SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**PROJEKT IZ BIOINFORMATIKE**

# Ukkonenov Algoritam

*Mislav Marković*

*Filip Matijević*

*Marko Raguž*

Voditelj: *Mislav Marković*

Zagreb, siječanj, 2019.

## Sadržaj

# 1. Uvod

Da bismo objasnili kako radi Ukkonenov algoritam potrebno je prvo objasniti što je to sufiksno stablo. Sufiksno stablo je je komprimirani trie koji sadrži sve sufikse zadatnog teksta gdje su ključevi sufiksi a vrijednosti su pozicije u originalnom tekstu.

Naivna implementacija sufiksnog stabla zahtjeva $O(n^2)$ ili čak $O(n^3)$ vremensku složenost.

Ukkonenov Algoritam je online (slovo po slovo) algoritam koji gradi sufiksno stablo u linearnom vremenu, predložen od Eska Ukkonena 1995 godine prema kojem je dobio i ime.

## 2. Opis algoritma

Struktura podataka, tj. Sufiksno stablo, koje na kraju trebamo stvoriti iz danog teksta je poput search trie-a ali umjesto jednog znaka po bridu kao oznaka brida koristit će se par brojeva [od, do]. To su pokazivači na originalni text te tako svaki brid predstavlja String proizvoljne duljine, ali zauzima samo O(1) memorije jer se spremaju samo 2 pokazivača. Ukoliko se radi o čvorovima njihov „do" pokazivač se automatski povećava kako se stablu dodaju novi znakovi, time u složenosti O(1) ažuriramo postojeće čvorove te nam samo preostaje dodavanje novog samostalnog sufiksa. Dodatno, čvorovi sadrže pokazivač „suffix link" koji se tijekom izgradnje stabla koristi kako bi se algoritam ispravno pozicionirao na aktivni čvor nakon dodavanja novog čvora u stablo. Stablo razlikuje dvije vrste čvorova, interni i rubni (leaf) čvor. Interni čvorovi nastaju kada se prilikom dodavanja novog znaka u stablo otkrije da je on implicitno već sadržan u stablu, tada algoritam uzima znakove sve dok ne pronađe sufiks koji više nije implicitno sadržan u stablu. U tom trenutku se čvor dijeli na dva čvora, ovisno gdje je otkriveno da sufiks više nije sadržan u stablu, te jedan čvor postaje dijete drugog čvora i time dobivamo novi interni čvor. Ukoliko je u istom koraku algoritma već nastao interni čvor, novi interni čvor i prošli stvoreni interni čvor treba povezati sufiksnom vezom u smjeru od prošlog internog čvora do novog internog čvora. Nakon stvaranja novog čvora aktivni čvor otkrivamo tako da pratimo sufiksnu vezu iz trenutačno aktivnog čvora, ukoliko ona postoji, inače se kao aktivni čvor postavlja korijen stabla.

Objasnit ćemo algoritam ukratko na jednostavnom primjeru. Za početak potrebne su dodatne varijable.

1. Aktivna točka - trojka( aktivni_čvor, aktivni_brid, aktivna_dužina). Govori nam gdje moramo početi sa umetanjem novog sufiksa.
2. Ostatak – Govori nam koliko suffiksa moramo umetnuti izravno. Npr ako je tekst abcdefg, a ostatak je 3 onda moramo procesirati zadnja 3 sufiksa "efg", "fg", te "g".

Uzmimo da nam je tekst “abcabd”. Prvo dodajemo specijalni znak $ na kraj teksta tako da je krajnji tekst “abcabd$”.

Korak 1: obrađeno : “”

aktivni_čvor: 0

aktivni_brid: none

aktivna_dužina: 0

ostatak: 0

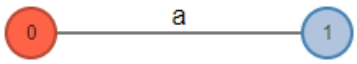

Korak 2:  obrađeno: “a”

aktivni_čvor: 0

aktivni_brid: none

aktivna_dužina: 0

ostatak: 0



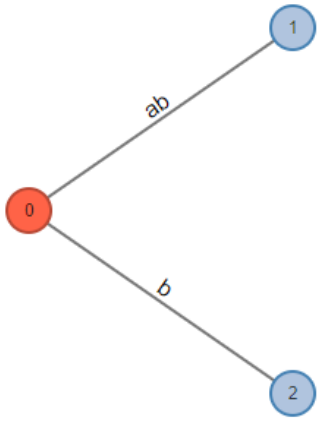

Korak 3:  obrađeno: “ab”

aktivni_čvor: 0

aktivni_brid: none

aktivna_dužina: 0

ostatak: 0



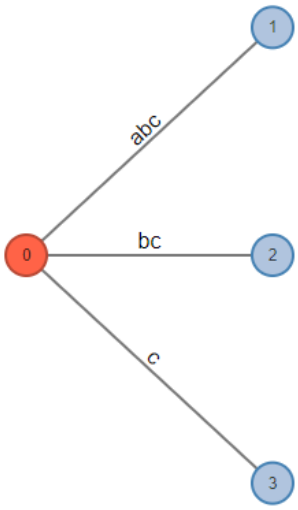

Korak 4:  obrađeno: “abc”

aktivni_čvor: 0

aktivni_brid: none

aktivna_dužina: 0

ostatak: 0



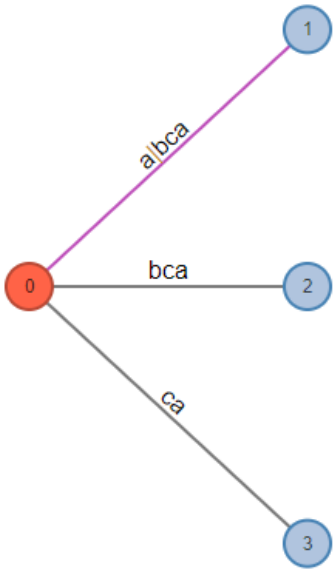

Korak 5: obrađeno: “abca”

aktivni_čvor: 0

aktivni_brid: a

aktivna_dužina: 1

ostatak: 1



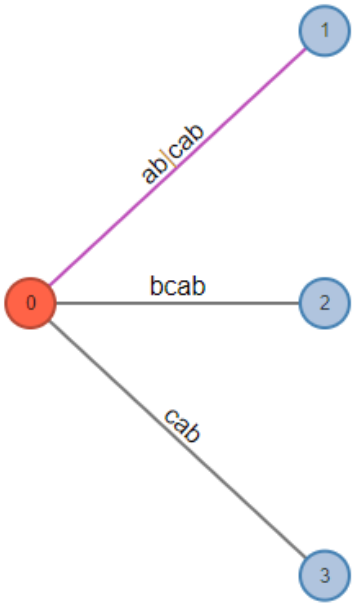

Korak 6:  obrađeno: “abcab”

aktivni_čvor: 0

aktivni_brid: a

aktivna_dužina: 2

ostatak: 2



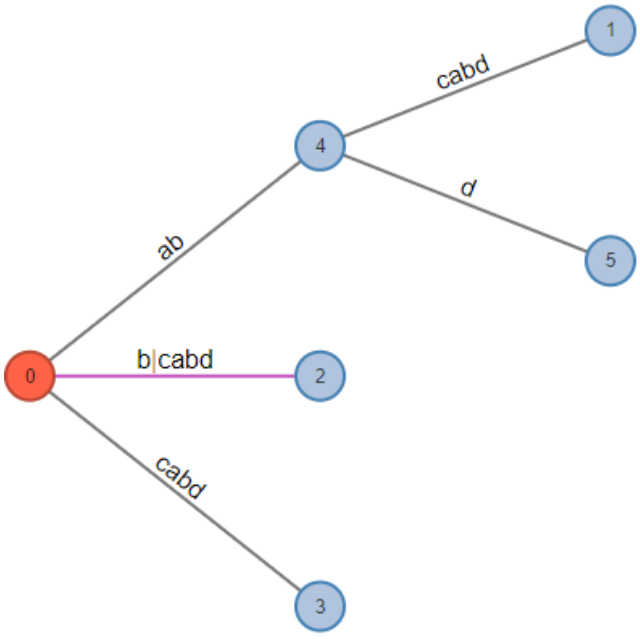
Korak 7:  obrađeno: “abcabd”

aktivni_čvor: 0

aktivni_brid: b

aktivna_dužina: 1

ostatak: 1



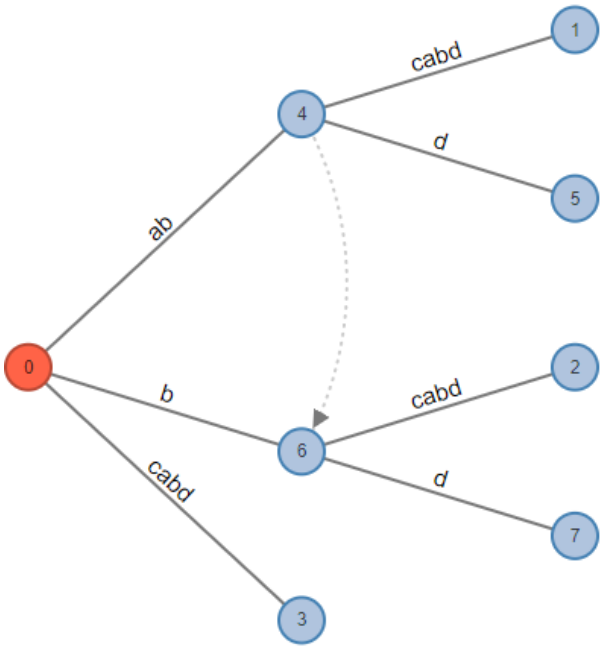

Korak 8:  obrađeno: “abcabd”

aktivni_čvor: 0

aktivni_brid: none

aktivna_dužina: 0

ostatak: 0



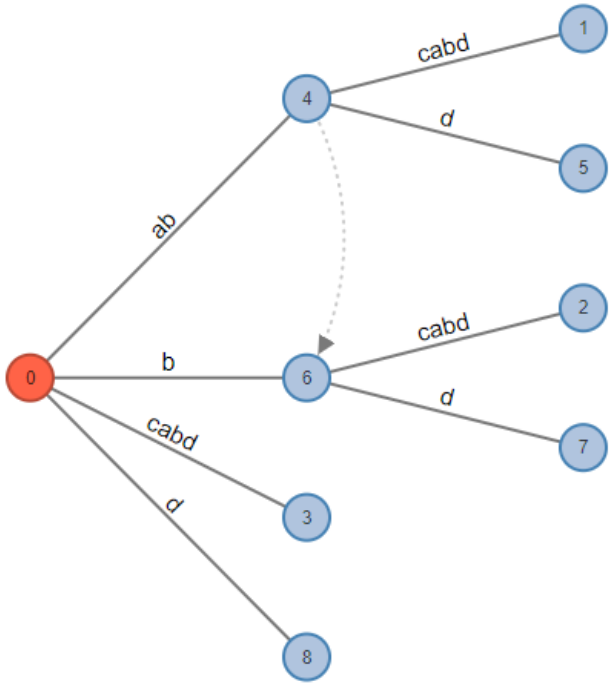

Korak 9:  obrađeno: “abcabd”

aktivni_čvor: 0

aktivni_brid: none

aktivna_dužina: 0

ostatak: 0



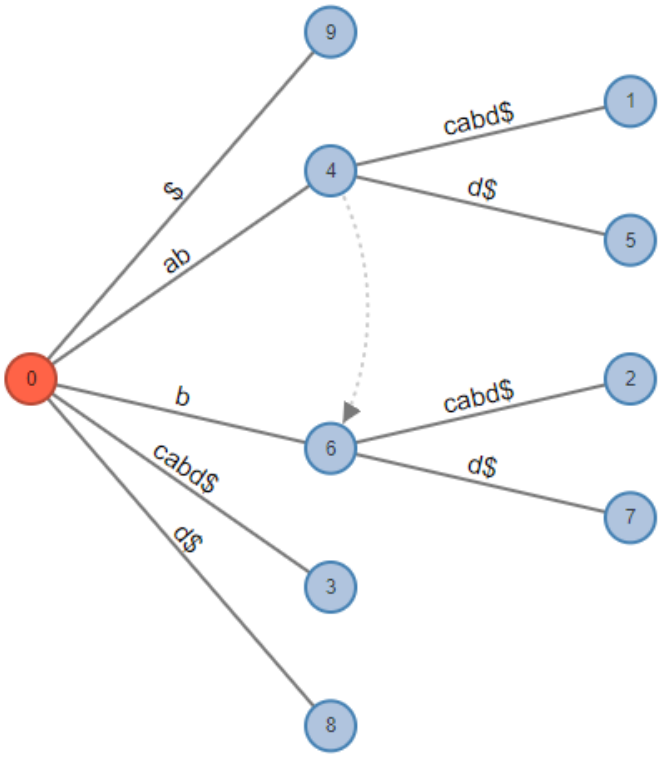

Korak 10:  obrađeno: “abcabd$”

aktivni_čvor: 0

aktivni_brid: none

aktivna_dužina: 0

ostatak: 0

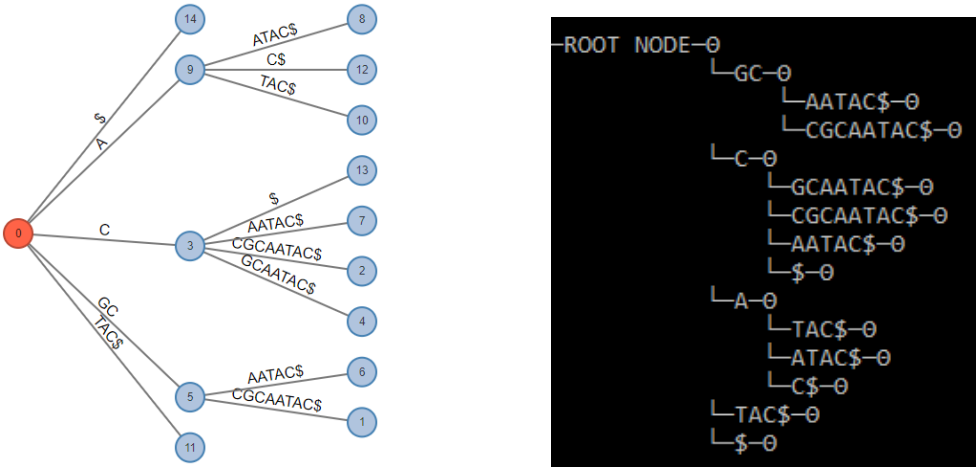



Time je sufiksno stablo dovršeno. Svi sufiksi se eksplicitno nalaze u stablu.

### Primjer generiranog stabla i opis ispisa

U nastavku je prikazan primjer ispisa sufiksnog stabla koristeći naše implementacije i alata korištenog u prijašnjem primjeru. Ulazni niz sastoji se od abecede [G, C, T, A] i duljine je 10 znakova

GCCGCAATAC



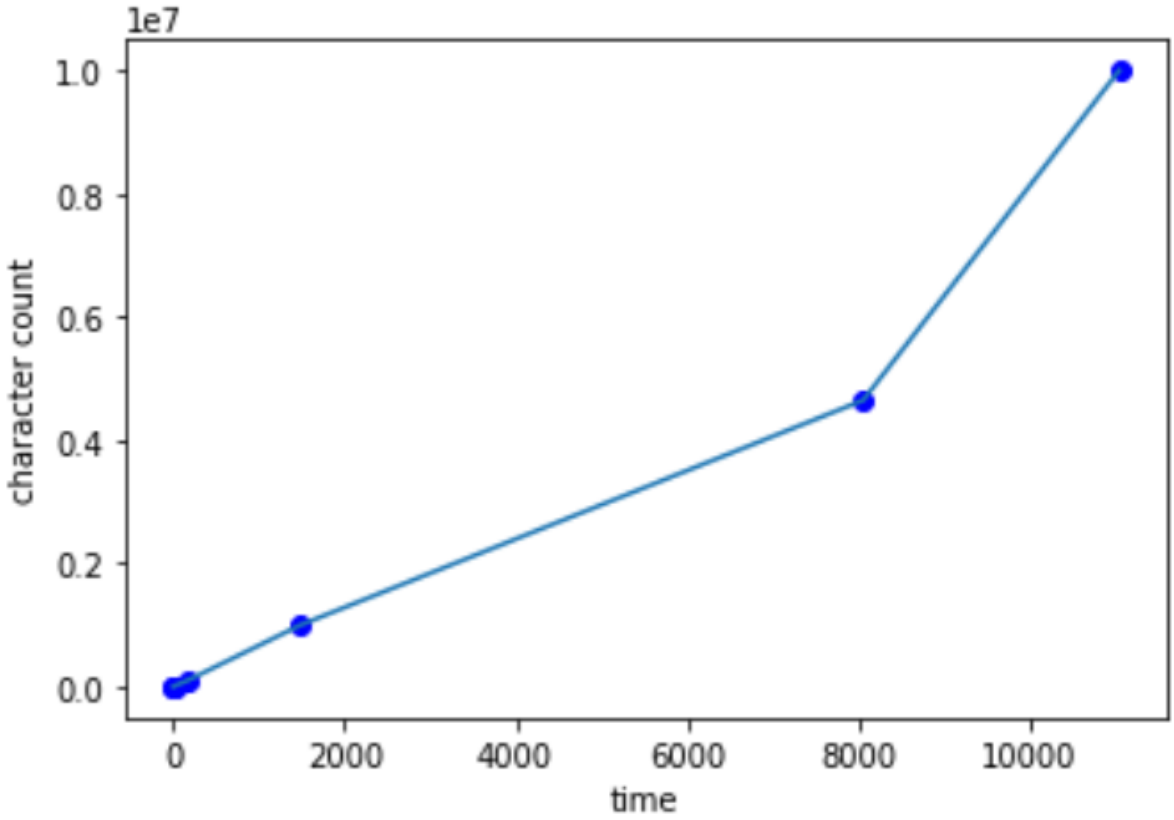

Usporedba ispisa stabla implementiranim algoritmom i online alatom

Algoritam obilazi stablo rekurzivno na način da prilikom pristupa svakom čvoru ispituje jeli taj čvor list ili ne. To je ujedno i uvjet prekida rekurzije. Rekurzivna funkcija kao argument prima čvor, a funkcija prolazi po djeci toga čvora i poziva samu sebe s djetetom kao argument ako to dijete nije list. Dodatan argument rekurzivne funkcije ispisa jest broj praznina koji je implementiran isključivo iz estetskih razloga.

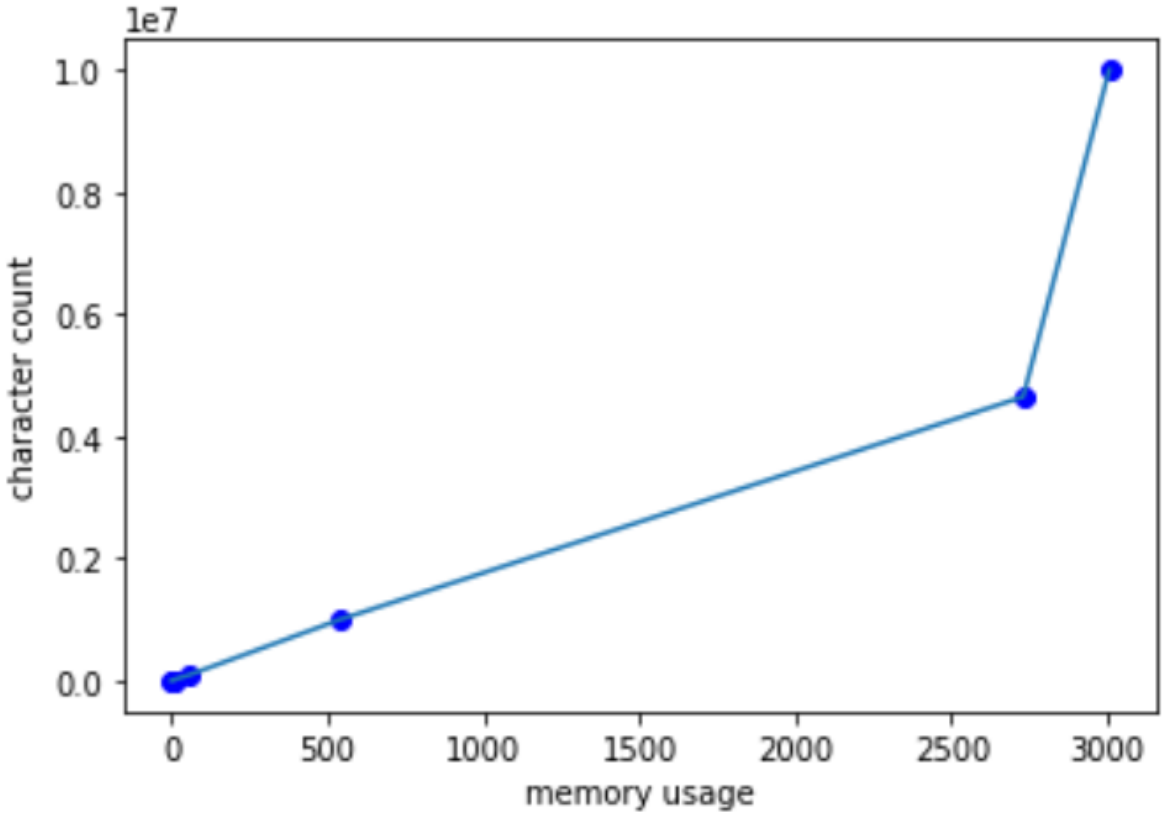
## 3. Rezultati testiranja

Algoritam smo testirali na generiranim tekstovima različite duljine, tj. 100, 1k, 10k, 100k, 1mil, te 10mil znakova. Uz generirane tekstove algoritam je testiran i na E. Coli genomu koji ima duljinu približno 4,4mil znakova. Tekstovi su generirani s abecedom od 4 znaka [A, G, C, T]. Vremenska i memorijska složenost je prikazana na grafovima. Algoritam je testiran na procesoru Intel(R) Core(TM) i7- 4710HQ CPU @ 2.50GHz 2.5GHz s radnom memorijom od 12GB.

| Broj znakova | Prosječno vrijeme [s] | Prosječna memorija [gb] |
|---|---|---|
| 100 | 0.000166357 | / |
| 1000 | 0.000915606 | / |
| 10,000 | 0.0191187 | 0,01 |
| 100,000 | 0.156729 | 0,1 |
| 1,000,000 | 1.45053 | 0,5 |
| 4,570,938 (e. coli) | 8.05753 | 2,7 |
| 10,000,000 | 10.0942 | 3 |




Vremenska složenost




Memorijska složenost

# 4. Zaključak

Testiranjem smo uočili da je i vremenska i memorijska složenost linearna, što je puno bolje od naivne implementacije sufiksnog stabla ($O(n^2)$ / $O(n^3)$) vremenska složenost). Ukkonenov algoritam je također on-line algoritam, u svakom trenutku ima sufiksno stablo spremno od trenutno pročitanog teksta, što može biti korisno ukoliko nemamo potpuni tekst u početku.

## 5. Literatura

1.) https://www.cs.helsinki.fi/u/ukkonen/SuffixT1.pdf

2.) https://en.wikipedia.org/wiki/Ukkonen%27s_algorithm

3.) https://stackoverflow.com/questions/9452701/ukkonens-suffix-tree-algorithm-in-plain-english

4.) https://en.wikipedia.org/wiki/Suffix_tree

5.) https://en.wikipedia.org/wiki/Trie