SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Izgradnja sufiksnog stabla - Ukkonenov algoritam

Frane Kurtović i Matija Šantl
Voditelj: *doc. dr. sc. Mirjana Domazet-Lošo*

Zagreb, siječanj 2015.

SADRŽAJ

1. Uvod **1**

1.1. Osnovni pojmovi 1

2. Opis algoritma **2**

2.1. Izgradnja sufiksnog stobla 2

2.2. Sufiksno stablo 4

2.3. On-line izgradnja sufiksnog stabla 5

3. Prikaz rada algoritma **10**

4. Rezultati testiranja **13**

4.1. Postavljanje sustava 13

4.2. Izgradnja ostvarenog rješenja 13

4.3. Escherichia Coli 14

4.4. Brochothrix Campestris 15

4.5. Sintetički testni primjer 16

5. Zaključak **18**

6. Literatura **19**

7. Sažetak **20**

1. Uvod

Sufiksno stablo (engl. *suffix tree*) je stablasta struktura podataka koja sadrži sve sufikse nekog niza znakova [6]. Sufiksno stablo na elegantan način rješava problem pretraživanja podniza u nizu, precizinije, podniz je moguće pronaći u vremenu proporcionalnom duljini podniza [5]. Jedna takva primjena je pronalazak najduljeg zajedničkog podniza dvaju nizova u vremenu porporcionalnom zbroju njihovih duljina [4].

Sufiksno stablo može biti izgrađeno u vremenu linearno ovisnom o duljini ulaznog niza. Današnji standard je *Ukkonenov* algoritam iz 1995. godine. To je prvi *on-line* algoritam za izgradnju sufiksnog stabla [5].

1.1. Osnovni pojmovi

Neka je $T = t_1 t_2 ... t_n$ niz znakova nad abecedom Σ. Svaki niz znakova x takav da je $T = uxv$ za neke (moguće i prazne) nizove u i v nazivamo podnizom od T.

U ovom tekstu korištene su sljedeće oznake:

- ulazni niz znakova je označen s T,
- duljina ulaznog niza znakova T je označena s n
- *prefiks* niza T je podniz $T[1, i]$ te je označen s T^i, $1 \leq i \leq n$
- *sufiks* niza T je podniz $T[i, n]$ te je označen s T_i, $1 \leq i \leq n + 1$
- posebno, T_{n+1} je prazan sufiks, $T_{n+1} = \epsilon$
- skup svih sufiksa niza znakova T je označen s $\sigma(T)$
- stoblo sufiksa (engl. *suffix trie*) je stoblo koje predstavlja $\sigma(T)$

2. Opis algoritma

Ukkonenov algoritam se može predstaviti kao linearna verzija algoritma za sufiksna stobla koji je kvadratne složenosti. Taj je algoritam opisan u poglavlju 2.1. Jednom transparentnom promjenom, koja je opisana u poglavlju 2.3, dolazimo do algoritma linearne složenosti, poznatijeg pod nazivom *Ukkonenov* algoritam za izgradju sufiksnog stabla u linearnom vremenu.

2.1. Izgradnja sufiksnog stobla

Formalno, sufiksno se stoblo može definirati kao petorka $STrie(T) = (Q \cup \perp, root, F, g, f)$, i možemo reći da je takvo stoblo prošireni deterministički konačni automat koji ima stablolik graf prijelaza koji predstavljaju stoblo $\sigma(T)$ koje je prošireno sa tzv. funkcijom sufiksa (engl. *suffix function*) f i pomoćnim stanjem $\perp$. Skup Q od stanja $STrie(T)$ se može staviti u jedan-naprema-jedan vezu s podnizovima niza T. Stanje koje odgovara podnizu x oznacavamo s $\overline{x}$.

Početno stanje $root$ odgovara praznom nizu ϵ, a skup F završnom skupu stanja $\sigma(T)$. Funkcija prijelaza g je definirana kao $g(\overline{x}, a) = \overline{y}$ za sve $\overline{x}$, $\overline{y}$ u Q tako da $y = xa$ pri čemu je $a \in \Sigma$.

Funkcija sufiksa f je definirana za svako stanje $\overline{x} \in Q$ na sljedeći način. Neka je $\overline{x} \neq root$. Onda $x = ay$ za neki $a \in \Sigma$, i $f(\overline{x}) = \overline{y}$. Dodatno, $f(root) = \perp$.

Pomoćno stanje $\perp$ nam omogućava pisanje algoritama u nastavcima tako da se izbjegne eksplicitna razlika između praznih i ne-praznim sufiksa. Stanje $\perp$ je spojeno na stoblo s $g(\perp, a) = root$ za svaki $a \in \Sigma$.

U skladu s [6], funkciju sufiksa $f(r)$ nazivamo sufiksnom vezom (engl. *suffix link*) stanja r.

On-line izgradnja sufiksnog stobla $STrie(T)$ radi se prolaskom niz T s lijeva na desno. Kao međurezultat izgradnje dobijamo $STrie(T^i)$ za $i = 0, 1, ..., n$. Ključna primjedba koja opisuje kako se $STrie(T^i)$ dobija iz $STrie(T^{i-1})$ je da se sufiksi od T^i mogu dobiti dodavanjem t_i na kraj svakog sufiksa od T^{i-1} i dodavanjam praznog

sufiksa, tj.

$$\sigma(T^i) = \sigma(T^{i-1}) \cup \epsilon$$

Prema definiciji, $STrie(T^{i-1})$ prihvaća σT^{i-1}. Kako bismo dobili da prihvaća i σT^i, moramo proučiti skup završnih stanja F_{i-1} od $STrie(T^{i-1})$. Ako $r \in F_{i-1}$ još nije t_i-prijelaz , prijelaz od r prema novom stanju se dodaje. Stanja u koja vode neki stari ili novi t_i-prijelazi iz nekog stanja F_{i-1} zajedno sa stanjem $root$ čine završna stanja F_i od $STrieT^i$.

Stanja $r \in F_{i-1}$ koja dobe nove prijelaze mogu se pronaći koristeći sufiksne veze na sljedeći način. Definicija sufiksne funkcije implicira da je $r \in F_{i-1}$ ako i samo ako $r = f^j(\overline{t_1...t_{i-1}})$ za neki $0 \leq j \leq i-1$. Zbog toga su sva stanja u F_{i-1} na putu sufiksne veze koja počinje na najdubljem stanju $\overline{t_1..t_{i-1}}$ od $STrie(T^{i-1})$ i završava na $\perp$. Taj se put zove granični put (engl. *boundary path*) od $STrie(T^{i-1})$.

Ako stanje $\overline{z}$ na graničnom putu još nema prijelaz na t_i, novo stanje $\overline{zt_i}$ i novi prijelaz $g(\overline{z}, t_i) = \overline{zt_i}$ se dodaje. Time osvježavamo funkciju prijelaza, g. Funkciju sufiksa, f, osvježavamo tako da nova stanja $\overline{zt_i}$ povezujemo s novim sufiksnim vezama koje čine put koji počinje u stanju $\overline{t_1t_2..t_i}$, tj. granični put od $STrie(T^i)$.

Prolazak preko F_{i-1} na graničnom putu se može odmah zaustaviti kada je pronađeno prvo stanje $\overline{z}$ takvo da stanje $\overline{zt_i}$ (a time i prijelaz $g(\overline{z}, t_i) = \overline{zt_i}$) već postoji. Neka je $\overline{zt_i}$ stanje. Onda $STrie(T^{i-1})$ mora sadržavati stanje $\overline{z't_i}$ i prijelaz $g(\overline{z'}, t_i) = \overline{z't_i}$ za svaki $\overline{z'} = f^j(\overline{z})$, $j \geq 1$. Drugim riječima, ako je zt_i podniz od T^{i-1} onda je svaki sufiks od zt_i podniz od T^{i-1}. Stanje $\overline{z}$ će uvijek postojati jer je stanje $\perp$ zadnje stanje na graničnom putu i stanje $\perp$ ima definiran prijelaz za svaki mogući t_i.

Kada je takav obilazak završio, stvorit ćemo novo stanje za svaku sufiks vezu koju smo pregledali tijekom obilaska. To implicira da će cijeli postupak trajati ovisno o veličini dobivenog automata.

U nastavku je prikazan algoritam za izgradnju stobla $STrieT^i$ iz $STrie(T^{i-1})$, pri čemu je *top* zapravo stanje $\overline{t_1...t_{i-1}}$;

```
1 r = top;
2 dok je g(r, t[i]) nedefinirano
3   stvori novo stanje r' i novi prijelaz g(r, t[i]) = r';
4   ako je r != top onda: stvori novu sufiksnu vezu f(oldr') = r';
5   oldr' = r';
6   r = f(r);
7 stvori novu sufiksnu vezu f(oldr') = g(r, t');
8 top = g(top, t[i])
```

Programski odsječak 2.1: Izgradnja $STrie(T^i)$ iz $STrie(T^{i-1})$

Počevši od $STrie(\epsilon)$, koje se sastoji samo od stanja $root$, $\perp$ i veza između njih, ponavljanjem procedure 2.1 za $t_i = t_1, t_2, ...t_n$ dobijamo $STrie(T)$. Algoritam je optimalan u smislu da je vremenska složenost proporcionalna konačnom rezultatu, $STrie(T)$, a to je proporcionalno s $|Q|$, tj. brojem različitih podnizova od T. Međutim, to može dovesti do kvadratne složenosti u $|T|$, ako je npr. $T = a^n b^n$.

2.2. Sufiksno stablo

Sufiksno stablo $STree(T)$ nad nizom T je struktura podataka koja predstavlja $STrie(T)$ u prostoru koji je linearan s duljinom $|T|$ niza T [6]. To se postiže tako da ne koristimo sva, već samo podskup stanja $Q' \cup \perp$ svih stanja od $STrie(T)$. Ta stanja nazivamo eksplictna stanja (engl. *explicit states*). Skup Q' se sastoji od granajućih stanja (engl. *branching state*), stanja koja imaju barem dva prijelaza, i listova (engl. *leaves*), stanja koja nemaju prijelaza. Po definiciji, stanje $root$ je uključeno u granajuća stanja. Sva preostala stanja $STrie(T)$ zovemo implicitna stanja (engl. *implicit states*) sufiksnog stabla $STree(T)$.

Niz znakova w koji se nalazi na putu u stoblu $STrie(T)$ između dva eksplicitna stanja s i r je u stablu $STree(T)$ predstavljen kao generalizirani prijelaz $g'(s, w) = r$. Kako bi uštedjeli na prostoru, niz znakova w je zapravo prikazan uređenim parom pokazivača (k, p) na T takvih da je $w = t_k...t_p$. Zbog toga, generalizirani prijelaz ima oblik $g'(s, (k, p)) = r$.

Prijelaz $g'(s, (k, p)) = r$ nazivamo *a-prijelaz* ako je $t_k = a$. Svako stanje s može imati najviše jedan *a-prijelaz* za svaki $a \in \Sigma$.

Prijelazi $g(\perp, a) = root$ su prikazani na sličan način. Neka je $\Sigma = a_1, a_2, ..., a_m$. Onda je prijelaz $g(\perp, a_j) = root$ prikazan kao $g(\perp, (-j, -j)) = root$ za $j = 1, ..., m$.

Zbog toga, sufiksno stablo $STree(T)$ ima dvije komponente: niz znakova T i samo stablo. Ono je linearne veličine u $|T|$ jer skup Q' ima najviše $|T|$ listova (postoji najviše jedan list za svaki ne-prazni sufiks) i zbog toga skup Q' mora sadržavati najviše $|T| - 1$ granajućih stanja. Može postojati najviše $2|T| - 2$ prijelaza između stanja u skupu Q', gdje svaki prijelaz zauzima konstatan prostor zbog korištenja pokazivača.

Stablo se dodatno proširuje sa sufiksnom funkcijom f', koja je sada definirana za sva granajuća stanja $\overline{x} \neq root$ kao $f'(\overline{x}) = \overline{y}$ gdje je y takvo granajuće stanje da je $x = ay$ za neki $a \in \Sigma$, i $f'(root) = \perp$.

Formalno, sufiksno stablo možemo predstaviti uređenom četvorkom kao $STree(T) = (Q' \cup \perp, root, g', f')$.

Na neko implicitno ili eksplicitno stanje sufiksnog stabla r se referenicramo pomoću uređenog para s, w gdje je s neko eksplicitno stanje koje prethodi stanju r a w je niz znakova koje posjećujemo na putu od s prema r u odgovarajućem sufiksnom stoblu. Referentni par (engl. *reference pair*) se naziva kanonskim (engl. *canonical*) ako je stanje s najbliži predak stanja r (i prema toma niz w je najkraći mogući). Za takvo eksplicitno stanje r, kanonski referentni par je (r, ϵ), tj. budući da koristimo pokazivače, taj par poprima oblik $(r, (p+1, p))$ pri čemu je $w = t_k...t_p$.

Eksplicitna završrna stanja dobivamo dodavanjem oznake kraja na T koja se ne pojavljuje nigdje u T.

2.3. On-line izgradnja sufiksnog stabla

Neka su $s_1 = \overline{t_1...t_{i-1}}$, $s_2, s_3, ..., s_i = root$, $s_{i+1} = \perp$ stanja sufiksnog stobla $STrie(T^{i-1})$ koja se nalaze na graničnom putu. Neka je j najmanji indeks takav da s_j nije list, i neka je j' najmanji indeks takav da $s_{j'}$ ima t_i-prijelaz.

Lema 1 *Algoritam 2.1 dodaje u $STrie(T^{i-1})$ jedan t_i-prijelaz za svako stanje s_h, $1 \leq h \leq j'$ takvo da je $1 \leq h \leq j$, novi prijelaz proširuje postojeće grane stobla koje završavaju na listu s_h, a za $j \leq h \leq j'$, stvara nove prijelaze koji se granaju iz s_h.*

Stanje s_j zovemo aktivnom točkom (engl. *active point*) a stanje $s_{j'}$ završnom točkom (engl. *end point*) sufiksnog stobla $STrie(T^{i-1})$. Ta stanja postoje kao eksplicitna ili implicitna i u sufiksnom stablu $STree(T^{i-1})$.

Prema *Lema 1*, algoritam 2.1 dodaje dvije skupine t_i-prijelaza u $STrie(T^{i-1})$.

1. Stanja na graničnom putu prije aktivne točke s_j dobijaju prijelaz. Ta stanja su listovi pa zbog toga svaki prijelaz mora proširiti postojeće grane sufiksnog stobla.

2. Stanja nakon aktivne točke s_j pa sve do završne točke $s_{j'}$ (ne uključujući) dobijaju novi prijelaz, a budući da ta stanja nisu listovi, ti prijelazi moraju započeti novu granu.

Prva skupina prijelaza se može implementirati kao osvježavanje desnog pokazivača od svakog prijelaza koji predstavlja granu. Neka je $g'(s, (k, i-1)) = r$ jedan takav prijelaz. Desni pokazivač mora pokazivati na zadnju poziciju, i-1, od T^{i-1}. Zbog toga što je stanje r list, svaki put koji vodi do r mora biti jedan sufiks od T^{i-1} koji se ne pojavljuje nigdje drugdje unutar T^{i-1}. Tada osvježeni prijelaz mora biti $g'(s, (k, i)) =$

r. Time se uopće ne mijenjaju stanje r i s već samo niz znakova koji nastaje obilaskom puta. Kako bi osvježavanje svakog takvog stanja zasebno bilo vremenski preskupo, koristi se sljedeći trik.

Svaki prijelaz stable $STrer(T^{i-1})$ koji vodi ka listu se zove otvoreni prijelaz (engl. *opem transition*). Takav je prijelaz oblika $g'(s,(k,i-1)) = r$. Budući da desni pokazivač mora pokazivati na zadnju poziciju $i-1$ od T^{i-1}, nije nužno da desni pokazivač ima ispravnu vrijednost, već se umjesto toga otvoreni prijelazi mogu predstaviti kao $g'(s,(k,\infty)) = r$ gdje ∞ označava da otvoreni prijelaz može "rasti". $g'(s,(k,\infty)) = r$ zapravo predstavlja granu bilo koje duljine između stanja s i imaginarnog stanja r koje je u "beskonačnosti". Zbog toga nije potrebno eksplicitno osvježavanje desnog pokazivača kada se umeće t_i u granu. Oznake ∞ se mogu zamijeniti s $n = |T|$ nakon što smo završili izgradnju stabla $STree(T)$.

Druga skupina prijelaza stvara sasvim nove grane koje počinju u stanjima s_h, $j \leq h \leq h'$. Traženje takvih stanja s_h zahtjeva malo pažnje budući da ne moraju nužno biti eksplicitna u tom trenutku. Takva ćemo stanja nalaziti na graničnom putu stabla $STree(T^{i-1})$ koristeći referentne parove i sufiksne veze.

Neka je $h = j$ i neka je (s,w) kanonski referentni par za s_h za recimo aktivnu točku. Kako je s_h na graničnom putu stobla $STrie(T^{i-1})$, w mora biti sufiks od T^{i-1}. Zbog toga je $(s,w) = (s,(k,i-1))$ za neki $k \leq i$.

Želimo li kreirati novu granu koja počinje u stanju na koje se referenciramo s $(s,(k,i-1))$, moramo najprije provjeriti da li se možda $(s,(k,i-1))$ već referencira na stanje $s_{j'}$. Ako se referencira onda smo gotovi. Inače moramo stvoriti novu granu. Stanje na koje se referenciramo s $(s,(k,i-1))$, s_h mora biti eksplicitno do sad. Ako stanje s_h nije eksplicitno, ono se stvara dijeljenjem prijelaza koje sadrži odgovarajuće implicitno stanje. Tek onda se stvara t_i-prijelaz iz stanja s_h. Taj prijelaz mora biti otvoreni prijelaz, tj. $g'(s_h,(i,\infty)) = s_{h'}$ pri čemu je $s_{h'}$ novi list. K tome, dodaje sufiksna veza $f'(s_h)$ ako je stanje s_h nastalo dijeljenjem prijelaza.

Nako toga se izgradnja pomiče na s_{h+1}. Budući da je referentni par za s_h bio $(s,(k,i-1))$, kanonski referentni par za s_{h+1} će biti $canonize(f'(s),(k,i-1))$ pri čemu $canonize$ stvara referentni par kanonskim tako da osvježi stanje i lijevi pokazivač. Gornje navedene operacije se tada ponavljaju za s_{h+1} pa sve dok se pronađe završna točka $s_{j'}$.

Na taj način radi funkcija $update$ koja je prikazana u programskom odsječku 2.2. Ta funkcija pretvara stablo $STree(T^{i-1})$ u $STree(T^i)$ umetanjem t_i-prijelaza iz druge skupine. Funkcija koristi pomoćne funkcije $canonize$ koje je spomenuta gore, te $test-and-split$ koja provjerava da li se dani referentni par referencira na završnu točku.

Ako se ne referencira na završnu točku tada funkcija $test - and - split$ stvara i vraća novo eksplicitno stanje za dani referentni par ako se taj referentni par već ne referancira na eksplicitno stanje. Funkcija $update$ vraća referentni par za završnu točku $s_{j'}$.

```
1 funkcija update(s, (k, i)):
2   // (s, (k, i - 1)) je kanonski referentni par za aktivnu tocku
3   stari_root = root
4   (zavrsna_tocka, r) = test_and_split(s, (k, i-1), t[i])
5
6   dok je zavrsna_tocka = False
7     stvori novi prijelaz g‘(r, (i, INF)) = r‘
8     ako je stari_root != root onda
9       stvori novu sufiksnu vezu f‘(stari_root) = r
10    stari_root  = r
11    (s, k) = canonize(f‘(s), (k, i-1))
12    (zavrsna_tocka, r) = test_and_split(s, (k, i- 1), t[i])
13
14  ako je stari_root != root onda
15    stvori novu sufiksnu vezu f‘(stari_root) = s
16
17  vrati (s, k)
```

Programski odsječak 2.2: Funkcija $update$

Funkcija $test - and - split$ provjerava da li je stanje s kanonskim referentnim parom $(s, (k, p))$ završna točka, tj. da li bi to stanje u stoblu $STrie(T^{i-1})$ imalo t_i-prijelaz. Simbol t_i je dan kao ulazni parametar t. Ako $(s, (k, p))$ nije završna točka, stanje (s, (k, p)) se napravi eksplicitnim ako još nije tako da se podijeli prijelaz.

```
1 funkcija test-and-split(s, (k, p), t):
2   ako je k <= p onda
3     neka je g‘(s, (k‘, p‘)) = s‘ t[k]-prijelaz iz s;
4     ako je t = t[k‘ + p - k _ 1] onda
5       vrati (True, s)
6     inace
7       zamijeni gornji t[k]-prijelaz s prijelazima
8         g‘(s, (k‘, k‘ + p - k)) = r
9         g‘(r, (k‘ + p - k + 1, p‘)) = s‘
10        pri cemu je r novo stanje
11        vrati (False, r)
12  inace
13    ako ne postoji t-prijelaz iz s
14      vrati (False, s)
15    inace
```

```
16        vrati (True, s)
```

Programski odsječak 2.3: Funkcija $test - and - split$

Odgovor na pitanje da li je neko stanje završna točka može se naći u konstantnom vremenu promatrajući pritom samo jedan prijelaz is s.

Funkcija $canonize$ za dani referentni par $(s, (k, p))$ za neko stanje r, nalazi i vraća stanje s' i lijevi pokazivač k' takvi da $(s', (k', p))$ bude referentni par za stanje r. Stanje s' je najbliži eksplicitni predak stanja r (ili stanje r ako je ono eksplicitno). Zbog toga niz znakova koji vode iz s' u r mora biti sufiks niza $t_k...t_p$ koj vodi iz s u r. Iz tog se razloga desni pokazivač ne mijenja, dok lijevi može postati k', $k' \geq k$.

```
1  funkcija canonize(s, (k, p)):
2    ako je p < k onda
3      vrati (s, k)
4    inace
5      pronadi t[k]-prijelaz g‘(s, (k‘, p‘)) = s‘ iz s
6      dok je p‘ - k‘ <= p - k
7        k = k + p‘ - k‘ + 1
8        s = s‘
9        ako je k <= p onda
10         nadi t[k]-prijelaz g‘(s, (k‘, p‘)) = s‘ iz s
11
12     vrati (s, k)
```

Programski odsječak 2.4: Funkcija $canonize$

Kako bi se omogućila daljnja konstrukcija stabla za simbol t_{i+1}, mora se pronaći aktivna točka stabla $STree(T^i)$. Stanje s_j je aktivna točka stabla $STree(T^{i-1})$ ako i samo ako je $s_j = \overline{t_j...t_{i-1}}$ pri čemu je $t_j...t_{i-1}$ najdulji sufiks od T^{i-1} koji se pojavljuje barem dva puta u T^{i-1}. Stanje $s_{j'}$ je završna točka stabla $STree(T^{i-1})$ ako i samo ako je $s_{j'} = \overline{t_{j'...t_{i-1}}}$ gdje je $t_{j'...t_{i-1}}$ najdulji sufiks od T^{i-1} takav da je $t_{j'...t_{i-1}}t_i$ podniz od T^{i-1}. To znači da ako je $s_{j'}$ završna točka stabla $STree(T^{i-1})$ onda je $t_{j'...t_{i-1}}t_i$ najdulji sufiks od T^i koji se pojavljuje barem dvaput u T^i, tj., stanje $g(s_{j'}, t_i)$ je aktivna točka stabla $STree(T^i)$.

Lema 2 *Neka je $(s, (k, i-1))$ referentni par završne točke s'_j stabla $STree(T^{i-1})$. Onda je (s, (k, i)) referentni par aktivne točke stabla $STree(T^i)$.*

```
1 stvori stanja root i start
2 za j = 1 do m
3   stvori prijelaz g‘(, (-j, -j)) = root
```

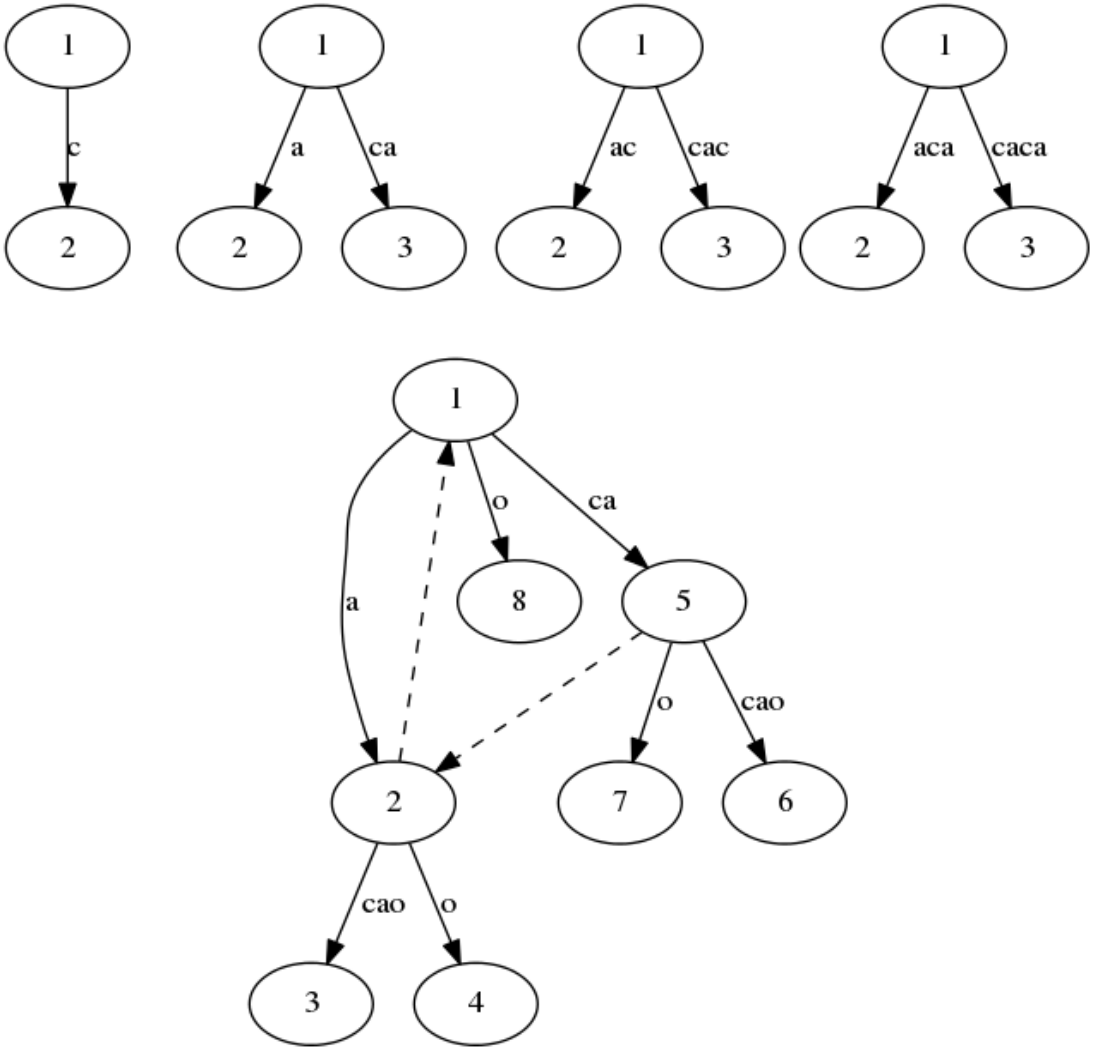
```
4 stvori sufiksnu vezu f‘(root) = start
5
6 s = root, k = 1, i = 0
7 dok t[i+1] nije zavrsni znak
8   i = i + 1
9   (s, k) = update(s, (k, i))
10  (s, k) = canonize(s, (k, i))
```

Programski odsječak 2.5: Konstrukcija sufiksnog stabla

Programski odsječak 2.5 konstruira sufiksno stablo $STree(T)$ za ulazni niz znakova $T = t_1..t_n$ on-line u vremenskoj složenosti $O(n)$.

3. Prikaz rada algoritma

Izgradnja sufiksnog stabla se odvija redom, dodavajući svaki put novi znak u stablo. Nakon svakog koraka sufiksno stablo je valjano.



Slika 3.1: Izgradnja sufiksnog stabla za riječ cacao

Na slici 3.1 je prikazana izgradnja stabla za već klasičnu riječ *cacao*. Na ovim

sufiksnim stablima čvor broj 1 uvijek predstavlja korijen stabla. Njegov roditelj nije prikazan jer se nikada ne mijenjaju veze iz ili prema tom čvoru.

Opis dodavanja svakog idućeg znaka iz riječi cacao.

1. Prije dodavanja znakova. Aktivna točka je u korijenu stabla, a završna točka je roditelj korijena.

2. Dodavanje znaka c. Ne postoji ništa prije aktivne točke. U aktivnoj točki se dodaje znak c čime nastaje novi prijelaz i novi čvor. Sufkisna veza vodi do završne točke, te time završava dodavanje novih čvorova. Prijelazom iz završne točke znakom c nastaje nova aktivna točka – korijen.

3. Dodavanje znaka a. Svi čvorovi prije aktivne točke (listovi) dobivaju automatski znak a na kraj. U aktivnoj točki se dodaje prijelaz a te nastaje novi čvor označen brojem 2. Iduće stanje je roditelj od korijena, no on već ima prijelaz za znak a te time ažuriranje stabla završava. Aktivna točka se dobiva prijelazom znakom a, a to je korijen stabla.

4. Dodavanje znaka c. Svi čvorovi prije aktivne točke (listovi) dobivaju automatski znak c na kraj. U aktivnoj točki, tj. u korijenu, već postoji prijelaz za znak c, što znači da je postupak ažuriranja stabla gotov. Nova aktivna točka postaje stanje do kojeg se dolazi koristeći znak c. To je stanje s oznakom $(root, c)$, tj. nalazi se na bridu $1 - 3$ iza slova c.

5. Dodavanje znaka a. Svi čvorovi prije aktivne točke (listovi) dobivaju automatski znak a na kraj. U aktivnoj točki, tj. iza slova c na bridu $1 - 3$ već postoji znak a što znači da je osvješavanje stabla završeno i da se aktivna točka samo pomiče koristeći znak a – brid $1 - 3$ iza ca $(root, ca)$.

6. Dodavanje znaka o. Svi čvorovi prije aktivne točke (listovi) dobivaju automatski znak o na kraj. U staroj aktivnoj točki, tj. iza slova ca na bridu $1 - 3$ ne postoji znak o što znači da je potrebno dodati novu granu na tom mjestu. To se radi tako da se stvara novi čvor na tom mjestu, na slici čvor 5, te iz njega izlaze dva brida, jedan koji vodi do starog čvora (broj 6 na slici) i jedan koji vodi do novostvorenog čvora (broj 7 na slici). Sljedeće stanje u koje algoritam dolazi je $(root, a)$. Dodavanjem znaka o se također mora razbiti brid te stvoriti novi čvor označen brojem 2, te iz njega izlaze dvije grane, jedna prema starom čvoru označenim brojem 3 i prema novom čvoru označeim brojem 4. Također dodaje se sufiksna veza iz prethodnog čvora označenog brojem 5 prema čvoru 2 jer je

to stanje upravo sufiks prošlog stanja. Iduće stanje u koje algoritam dolazi je sufiks od $(root, a)$, a to je $(root, \epsilon)$. Iz njega ne postoji prijelaz za slovo o pa se stvara novi čvor označen brojem 8 te se stvara sufiksna veza iz prošlog čvora do trenutnog čvora, tj. iz čvora 2 do čvora 1. Nova aktivna točka je $(root, \epsilon)$.

4. Rezultati testiranja

Za testiranje programske izvedbe *Ukkonenovog* algoritma za izgradnju sufiksnog stabla koristili smo *Bio-Linux 8* platformu koja se može preuzeti na [1].

4.1. Postavljanje sustava

Bio-Linux 8 je moćna, besplatna platforma prilagođena potrebama u bioinformatici koja se može instalirati na osobno računalo ili za to posebno predviđeni poslužitelj. *Bio-Linux 8* uključuje više of 250 bioinformatičkih paketa na bazni operacijski sustav *GNU/Linux Ubuntu 14.04 LTS*.

Međutim, naša programska izvedba *Ukkonenovog* algoritma zahtijeva instalaciju dodatnog paketa koji je potreban za iscrtavanje sufiksnog stabla. Radi se o paketu *graphviz* koji je moguće instalirati sljedećom naredbom

```
1 sudo apt-get install libgraphviz-dev
```

4.2. Izgradnja ostvarenog rješenja

Kako bi olakšali komajliranje ostvarenog programskog rješenja, koristili smo *Makefile* datoteku. Ona omogućuje definiranje pravila kompajliranja izvornog teksta ostvarenog rješenja prema kojima se izgrađuju izvršne datoteke. Za potrebe ovog projekta izradili, kao rezultat izgradnje ostvarenog rješenja nastaju sljedeće izvršne datoteke:

- trie - izvršna datoteka za izgradnju sufiksnog stobla
- test_tree - izvršna datoteka koja provjerava ispravan rad izgradnje sufiksnog stabla
- visualize_tree - izvršna datoteka koja prilikom izgradnje sufiksnog stabla, nakon svakog koraka, iscrtava trenutno stanje sufiksnog stabla

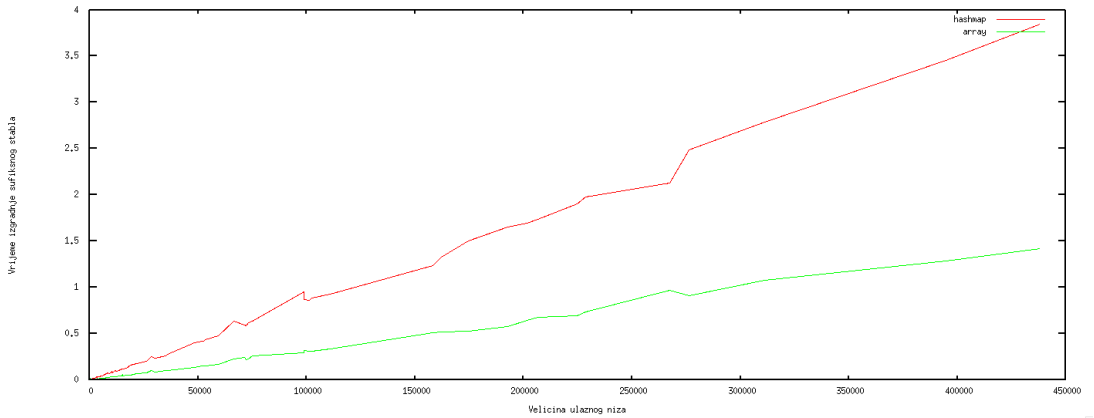
– create_tests - izvršna datoteka koja čita genom u *FASTA* formatu, te u zasebne datoteke sprema genome za potrebe testiranja

– test_performance - izvršna datoteka koja čita testove koji se nalaze u zasebnim datotekama te prilikom izgradnje sufiksnog stabla mjeri utrošeno vrijeme i memoriju

4.3. Escherichia Coli

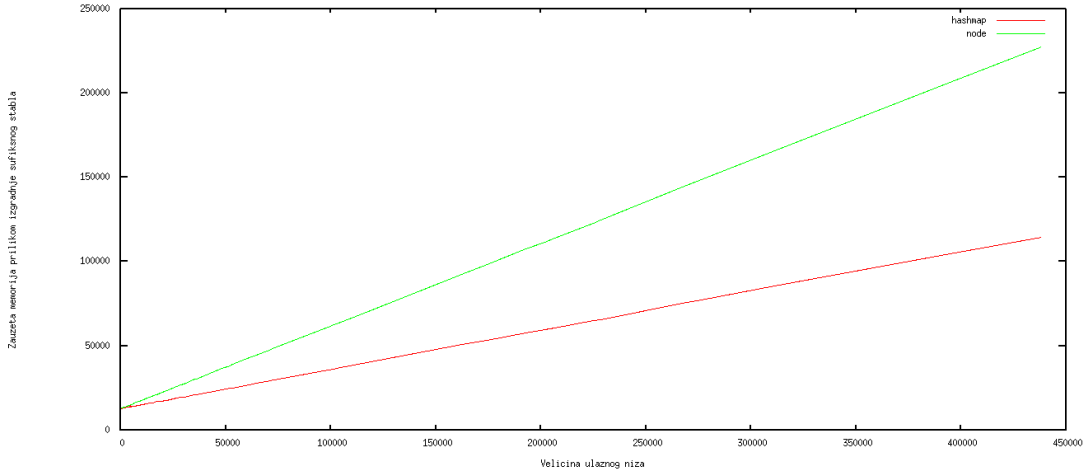
Genom bakterije *Escherichia Coli* preuzet je s [2].

Nakon preuzimanja komprimirane datoteke i raspakiravanja iste, korištena je izvršna datoteke *create_tests* koja je kreirala zasebne datotke za testiranje u kazalu */tests/*. Izvršnom datotekom *test_performance* su zatim provedeni svi testovi te spremljeni rezultati izvođenja. Alatom *gnuplot* su zatim iscrtane ovisnosti vremena izgradnje i memorije potrebne za izgradnju sufiksnog stabla u ovisnosti o veličini ulaznog niza.



Slika 4.1: Vremenska ovisnost o veličini ulaznog niza

Na slici 4.1 možemo vidjeti dva grafa. Crvenom je bojom označena vremenska ovisnost izgradnje sufiksnog stabla u slučaju kada ja kao struktura podataka čvora korištena *hash* tablica dok je zelenom bojom označena vremenska ovisnost izgradnje sufiksnog stabla u slučaju kada ja kao struktura podataka čvora korišten *niz*.



Slika 4.2: Memorijska ovisnost o veličini ulaznog niza

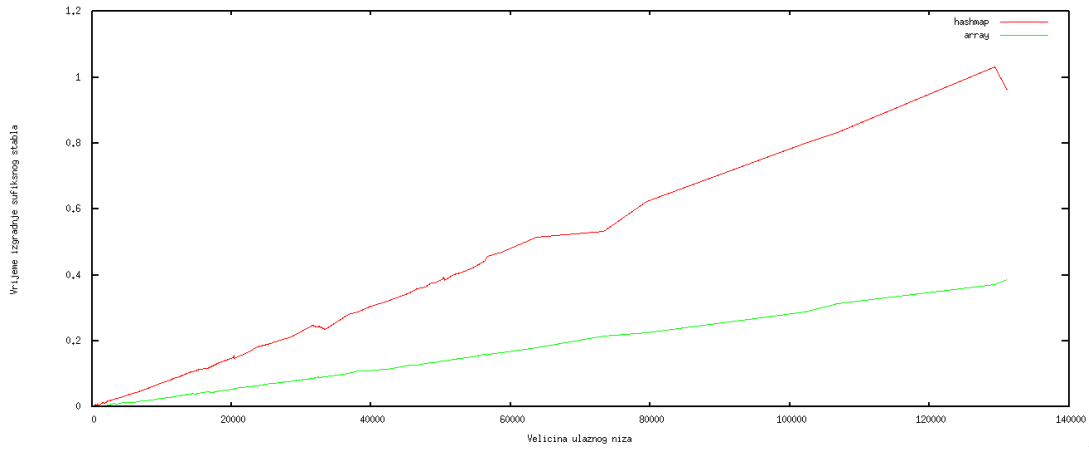
Slika 4.2 također koristi iste oznake kao i one za sliku 4.1.

Na obje se slike može uočiti **linearan** porast vremena izvođenja, odnosno, potrebne memorije prilikom izgradnje sufiksnog stabla nad genomom bakterije *Escherichia Coli*.

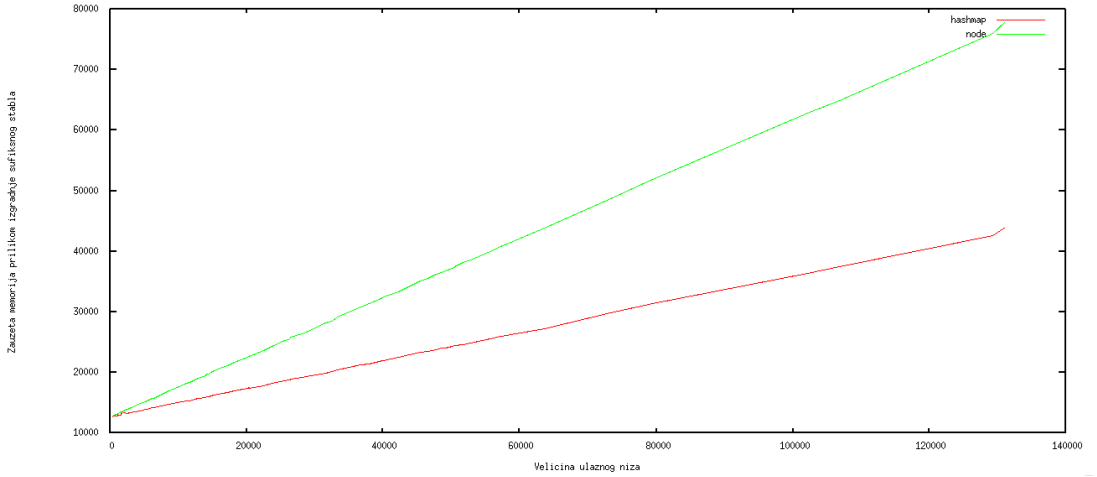
4.4. Brochothrix Campestris

Genom bakterije *Brochothrix Campestris* preuzet je s [3].

U nastavku su prikazane performanse ostvarenog programskog rješenja nad genomom bakterije *Brochothrix Campestris*.



Slika 4.3: Vremenska ovisnost o veličini ulaznog niza



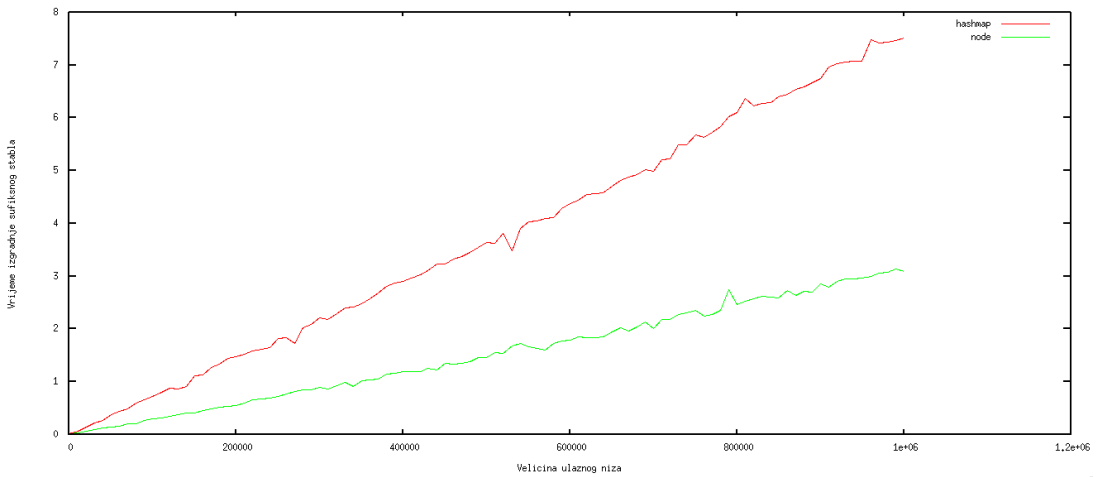
Slika 4.4: Memorijska ovisnost o veličini ulaznog niza

Na slici 4.3 te slici 4.4, možemo uočiti **linearan** porast vremena izvođenja, odnosno potrebne memorije prilikom izgradnje sufiksnog stabla nad genomom bakterije *Brochothrix Campestris*.

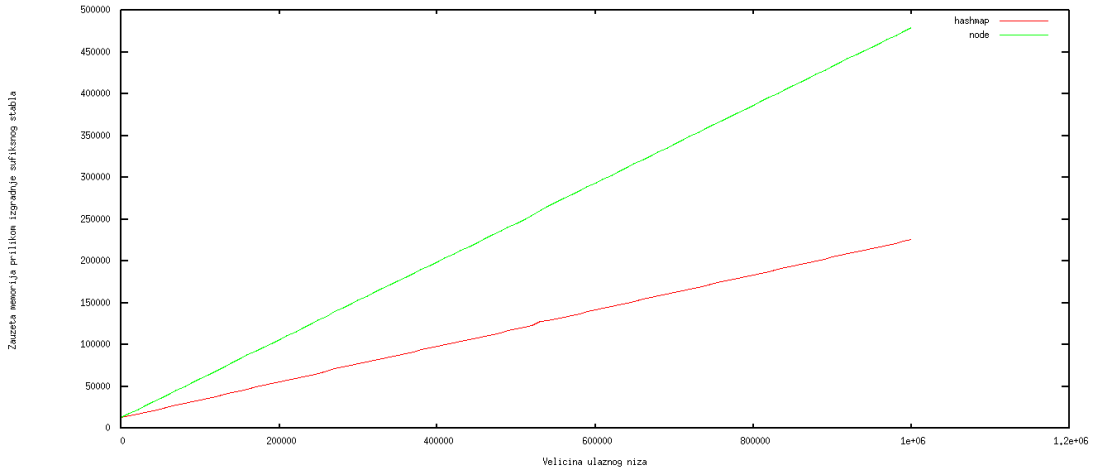
4.5. Sintetički testni primjer

Za dodatno testiranje performansi, korišteni su i sintetički testni primjeri generirani pseudo-slučajnim generatorom. Raspon veličine ulaznog niza je od 100 do 1000000, dok je veličina abecede pet znakova.

U nastavku su prikazane performanse ostvarenog programskog rješenja na sintetičkim testinim primjerima.



Slika 4.5: Vremenska ovisnost o veličini ulaznog niza



Slika 4.6: Memorijska ovisnost o veličini ulaznog niza

Na slici 4.5 te slici 4.6, opet uočavamo **linearan** porast vremena izvođenja, odnosno potrebne memorije prilikom izgradnje sufiksnog stabla na sintetičkim testnim primjerima.

5. Zaključak

Ukkonenov algoritam za izgradnju sufiksnog stabla ima dvije prednosti. Prva je ta što je vremenska složenost izgradnje stabla linearna s obzirom na duljinu ulaznog niza znakova, tj. ako je T ulazni niz znakova, onda je vremenska složenost *Ukkonenovog* algortima $O(n)$ pri čemu je $n = |T|$.

Druga prednost je što ima poželjno svojstvo procesiranja simbola s lijeva na desno te pritom ima izgrađeno sufiksno stablo za pročitani dio ulaznog niza, tj. ako smo pročitali prvih k znakova ulaznog niza, $T = t_1 t_2 ... t_n$, nakon obrade simbola t_k, sufiksno stablo $STree(T^k)$ će biti u potpunosti izgrađeno.

Ostvareno programsko rješenje pokazuje da izgradnja sufiksnog stabla *Ukkonenovim* algoritmom, i vremenski i memorijski, linearno ovisi o veličini ulaznog niza T.

6. Literatura

[1] Bio linux. http://environmentalomics.org/bio-linux-download/, Sijecanj 2015.

[2] Genom escherichia coli. ftp://ftp.ensemblgenomes.org/pub/bacteria/release-25/fasta/bacteria_11_collection/escherichia_coli_07798/dna/Escherichia_coli_07798.GCA_000303655.1.25.dna.genome.fa.gz, Sijecanj 2015.

[3] Genom brochothrix campestris. ftp://ftp.ensemblgenomes.org/pub/bacteria/release-25/fasta/bacteria_80_collection/brochothrix_campestris_fsl_f6_1037/dna/Brochothrix_campestris_fsl_f6_1037.GCA_000525915.1.25.dna.genome.fa.gz, Sijecanj 2015.

[4] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences*. University of California, Davis Cambridge University Press, 1997.

[5] Mile Sikic i Mirjana Domazet-Loso. *Bioinformatika*. Sveuciliste u Zagrebu, Fakultet elektrotehnike i racunarstva, 2013.

[6] Esko Ukkonen. *On-line construction of suffix trees*. Department of Computer Science, University of Helsinki.

7. Sažetak

Ukkonenov algoritam efikasno rješava problem izgradnje sufiksnog stabla. Vremenska i memorijska složenost je $O(n)$, tj. optimalna. Unatoč tome u praksi je memorijska konstanta često prevelika jer sufiksno stablo zauzima 10 do 50 puta više memorije nego li je veličina ulaznog niza. Zbog toga su su se dosta počela istraživati sufiksna polja koja zauzimaju puno manje memorije, a mogu u jednakoj složenosti odgovarati na skoro sve upite kao i sufiksna stabla.

Unatoč tome ovo je jedan od jednostavnijih efikasnih *on-line* algoritama izgradnje sufiksnog stabla.