SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Izgradnja sufiksnog polja korištenjem Kärkkäinen – Sandersovog algoritma

Projekt iz predmeta Bioinformatika

Voditelji:
dr.sc. Mirjana Domazet-Lošo
doc.dr.sc. Mile Šikić

Studenti:
Petra Bevandić, 0036449075
Miranda Kreković, 0036449278
Domagoj Šalković, 0036449374
Petra Vučković, 0036452457
Vedran Vukotić, 0112019329

Zagreb, siječanj 2014.

Sadržaj

1. Uvod

Ubrzan razvoj računala zadnjih pedesetak godina omogućio je njihovu primjenu u rješavanju niza kompleksnih problema od obrade prirodnog jezika do biologije i medicine. Prelaskom fokusa razvoja same računalne tehnologije na osobna računala, pokazalo se da ima smisla razvijati postupke koji rješavaju ove probleme korištenjem računala opće namjene. Ovo je zapravo značilo da se u razvoju samih algoritama većina energije usmjerava na vremensku i memorijsku učinkovitost samih postupaka.

U već spomenutim područjima obrade prirodnog jezika, odnosno biologije i medicine, jedan od čestih problema jest usporedba sekvenci znakova – primjerice za usporedbu sličnosti sekvenci gena ili pronalazak tražene riječi. Jednostavna, a opet vremenski učinkovita struktura koja je ovako nešto omogućavala bila su sufiksna stabla, prvi puta opisana 1973. godine. Sufiksna stabla predstavljaju strukturu u koju se pohranjuju svi mogući sufiksi nekog niza. Primjenjuju se za indeksiranje tekstova, traženje uzoraka, usporedbu nizova i traženje zajedničkih podnizova (maksimalnog podniza, broja svih podnizova itd.) Sufiksnim je stablima ove zadatke moguće izvesti u linearnom vremenu u ovisnosti o duljini promatranog niza.

Ipak, sufiksna stabla imaju i jedan problem, a to je njihova prostorna složenost (memorijsko zauzeće) koje iznosi O($n \log n$). Kao odgovor na ovaj problem, 1990. godine uvedena je nova struktura – sufiksno polje. Sufiksno polje predstavlja strukturu koja pohranjuje niz brojeva koji predstavljaju indekse abecedno poredanih sufiksa u originalnom nizu. Sufiksna polja imaju sve mogućnosti sufiksnih stabala, ali uz linearnu prostornu složenost.

Razvojem algoritama za izgradnju samih sufiksnih polja pokazalo se da ih je moguće izgraditi u linearnom vremenu. Jedan od algoritama koji rade na taj način jest naslovni Kärkkäinen – Sandersov algoritam, koji je implementiran u sklopu ovog projekta.

Ovaj rad sastoji se od tri dijela. U prvome dijelu objašnjava se način rada samoga algoritma. U drugome dijelu prikazani su svi koraci primjene tog algoritma na jednostavnom primjenu. Naposljetku, prikazane su i uspoređene performanse različitih implementacije Kärkkäinen – Sandersovog algoritma.

2. Opis algoritma

2.1 Definicije

U opisu algoritma razmatra se znakovni niz *T* duljine *n*. Za svaki *i*, *j* $\in \mathbb{N}_0$ definiramo:

- $[i...j] := \{i, i+1, ..., j\}$
- $[i...j) := [i...j-1]$
- *T*[*i*] je *i* - ti znak niza *T*
- $T[i...j] := T[i]T[i+1]...T[j]$ je podniz od *i* - tog do *j* - tog znaka
- sva brojanja počinju s 0, pr. $T = T[0...n-1]$
- |*T*| označava duljinu niza, pr. |*T*| = *n*
- Spajanje nizova *X* i *Y* označeno je s $X \cdot Y$, pr. $T = T[0...i-1] \cdot T[i...n-1]$, za $i \in [1...n)$

2.2 Leksikografsko imenovanje

Neka je dan skup znakovnih nizova *S*. Preslikavanje $\phi : S \to [0...|S|)$ naziva se leksikografsko imenovanje ako za svaki *X*, *Y* $\in S$ vrijedi $X <_{lex} Y \Leftrightarrow \phi(X) < \phi(Y)$. Preslikavanje $\phi(X)$ naziva se ime ili rang od *X*.

Opisani algoritam koristi sljedeću lemu kako bi broj usporedbi pri spajanju nizova smanjio na broj usporedbi njihovih leksikografskih imena.

Lema. Neka je dan skup $S \subseteq \sum^t$ znakovnih nizova duljine *t* i leksikografskog imena ϕ. Neka su $X_1, ..., X_k \in S$ i $Y_1, ..., Y_k \in S$ znakovni nizovi iz *S*. Leksikografski odnos između spojenih nizova $X_1 \cdot X_2 \cdot ... \cdot X_k$ i $Y_1 \cdot Y_2 \cdot ... \cdot Y_k$ jednak je leksikografskom odnosu između spojenih leksikografskih imena nizova:

$$X_1 \cdot X_2 \cdot ... \cdot X_k <_{lex} Y_1 \cdot Y_2 \cdot ... \cdot Y_l$$
$$\Leftrightarrow \phi(X_1) \cdot \phi(X_2) \cdot ... \cdot \phi(X_k) <_{lex} \phi(Y_1) \cdot \phi(Y_2) \cdot ... \cdot \phi(Y_l)$$

2.3 Algoritam

Algoritam se može opisati kroz tri glavna koraka:

1. Konstrukcija sufiksnog polja A^{12} svih sufiksa koji započinju na indeksima $i \not\equiv 0 \pmod 3$. To se postiže rekurzivnim pozivom algoritma za znakovni niz čija duljina iznosi $2/3$ duljine početnog niza.
2. Konstrukcija sufiksnog polja A^0 preostalih sufiksa koristeći rezultat prvog koraka.
3. Spajanje nizova A^{12} i A^0 u konačno sufiksno polje.

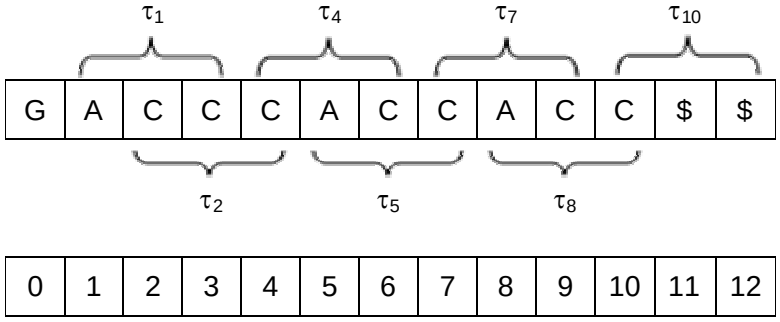
2.3.1 Konstrukcija sufiksnog polja A^{12}

Cilj prvog koraka algoritma stvoriti je sufiksno polje A^{12} promatranog teksta *T* duljine *n*, za sve sufikse $T[i..n-1]$, gdje vrijedi $0 < i < n$ i $i \not\equiv 0 \pmod 3$. Kako bi se algoritam mogao rekurzivno pozvati, stvara se novi tekst *T'* čije će sufiksno polje biti korišteno pri računanju A^{12}. Postupak se provodi kroz sljedeća četiri koraka:

a. Leksikografsko imenovanje trojki svih tripleta $T[i..i+2]$
b. Konstrukcija teksta *T'* od imena tripleta
c. Konstrukcija sufiksnog polja *A'* od *T'* (rekurzivno)
d. Transformacija *A*' u A^{12}

2.3.1.1 Leksikografsko imenovanje trojki

Triplet je podniz znakovnog niza duljine 3. U ovome koraku promatraju se tripleti $T[i...i+2]$ koji počinju na pozicijama $i \not\equiv 0 \pmod 3$. Neka znak $ predstavlja znak koji se ne nalazi u originalnom nizu i koji je leksikografski manji od svih ostalih znakova. Na kraj niza *T* dodaje se $$$ kako bismo mogli izračunati triplete i za pozicije $[n-2...n]$.



Primjer. Neka je *T* = GACCCACCACC. Iz niza se redom izdvajaju svi tripleti koji započinju na pozicijama $\{i \mid i \in [1...n + (n_0 - n_1)] \wedge i \not\equiv 0 (\mathrm{mod}\, 3)\}$. U ovome primjeru niz *T* duljine je 11 znakova, a $i \in \{1, 2, 4, 5, 7, 8, 10, \}$. Niz tripleta sortira se pomoću tri prolaza *radix sort*-a. Tripleti se prvo sortiraju prema posljednjem, zatim prema drugom, te naposljetku prema prvome znaku. Sortiranim tripletima dodjeljuju se leksikografska imena na način da jednaki tripleti imaju i jednaka imena. Leksikografska imena započinju s 0 te se povećavaju za 1 za svaki triplet različit od prethodnog.

početni tripleti		1. prolaz *radix sort*-a		2. prolaz *radix sort*-a		3. prolaz *radix sort*-a		
i	$T[i...i+2]$	i	$T[i...i+2]$	i	$T[i...i+2]$	i	$T[i...i+2]$	τ_i
1	ACC	10	C$$	10	C$$	1	ACC	0
2	CCC	1	ACC	4	CAC	5	ACC	0
4	CAC	2	CCC	7	CAC	8	ACC	0
5	ACC	4	CAC	1	ACC	10	C$$	1
7	CAC	5	ACC	2	CCC	4	CAC	2
8	ACC	7	CAC	5	ACC	7	CAC	2
10	C$$	8	ACC	8	ACC	2	CCC	3

2.3.1.2 Konstrukcija *T'*

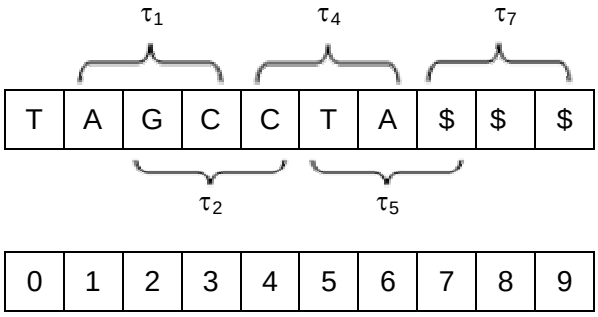
Znakovni niz *T'* nastaje ulančavanjem nizova t_1 i t_2, gdje su t_1 i t_2 nastali spajanjem leksikografskih imena tripleta:

$$t_1 = \tau_1\tau_4\tau_7...\tau_{1+3(n0-1)},$$

$$t_2 = \tau_2\tau_5\tau_8...\tau_{2+3(n2-1)},$$

gdje $n_j = \left\lceil \frac{n-j}{3} \right\rceil$, $j \in \{0,1,2\}$. n_j predstavlja broj tripleta znakovnog niza duljine n koji započinju na pozicijama $i \equiv j(\mathrm{mod}\,3)$.

Kako bi se osiguralo da t_1 uvijek završava sa specijalnim karakterom $, u slučaju kada za duljinu niza vrijedi $n \equiv 1(\mathrm{mod}\,3) \Leftrightarrow n_0 - n_1 = 1$ dodatan triplet $$$ uključuje se u niz i dodaje kao posljednji u podniz t_1. Zbog toga podniz t_1 sadrži $n_1 + (n_0 - n_1) = n_0$ tripleta, a ne n_1 triplet. Podniz t_2 sadrži n_2 tripleta. Primjer kada je duljina $n = 7$ dan je u nastavku:



Tada $n_0 = \left\lceil \frac{7}{3} \right\rceil = 3$, $n_1 = \left\lceil \frac{6}{3} \right\rceil = 2$ i $n_2 = \left\lceil \frac{5}{3} \right\rceil = 2$. Podniz t_1 sadrži tri tripleta, $t_1 = \tau_1\tau_4\tau_7$, a t_2 dva, $t_2 = \tau_2\tau_5$.

U ranije navedeno primjeru, *T* = GACCCACCACC, leksikografski smo sortirali i imenovali triplete. U ovome koraku leksikografska imena poredaju se u podnizove t_1 i t_2, i stvara se niz *T'*.

$$T' = \langle \tau_{1+3i} \mid i \in [0...n_0) \rangle \cdot \langle \tau_{2+3i} \mid i \in [0...n_2) \rangle, \qquad n = 11,\ n_0 = \left\lceil \frac{11}{3} \right\rceil = 4,\ n_2 = \left\lceil \frac{11-2}{3} \right\rceil = 3$$

T' =	τ_1	τ_4	τ_7	τ_{10}	τ_2	τ_5	τ_8
=	0	2	2	1	3	0	0
=	ACC	CAC	CAC	C$$	CCC	ACC	ACC

2.3.1.3 Konstrukcija sufiksnog polja *A'* iz *T'*

Duljina znakovnog niza *T'* iznosi $\left\lceil \frac{2n-1}{3} \right\rceil$. U ovome koraku primjenjuje se rekurzivni poziv kako bi se stvorilo sufiksno polje *A'* niza *T'*. Ako su leksikografska imena τ_i jedinstvena u *T'*, tada se sufiksno polje *A'* može direktno izračunati iz *T'*, bez rekurzije.

Pošto u primjeru *T* = GACCCACCACC polje *T'* iznosi 0221300, potreban je rekurzivni poziv. U prvom koraku rekurzije polje *T'* iznosi 34021. Pošto su svi znakovi jedinstveni, sufiksno polje računa se direktno, *A'* = 24301. Nakon računanja preostalih koraka i povratka iz rekurzije, dobiva se sufiksno polje *A'* početnog niza *T'* koje je jednako:

$A'[0]$	=	6	= 0
$A'[1]$	=	5	= 00
$A'[2]$	=	0	= 0221300
$A'[3]$	=	3	= 1300
$A'[4]$	=	2	= 21300
$A'[5]$	=	1	= 221300
$A'[6]$	=	4	= 300

2.3.1.4 Transformacija *A'* u A^{12}

Sufiksi koji se nalaze u podnizu t_2 počinju na pozicijama $i = j + n_0$ u *T'* i za svakog je poznata točna pozicija *k* u originalnom tekstu *T*, $k = 2 + 3j = 2 + 3(i - n_0)$. Sufiksi koji se nalaze u podnizu t_1 odgovaraju sufiksima na pozicijama $k = 1 + 3i$ teksta *T*. Formula za pretvorbu niza *A'* u A^{12} je sljedeća:

$$A^{12}[i] = \begin{cases} 1 + 3A'[i] & za\ A'[i] < n_0 \\ 2 + 3(A'[i] - n_0) & za\ A'[i] \geq n_0 \end{cases}$$

U prethodnome primjeru $n_0 = 4$ pa polje A^{12} iznosi:

$$\begin{array}{lllll} A'[0] = 6 & \rightarrow & A^{12}[0] = 8 \\ A'[1] = 5 & \rightarrow & A^{12}[1] = 5 \\ A'[2] = 0 & \rightarrow & A^{12}[2] = 1 \\ A'[3] = 3 & \rightarrow & A^{12}[3] = 10 \\ A'[4] = 2 & \rightarrow & A^{12}[4] = 7 \\ A'[5] = 1 & \rightarrow & A^{12}[5] = 4 \\ A'[6] = 4 & \rightarrow & A^{12}[6] = 2 \end{array}$$

2.3.2 Računanje A^0 iz A^{12}

Iz niza A^{12} izdvajaju se sufiksi T_i koji počinju na pozicijama *i*, $i \equiv 1 \pmod 3$. Vrijednosti brojeva *i* - 1 pohranjuju se u novi niz A^0 prema istom poretku. Zatim se koristi *radix sort* kako bi se niz A^0 sortirao prema prvome znaku. Time se dobiva ispravan leksikografski redoslijed jer za svaki *i* < *j* vrijedi:

$$T\left[A^0[i]\right] < T\left[A^0[j]\right] \text{ ili}$$
$$T\left[A^0[i]\right] = T\left[A^0[j]\right] \wedge T\left[A^0[i]+1..n-1\right] <_{lex} T\left[A^0[j]+1..n-1\right]$$

Iz prethodnog primjera:

$$\begin{array}{lll} A^{12}[0] = 8 & & \\ A^{12}[1] = 5 & & \\ A^{12}[2] = 1 & & \\ A^{12}[3] = 10 & \rightarrow & \begin{array}{l} A^0[0] = 0 \\ A^0[1] = 9 \\ A^0[2] = 6 \\ A^0[3] = 3 \end{array} \\ A^{12}[4] = 7 & & \\ A^{12}[5] = 4 & & \\ A^{12}[6] = 2 & & \end{array}$$

početne vrijednosti niza	nakon *radix sort*-a prema početnom slovu
$A^0[0]=0 \rightarrow$ GACCCACCACC	$A^0[3]=9 \rightarrow$ **C**C
$A^0[1]=9 \rightarrow$ CC	$A^0[2]=6 \rightarrow$ **C**CACC
$A^0[2]=6 \rightarrow$ CCACC	$A^0[1]=3 \rightarrow$ **C**CACCACC
$A^0[3]=3 \rightarrow$ CCACCACC	$A^0[0]=0 \rightarrow$ **G**ACCCACCACC

2.3.3 Spajanje A^{12} i A^0 u sufiksno polje A

U posljednjem koraku algoritma potrebno je ispravnim redoslijedom spojiti elemente dva poredana sufiksna polja, A^{12} i A^0. Dva pokazivača postave se na početak svakog niza, a zatim se uspoređuju sufiksi na mjestima određenim pokazivačima. Odabire se manji sufiks, te se njegov pokazivač pomiče za jedno mjesto u desno. Ako je $n \equiv 1 \pmod 3$, prvi sufiks polja A^{12} se preskače.

Za računanje leksikografskog ranga sufiksa iz A^{12}, konstruira se inverzno polje R^{12} za koje vrijedi $R^{12}\left[A^{12}[i]\right]=i+1$. Dva sufiksa polja A^{12} i A^0 uspoređuju se na sljedeće načine:

1. Ako vrijedi $i \equiv 0 \pmod 3$ i $j \equiv 1 \pmod 3$:

$$T[i..n-1] <_{lex} T[j..n-1] \Leftrightarrow \left(T[i] <_{lex} T[j]\right) \vee \left(T[i] =_{lex} T[j] \wedge R^{12}[i+1] < R^{12}[j+1]\right)$$

2. Ako vrijedi $i \equiv 0 \pmod 3$ i $j \equiv 2 \pmod 3$:

$$T[i..n-1] <_{lex} T[j..n-1] \Leftrightarrow \left(T[i..i+1] <_{lex} T[j..j+1]\right) \vee \left(T[i..i+1] =_{lex} T[j..j+1] \wedge R^{12}[i+2] < R^{12}[j+2]\right)$$

Iz prethodnih koraka algoritma za slučaj T = GACCCACCACC poznata su sljedeća polja i vrijednosti, te se računa polje R^{12}:

i	0	1	2	3	4	5	6	7	8	9	10	11	12
T	G	A	C	C	C	A	C	C	A	C	C	$	$
R^{12}	0	3	7	0	6	2	0	5	1	0	4	0	0

$u=0$ ↓

A^{12}	8	5	1	10	7	4	2
A^0	9	6	3	0			

$v=0$ ↑

U prvome koraku uspoređuju se $T[9..10]=CC$ i $T[8..9]=AC$. Pošto je $CC>AC$, kao prvi element konačnog sufiksnog polja postavlja se vrijednost $A[0]=8$ te se pokazivač polja A^{12} pomiče na drugi element.

$u=1$ ↓

A^{12}	8	5	1	10	7	4	2
A^0	9	6	3	0			

$v=0$ ↑

Sada se uspoređuju $T[9..10]=CC$ i $T[5..6]=AC$. Vrijedi ista nejednakost pa $A[1]=5$ i pokazivač $u=2$. Identičan postupak ponavlja i za sljedeći element. Sufiksno polje *A* tada sadrži elemente *A* = 8 5 1, $u=4$, a $v=0$.

$u=4$ ↓

A^{12}	8	5	1	10	7	4	2
A^0	9	6	3	0			

$v=0$ ↑

U ovome slučaju znakovi $T[10]=C$ i $T[9]=C$ su jednaki, pa se uspoređuju njihovi rangovi, $R^{12}[11]=0$ i $R^{12}[10]=4$. Pošto vrijedi $R^{12}[11]<R^{12}[10]$, u polje *A* unosi se vrijednost $A[3]=10$ i pokazivač u povećava se za 1. Postupak se ponavlja za svaki element dok se ne dođe do kraja jednog niza. Tada se ostatak drugog niza prepiše u sufiksno polje *A* i algoritam je završen.

Naposljetku, sufiksno polje *A* = 8 5 1 10 7 4 9 6 3 2 0.

$A[0]$	=	8	→	ACC
$A[1]$	=	5	→	ACCACC
$A[2]$	=	1	→	ACCCACCACC
$A[3]$	=	10	→	C
$A[4]$	=	7	→	CACC
$A[5]$	=	4	→	CACCACC
$A[6]$	=	9	→	CC
$A[7]$	=	6	→	CCACC
$A[8]$	=	3	→	CCACCACC
$A[9]$	=	2	→	CCCACCACC
$A[10]$	=	0	→	GACCCACCACC

3. Primjer izvođenja algoritma

Kako je spomenuto u opisu samog algoritma, ovisno o ulaznom nizu, moguće je da se pojavi potreba za izradom sufiksnog polja *T'*, odnosno da je funkciju koja računa sufiksno polje potrebno rekurzivno pozvati više puta. U nastavku je prikazan primjer rada implementacije algoritma za jednostavniji niz za koji nije potreban rekurzivan poziv i složeniji koji ima jedan rekurzivan poziv.

3.1 Primjer bez rekurzije

Kako bi se implementirana funkcija mogla rekurzivno pozivati, najprije je potrebno niz znakova pretvoriti u niz brojeva.

Neka je zadan niz PBPVDSVVMK. Svakom znaku pridružuje se broj prema abecednome poretku.

P	B	P	V	D	S	V	V	M	K
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
5	1	5	7	2	6	7	7	4	3

Pošto je duljina niza jednaka 10 (pri dijeljenju s 3 daje ostatak 1), na ulazni niz nadodaju tri nule. Zatim se dani niz indeksira, te se indeksi razvrstavaju u tri grupe, ovisno o ostatku koji daju pri dijeljenju s 3.

Tekst:	5	1	5	7	2	6	7	7	8	9	0	0	0
Indeks:	0	1	2	3	4	5	6	7	8	9	10	11	12

Iz niza se zatim izdvajaju oni indeksi koji pri dijeljenju s 3 daju ostatak 1 i 2. Slika dodatno prikazuje trojke iz originalnog niza čije se početne pozicije nalaze na odabranim indeksima.

Grupa 1				Grupa 2		
1 5 7	2 6 7	7 4 3	0 0 0	5 7 2	6 7 7	4 3 0
1	4	7	10	2	5	8

Snaga Kärkkäinen – Sandersovog algoritma leži u sljedećem triku – s obzirom da su trojke konstantne duljine, moguće ih je poredati radix sortom, krenuvši od zadnjeg elementa trojke prema prvome.

Grupa 1 i 2						
0 0 **0**	7 4 **3**	1 5 **7**	2 6 **7**	4 3 **0**	5 7 **2**	6 7 **7**
10	7	1	4	8	2	5

Grupa 1 i 2						
0 **0** 0	7 **4** 3	1 **5** 7	2 **6** 7	4 **3** 0	5 **7** 2	6 **7** 7
10	7	1	4	8	2	5

Grupa 1 i 2						
0 0 0	**1** 5 7	**2** 6 7	**7** 4 3	**4** 3 0	**5** 7 2	**6** 7 7
10	1	4	7	8	2	5

Nakon što su trojke poredane, potrebno ih je „imenovati“ – svakoj trojci pridijeliti redni broj. Iako su trojke grupirane ovisno o grupi kojoj pripadaju imenovanje se vrši u odnosu na trojke iz obje grupe. Ako su dvije trojke jednake dobivaju isto ime. Imenovanje je potrebno da bi se mogla usporediti dva sufiksa prilikom spajanja polja S^{12} i S^{0}.

	Grupa 1				Grupa 2		
	1 5 7	2 6 7	7 4 3	0 0 0	5 7 2	6 6 7	4 3 0
Indeks:	1	4	7	10	2	5	8
Ime:	2	3	7	1	5	6	4

Iz imenovanog je niza sada moguće izgraditi SA^{12}, sufiksno polje koje sadrži sortirane indekse trojki u A^{12}.

	Grupa 1				Grupa 2		
	1 5 7	2 6 7	7 4 3	0 0 0	5 7 2	6 6 7	4 3 0
Indeks:	1	4	7	10	2	5	8
Ime:	2	3	7	1	5	6	4
SA:	3	0	1	6	4	5	2

SA^{12} sada je moguće iskoristiti za sortiranje S^0 polje koje sadrži sortirane indekse trojki iz originalnog niza koji se nalaze na pozicijama koje pri dijeljenju s 3 daju ostatak 0.

	Grupa 0				Grupa 1				Grupa 2		
	3 0 0	5 1 5	7 2 6	7 7 4	1 5 7	2 6 7	7 4 3	0 0 0	5 7 2	6 6 7	4 3 0
Indeks:	9	0	3	6	1	4	7	10	2	5	8
Ime:	1	2	3	4	2	3	7	1	5	6	4
SA:					3	0	1	6	4	5	2

Indeks s S^{12} konačno se modificira tako da sadrži originalne indekse iz originalnog niza. Konačno se S^{12} i S^0 kombiniraju u jedan indeks slijednom usporedbom trojki koje indeksiraju ova dva polja.

	Grupa 0				Grupa 1				Grupa 2		
	3 0 0	5 1 5	7 2 6	7 7 4	1 5 7	2 6 7	7 4 3	0 0 0	5 7 2	6 6 7	4 3 0
Indeks:	9	0	3	6	1	4	7	10	2	5	8
Ime:	1	2	3	4	2	3	7	1	5	6	4
Indeks':	9	0	3	6	10	1	4	8	2	5	7
SA:	1	4	9	8		0	2	5	3	7	6

3.2 Primjer s rekurzijom

Primjer koji se sortira jest primjer koji je korišten i prilikom opisa algoritma: GACCACCACC.

Algoritam opet počinje pretvorbom niza u brojeve.

G	A	C	C	C	A	C	C	A	C	C
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
3	1	2	2	2	1	2	2	1	2	2

Ulaznom se nizu nadodaju dvije nule te se niz zatim indeksira.

Tekst:	3	1	2	2	2	1	2	2	1	2	2	0	0	0
Indeks:	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Izdvajaju se trojke koje se nalaze na indeksima čiji je ostatak prilikom dijeljenja s 3 jednak 1 i 2.

Grupa 1				Grupa 2		
1 2 2	2 1 2	2 1 2	2 0 0	2 2 2	1 2 2	1 2 2
1	4	7	10	2	5	8

Izdvojene trojke opet se sortiraju radix sortom.

Grupa 1 i 2						
2 0 **0**	1 2 **2**	2 1 **2**	2 1 **2**	2 2 **2**	1 2 **2**	1 2 **2**
10	1	4	7	2	5	8

Grupa 1 i 2						
2 **0** 0	2 **1** 2	2 **1** 2	1 **2** 2	2 **2** 2	1 **2** 2	1 **2** 2
10	4	7	1	2	5	8

Grupa 1 i 2						
1 2 2	**2** 0 0	**2** 1 2	**2** 1 2	**1** 2 2	**1** 2 2	**2** 2 2
1	10	4	7	5	8	2

Sortirane trojke zatim se imenuju kako je prethodno opisano. Vidljivo je da postoje trojke koje su jednake. Ovo znači da je nemoguće na temelju početne trojke znati koji sufiks dolazi prije, odnosno da usporedba prva tri znaka postojećih sufiksa nije dovoljna da bi se sortirao niz.

	Grupa 1				Grupa 2		
	1 2 2	2 1 2	2 1 2	2 0 0	2 2 2	1 2 2	1 2 2
Indeks:	1	4	7	10	2	5	8
Ime:	1	3	3	2	4	1	1

Polje s imenima sad predstavlja ulaz u novi poziv funkcije za izgradnju sufiksnog polja. Ideja iza ovog rekurzivnog poziva jest da je očito da za razlikovanje sufiksa nije dovoljno promatrati samo prva tri znaka. Rekurzivnom se pozivom zapravo uspoređuje prvih 9 znakova samog sufiksa, no s obzirom da već znamo indekse trojki, tih se devet znakova može usporediti uspoređivanjem već izračunatih imena trojki. Svi su koraci jednaki kako je prethodno opisano.

Test:	1	3	3	2	4	1	1	0	0	0
Indeks:	0	1	2	3	4	5	6	7	8	9

Gradi se novi S^{12} i sortira.

Grupa 1 i 2				
0 0 **0**	4 1 **1**	3 3 **2**	1 1 **0**	3 2 **4**
7	4	1	5	2

Grupa 1 i 2				
0 **0** 0	4 **1** 1	3 **3** 2	1 **1** 0	3 **2** 4
7	4	1	5	2

Grupa 1 i 2				
0 0 0	**3** 3 2	**4** 1 1	**1** 1 0	**3** 2 4
7	1	4	5	2

Konačno se imenuju sortirani indeksi, ovaj put su jedinstveni pa je moguće izgraditi sufiksno polje za ulazni niz.

	Grupa 1						Grupa 2					
Indeks:	0	0	0	3	3	2	4	1	1	1	1	0	3	2	4
Ime:		7			1			4			5			2	

Iz imenovanog se polja gradi sufiksno polje S^{12}.

	Grupa 1									Grupa 2					
Indeks:	0	0	0	3	3	2	4	1	1	1	1	0	3	2	4
Ime:		7			1			4			5			2	
SA:		2			4			3			0			1	

Konačno se korištenjem S^{12} sortira S^{0}, te se njihovim spajanjem gradi konačno sufiksno polje.

	Grupa 0									Grupa 1									Grupa 2					
	1	0	0	1	3	3	2	4	1	3	3	2	4	1	1	0	0	0	3	2	4	1	1	0
Indeks:		6			0			3			1			4			7			2			5	
Ime:		1			2			3			4			5			1			3			2	
Indeks':		6			0			3			7			5			2			1			4	
SA:		6			5			0						3			2			1			4	

Povratkom iz rekurzije S^{12} sadrži sortirane sufikse na pozicijama koje prilikom dijeljenja s 3 daju 1 ili 2. Iz ovog se polja iznova generiraju imena sufiksa – s obzirom da je polje dobiveno pozivom funkcije za izgradnju sufiksnog polja, ovo je moguće napraviti automatski – činjenica da je S^{12} dobiveno rekurzijom garantira da su svi sufiksi jedinstveni.

	Grupa 1				Grupa 2		
	1 2 2	2 1 2	2 1 2	2 0 0	2 2 2	1 2 2	1 2 2
Indeks:	1	4	7	10	2	5	8
Ime:	3	6	5	4	7	2	1
SA:	6	5	0	3	2	1	4

Konačno se S^{12} originalnog ulaznog niza koristi za generiranje S^0, te se S^{12} i S^0 spajaju u jedinstveno sufiksno polje.

	Grupa 0				Grupa 1				Grupa 2		
	3 1 2	2 2 0	2 1 1	2 1 1	1 2 2	2 1 2	2 1 2	2 0 0	2 2 2	1 2 2	1 2 2
Indeks:	0	9	6	3	1	4	7	10	2	5	8
Ime:	4	1	2	3	3	6	5	4	7	2	1
Indeks':	9	6	3	0	8	5	1	10	7	4	2
SA:	8	5	1	10	7	4	9	6	3	2	0

4. Testiranje imeplementacije i rezultati

Kärkkäinen – Sanders algoritam implementiran je u pet različitih jezika: Perl (Petra Bevandić), Java (Miranda Kreković), Python (Domagoj Šalković), Octave (Vedran Vukotić), C# (Petra Vučković).

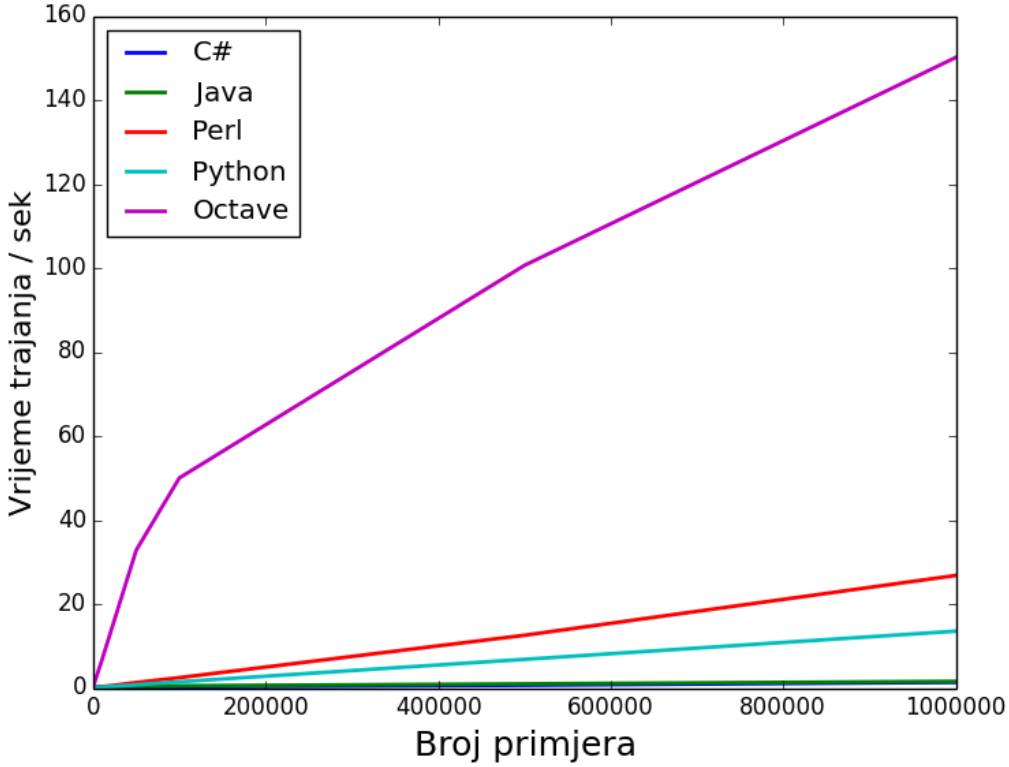
Obavljena su mjerenja performansi kako bi se mogle usporediti brzine izvođenja algoritama te memorija koju pojedini algoritam zauzima tijekom izvođenja. Testiranja su obavljena sa različitim veličinama ulaza. Veličine stringova koji su korišteni kao ulazi su: 100, 500, 1 000, 5 000, 10 000, 50 000, 100 000, 500 000 i 1 000 000. Također da bi testiranja bila realna, sva mjerenja su obavljena istom računalu i na istim primjerima.
U *tablici 1* prikazana su obavljena mjerenja vremena izvođenja u sekundama.

Broj znakova	**100**	**500**	**1000**	**5000**	**10000**	**50000**	**100000**	**500000**	**1000000**
C#	0.173	0.177	0.151	0.142	0.169	0.211.	0.263	0.653	1.342
Java	0.176	0.172	0.199	0.216	0.291	0.402	0.523	0.976	1.574
Perl	0.128	0.076	0.100	0.181	0.283	1.308	2.463	12.553	26.795
Python	0.111	0.112	0.112	0.155	0.214	0.782	1.471	6.820	13.532
Octave	0.335	0.584	0.881	3.396	6.504	32.826	50.013	100.612	150.116

Tablica 1 Mjerenja trajanja izvođenja programa za različite duljine ulaza.

Također, na *slici 1* su prikazana mjerenja u obliku grafova. Svaki jezik prikazan je različitom bojom. Na *x* osi nalazi se duljina ulaznog znakovnog niza, a na *y* osi nalazi se vrijeme izvođenja, odnosno vrijeme koje je potrebno da program iz ulaznog niza izračuna sufiksno polje.

Sa slike se može uočiti da su programski jezici C# i Java najbrže izvršavali algoritam. Kärkkäinen – Sanders algoritam u teoriji ima vremensku složenost *O(n)* pa su očekivani i takvi rezultati. Prikazani grafovi su linearni, tako da se može zaključiti da i implementirani algoritam ima vremensku složenost *O(n).* Naravno, rezultati ovise o samome jeziku u kojima je algoritam implementiran, a i o načinu implementacije samog algoritma.

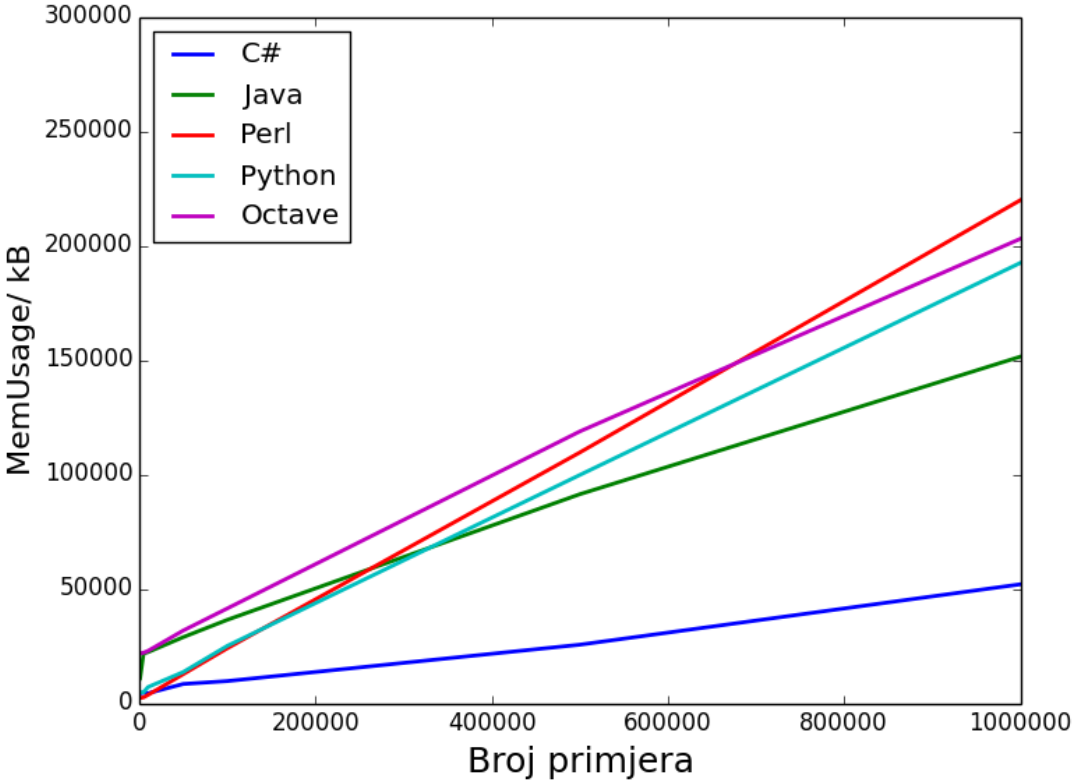


Slika 1 Ovisnost vremena izvođenja o broju duljini ulaznog niza.

Broj znakova	100	500	1000	5000	10000	50000	100000	500000	1000000
C#	4576	4580	4316	4580	4320	8400	9668	25564	51960
Java	10360	10360	10624	21716	22292	28852	36428	91320	151448
Perl	2008	2272	2272	2540	3592	12668	23964	109560	219848
Python	4756	4448	4756	4756	7032	13612	25040	99724	192436
Octave	21956	21720	21740	22060	22848	31740	41408	118756	202955

Tablica 2 Količina zauzete memorije u kB.

U *tablici 2* prikazani su rezultati zauzeća memorije pojedinog programa u kB ovisno o duljini ulaznog niza. Kao što je i logično veći ulazni niz, zauzima više memorije, što je prikazano i grafički na *slici 2.* Svi programski jezici, osim C# u ovom mjerenju daju slične rezultate, dok C# zauzima najmanje memorije od svih ostalih. Konačno iz grafova je vidljivo da je i zauzeće memorije linearno u ovisnosti o duljini ulaznog niza.



Slika 2 Ovisnost zauzeća memorije o veličini ulaznog niza.

5. Upute za korištenje

Kod pokretanja programa kao argumenti predaju se putanje do ulazne i izlazne datoteke. U ulaznoj datoteci treba se nalaziti znakovni niz za kojeg se želi izračunati sufiksno polje, te se to polje zapisuje u izlaznu datoteku.

Implementirani algoritmi prihvaćaju dva tipa ulaznih datoteka: običan tekst i FASTA format. Datoteka u kojoj se nalazi FASTA format raspoznaje se po prvom znaku, odnosno ako je prvi znak datoteke oznaka komentara (;) ili oznaka sekvence (>) datoteka je FASTA formata, inače je obična tekstualna datoteka.

Kod FASTA formata linije koje počinju s znakom > označavaju početak sekvence u sljedećem retku. Također, svaki student u svom programskom rješenju implementirao je mogućnost da se u jednoj datoteci nalazi više sekvenca. U tom slučaju izlazi, odnosno sufiksna polja određenih sekvenci, ispisana su u datoteci svaka u svojem retku, te se između njih nalazi prazan redak. Isto tako, algoritmi su implementirani tako da u FASTA formatu propuštaju sva slova i znakove: crtica (-) te zvjezdica (*). Ako je ulazni znak malo slovo, ono se automatski prebacuje u veliko tiskano slovo.

Prikazan je primjer FASTA formata ulazne datoteke:

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWG
ATVITNLFSAIPYIGTNLVEWIWGGFSVDKATLNRFFAFHFILPFTMVALAGV
HLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLGLLILILLLLLLALLSPDML
GDPDNHMPADPLNTPLHIKPEWYFLFAYAILRSVPNKLGGVLALFLSIVIL
GLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIGQM
ASILYFSIILAFLPIAGXIENY
```

Izlazna datoteka, odnosno sufiksni niz za odgovarajući ulaz je:

167 88 272 35 101 277 187 60 99 201 231 151 262 33 81 51 185 1 136 79 124 239 156 162 160 168 122 281 110 19 70 179 252 87 184 86 89 128 91 182 137 107 214 273 203 36 58 267 76 95 48 233 50 159 75 74 197 24 139 210 118 259 44 7 112 247 13 65 102 198 278 38 109 90 5 175 104 164 129 220 216 276 280 258 6 246 64 257 269 134 176 270 208 142 144 92 188 264 25 126 61 30 54 206 72 10 80 135 219 125 195 177 271 100 200 150 0 183 202 57 232 158 196 138 209 117 108 174 215 141 143 29 149 140 28 148 27 147 146 145 152 240 211 83 170 93 274 41 225 189 204 153 228 105 241 236 119 243 68 265 16 2 261 32 238 37 157 26 224 223 165 212 46 97 163 9 194 56 67 114 115 84 22 171 282 166 155 161 178 127 213 94 275 116 173 169 227 193 250 42 62 254 130 230 260 45 249 85 8 226 221 190 59 123 121 18 47 268 263 205 218 222 113 154 229 248 77 191 14 34 106 23 111 4 256 133 82 235 242 31 237 96 55 66 172 120 217 52 244 20 98 78 69 251 103 207 53 199 40 192 49 73 43 245 71 21 234 180 279 283 186 181 266 12 63 15 253 17 3 255 132 39 11 131

U običnim tekstualnim datotekama može se nalaziti samo jedna sekvenca, odnosno sve što se nalazi u datoteci spaja se u niz te se tvori sufiksno polje tog znakovnog niza. Također, u običnoj tekstualnoj datoteci mogu se nalaziti svi znakovi.

6. Zaključak

Potreba za efikasnim algoritmima i strukturama podataka koje mogu izvršavati niz operacija nad dugačkim sekvencama znakova dovela je do uvođenja sufiksnih polja. Sufiksna polja su struktura koja pohranjuje indekse sortiranih sufiksa nekog niza te se mogu primjenjivati za usporedbu niza, pronalaženje zajedničkih podnizova, pronalaženje uzoraka i sl.

U sklopu ovog projekta bilo je potrebno implementirati Kärkkäinen – Sandersov algoritma koji konstruira sufiksno polje uz linearnu vremensku složenost ovisno o duljini ulaznog niza. Opisan je sam algoritam, te kroz primjere izložen rad konkretne implementacije algoritma. Konačno, s obzirom da je sam algoritam implementiran u nekoliko različitih programskih jezika, dana je usporedba performansi različitih implementacija.

Dok postoje razlike u performansama u ovisnosti o odabranom programskom jeziku (interpreterski jezici su nešto sporiji od jezika koji se kompajjliraju u izvršnu datoteku), sve implementacije pokazuju linearnu vremensku i prostornu složenost u ovisnosti o duljini ulaznog niza.

7. Literatura

1. Kärkkäinen, J., Sanders, P. Simple Linear Work Suffix Array Construction. *Proceedings of International Colloquium on Automata, Languages and Programming 2003,* stranice 943-955, 2003.

2. Kärkkäinen, J., Sanders, P. i Burkhart, S. Linear work suffix array construction. *Journal of the ACM,*53(6), stranice 918-936, 2006