prefiks drugog zadanog niza, potrebno je provjeriti i drugu stranu, odnosno da je prefiks niza nad kojim je izgrađeno polje sufiks drugog niza. U tu svrhu izgradit ćemo poboljšano sufiksno polje nad drugim nizom i tada možemo iskoristiti opisani algoritam.

4. Rezultati

Testiranje je provedeno na osobnom prijenosnom računalu s procesorom Intel Core i7-4700MQ @ 2.40 GHz i 8 GB RAM-a na operacijskom sustavu Windows 8.1 (64-bit). Implementacija je ostvarena u programskom jeziku C++[1]. Kako nije moguće postići jednake uvjete pri svakom pokretanju programa, svako je mjerenje izvršeno 10 puta te je aritmetička sredina tih mjerenja smatrana konačnim rezultatom.

Ulazni podaci korišteni za testiranje su rezultati očitanja dobivenih sekvenciranjem genoma preuzeti s web-stranice *National Center for Biotechnology Information*[2]. Konkretno, testiranje je provedeno DNA sekvencama 11. kromosoma čovjeka (1), kromosoma gljivice *Pichia sorbitophila* (2) te kromosoma bakterije *Escherichia coli* (3).

Prikazane su usporedbe rezultata provođenja upita nad poboljšanim sufiksnim poljem te nad sufiksnim stablom izgrađenim Ukkonenovim algoritmom koji radi u linearnom vremenu (Ukkonen, 1995).

Tablica 4.1: Vrijeme izgradnje ESA i sufiksnog stabla (sva vremena u ms).

S	bp	SA	LCP	*childtab*	ΣESA	suf. stablo
1	185 035	35	7	46	96	620
2	2 121 241	561	212	513	1 334	7 495
3	4 641 652	1 616	493	1 138	3 357	16 647

U tablici 4.1 vidljivo je da vrijeme izgradnje raste gotovo linearno s povećanjem broja znakova ulaznog niza S, što je i očekivano s obzirom na linearnu vremensku složenost.

[1] https://github.com/ahadviger/esa
[2] http://www.ncbi.nlm.nih.gov/

Tablica 4.2: Vrijeme pretraživanja svih pojavljivanja uzorka P u ulaznom nizu *E. Coli* (4 641 652 bp) koristeći ESA.

P	broj pojavljivanja	vrijeme [ms]
A	1 142 742	25
G	1 177 437	32
CG	346 793	5
TA	212 024	4
TGC	95 270	2
AGT	49 792	1
ATGC	21 746	1
GTCG	17 270	<1

Tablica 4.3: Vrijeme pretraživanja ulaznog niza *E.coli* (4 641 652 bp) uzorkom P.

$\|P\|$	ESA [ms]	suf. stablo [ms]
1 131 755	4	16
1 358 105	4	29
1 629 725	5	22
1 955 669	5	25
2 346 802	7	29
2 816 162	9	34
3 379 394	10	40
4 055 272	12	47

Tablica 4.4: Vrijeme pretraživanja sufiks-prefiks preklapanja ulaznog niza ATATAT... (937 944 bp) i niza P koji je uvijek odabran kao sufiks ulaznog niza pa je duljina maksimalnog preklapanja uvijek jednaka duljini uzorka. Iz mjerenja su izuzeta vremena izgradnje potrebnih struktura.

$\|P\|$	broj preklapanja	ESA [ms]	suf. stablo [ms]
81 920	2006	1	27
163 840	3897	3	28
327 680	1167	3	32
655 360	2262	5	44
781 250	6594	6	45

Vremena pretraživanja svih pojavljivanja preklapanja ne ovise samo o duljini nizova, već i o broju pronađenih pojavljivanja. Poboljšano sufiksno polje pokazuje nešto bolje rezultate nego sufiksno stablo, no obje implementacije uvelike nadmašuju naivne metode.

5. Zaključak

Razvijanje efikasnih struktura za indeksiranje teksta važno je za napredak u znanosti jer su brzine računalnih procesora dosegle svoj vrhunac, a potrebno je u što kraćem vremenskom roku obrađivati sve veće količine podataka kako bi se izveli zaključci ključni za nove znanstvene spoznaje i poboljšanje kvalitete života ljudi. Poboljšano sufiksno polje moćna je i efikasna struktura prilagođena za rješavanje najkompleksnijih problema koji se javljaju u bioinformatici, prvenstveno pretraživanju veoma dugačkih dugačkih DNA sekvenci. Obično je sufiksno polje dobra baza za kompleksnije strukture za indeksiranje i pretraživanje nizova, no samo po sebi manje je funkcionalno od druge popularne strukture, sufiksnog stabla. Danas su dostupni algoritmi za linearnu izgradnju pojedinih dijelova poboljšanog sufiksnog polja, kao što je SA-IS algoritam za izgradnju sufiksnog polja. Zbog linearne složenosti izgradnje, memorijski učinkovitije izvedbe i jednostavnije implementacije, poboljšano sufiksno polje ima bolja svojstva od sufiksnih stabala kojima je u potpunosti ekvivalentno.

Literatura

Mohamed Ibrahim Abouelhoda, Stefan Kurtz, i Enno Ohlebusch. The enhanced suffix array and its applications to genome analysis. U *Algorithms in Bioinformatics*, stranice 449–463. Springer, 2002.

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

Srinivas Aluru. Suffix trees and suffix arrays. *Handbook of Data Structures and Applications*, 2004.

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

Johannes Fischer. Inducing the lcp-array. U *Algorithms and Data Structures*, stranice 374–385. Springer, 2011.

Mile Šikić i Mirjana Domazet-Lošo. Bioinformatika - skripta. 2013.

Mile Šikić i Mirjana Domazet-Lošo. Bioinformatika - predavanja. 2014.

Juha Kärkkäinen i Peter Sanders. Simple linear work suffix array construction. U *Automata, Languages and Programming*, stranice 943–955. Springer, 2003.

Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, i Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. U *Combinatorial pattern matching*, stranice 181–192. Springer, 2001.

Donald E Knuth, James H Morris, Jr, 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. *Journal of Discrete Algorithms*, 3(2):143–156, 2005.

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. *ACM Transactions on Information Systems (TOIS)*, 31(3):15, 2013.

Ge Nong, Sen Zhang, i Wai Hong Chan. Linear suffix array construction by almost pure induced-sorting. U *Data Compression Conference, 2009. DCC'09.*, stranice 193–202. IEEE, 2009.

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.

Poboljšano sufiksno polje

Sažetak

U bioinformatici često se javlja problem pretraživanja veoma dugačkih nizova (DNA sljedova) s različitim uzorcima pretrage. Kako bi se problem riješio u razumnom vremenu, potrebno je koristiti efikasne strukture indeksiranja teksta, kao što su sufiksno stablo i poboljšano sufiksno polje. Poboljšano sufiksno polje sastoji se od običnog sufiksnog polja i dodatnih struktura (polje najduljih zajedničkih prefiksa, tablica djece). Uz manje memorijsko zauzeće, jednostavniju implementaciju i linearno vrijeme izgradnje, koristeći algoritme kao što je SA-IS, u praksi nadmašuje sufiksno stablo zadržavajući svu kompleksnu funkcionalnost.

Ključne riječi: bioinformatika, poboljšano sufiksno polje, SA-IS

Enhanced Suffix Array

Abstract

Bioinformatics problems often require pattern matching large texts (DNA sequences) with different patterns. To find a solution to these problems in a reasonable time, it is required to use efficient text indexing structures, like suffix tree or enhanced suffix array. It is made of a regular suffix array and enhanced with several additional structures (longest common prefix array, child table). With less memory consumption, simpler implementation and linear construction complexity, using SA-IS algorithm, it shows better results in practice than suffix trees while keeping all the complex functionality.

Keywords: bioinformatics, enhanced suffix array, SA-IS