SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKTNI IZVJEŠTAJ

Određivanje LCP polja korištenjem modificiranog algoritma SA-IS

Pero Skoko, Nikola Zadravec, Mislav Požek

Zagreb, siječanj, 2019

Sadržaj

1. Uvod

Polje najduljih zajedničkih prefiksa (*engl. LCP-array*) koristi se u kombinaciji sa sufiksnim poljem u mnogim primjenama obrade tekstualnih podataka. Neke direktne primjene su u području kompresije podataka, prepoznavanja ponavljanja, potpunog i djelomičnog podudaranja nizova znakova. Dodatno sufiksna polja koriste se u izradi sufiksnih stabala, koja imaju i širu primjenu.

Zbog široke primjene, brz pronalazak sufiksnih i LCP polja postao je važan cilj. Sufiksna polja moguće je pronaći u linearnom vremenu izradom sufiksnog stabla, no u primjeni se preferira direktna izgradnja sufiksnog polja radi očuvanja memorije.

S druge strane, brza izrada LCP polja ostaje relativno neistražen problem. Iako je LCP polje moguće izraditi u linearnom vremenu [5], algoritmi za izradu LCP polja zbog manjka pozornosti nisu dostigli razinu algoritama dostupnih za izgradnju sufiksnih polja.

Fischer, inspiriran brzim SA-IS algoritmom u linearnom vremenu za pronalazak sufiksnih polja temeljenom na induciranom sortiranju [2], pokazuje kako prilagoditi algoritam tako da inducira i LCP polje [1]. Time opisuje novi algoritam za izradu LCP polja u linearnom (amortiziranom) vremenu.

U ovom projektu ostvarili smo implementaciju algoritma za izradu LCP polja modificiranim SA-IS algoritmom temeljenu na Fischerovom radu te ju usporedili s prijašnjim dostupnim implementacijama na sintetskim podatcima i genomu E. Coli.

2. Opis algoritma

SA-IS algoritam sastoji se od nekoliko ključnih koraka:

1. Određivanje tipova sufiksa
2. Rekurzivno sortiranje S* sufiksa
3. Induciranje poretka sufiksa L-tipa
4. Induciranje poretka sufiksa S-tipa

Modifikacija SA-IS algoritma za izračun LCP polja dodaje paralelne korake originalnom SA-IS algoritmu:

1. Skaliranje LCP vrijednosti dobivenih rekurzijom u koraku 2. na prave vrijednosti za S* sufikse
2. Induciranje LCP vrijednosti tokom induciranja sufiksa L-tipa
3. Induciranje LCP vrijednosti tokom induciranja sufiksa S-tipa
4. Popravak LCP vrijednosti na međama S i L tipova sufiksa tokom oba indukcijska prolaska

U nastavku opisujemo modificiran SA-IS algoritam kroz primjer niza „aacabcaba”.

Definiramo dodatan završni znak $ koji je leksikografski manji od svakog drugog znaka u abecedi.

i	0	1	2	3	4	5	6	7	8	9
T	a	a	c	a	b	c	a	b	a	$

2.1 Određivanje tipa sufiksa

Pridodjeljujemo tip svakom sufiksu niza. Sufiks na indeksu i je S-tipa ako je leksikografski manji od sufiksa na indeksu i+1, inače je sufiks L-tipa.

Dodatno, definiramo da je sufiks na indeksu i S*-tipa akko je S-tipa i sufiks na indeksu i-1 je L-tipa.

Prazan sufiks je S-tipa.

Tipove sufiksa moguće je odrediti jednim prolaskom kroz niz s desna na lijevo u linearnom vremenu.

i	0	1	2	3	4	5	6	7	8	9

T	a	a	c	a	b	c	a	b	a	$
tip	S*	S	L	S*	S	L	S*	L	L	S*

2.2 Određivanje odjeljaka

Cilj algoritma odrediti je dva polja; sufiksno polje (SA) i polje najduljih zajedničkih prefiksa (LCP). Sufiksno polje je uređeno polje indeksa početaka sufiksa u nizu. Pritom je bitno primjetiti da je moguće podijeliti sufiksno polje u odjeljke po početnim znakovima sufiksa koji će se nalaziti unutar njih. Odjeljci su leksikografski sortirani te je svaki dugačak točno onoliko pozicija koliko ima ponavljanja odgovarajućeg znaka unuar zadanog niza. Gornji niz ima 5 pojavljivanja slova a, 2 pojavljivanja slova b, 2 pojavljivanja slova c i jedno pojavljivanje znaka $, dakle sufiksno polje dijelimo na 4 odjeljka, veličina redom 1,4,2,2.

i	0	1	2	3	4	5	6	7	8	9
odjeljak	$	a	a	a	a	a	b	b	c	c
SA	9									

Dodatno možemo svaki odijeljak rastaviti po tipu sufiksa koji ulaze unutra, imajući pritom na umu da su svi sufiksi L-tipa koji počinju na isto slovo leksikografski manji od sufiksa S-tipa koji počinju na isto slovo.

Podjela polja na odjeljke je vremenski linearna operacija prebrojavanja slova u ulaznoj riječi.

Prvi element SA odgovara praznom sufiksu i uvijek je jednak duljini niza -1.

2.3 Sortiranje S* sufiksa

Početno ubacujemo S* sufikse u odgovarajuće odjeljke nesortirano, tj. prema redoslijedu pojavljivanja u originalnom slijedu. S* sufiksi su na indeksima [0,3,6]

i	0	1	2	3	4	5	6	7	8	9
odjeljak	$	a	a	a	a	a	b	b	c	c
tip	S	L	S	S	S	S	L	S	L	L
SA	9		0	3	6					

Induciramo pretpostavljeni poredak L-induciranjem i S-induciranjem. Pravila S i L induciranja nalaze se u potpoglavljima 2.4 i 2.5 te ih nećemo ponavljati ovdje. Ova operacija je linearna u vremenu.

i	0	1	2	3	4	5	6	7	8	9
odjeljak	$	a	a	a	a	a	b	b	c	c
tip	S	L	S	S	S	S	L	S	L	L
SA	**9**	8	**0**	**6**	**3**	1	7	4	2	5

Ovim postupkom dobili smo točan relativan poredak S* sufiksa, dok oni možda i dalje nisu dobro poredani s obzirom na druge S sufikse unutar istog odjeljka.

Definiramo S* podniz kao podniz originalnog niza razapetog dvama susjednim S* sufiksnim pozicijama unutar originalnog niza.

Unutar primjera imamo 3 S* podniza:

R0 = a a c a (0..3)

R1 = a b c a (3..6)

R2 = a b a $ (6..9)

Pridodajemo S* podnizovima oznake:

vi e [1..n>

takve da

vi < vj ako Ri < Rj i vi = vj ako Ri = Rj.

Oznake pridodajemo pomoću trenutnog poretka S* sufiksa u SA. Prvi S* u SA dobiva oznaku 0, svaki idući uspoređujemo po sadržaju s prošlim i ako su leksikografski jednaki dobiva istu oznaku, u suprotnom za 1 veću. Time je pridodjeljivanje oznaka također linearno u vremenu. (ovdje ne pridodajemo oznaku završnom znaku, njega ignoriramo u pokušaju sortiranja pošto ima samo jednu moguću poziciju)

U našem primjeru dobivamo:

i	R1	R2	R3
SA	0	3	6
vi	0	2	1

Oblikujemo novi niz T’ = v1 v2 v3 ... $ te rekurzivno pozivamo algoritam kako bismo izgenerirali sufiksno polje za T’. Ukoliko je broj jedinstvenih oznaka (abeceda) T’ jednak broju S* podnizova, sufiksno polje možemo sortirati običnim *bucket* sortom, u suprotnom rekurzivno ponavljamo algoritam.

U oba slučaja znamo vrijednosti LCP-a novog niza. Ukoliko nije došlo do rekurzivnog poziva, novi LCP je ispunjen nulama pošto niti jedan sufiks ne počinje istom oznakom (brojem).

To je slučaj u našem primjeru, te dobivamo:

i	0	1	2	3
odjeljak	$	0	1	2
SA’	3	0	2	1
LCP’	x	0	0	0

Sortirani sufiksi niza T’ označavaju sortirane S* podnizove (odnosno sortirane S* sufikse) originalnog niza T.

Vrijednosti LCP’ polja T’ potrebno je skalirati kako bi odgovarale LCP vrijednostima S* sufiksa. To radimo naivnom usporedbom podnizova susjednih S* sufiksa (uključujući i podniz $).

i	0	1	2	3
odjeljak	$	0	1	2
SA’	3	0	2	1
LCP’	x	0	0	0
LCP	x	0	1	2

LCP vrijednost S* sufiksa ne odgovara preklapanju S* podnizova u slučaju da su dva ili više susjednih S* podnizova jednaki. U tom trenutku na konačnu LCP vrijednost potrebno je dodati duljine svih prošlih identičnih S* podnizova. Zbog mogućnosti da sumu pamtimo kroz prolazak kroz S* sufikse, za svaki novi ponavljajući sufiks dovoljno je dodati duljinu prošlog na njegovu sumu što čini ovu operaciju linearnom u vremenu.

To u našem primjeru nije slučaj.

Ubacujemo S* sufikse u **novo** sufiksno polje redoslijedom kojim se pojavljuju u sufiksnom polju T’ u odgovarajuće odjeljke S-tipa. Također ubacujemo skalirane LCP’ vrijednost S* sufiksa na isto mjesto u LCP polju.

i	0	1	2	3	4	5	6	7	8	9
T	a	a	c	a	b	c	a	b	a	$
tip	S*	S	L	S*	S	L	S*	L	L	S*

i	0	1	2	3	4	5	6	7	8	9
odjeljak	$	a	a	a	a	a	b	b	c	c
tip	S	L	S	S	S	S	L	S	L	L
SA	9		0	6	3					
LCP	x		0	1	2					

2.4 Induciranje poretka sufiksa L-tipa

Induciramo položaje L-sufiksa prolaskom kroz sufiksno polje s lijeva na desno.

Za svaku poziciju u sufiksnom polju (koja predstavlja sufiks s odmakom SA[i]), ako je sufiks s jednim znakom više (dakle sufiks s odmakom SA[i]-1) L-tipa, upisujemo SA[i]-1 na početak odgovarajućeg odjeljka L-tipa u sufiksnom polju.

U koraku *i* potencijalno induciramo sufiks L-tipa SA[i]-1 na prvu slobodnu lokaciju u L-odjeljku odgovarajućeg početnog slova.

Ukoliko je inducirani sufiks prvi sufiks u svom L-odjeljku znamo da je također prvi sufiks koji počinje na slovo odjeljka te mu pridodajemo LCP vrijednost 0. U našem primjeru to vrijedi za pozicije 1, 6 i 8.

U suprotnom, neki prijašnji korak *i'* inducirao je zadnji sufiks SA[i']-1 L-tipa u odjeljak istog slova (u primjeru, i = 4, SA[8] = 5 = SA[3]-1, *i'* = 3). U slučaju da su *i'* i *i* u različitim odjelcima znali bismo da novi inducirani sufiks dijeli samo početno slovo s prošlim te bismo postavili LPC[i] = 1. Ukoliko su koraci *i* i *i'* u istom odjeljku, LCP vrijednost induciranog L sufiksa određujemo kao minimalnu LCP vrijednost u rasponu indeksa [*i'+1, i*] + 1.

U primjeru to vrijedi za L sufiks u odjeljku c na indeksu 9. Njega inducira korak *i=4*, a prošli L sufiks u istom odjeljku inducirao je *i'=3*. Oba koraka su u istom odjeljku te je *min(LCP[i'+1, i]) = min(LCP[4]) = LCP[4] = 2,* pa je LCP[9] = 2 + 1 = 3.

i	0	1	2	3	4	5	6	7	8	9
odjeljak	$	a	a	a	a	a	b	b	c	c
tip	S	L	S	S	S	S	L	S	L	L
SA	9	8	0	6	3		7		5	2
LCP	x	0	0	1	2		0		0	3

Ovime smo inducirali pozicije svih sufiksa L-tipa unutar sufiksnog polja i odgovarajuće LCP vrijednosti.

2.5 Induciranje poretka sufiksa S-tipa

Induciramo položaje S-sufiksa prolaskom kroz sufiksno polje s **desna na lijevo**.

Za svaku poziciju u sufiksnom polju (koja predstavlja sufiks s odmakom SA[i]), ako je sufiks s jednim znakom više (dakle sufiks s odmakom SA[i]-1) S-tipa, upisujemo SA[i]-1 na **kraj** odgovarajućeg odjeljka S-tipa u sufiksnom polju.

U koraku *i* potencijalno induciramo sufiks S-tipa SA[i]-1 na zadnju slobodnu lokaciju u S-odjeljku odgovarajućeg početnog slova.

Ukoliko je inducirani sufiks prvi sufiks u svom S-odjeljku naivno računamo LCP vrijednost tog sufiksa s obzirom na zadnji sufiks L-tipa u istom odjeljku.

U suprotnom, neki prijašnji korak *i'* inducirao je zadnji sufiks SA[i']-1 S-tipa u odjeljak istog slova (u primjeru, i = 7, *i'* = 9). U slučaju da su *i'* i *i* u različitim odjelcima znali bismo da novi inducirani sufiks dijeli samo početno slovo s prošlim te bismo postavili LCP = 1 od prošlog sufiksa. Ukoliko su koraci *i* i *i'* u istom odjeljku, LCP vrijednost induciranog L sufiksa određujemo kao minimalnu LCP vrijednost u rasponu indeksa [*i+1, i'*] + 1.

i	0	1	2	3	4	5	6	7	8	9
T	a	a	c	a	b	c	a	b	a	$
tip	S*	S	L	S*	S	L	S*	L	L	S*

i	0	1	2	3	4	5	6	7	8	9
odjeljak	$	a	a	a	a	a	b	b	c	c
tip	S	L	S	S	S	S	L	S	L	L
SA	9	8	**0**	**6**	**3**	**1**	7	**4**	5	2
LCP	X	0	0	**1**	**2**	**1**	0	**1**	0	3

Ovime smo inducirali pozicije svih sufiksa unutar sufiksnog polja te imamo konačno sufiksno i LCP polje te je algoritam priveden kraju.

3. Opis implementacije

Implementacija modificiranog SA-IS algoritma ostvarena je u programskom jeziku C++.

Korištene su samo standardne biblioteke.

Implementacija je ostvarena u objekto orijentiranoj programskoj paradigmi. Algoritam implementiran je kroz 3 razreda:

- SuffixStructure – osnovni razred, zadužen za održavanje stanja sufiksnih i LCP polja i implementaciju indukcijskih funkcija
- StarSuffixStructure – implementacija SuffixStructure razreda zadužena za organizaciju podataka vezanih za rekurzivni korak sortiranja S* sufiksa algoritma
- StringSuffixStructure – implementacija SuffixStructure razreda zadužena za pohranjivanje tekstualnih ciljnih slijedova

4. Rezultati

Našu implementaciju usporedili smo s dvije druge dostupne implementacije modificiranog SA-IS algoritma [3][4] (u nastavku originalna i *novija* implementacija). Mjerili smo memorijsko zauzeće i vrijeme izvođenja na sintetskim podatcima i referentnom genomu escherichia coli.

4.1 Sintetski primjeri

Sintetski primjeri su nizovi nasumično odabranih znakova različitih duljina generiranih pseudo-slučajnim izborom iz abecede malih i velikih slova.

Testirali smo implementacije na 10 različitih duljina nizova: 64, 128, 256, 512, 1024, 2048, 16384, 262144, 524288 i 1048576 znakova.

Za svaku duljinu generirali smo 10 nizova. U nastavku su prikazani prosjeci vremena izvođenja i memorijskog zauzeća za svaku duljinu.

Broj znakova	64	128	256	512	1024	2048	16k	262k	524k	1M
Originalna implementacija	5.12	4.56	4.13	4.68	7.84	5.36	10.55	75.95	127.47	2551.5
Novija implementacija	6.18	4.79	3.79	3.94	7.64	5.69	15.19	67.03	132.04	257.58
Naša implementacija	2.2	2.14	3.04	3.64	4.57	8.42	94.06	2053	4813	11062

Tablica 1 – prosječno vrijeme izvođenja za sve tri implementacije i svaku duljinu niza (u ms)

Broj znakova	64	128	256	512	1024	2048	16k	262k	524k	1M
Originalna implementacija	1.58	1.57	1.6	1.58	1.58	1.58	1.74	3.74	5.85	10.6
Novija implementacija	1.59	1.59	1.57	1.59	1.57	1.61	1.71	3.73	5.83	10.6
Naša implementacija	3.47	3.51	3.54	3.58	3.64	3.8	6.29	48	90	170

Tablica 2 – prosječno memorijsko zauzeće za sve tri implementacije i svaku duljinu niza (u MB)

4.2 Genom E. Coli

Implementacija	Memorijsko zauzeće (MiB)	Vrijeme izvođenja (s)
Originalna implementacija	64.7	2.8
Novija implementacija	44.6	1.4
Naša implementacija	444.7	39.05

Tablica 3 – prikaz memorijskog zauzeća i vremena izvođenja za sve tri implementacija algoritma na slijedu genoma E. Coli

4.3 Analiza linearnosti

Zbog nepreciznog mjerenja vremena izvođenja, odlučili smo generirati dodatne sintetske primjere te mjeriti direktno vrijeme izvođenja samog algoritma za noviju i našu implementaciju (bez učitavanja datoteka).

Generirali smo dva skupa novih sintetskih primjera:

- Sintetski primjeri za duljine niza potencija broja 10 do 10^7 znakova uz punu abecedu engleskog jezika
- Sintetski primjeri za duljine niza potencija broja 10 do 10^7 znakova uz abecedu { A, C, G, T }

U nastavku su navedeni rezultati mjerenja vremena na novim podatcima uz novu metodu.

Duljina niza	*Novija* implementacija (ms)	Naša implementacija (ms)	Omjer vremena izvođenja (naša / njihova)
10	0.009	0.008	0.888889
100	0.017	0.161	9.470588
1000	0.132	1.46	11.06061
10000	2.128	38.313	18.00423
100000	22.12	520.981	23.55249
1000000	236.881	9292.248	39.22749
10000000	5555.742	135928.5	24.46632

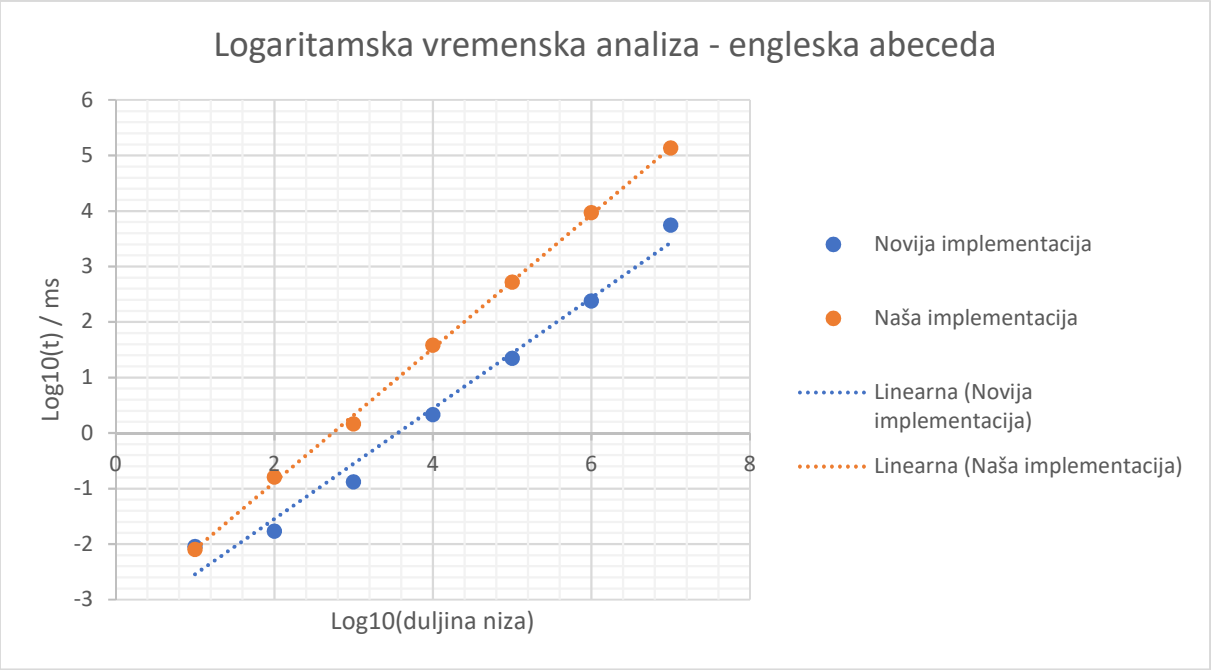
Tablica 4 – prikaz vremena izvođenja novije i naše implementacije na generiranim nizovima engleske abecede

Duljina niza	*Novija* implementacija (ms)	Naša implementacija (ms)	Omjer vremena izvođenja (naša / njihova)
10	0.009	0.006	0.67
100	0.02	0.125	6.25
1000	0.224	1.347	6.01
10000	2.043	14.098	6.90
100000	22.278	195.245	8.76
1000000	246.743	3553.14	14.40
10000000	4612.831	62164.76	13.48

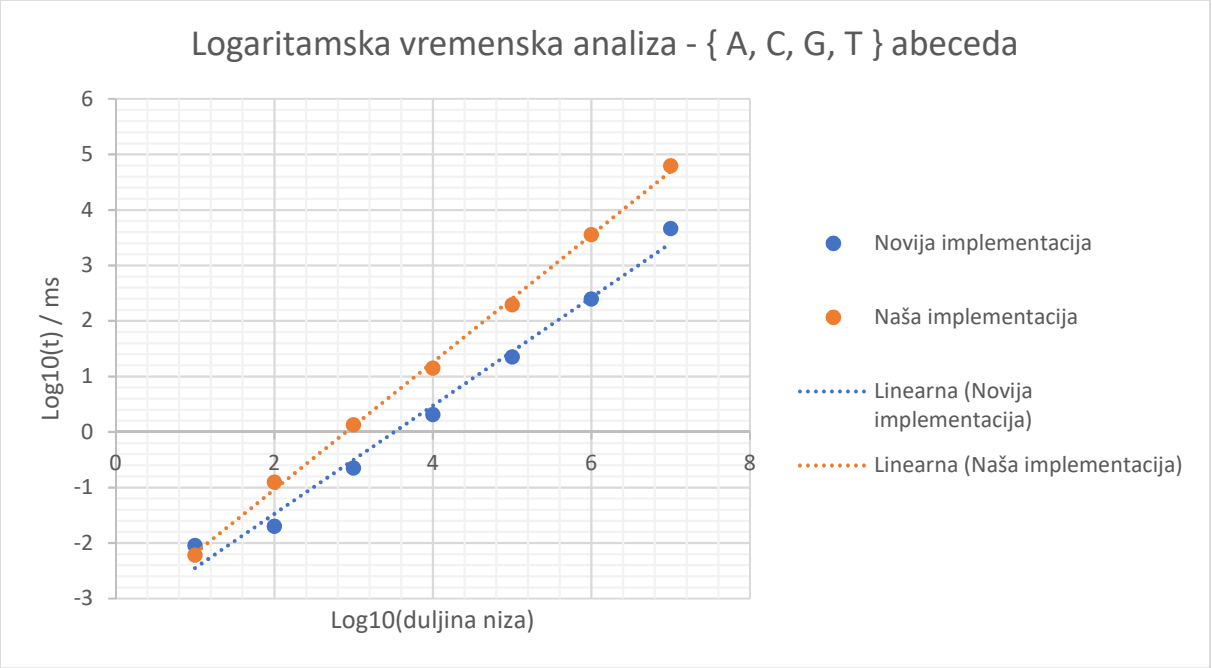
Tablica 5 – prikaz vremena izvođenja novije i naše implementacije na generiranim nizovima abecede { A, C, G, T }

U tablicama su crveno označeni redci pripadajućih nizova duljine 10 pošto smatramo da rezolucija mjerenja vremena nije bila dovoljno velika da bi precizno izmjerila te primjere.

U nastavku prilažemo grafičku analizu linearnosti podataka. Obije osi su logaritmirane radi jasnijeg prikaza (svojstvo linearnosti je očuvano).



Graf 1 – analiza vremena izvođenja na sintetskim podatcima engleske abecede



Graf 2 – analiza vremena izvođenja na sintetskim podatcima abecede {A, C, G, T}

5. Zaključak

Rezultati naše implementacije su lošiji (duže vrijeme izvođenja i veće zauzeće memorije) od postojećih, no postigli smo svojstvo linearnosti.

Za poboljšanje implementacije predlažemo da se kod izračunavanja minimalne vrijednosti za svaki odjeljak odredi hoće li se za njega koristiti mapa ili će se traženi minimum izračunati trčanjem po intervalu u polju LCP, ovisno o veličini odjeljka te da se upotrebom prikladnijeg memorijskog alokatora za strukture podataka iz STL-a smanji vršno memorijsko zauzeće i vrijeme izvođenja.

6. Literatura

[1] J. Fischer. Inducing the LCP-Array. Algorithms and Data Structures, pages 374-385. Springer, 2011.

[2] G. Nong, S. Zhang, and W. H. Chan. Linear suffix array construction by almost pure induced sorting. In *Proc. DCC,* pages 193 – 202. IEEE Press, 2009.

[3] J. Fischer. Proc. 12th Algorithms and Data Structures Symposium (WADS'11), Lecture Notes in Computer Science 6844, 374-385, Springer-Verlag, 2011. http://algo2.iti.kit.edu/english/1828.php

[4] J. Fischer, F. Kurpicz. ad-hoc-implementation of modified SA-IS algorithm, https://github.com/kurpicz/sais-lite-lcp

[5] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proc. CPM,* volume 2089 of *LNCS*, pages 181-192. Springer, 2001.