SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4188

SUFIKSNO STABLO

Tomislav Šebrek

Zagreb, lipanj 2015.

Sadržaj

1. Uvod

Mnogi algoritmi u bioinformatici temelje se na analizi genoma, tj. njegove znakovne reprezentacije, a najčešći upiti koji se nad tim znakovnim nizom postavljaju su traženje preklapanja s nekim drugim, često puno kraćim, znakovnim nizom. Kako su znakovni nizovi koji predstavljaju genom reda veličine od 10^6 do 10^9 znakova, upiti se moraju izvršavati brzo i neovisno o duljini početnog znakovnog niza, a kako bi izvršavanje upita bilo uopće moguće, potrebno je imati strukturu podataka koja će, s jedne strane, što sažetije sadržavati reprezentaciju znakovnog niza, a s druge strane, biti organizirana tako da potrebne upite obavlja u linearnom vremenu.

Kada govorimo o strukturama podataka, postoje dvije strukture podataka koje omogućavaju manipulaciju nad znakovnim nizom, a to su sufiksno polje i sufiksno stablo.

Obje strukture podataka sadrže sve sufikse početnog niza, samo ih pohranjuju na drugačiji način. Sufiksno polje je polje koje sadrži indekse abecedno poredanih sufiksa početnog znakovnog niza, dok u sufiksnom stablu put od korijena do lista predstavlja jedan od sufiksa.

Razlika sufiksnog polja i sufiksnog stabla je u količini memorije potrebne za reprezentaciju znakovnog niza. Sufiksna stabla memorijski su puno zahtjevnija od sufiksnog polja budući da za pohranu istog znakovnog niza traže više informacija. S druge strane, pohranjeni višak informacija omogućava strukturi sufiksnog stabla da neke upite nad znakovnim nizom izvršava brže od sufiksnog polja. Stoga možemo zaključiti da obje strukture imaju svoje prednosti i mane.

U nastavku ovog rada posvetit ćemo se strukturi sufiksnog stabla, njenoj izgradnji u linearnom vremenu i pronalaženju sufiks-prefiks preklapanja između znakovnih nizova kao jednoj od primjena ove strukture.

2. Sufiksno stablo

Sufiksno stablo je struktura podataka koja sadrži sve sufikse određenog znakovnog niza. Glavno svojstvo ove strukture je što omogućava razne manipulacije nad znakovnim nizom, kao što su provjera podudaranja s nekim drugim znakovnim nizom, broj ponavljanja određenog znakovnog niza u nizu nad kojim je stablo izgrađeno i sl. u linearnom vremenu ovisno o duljini podniza. Ovo svojstvo strukture vrlo je moćno jer nam omogućava da nakon što jednom izgradimo stablo, sve upite nad njim provodimo neovisno o duljini početnog niza, koje kod nekih primjera, kao što je reprezentacija genoma, može biti jako veliko. Prije nastavka definirat ćemo neke osnovne pojmove i oznake koje ćemo primjenjivati.

Niz znakova nad kojim ćemo graditi stablo označavat ćemo slovom `S`, broj znakova koji niz `S` sadrži označavat ćemo slovom `n`, a pojedini znak na određenoj poziciji označavat ćemo oznakom `S[i]`, gdje indeksacija započinje nulom.

Definicija. Podniz znakovnog niza `S`, s oznakom `S[i,j]`, je niz znakova koji započinje na i-toj poziciji, a završava uključivo na j-toj poziciji.

Definicija. Prefiks znakovnog niza `S`, s oznakom `S[0,i]`, je podniz znakovnog niza `S`, koji počinje prvim znakom, a završava uključivo na poziciji `i`.

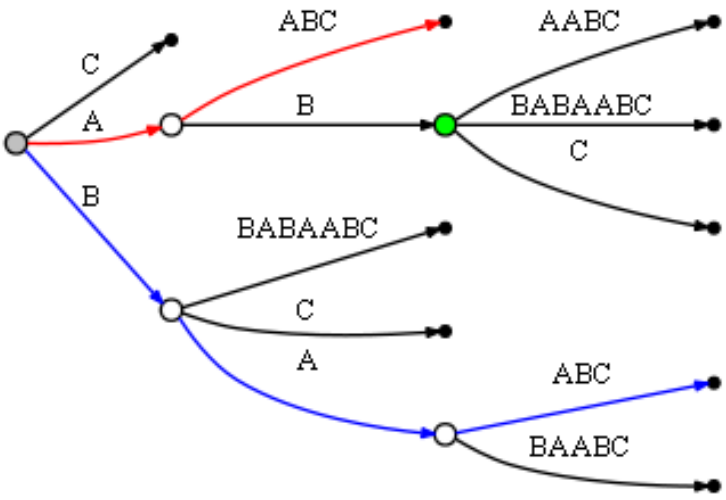
Definicija. Sufiks znakovnog niza `S`, s oznakom `S[i,n-1]`, je podniz znakovnog niza `S`, koji započinje na poziciji i, a završava uključivo zadnjim znakom znakovnog niza `S`.

U nastavku iznosimo formalnu definiciju sufiksnog stabla temeljenu na sljedećoj klasifikaciji čvorova.

Definicija. Korijen je čvor koji nema roditelja, list je čvor koji nema djece, a unutarnji čvor je čvor koji nije niti korijen niti list.

Definicija. Sufiksno stablo je struktura podataka koja je jedinstveno određena korijenom stabla, skupom čvorova i bridovima koji predstavljaju prijelaze, a označeni su podnizovima znakovnog niza nad kojim je stablo izgrađeno.

Na slici 2.1. prikazano je stablo za znakovni niz „ABBABAABC“. Sva stabla, radi preglednosti, bit će orijentirana tako da im krajnje lijevi čvor predstavlja korijen stabla, a samo stablo razvija se s lijeva na desno.



Slika 2.1. Prikaz sufiksnog stabla izgrađenog nad znakovni nizom: „ABBABAABC“. Bridovi označeni plavom bojom predstavljaju sufiks „BAABC“, a bridovi označeni zelenom bojom predstavljaju sufiks „AABC“. Unutarnji čvor predstavljen zelenom bojom ima oznaku puta „AB“.

Definicija. Oznaka puta čvora v je znakovni niz koji se dobije nadovezivanjem oznaka bridova na putu od korijena do čvora v.

Svaki sufiks u sufiksnom stablu određen je putom koji se proteže od korijena stabla, preko određenog broja unutarnjih čvorova, do jednog od listova, tj. svaka oznaka puta nekog lista predstavlja jedan od sufiksa znakovnog niza nad kojim je izgrađeno stablo. Svaki list jedinstveno određuje sufiks znakovnog niza S.

Iz primjera na slici 2.1. možemo zaključiti da je oznaka puta unutarnjeg čvora označenog zelenom bojom „AB“. Put od korijena do lista označen crvenom bojom predstavlja sufiks „AABC“, dok put označen plavom bojom predstavlja sufiks „BAABC“.

Dodatno, možemo zaključiti na sljedeće svojstvo sufiksnog stabla: „Korijen stabla ima onoliko djece koliko ima različitih znakova u znakovnom nizu S, budući da sufiksno stablo sadrži sve sufikse niza, a put kojim je svaki sufiks određen započinje korijenom.

Prilikom izgradnje sufiksnog stabla, na kraj znakovnog niza postavlja se oznaka kraja niza koja ima svojstvo da je leksikografski veća od svih znakova koji se pojavljuju u znakovnom nizu te se po prvi i jedini puta pojavljuje na kraju niza. U nastavku teksta znak kraja niza označavat ćemo znakom „$“. Zadaća znaka kraja niza je osigurati da svaki sufiks bude sadržan u stablu.

Definicija. Implicitno sufiksno stablo je stablo izgrađeno nad znakovnim nizom koji na svome kraju nema oznaku kraja niza.

Sufiksno stablo prikazano na slici 2.1. je implicitno zato što znakovni niz nad kojim je izgrađeno ne sadrži znak „$“ na svome kraju. Ipak, ulogu znaka kraja niza u tom primjeru preuzima znak „C“ zato što zadovoljava svojstva znaka kraja niza budući da se po prvi puta pojavljuje, nalazi se na kraju i abecedno je veći od svih znakova znakovnog niza.

U nastavku teksta opisujemo algoritam izgradnje sufiksnog stabla u linearnoj složenosti. Naime, iako smo rekli da je temeljno svojstvo stabla da se upiti nad njim izvode neovisno o duljini znakovnog niza nad kojim je izgrađen, brzina izgradnje stabla jednako je važan faktor. Primjeri zapisa genoma, nad kojim se često provode upiti, reda su veličine od 10^6 do 10^9 znakova te tako veliki znakovni nizovi ne trpe kvadratnu složenost, u kojoj se sufiksno stablo može izgraditi na trivijalan način.

3. Naivni Ukkonenov algoritam

Ukkonenov algoritam izgradnje sufiksnog stabla temelji se na izgradnji implicitnih sufiksnih stabala za svaki pojedini prefiks znakovnog niza.

Tako je za znakovni niz „ABCD“ potrebno izgraditi stablo za prvi prefiks, a to je „A“. Prethodno stablo nadograđujemo svim sufiksima sljedećeg prefiksa („AB“), a to su sufiksi „AB“ i „A“. U sljedećem koraku stablo nadograđujemo znakovnim nizovima: „ABC“, „BC“ i „C“ koji predstavljaju sve sufikse prefiksa „ABC“. U posljednjem koraku gradimo stablo za sve sufikse zadanog znakovnog niza, a to su redom: „ABCD“, „BCD“, „CD“ i „D“.

Prije nastavka potrebno je otkloniti potencijalnu nedoumicu. Naime, u prvom smo koraku izgradili sufiksno stablo za znak „A“ koji ne predstavlja niti jedan sufiks znakovnog niza „ABCD“. Zapravo, stabla izgrađena u prvih $n-1$ koraka ne predstavljaju konačno stablo, nego su kostur za proširenje sufiksima sljedećeg koraka. Tako znakovni niz „A“, koji je dodan u stablo u prvom koraku, služi kao temelj za proširenje do znakovnog niza „AB“, koji se preko znakovnog niza „ABC“ proširuje do sufiksa „ABCD“, a nalazi se u konačnom sufiksnom stablu. Jednako tako se znak „B“, koji je dodan u 2. koraku, proširuje preko znakovnog niza „BC“ do sufiksa „BCD“.

Općenito vrijedi:

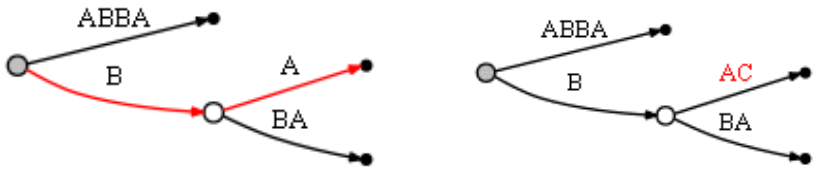
- u i-tom koraku izgradili smo T_i za sve sufikse podniza S[0,i]
- u (i+1)-om koraku proširuje se prethodno izgrađeno stablo T_i za sve sufikse podniza S[0,i+1] te se tako dobiva stablo T_{i+1}
- sufiksi dodani u (i+1)-om koraku nazivaju se proširenja
- svako proširenje može biti ostvareno u 3 slučaja
    - produljivanjem brida (implicitno proširenje)
    - dodavanjem novog brida (eksplicitno proširenje)
    - zanemarivanjem proširenja

Vrste proširenja bit će opisane u nastavku.

3.1. Vrste proširenja

U (i+1)-om koraku cilj svakog j-tog proširenja je osigurati da se podniz S[j, i+1] nalazi u stablu. Naime, kako smo u prethodnom koraku nadogradili stablo za sve sufikse od S[0, i], primijetimo da će podniz S[j, i] uvijek postojati za j-to proširenje, a vrsta proširenja ovisit će o nastavku puta i znaku S[i+1].

Slučaj 1. Ako put S[j, i] završava kao list, brid grane koja vodi do lista proširuje se znakom S[i+1]. Ovakva vrsta proširenja naziva se implicitno proširenje.



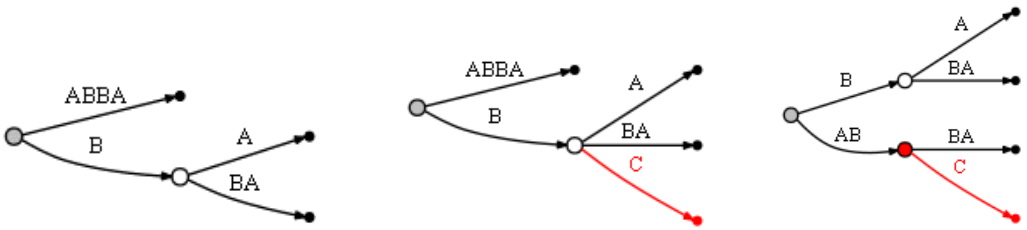
(a) (b)

Slika 3.1. Implicitno proširenje. (a) Stablo prije proširenja. (b) Stablo nakon proširenja znakovnim nizom „BAC“.

Slika 3.1. (a) prikazuje stablo prije proširenja. Ako stablo proširimo znakovnim nizom S[j,i+1] = „BAC“, proširenje će biti implicitno zato što put S[j, i] = „BA“ završava listom (taj put je prikazan crvenom bojom). Implicitno proširenje svodi se na proširenje oznake zadnjeg brida na putu zadnjim znakom znakovnog niza kojim obavljamo proširenje (to je u našem slučaju znak S[i+1] = „C“). Tako oznaka brida „A“ postaje „AC“. Rezultat implicitnog proširenja prikazan je na slici 3.1. (b).

Slučaj 2. Ako se iz puta S[j, i] ne nastavlja niti jedan put koji počinje znakom S[i+1] potrebno je dodati novu granu koja vodi do lista, a označena je znakom S[i+1]. Pri tome, ako put S[j, i] ne završava u nekom od unutarnjih čvorova potrebno je dodati novi unutarnji čvor koji razdvaja put prema listu i nastavak puta. Ovakva vrsta proširenja naziva se eksplicitno proširenje.

Slika 3.2. (a) prikazuje stablo prije proširenja. Ako stablo proširimo znakovnim nizom S[j, i+1] = „BC“, proširenje će biti eksplicitno zato za put S[j, i] = „B“ ne postoji nastavak puta koji počinje završnim znakom znakovnog niza (u ovom slučaju S[i+1] = „C“). Kako put „B“ završava unutarnjim čvorom, eksplicitno proširenje stabla svodi se na dodavanje brida koji povezuje kraj puta s novostvorenim listom, a označen je zadnjim znakom znakovnog niza. Rezultat proširenja prikazan je na slici 3.2. (b).



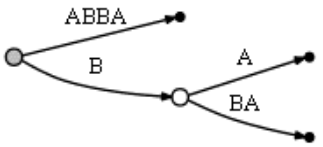
(a) (b) (c)

Slika 3.2. Eksplicitno proširenje. (a) Stablo prije proširenja. (b) Stablo nakon proširenja znakovnim nizom „BC“. (c) Stablo nakon proširenja znakovnim nizom „ABC“.

Ako stablo iz istog primjera proširimo znakovnim nizom S[j, i+1] = „ABC“, proširenje, iako će isto biti eksplicitno, zahtijeva stvaranje novog unutarnjeg čvora budući da put S[j, i] = „AB“ ne završava postojećim unutarnjim čvorom. Novostvoreni unutarnji čvor (označen crvenom bojom na slici 3.2. (c)) predstavlja grananje koje niz „AB“ nastavlja prema sufiksu „ABBA“, koji je postojao u stablu prije proširenja, te prema sufiksu „ABC“, koji će u stablu postojati nakon proširenja. Proširenje se svodi na dodavanje brida koji povezuje novostvoreni unutarnji čvor s novostvorenim listom, a označen je zadnjim znakom znakovnog niza (S[i+1] = „C“). Rezultat proširenja prikazan je na slici 3.2. (c).

Primijetite da se slučajevi eksplicitnog proširenja razlikuju u stvaranju novog unutarnjeg čvora, dok su im stvaranje novog lista te dodavanje novog brida, koji je označen zadnjim znakom znakovnog niza, zajednički.

Slučaj 3. Ako postoji put koji se iz puta S[j, i] nastavlja znakom S[i+1], proširenje treba zanemariti budući da je put S[j, i+1] već sadržan u stablu.



Slika 3.3. Stablo nad kojim se zanemaruje proširenje nizom „BB“.

Na slici 3.3. prikazano je stablo prije proširenja. Ako stablo pokušamo proširiti znakovnim nizom S[j, i+1] = „BB“, prema slučaju br. 3, neće biti promjena u stablu budući da se znakovni niz „BB“ u njemu već nalazi.

3.2. Algoritam i analiza složenosti

U nastavku je prikazan pseudokod osnovne izvedbe Ukkonenovog algoritma:

```
izgradi stablo T0
za i=1 do n-1 činiti
{
   // gradi se stablo Ti+1 na temelju stabla Ti
   za j=0 do i činiti
   {
      // potrebno je osigurati postojanje S[j, i+1] u stablu Ti+1
      ako put S[j,i] završava kao list onda
          implicitno proširenje (slučaj 1)
      inače
      {
         ako je put S[j,i+1] već sadržan onda
            ne radi ništa (slučaj 3)
         inače
            eksplicitno proširenje (slučaj 2)
      }
   }
}
```

Algoritam 3.1. Naivni Ukkonenov algoritam

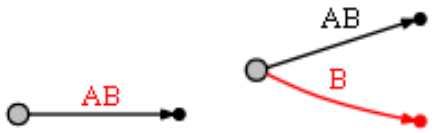
Prema gore opisanom algoritmu provjera slučaja za podniz S[j,i] izvršit će se $\frac{n\cdot(n+1)}{2}$ puta. Kako se provjera slučaja izvodi u složenosti $O(m)$ gdje je m duljina podniza, a u najgorem slučaju $O(n)$, budući da provjeru slučaja za svaki podniz provodimo počevši od korijena stabla. Zato je složenost naivne verzije algoritma $O(n^3)$.

3.3. Primjer izgradnje stabla

Pogledajmo sada primjer izgradnje stabla za znakovni niz „ABBA$“.

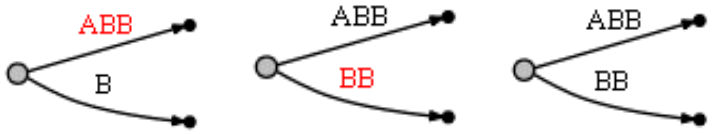
Slika 3.4. 1. korak izgradnje sufiksnog stabla za znakovni niz „ABBA$“

U prvom koraku gradimo stablo za znak „A“. Stablo na slici 3.4. predstavlja stablo T_0 iz algoritma.



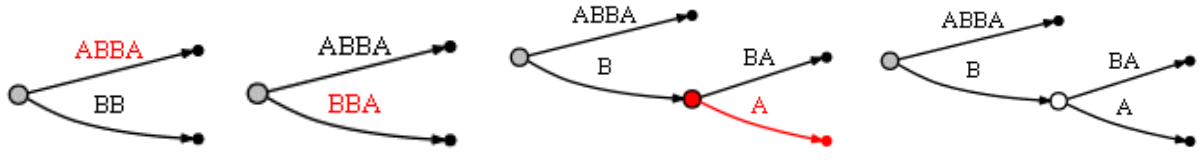
Slika 3.5. 2. korak izgradnje sufiksnog stabla za znakovni niz „ABBA$“

U drugom koraku algoritma moramo osigurati postojanje sljedećih znakovnih nizova: „AB“ i „B“ proširujući stablo T_0 izgrađeno u prethodnom koraku. Stablo T_0 nizom „AB“ proširit ćemo implicitno, a zatim eksplicitno nizom znakova „B“. Rezultat ovih proširenja prikazan je na slici 3.5.



Slika 3.6. 3. korak izgradnje sufiksnog stabla za znakovni niz „ABBA$“

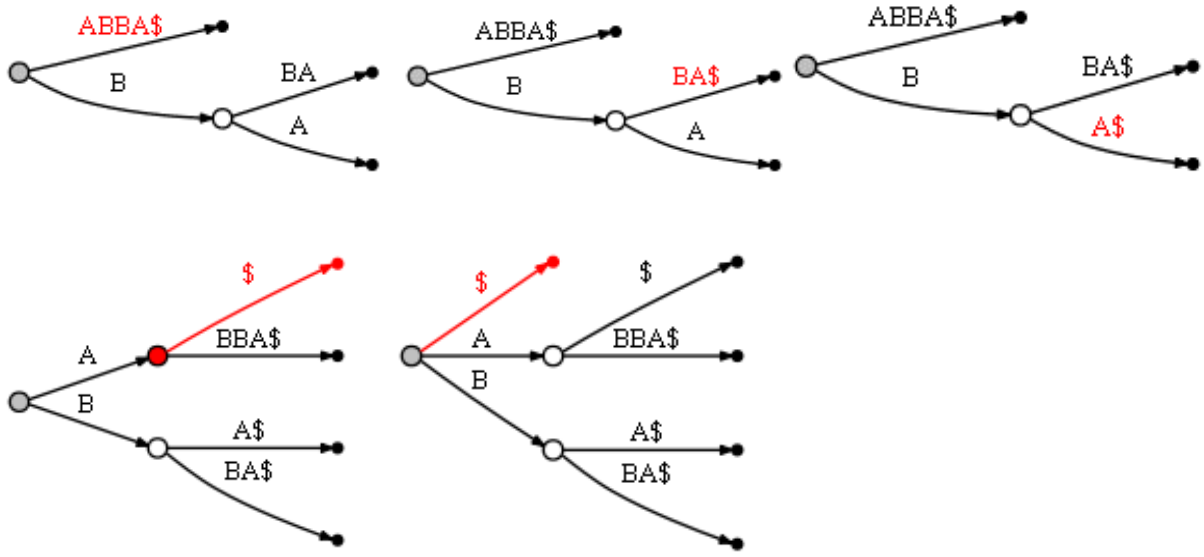
U trećem koraku stablo T_1, koje je izgrađeno u prethodnom koraku, proširujemo znakovnim nizovima „ABB“, „BB“ i „B“. Za znakovne nizove „ABB“ i „BB“ primjenjujemo implicitno proširenje, dok za „B“ ne poduzimamo nikakvu akciju zato što je taj znakovni niz sadržan u stablu. Izgradnja stabla T_2 prikazana je na slici 3.6.



Slika 3.7. 4. korak izgradnje sufiksnog stabla za znakovni niz „ABBA$“

U četvrtom koraku proširujemo stablo svim sufiksima prefiksa „ABBA“, a to su redom: „ABBA“, „BBA“, „BA“ i „A“. Prva dva sufiksa proširuju stablo T_2 implicitno, treći ga

proširuje eksplicitno dok se za zadnji sufiks ne poduzima nikakva akcija budući da je sam već sadržan u stablu nakon svih prethodnih proširenja. Zaustavimo se na tren i primijetimo važnost znaka kraja niza ('$'). Bez njega bi 4. korak bio ujedno i posljednji korak algoritma te bi konačno stablo izgledalo kao na slici 3.7. koje ne sadrži jedan sufiks – upravo onaj za koji nismo poduzeli nikakvu akciju u trenutnom koraku. Ovaj problem upravo rješava znak kraja niza ('$') koji se prvi puta pojavljuje i leksikografski je veći od svih znakova u stablu pa tako u sljedećem koraku svi sufiksi proširuju stablo eksplicitno ili implicitno.



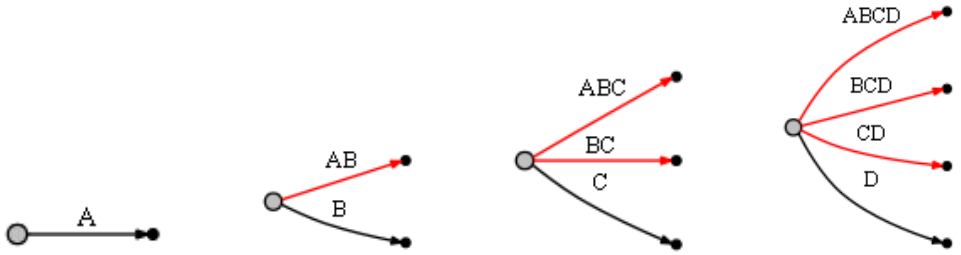
Slika 3.8. 5. korak izgradnje sufiksnog stabla za znakovni niz „ABBA$“

U zadnjem koraku stablo proširujemo svim sufiksima znakovnog niza „ABBA$“, a to su redom: „ABBA$“, „BBA$“, „BA$“, „A$“ i „$“. Primijetite da u zadnjem koraku nije moguće primijeniti slučaj br. 3 zato što niz koji na svom kraju sadrži znak kraja niza nikako ne može biti prethodno sadržan u stablu. Zbog toga nakon zadnjeg proširenja, a to je proširenje samim znakom kraja niza „$“, svi sufiksi bit će sadržani u stablu. Konačno sufiksno stablo prikazano je na slici 3.8.

4. Linearni Ukkonenov algoritam

4.1. Eliminacija provjere implicitnog proširenja

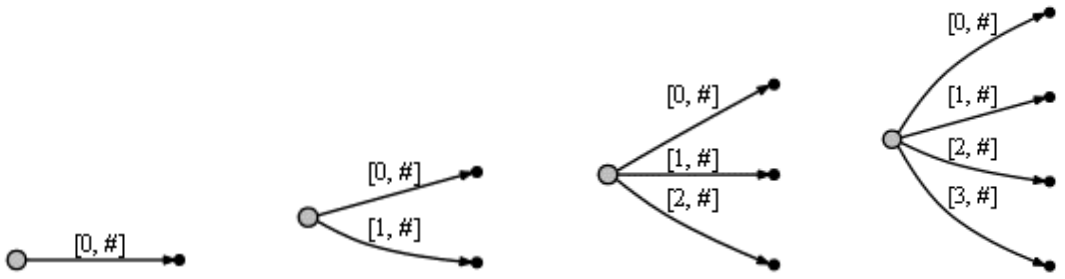
Do prvog poboljšanja algoritma doći ćemo intuitivno, preko primjera izgradnje stabla za znakovni niz „ABCD“. Koraci izgradnje algoritma prikazani su na slici 4.1.



Slika 4.1. Koraci izgradnje sufiksnog stabla za znakovni niz „ABCD“

Primijetimo da će na kraju i-tog koraka algoritma, svaki put završavati znakom S[i], a kako svaki kraj puta završava listom, slijedi da će na kraju i-tog koraka svaka oznaka brida koji završava listom, završavati znakom S[i]. Iz slike 4.1. možemo uočiti kako se dodavanje sufiksa: S[0, i], S[1, i], ..., S[i-1, i] zapravo svodi na implicitno proširenje svih putova stabla T_{i-1}, kreiranog u prethodnom koraku. Ovo opažanje može se sažeto definirati pravilom: „jednom list, uvijek list“, tj. grana koja završava u listu uvijek će završavati u listu.

Bridove stabla označavat ćemo pomoću dvije kazaljke na sljedeći način: `[od, do]`. Kazaljka `od` pokazuje na početni znak kojim je označen brid, dok kazaljka `do` pokazuje na završni znak. Dodatno, kod bridova koji završavaju listom, kazaljka `do` ne pokazuje na fiksnu poziciju u znakovnom nizu, nego na znak S[i] te predstavlja kraj niza u trenutnom koraku.



(i) *(ii)* *(iii)* *(iv)*

Slika 4.2. Koraci izgradnje sufiksnog stabla za znakovni niz „ABCD“ uz označavanje bridova pomoću kazaljki

Na slici 4.2. (i) prikazan je prvi korak izgradnje algoritma za `i=0`. Brid je označen oznakom [0, #], što predstavlja S[0, 0], tj. znak „A“ budući da je brid označen u prvom koraku (`i=0`), a oznaka trenutnog kraja (#) jednaka je 0. U sljedećem koraku brid je označen istom oznakom [0, #], ali ona sada predstavlja drugi kontekst. Naime, budući da se radi o koraku za `i=1`, u ovom slučaju brid označava podniz S[0, 1], a predstavlja znakovni niz „AB“. U trećem koraku brid je predstavljen istom oznakom, ali uz značenje S[0, 2] i predstavlja znakovni niz „ABC“, dok u završnom koraku ista oznaka definira raspon znakova S[0, 3] i predstavlja sufiks „ABCD“.

Činjenica. Predstavljanje bridova koji završavaju listom kazaljkom trenutnog kraja, koja je ovisna o kontekstu algoritma, tj. koraku u kojem obavljamo proširenja, omogućuje nam da sva implicitna proširenja cijelog algoritma izvedemo u složenosti $O(1)$.

4.2. Redukcija na kvadratnu složenost

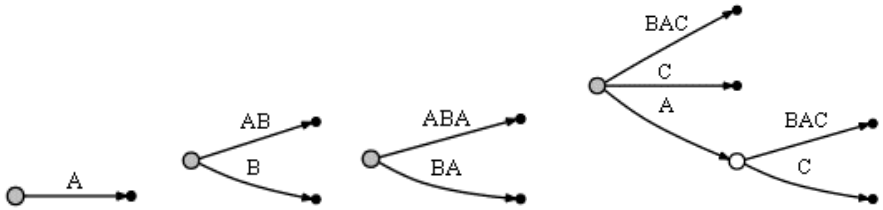
Eliminacijom implicitnog proširenja napravili smo velik korak prema reduciranju složenosti, budući da sva implicitna proširenja obavljamo u $O(1)$, a pseudokod algoritma možemo zamisliti ovako:

```
za i=0 do n-1 činiti
{
   // t – pozicija koja određuje sufiks S[t, i] koji nije sadržan u stablu
         nakon implicitnog proširenja prethodno izgrađenog stabla Ti-1
   za j=t do i činiti
   {
      ako S[j, i] nije sadržan u stablu onda
            dodati sufiks S[j, i]
   }
}
```

Algoritam 4.1. Kvadratni Ukkonenov algoritam

Postavlja se pitanje kako odrediti poziciju `t`, tj. kako odrediti sufikse koji nisu obuhvaćeni implicitnim proširenjem stabla izgrađenog u prethodnom koraku. Osvrnimo se još jednom na primjer pomoću kojeg smo zaključili na eliminaciju provjere implicitnog proširenja, a to je izgradnja stabla za znakovni niz „ABCD“ prikazan na slici 4.2. Za taj primjer vrijedi `t=i`, tj. jedini sufiks koji je potrebno dodati

u stablo u svakom koraku je S[i] budući da su sufiksi S[0, i], ..., S[i-1, i] obuhvaćeni implicitnim proširenjem stabla T_{i-1}, izgrađenom u prethodnom koraku, što je na slici 4.1. prikazano crvenom bojom, no to neće biti općeniti slučaj. Pogledajmo primjer izgradnje stabla za znakovni niz „ABAC“.



Slika 4.3. Koraci izgradnje sufiksnog stabla za znakovni niz „ABAC“.

U prvom koraku izgradili smo stablo za znak „A“, a u sljedećem koraku dodali znak „B“. Primijetite da se „A“ grana budući da vodi do lista implicitno proširuje u složenosti $O(1)$. U trećem koraku obje su grane (tj. prethodno izgrađeno stablo) implicitno proširene, ali u tom koraku nismo dodali novi sufiks (budući da se sufiks „A“ već nalazi u stablu). Zadnjem koraku algoritma treba posvetiti posebnu dodatnu pažnju. Prije dodavanja novih sufiksa stablo se ponovno implicitno proširuje te stablo sadrži dvije grane „ABAC“ i „BAC“ (taj korak nije prikazan na slici 4.3.) te nam preostaje dodati sufikse: „AC“ i „C“, tj `t=i-1`. Dakle, možemo zaključiti:

Činjenica. U i-tom koraku potrebno je dodati onoliko sufiksa koliko ih je u prethodnom koraku preskočeno, tj. već se nalazilo u stablu prema 3. slučaju.

U implementaciji za pamćenje broja sufiksa koje treba dodati, koristit ćemo varijablu `podsjetnik`, koju ćemo uvećati za 1 u trenutku kada započnemo novi korak, budući da moramo dodati novi sufiks za znak S[i], a umanjivati za 1 svaki puta kada dodamo novi sufiks. Tako u svakom koraku znamo koliko sufiksa nije obuhvaćeno implicitnim proširenjem prethodnog stabla, tj. koliko je novih sufiksa potrebno dodati.

Sada kada znamo odrediti koje sufikse je potrebno dodati u određenom koraku, spremni smo za sljedeće poboljšanje koje će reducirati složenost algoritma na $O(n^2)$. Naime, ako u nekom koraku vrijedi da se niz „ABC“ nalazi u stablu, tada se i nizovi „BC“ i „C“ također nalaze u stablu budući da su sufiksi niza „ABC“. Možemo poopćiti,

Činjenica. Ako u je i-tom koraku sufiks S[j,i] sadržan u stablu, tada su svi sufiksi S[j+1,i], ..., S[i,i] također sadržani u stablu.

Algoritam možemo modificirati ovako:

```
podsjetnik = 0
za i=0 do n-1 činiti
{
   podsjetnik++
   za j=i-podsjetnik+1 do i činiti
   {
      ako S[j, i] nije sadržan u stablu onda {
            dodati sufiks S[j, i]
            podsjetnik--
      }
      inače
         prekini petlju
   }
}
```

Algoritam 4.2. Kvadratni Ukkonenov algoritam.

4.3. Aktivna pozicija

Zamislimo situaciju u kojoj gradimo sufiksno stablo za niz „ABCDABCE“ te da smo izgradili podstablo za niz „ABCD“. U sljedećem koraku bit će potrebno provjeriti nalazi li se „A“ u prethodno izgrađenom stablu. Jednako tako, istu provjeru morat ćemo provoditi nad znakovnim nizovima „AB“ i „ABC“. Ako za svaki znakovni niz: „A“, „AB“ i „ABC“ provjeru započinjemo od korijena, drastično ćemo usporiti algoritam. Kako bi provjeru obavili u $O(1)$ koristimo sljedeću činjenicu:

Činjenica. Provjera 3. slučaja za podniz S[j, i] svodi se na provjeru podudaranja znaka S[i] i sljedećeg znaka nastavka puta S[j, i-1] koji se nalazi u stablu.

Kako bi ostvarili provjeru 3. slučaja u složenosti $O(1)$ stanje zadnje preskočenog sufiksa bit će potrebno održavati u 3 varijable:

`aktivni_cvor` – predstavlja posljednji čvor na kojem završava put S[i,j]

`aktivni_brid` – predstavlja granu u kojoj završava put S[i,j] za aktivni čvor

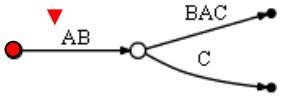
`aktivna_duljina` – predstavlja udaljenost kraja puta S[i,j] od aktivnog čvora

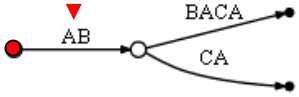
Provjera 3. slučaja svodi se na provjeru podudaranja znaka S[i] sa znakom `oznaka[aktivna_duljina]`, gdje je `oznaka`, znakovni niz kojim je označen aktivni brid.

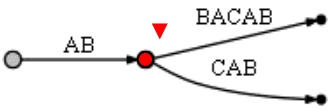
Definicija. Aktivna pozicija, nastavak je puta zadnjeg sufiksa za kojeg je ustanovljeno da se nalazi u stablu, a označena je trojkom kontrolnih varijabli i označava se (`aktivni_cvor`, `aktnivni_brid`, `aktivna_duljina`).

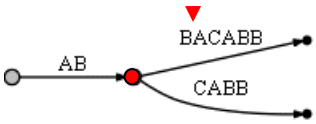
Na slici 4.4. prikazani su koraci proširenja stabla prilikom nadogradnje znakovnim nizom „ABBD“. Na svakom stablu crvenom je bojom označen aktivni čvor prije dodavanja sufiksa, dok je crvenom strelicom označena aktivna pozicija na kojoj moramo provjeriti zadnji znak.

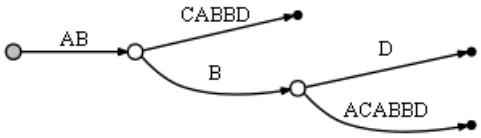
SUFIKS	STANJE STABLA	STANJE VARIJABLI

„A“		`aktivni_cvor = korijen`
		`aktivni_brid = ∅`
		`aktivna_duljina = 0`

„AB“		`aktivni_cvor = korijen`
		`aktivni_brid = 'A'`
		`aktivna_duljina = 1`

„ABB“		`aktivni_cvor = interni`
		`aktivni_brid = ∅`
		`aktivna_duljina = 0`

„ABBD“		`aktivni_cvor = interni`
		`aktivni_brid = 'B'`
		`aktivna_duljina = 1`

„BDD“		?

Slika 4.4. Koraci nadogradnje grane znakovnim nizom „ABBD“. Svaki redak predstavlja jedan od koraka izgradnje, a svaki stupac prikazuje sufiks koji se dodaje, stablo i stanje kontrolnih varijabli prije dodavanja sufiksa.

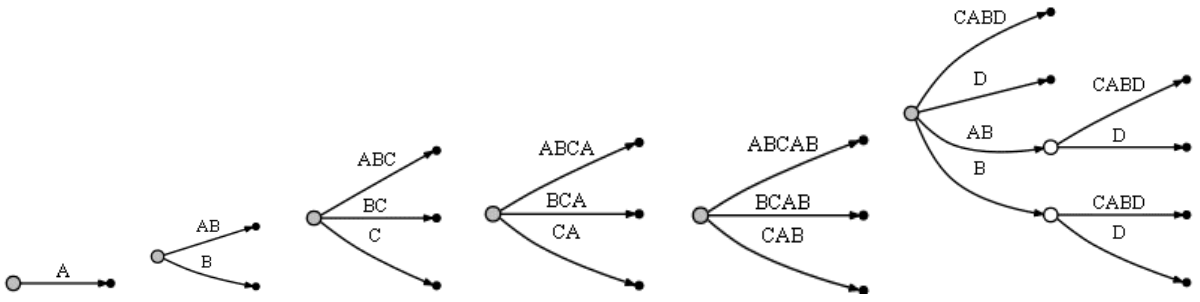
U prvom koraku moramo provjeriti postoji li znak „A“ u stablu, tj. na aktivnoj poziciji određenoj pomoću tri varijable. Naime, kako je `aktivna_duljina=0`, to znači da moramo provjeriti podudaranje prvog znaka oznake svih bridovima koji nastavljaju aktivni čvor. Budući da postoji grana koja počinje slovom „A“ `aktivni_brid` postavljamo na znak „A“, a varijablu `aktivna_duljina` na 1.

U sljedećem koraku dodajemo sufiks „AB“, a provjera se svodi na provjeru poklapanja znaka „B“ s aktivnom pozicijom, budući da ona označava kraj puta sufiksa nad kojim je primijenjen slučaj br. 3 u prethodnom koraku. Kako se na aktivnoj poziciji nalazi znak „B“ u složenosti $O(1)$ zaključujemo da se sufiks „AB“ nalazi u stablu. Sada nam preostaje osvježiti strukturu aktivne pozicije. `aktivni_čvor` postaje sljedeći unutarnji čvor na putu zato što aktivna duljina postaje jednaka broju znakova kojima je označen brid, `aktivna_duljina` postaje 0, dok `aktivni_brid` postavljamo na $\emptyset$ budući da se nastavak može ostvariti kroz bilo koju granu u sljedećem koraku. Primijetite da se stablo prije svakog koraka implicitno proširuje znakom S[i].

Postavlja se pitanje kako osvježiti strukturu nakon dodavanja novog brida. Upravo to opisuju 3 pravila Ukkonenovog algoritma, dok prvo pravilo glasi.

Pravilo 1. Nakon dodavanja čvora, ako je `aktivni_cvor` korijen stabla, `aktivni_cvor` ostaje korijen stabla, `aktivni_brid` postaje prvi znak novog sufiksa kojeg je potrebno dodati, a `aktivna_duljina` umanjuje se za 1.

Kako bi suštinski shvatili bit ovog pravila pogledajmo primjer izgradnje za niz „ABCABD“. Koraci izgradnje stabla prikazani su na slici 4.5.



Slika 4.5. Koraci izgradnje sufiksnog stabla za znakovni niz „ABCABD“.

Nakon zaključivanja kako niz „AB“ pripada stablu (za `i=4`), varijable su postavljene na sljedeće vrijednosti:

```
aktivni_brid = korijen stabla
aktivni_brid = 'A'
aktivna_duljina = 2
podsjetnik = 3
```

Naime, sljedeći sufiks koji ćemo pokušati provjeravati bit će „ABD“, a ako se nalazi u stablu, onda se nalazi u bridu korijena stabla koji počinje znakom A i nalazi se nakon 2 znaka u oznaci brida. Budući da se na toj poziciji ne nalazi znak „D“ (nego znak „C“) primjenjujemo eksplicitno proširenje, a strukturu osvježavamo koristeći 1. pravilo algoritma.

```
aktivni_brid = korijen stabla
aktivni_brid = 'B'
aktivna_duljina = 1
podsjetnik = 2
```

Sada možemo detaljnije pojasniti 1. pravilo algoritma. Naime, u prethodnom koraku pokušali smo dodati sufiks „AB“ i zaključili da je već sadržan u stablu, a time istu stvar možemo zaključiti i za niz „B“. U sljedećem koraku smo zaključili da se niz „ABD“ ne nalazi u stablu te smo obavili eksplicitno proširenje, a istu provjeru moramo učiniti i za sljedeći sufiks „BD“. Kako se niz „B“ već nalazi u stablu (što znamo iz prethodnog koraka), ostaje samo pitanje u kojem dijelu stabla završava put „B“. Počinje bridom čija oznaka započinje znakom „B“ (zato aktivni brid postavljamo na prvi znak novog sufiksa kojeg je potrebno dodati.) i od korijena je udaljen za `aktivna_duljina-1` budući da je sufiks za jedan znak kraći od prethodnog sufiksa koje je u stablo eksplicitno dodan (u našem slučaju to je bio sufiks „ABD“).

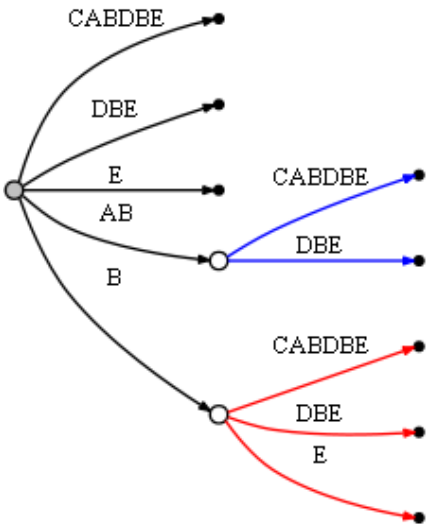
Ostaje nam još komentirati ograničenje da `aktivni_cvor` mora biti korijen stabla. Da bi odgovorili na to pitanje zamislimo aktivnu poziciju određenu unutarnjim čvorom koji nije korijen stabla te da smo dodali sufiks xα. Sljedeći sufiks koji je potrebno dodati je α na poziciju koja je za 1 znak kraća nego pozicija sufiksa xα. Problem je što `aktivna_duljina-1` ne pokazuje ukupnu duljinu od korijena, nego duljinu od trenutnog aktivnog čvora kojim put α ne prolazi. Kako ne bi za svaki novi sufiks pretraživanje započeli od korijena uvodimo posljednje poboljšanje – sufiksne veze.

4.4. Sufiksne veze

Redukciju s $O(n^2)$ na $O(n)$ temeljit ćemo na sljedećoj činjenici:

Činjenica. Svi sufiksi podstabla koji se nastavljaju na čvor s oznakom puta xα, sadržani su u podstablu koje se nastavlja na čvor s oznakom puta α.

Naime, neka se u skupu β_1 nalaze svi sufiksi koji započinju u podstablu s oznakom puta xα. Svi sufiksi koji započinju znakovima xα stoga su $x\alpha\beta_1$. Sada, kako je α sufiks znakovnog niza xα, isti skup sufiksa β_1 mora biti sadržan u skupu β_2, u kojem se nalaze svi znakovni nizovi koji započinju u podstablu s oznakom puta α.



Slika 4.6. Prikaz stabla za znakovni niz „ABCABDBE“. Podstablo označeno plavom bojom podskup je podstabla koje je označeno crvenom bojom

Pogledajmo primjer stabla na slici 4.6. U ovom slučaju podstablo β_1 (na slici 4.6. prikazano plavom bojom), koje nastavlja znakovni niz xα = „AB“ određeno je skupom {„CABDBE“, „DBE“}. Podstablo β_2 (na slici 4.6. prikazano crvenom bojom), koje nastavlja put označen sa α = „B“ (na slici 4.6. prikazano plavom bojom) sadrži skup {„CABDBE“, „DBE“, „E“}. Primijetimo da je skup β1 podskup skupa β2.

Ako bi u stablo bilo potrebno dodati sljedeći sufiks: „ABCF“, stvorili bi novu granu u podstablu označenom plavom bojom. Prema algoritmu, sljedeći sufiks koji je potrebno dodati je „BCF“, a dodavanje se svodi na dodavanje nove grane u crveno podstablo na istoj poziciji kao i u plavom podstablu prema zadnje navedenoj činjenici.

Iako znamo točnu poziciju na koju trebamo dodati novu granu, ne znamo poziciju unutarnjeg čvora kojem crveno podstablo započinje te pretragu ponovno moramo započeti od korijena stabla.

Kako pretragu ne bi morali provoditi od korijena definirat ćemo novi pojam.

Definicija. Sufiksna veza čvora v s oznakom puta xα je pokazivač na čvor s(v) s oznakom puta α.

Sufiksne veze prikazivat ćemo iscrtkanom strelicom.

Činjenica. Ako je u j-tom proširenju i-tog koraka dodan novi unutarnji čvor s oznakom puta xα, a α nije Ø, u (j+1)-om proširenju bit će ostvaren jedan od dva slučaja:

i. novi čvor s oznakom puta α bit će kreiran u (j+1)-om proširenju
ii. čvor s oznakom puta α već postoji u stablu

Na temelju ove činjenice možemo formirati posljednja dva pravila algoritma.

Pravilo 2.

(i) Ako nakon dodavanja nove grane u stablo vrijedi da to nije prvo dodavanje u trenutnom koraku, prethodna dva čvora nad kojima je dodana nova grana potrebno je povezati sufiksnom vezom.

(ii) Ako je u trenutnom proširenju sufiks sadržan u sufiksnom stablu, posljednji čvor nad kojim je dodana nova grana povezujemo s aktivnim čvorom koji sadrži sufiks u trenutnom proširenju.

Posljedica primjene pravila br. 2 je da će svaki unutarnji čvor imati sufiksnu vezu prema nekom drugom čvoru, a pokazivat će na čvor kojim započinje drugo podstablo u koje je potrebno dodati novu granu. Na ovaj način ne moramo pretragu svaki puta započinjati ispočetka, nego je dovoljno slijediti sufiksne veze kreirane u ranijim koracima. Napomenimo još da ako unutarnji čvor nema sufiksnu vezu, podrazumijeva se sufiksna veza s korijenom stabla.

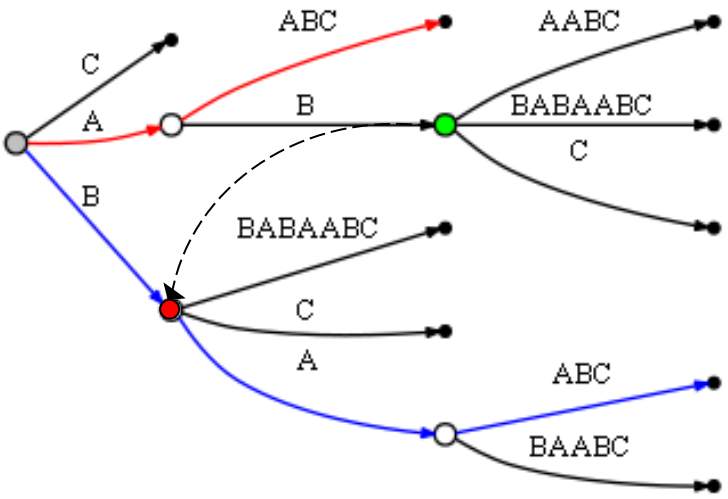
Ako smo dosljedno primjenjivali 2. pravilo, svaki unutarnji čvor sufiksnom vezom pokazuje na drugi čvor koji sadrži prošireni skup znakovnih nizova u podstablu kojem

je on korijen. Kako sufiksna veza pokazuje poziciju aktivnog čvora za dodavanje sljedećeg sufiksa možemo se osloboditi ograničenja na vrijednost aktivnog čvora prvog pravila i definirati zadnje pravilo algoritma.

Pravilo 3. Nakon dodavanja nove grane na čvor koji nije korijen, aktivnu poziciju modificiramo tako da aktivni čvor postavimo na čvor na koji pokazuje sufiksna veza, dok aktivnu duljinu i aktivni brid ne mijenjamo.

Pravilom 3 zapravo samo slijedimo sufiksnu vezu nad unutarnjim čvorom koja nam predstavlja početak podstabla za kojeg znamo mjesto na kojem trebamo dodati novi čvor.

Zamislimo situaciju u kojoj dodajemo sufiks „ABAAD“. Prije obrade sufiksa, aktivna pozicija bila bi (n1, 'A', 2), gdje je n1 unutarnji čvor označen zelenom bojom na slici 4.7. Sljedeći sufiks koji je potrebno dodati je „BAAD“, a prema 3. pravilu i definiciji sufiksne veze (koja je na slici prikazana iscrtkanom strelicom od čvora n1 prema čvoru n2) nalazi se na poziciji (n2, 'A', 2), gdje je n2 unutarnji čvor označen crvenom bojom na slici 4.7. Sada nailazimo na problem. Naime, duljina brida manja je od aktivne duljine što znači da se aktivna pozicija ne nalazi na aktivnom bridu nego u podstablu prema kojem aktivni brid vodi. Ovaj problem naziva se „problem propagacije aktivne pozicije“.



Slika 4.7. Sufiksno stablo izgrađeno nad znakovni nizom: „ABBABAABC“

Činjenica. Ako je duljina aktivnog brida manja ili jednaka vrijednosti aktivne duljine, aktivnu poziciju modificiramo tako da aktivni čvor postavljamo na čvor prema kojem aktivni brid vodi, aktivni brid na znak kojim reducirani sufiks započinje, a aktivnu duljinu umanjujemo za duljinu brida.

Primjenom gore navedene činjenice problem propagacije riješili bi tako da bi aktivnu poziciju postavili na (n3, 'A', 1).

Postavlja se pitanje utječe li problem propagacije na složenost algoritma budući da je broj potencijalno mogućih propagacija aktivne pozicije jednak n. Odgovor je ne. Naime, ako će uistinu biti potrebno modificirati aktivnu poziciju, na novoj će aktivnoj poziciji biti kreiran novi unutarnji čvor do kojeg će voditi sufiksna veza iz prethodno kreiranog unutarnjeg čvora. To znači da će za sve buduće sufikse, koje će biti potrebno dodati preko čvora koji zahtjeva propagaciju aktivne pozicije, kreirana sufiksna veza pokazivati na čvor koji je već propagiran tako da se sve propagacije odvijaju amortizirano ovisno o duljini niza.

Pogledajmo sada konačnu verziju Ukkonenovog algoritma.

```
podsjetnik = 0;
aktivna_pozicija = (korijen, Ø, 0)

za i = 0 do n-1 činiti {
   podsjetnik++;
   dok je podsjetnik > 0 činiti {
      ako je aktivna_duljina = 0 onda
         aktivni_brid = Ø

      ako aktivni_cvor ne sadrži brid koji započinje s aktivni_brid onda {
         dodaj brid s oznakom(i, #)
         ako je potrebno dodati sufiksnu vezu onda
            dodaj sufiksnu vezu     // pravilo 2 i)
      }
      inače {
         ako je potrebno propagirati aktivnu poziciju onda {
            propagiraj aktivnu poziciju
            vrati se na početak dok je petlje
         }
         ako je znak sadrzan na aktivnoj poziciji onda {
            aktivna_duljina++
            ako je potrebno dodati sufiksnu vezu onda
               dodaj sufiksnu vezu        // pravilo 2 ii)
            prekini izvođenje dok je petlje
         }

         stvori novi čvor i dodaj novi brid s oznakom (i, #)
         ako je potrebno dodati sufiksnu vezu onda
            dodaj sufiksnu vezu
      }
      podsjetnik--
      ako je akticni_cvor = korijen onda
         osvjezi aktivnu poziciju prema 1. pravilu    // pravilo 1
      inace
         osvjezi aktivnu poziciju prema 3. pravilu    // pravilo 3
   }
}
```

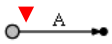
Algoritam 4.3. Linearni Ukkonenov algoritam.

5. Primjer izgradnje linearnim Ukkonenovim algoritmom

U nastavku ćemo prikazati postupak izgradnje stabla za znakovni niz „ABBBABA$“, kojim su obuhvaćeni svi slučajevi algoritma. Nakon opisa svakog koraka bit će prikazano stanje stabla T_i na kraju koraka, kao i stanja kontrolnih varijabli koje određuju aktivnu poziciju na kraju svakog koraka (aktivna pozicija na slikama će biti prikazana crvenim trokutom).

korak (`i=0, `**`A`**`BBBABA$`)

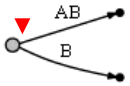
U prvom koraku izgradili smo stablo za znak „A“, a na kraju ovog koraka varijabla `podsjetnik` bit će postavljena na 0 budući da smo obradili sve sufikse trenutnog koraka, a uvećat će se za 1 na početku sljedećeg koraka.




```
aktivni_cvor = korijen
aktivni_brid = Ø
aktivna_duljina = 0
podsjetnik = 0
```

korak (`i=1, A`**`B`**`BBABA$`)

U ovom koraku potrebno je dodati samo sufiks „B“ budući da je varijabla `podsjetnik` uvećana na vrijednost 1 kao prva operacija trenutnog koraka, a stablo je prije obrade sufiksa implicitno prošireno znakom S[i] = „B“. Kako se na aktivnu poziciju ne nastavlja znak S[i] = „B“, u stablo iz aktivnog čvora, koji je korijen stabla, dodajemo brid s oznakom S[i] = „B“. Varijablu `podsjetnik` postavljamo na 0.

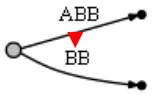



```
aktivni_cvor = korijen
aktivni_brid = Ø
aktivna_duljina = 0
podsjetnik = 0
```

korak (`i=2, AB`**`B`**`BABA$`)

Prije obrade znaka S[i] = „B“, stablo će biti implicitno prošireno upravo tim znakom te će sadržavati putove s oznakama „ABB“ i „BB“. Kako se znak „B“

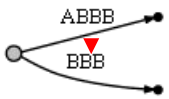
nalazi na nastavku aktivne pozicije, ne poduzima se nikakva akcija, a aktivna pozicija postavlja se na (korijen, 'B', 1). Prisjetimo se još jednom, aktivni brid nam govori na kojem bridu se nalazi aktivna pozicija, a aktivna duljina koliko je znakova udaljena od aktivnog čvora. Varijabla `podsjetnik` je na kraju koraka postavljena na 1 budući da jedan sufiks (sufiks „B“) nije dodan u stablo.




```
aktivni_cvor = korijen
aktivni_brid = B
aktivna_duljina = 1
podsjetnik = 1
```

korak (`i=3, ABBBABA$`)

Prvom operacijom 3. koraka `podsjetnik` uvećavamo na vrijednost 2, što znači da će u trenutnom koraku biti potrebno dodati sufikse S[i-1, i] = „BB“ i S[i, i] = „B“. Prije obrade znaka S[i] = „B“, prethodno stablo ponovno će biti implicitno prošireno tako da zapravo sadrži dvije grane: „ABBB“ i „BBB“. Kako se znak „B“ nalazi na nastavku aktivne pozicije, mijenjamo ju na vrijednost (korijen, 'B', 2), `podsjetnik` uvećavamo na 2, budući da nismo dodali nijedan sufiks i završavamo s korakom.

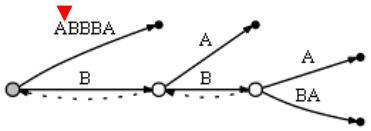



```
aktivni_cvor = korijen
aktivni_brid = B
aktivna_duljina = 2
podsjetnik = 2
```

korak (`i=4, ABBBBABA$`)

U ovom koraku potrebno je obraditi sufiks „BBA“ budući da je uvećanjem varijable `podsjetnik`, ista postavljena na vrijednost 3. Prije obrade znaka S[i] = „A“, koji uz kontrolne varijable predstavlja sufiks „BBA“, prethodno stablo ponovno će biti implicitno prošireno tako da zapravo sadrži dvije grane: „ABBBA“ i „BBBA“. Obrada sufiksa „BBA“ započinje provjerom znaka S[i] = „A“

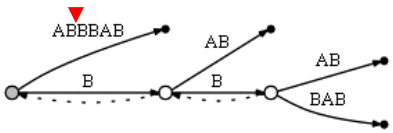
na aktivnoj poziciji. Kako znak S[i] = „A“ ne nastavlja aktivnu poziciju, na njoj stvaramo novi unutarnji čvor te nad njim dodajemo novi brid označen s S[i] = „A“. Ovime smo dodali sufiks „BBA“ u stablo, a kako je sufiks dodan u trenutku kada je aktivni čvor postavljen na korijen stabla, aktivnu poziciju modificiramo na (korijen, 'B', 1) prema prvom pravilu Ukkonenovog algoritma, budući da je sljedeći sufiks koji je potrebno dodati „BA“. Kako znak S[i] = „A“ ne nastavlja ni novomodificiranu aktivnu poziciju dodajemo novi unutarnji čvor na mjesto aktivne pozicije i novi brid označen znakom S[i] = „A“. Nakon ovog eksplicitnog proširenja ponovno primjenjujemo prvo pravilo i modificiramo aktivnu poziciju na (korijen, 'A', 1). Prema drugom pravilu potrebno je dodati sufiksnu vezu između zadnja dva čvora nad kojima je dodan novi brid. Kako znak „A“ nastavlja novu aktivnu poziciju ne poduzimamo dodatnu akciju, a na kraju koraka postavljamo `podsjetnik` na vrijednost 1 budući da toliko sufiksa nismo obradili u trenutnom koraku.




```
aktivni_cvor = korijen
aktivni_brid = A
aktivna_duljina = 1
podsjetnik = 1
```

korak (`i=5, ABBBABA$`)

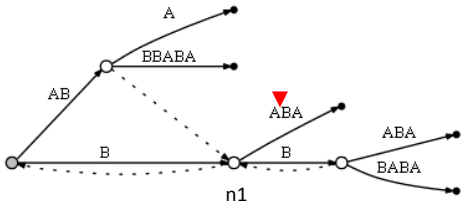
U ovom koraku trebamo obraditi sufiks „AB“ budući da je varijabla `podsjetnik` nakon uvećanja postavljena na vrijednost 2. Znak S[i] = „B“ nastavlja aktivnu poziciju te ga ne dodajemo u stablo, nego modificiramo aktivnu poziciju na (korijen, 'A', 2) i povećavamo vrijednost varijable `podsjetnik` budući da sufiks „AB“ nije obrađen.




```
aktivni_cvor = korijen
aktivni_brid = A
aktivna_duljina = 2
podsjetnik = 2
```

korak (`i=6, ABBBAB`**`A`**`$`)

U ovom koraku trebamo obraditi sufiks „ABA“ budući da varijabla `podsjetnik` postaje 3. Kako znak S[i] = „A“ nije sadržan u nastavku aktivne pozicije dodajemo novi čvor, smanjujemo varijablu `podsjetnik` i osvježavamo aktivnu poziciju na (korijen, 'B', 1) budući da je sljedeći sufiks koji trebamo dodati „BA“, a počinje znakom „B“. Kako se nova aktivna pozicija nalazi nakon zadnjeg znaka oznake aktivnog brida potrebno je propagirati aktivnu poziciju niz stablo, a aktivnu duljinu smanjiti za duljinu oznake brida. Tako aktivna duljina postaje 0, a aktivna pozicija (n1, Ø, 0). Kako se znak S[i] = „A“ nalazi u nastavku aktivne pozicije, modificiramo je na (n1, 'A', 1). Prije završetka koraka potrebno je zadnji čvor nad kojim je dodan novi brid povezati sufiksnom vezom s aktivnom čvorom, prema drugom dijelu drugog pravila Ukkonenovog algoritma. Varijabla `podsjetnik` na kraju je postavljena na 2.

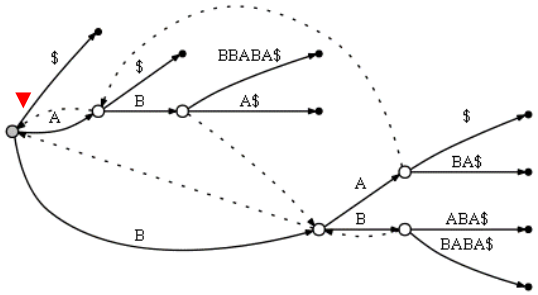



```
aktivni_cvor = n1
aktivni_brid = A
aktivna_duljina = 1
podsjetnik = 2
```

korak (`i=7, ABBBABA`**`$`**)

U ovom koraku trebamo obraditi sufiks „BA$“ zato što varijabla `podsjetnik` postaje 3. Znak S[i] = „$“ ne nastavlja aktivnu poziciju te stvaramo novi čvor na aktivnoj poziciji i nad njim dodajemo brid s oznakom S[i] = „$“. Kako aktivni čvor nije korijen stabla primjenjujemo 3. pravilo Ukkonenovog algoritma te aktivnu poziciju modificiramo tako da aktivni čvor postavimo na čvor na koji pokazuje sufiksna veza. Prisjetimo se da ako unutarnji čvor nema eksplicitno postavljenu sufiksnu vezu, podrazumijeva se sufiksna veza prema korijenu. Nova aktivna pozicija određena je trojkom (korijen, 'A', 1). Na toj poziciji stvaramo novi čvor te aktivnu poziciju modificiramo na (korijen, Ø, 0) te budući da smo u trenutnom koraku dodali više od jednog novog brida prema drugom pravilu Ukkonenovog algoritma, prethodna dva čvora koja smo eksplicitno

proširili povezujemo sufiksnom vezom. Nakon dodavanja novog brida nad korijen ponovno primjenjujemo drugo pravilo.



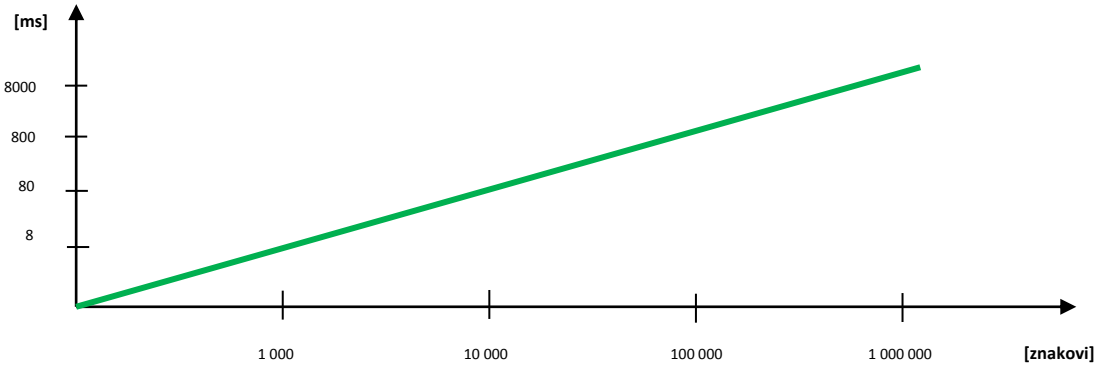

```
aktivni_cvor = korijen
aktivni_brid = Ø
aktivna_duljina = 0
podsjetnik = 0
```

6. Rezultati i analiza složenosti

Implementaciju Ukkonenovog algoritma testirat ćemo na primjerima od 1kB do 1MB, a rezultati testiranja prikazani su u tablici 6.1. i na slici 6.1. Programsko rješenje ostvareno je u programskom jeziku C++.

Tablica 6.1. Rezultati testiranja brzine izgradnje sufiksnog stabla

Veličina niza (broj znakova)	Vrijeme (milisekunde)
1000	10
10000	85
100000	825
1000000	8520



Slika 6.1. Graf prikazuje ovisnost brzine izgradnje sufiksnog stabla u milisekundama prema duljini znakova nad kojima se stablo gradi.

Sada kada smo se uvjerili kako algoritam gradi stablo u linearnom vremenu potrebno je to i formalno pokazati.

Pogledajmo algoritam 4.2. Obje petlje se izvode u amortiziranoj linearnoj složenosti ovisno o duljini znakovnog niza. Naime, kako smo sva implicitna proširenja sveli na složenost $O(1)$, kao i provjeru podudaranja sufiksa s trenutno izrađenim stablom (slučaj br. 3), broj iteracija koje će amortizirano izvesti ugniježđene petlje algoritma 4.2. jednak je duljini znakovnog niza, budući da je upravo toliko sufiksa potrebno

eksplicitno dodati u stablo. Kako aktivna pozicija označava mjesto na koje moramo postaviti sljedeći sufiks, a sufiksna veza poziciju čvora na koji se aktivna pozicija odnosi za sljedeći sufiks koji je potrebno dodati, svako eksplicitno proširenje izvodimo u složenosti $O(1)$, što cijeli algoritam zbog n iteracija dovodi do složenosti $O(n)$. Prisjetimo se još kako problem propagacije aktivne pozicije ne narušava linearnu složenost zato što se odvija amortizirano, u ovisnosti o duljini znakovnog niza.

Brzinu izgradnje usporedit ćemo s implementacijom SAIS algoritma[1]. Napomenimo kako je SAIS algoritam implementiran u programskom jeziku Java.

Tablica 6.2. Usporedba brzine izgradnje sufiksnog stabla Ukkonenovim algoritmom sa SAIS algoritmom za izgradnju sufiksnog polja

Veličina niza (broj znakova)	**Vrijeme – sufiksno stablo (milisekunde)**	**Vrijeme – sufiksno polje (milisekunde)**
1000	10	8
10000	85	30
100000	825	350
1000000	8520	3350

Iz tablice 6.2. možemo uočiti kako je izgradnja sufiksnog polja SAIS algoritmom brža od Ukkonenovog algoritma za izgradnju sufiksnog stabla. Glavni razlog je taj što SAIS algoritam stvara „samo“ sufiksno polje, tj. polje leksikografski sortiranih sufiksa, dok Ukkonenov algoritam stvara strukturu stabla pogodniju za izvršavanje budućih upita nad strukturom.

Drugi razlog implementacijske je prirode. Čvor stabla Ukkonenovog algoritma predstavljen je hash-mapom kako bi algoritam efikasnije izvršavao upite nad sufiksnim stablom.

[1] SAIS je algoritam za izgradnju sufiksnog polja u linearnom vremenu.

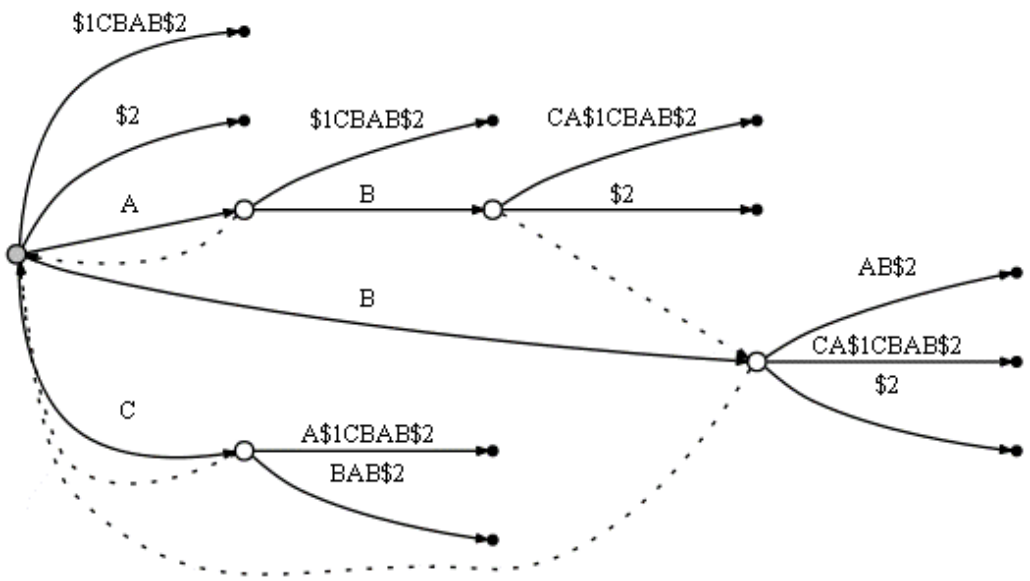
7. Primjene sufiksnog stabla

Kao što je već spomenuto u uvodu, primjena sufiksnog stabla ima jako puno, a u ovom radu posebnu pažnju ćemo posvetiti pronalaženju sufiks-prefiks preklapanja između n znakovnih nizova. Kako bi ova pretraga bila moguća moramo poopćiti Ukkonenov algoritam na izgradnju sufiksnog stabla za n znakovnih nizova.

7.1. Poopćeno sufiksno stablo

Definicija. Sufiksno stablo izgrađeno za znakovni niz $S_1\$_1...S_n\$_n$ nazivamo poopćeno sufiksno stablo gdje su S_i znakovni nizovi, a $\$_i$ znakovni kraja niza pri čemu vrijedi ako je j > i onda $\$_j > \$_i$.

Na slici 7.1. prikazano je poopćeno sufiksno stablo za znakovne nizove „ABCA“ i „CBAB“.



Slika 7.1. Poopćeno sufiksno stablo izgrađeno nad znakovni nizom: „ABCA$$_1$CBAB$$_2$“

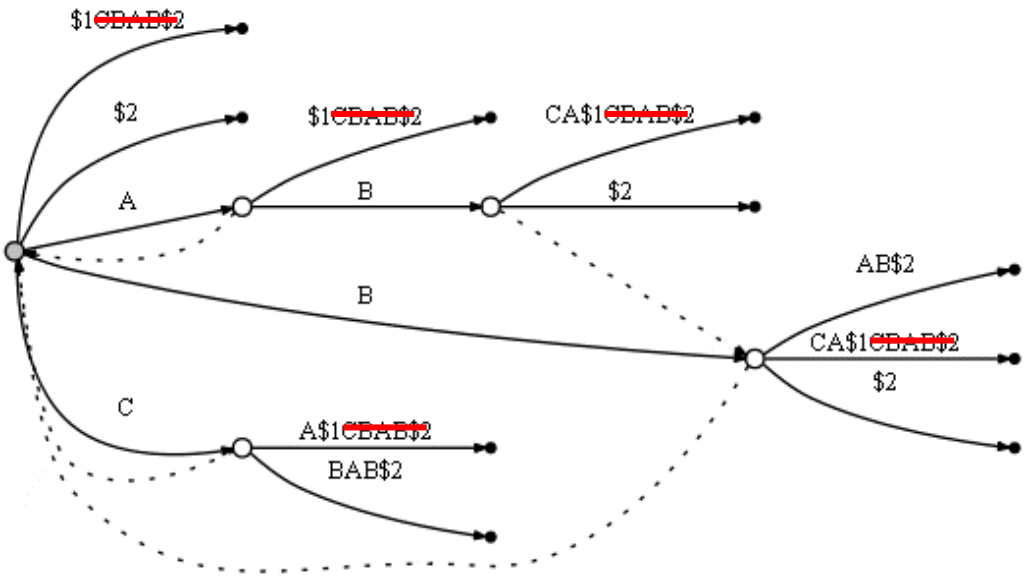
Problem koji se javlja kod ovog pristupa je da sufiksi prvog znakovnog niza ne završavaju listom, nego se nastavljaju svim znakovima drugog znakovnog niza. Tako sufiks „CA$$_1$“ prvog znakovnog niza ne završava listom, nego je dio puta „CA$$_1$CBAB$$_2$“ koji završava listom. Iako je to u skladu s definicijom sufiksnog stabla budući da CA$$_1$ nije sufiks originalnog niza nad kojim je napravljeno stablo

(„ABCA$$_1$CBAB$$_2$“) cilj nam je modificirati algoritam tako da svi sufiksi, bilo kojeg niza, budu prikazani u stablu. Modifikaciju temeljimo na sljedećoj činjenici.

Činjenica. Nakon što je grana stabla implicitno proširena znakom $$_i$, nad njom se neće provoditi eksplicitno proširenje u nastavku algoritma, nego će biti implicitno proširivana svim znakovima koji slijede u znakovnom nizu.

Gore navedena činjenica slijedi iz definicije znaka kraja niza. Da bi grana koja sadrži znak kraja niza $$_i$ bila eksplicitno proširena, nastavak niza trebao bi sadržavati znak $$_i$. Kako se $$_i$ pojavljuje samo jednom i označava kraj i-tog znakovnog niza, zaključujemo da nad granom koja sadrži znak kraja niza $i ne obavljaju eksplicitna proširenja. Sada možemo zaključiti na sljedeće:

Činjenica. Ako je grana označena s više znakova $$_i$, sufiks oznake nakon prvog znaka kraja niza ($$_i$) se odbacuje.



Slika 7.2. Poopćeno sufiksno stablo za nizom: „ABCA$1CBAB$2“ uz reduciranje oznaka.

Na slici 7.2. prikazano je reduciranje oznake znakovnog niza nakon znaka kraja niza. Postupak je ispravan budući da su znakovni nizovi koji su izbačeni iz oznake, zapravo sufiksi početnog znakovnog niza koji se nalaze u stablu.

Ovim postupkom znakovni niz sufiks CA$$_1$ koji je sufiks prvog znakovnog niza, završava u listu.

7.2. Sufiks-prefiks preklapanje

Definicija. Sufiks-prefiks preklapanje dva niza je znakovni niz za koji vrijedi da je ujedno sufiks jednog niza i prefiks drugog znakovnog niza.

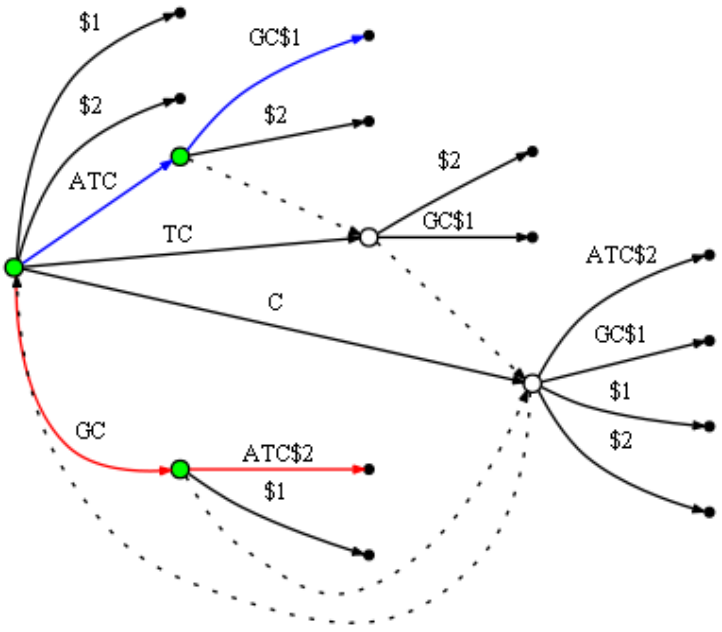
Tako je za znakovne nizove „ATCGC“ i „GCATC“ maksimalno sufiks-prefiks preklapanje jednako znakovnom nizu „ATC“ budući da je ono prefiks prvog znakovnog niza i sufiks drugog. Znakovni niz „GC“ je također sufiks-prefiks preklapanje, ali nije najveće.

Algoritam pronalaženja temeljit ćemo na svojstvu terminalnosti unutarnjeg čvora.

Definicija. Unutarnji čvor je terminalan ako sadrži granu koja je označena isključivo znakom kraja niza $\$_i$.

Prema definiciji slijedi da je oznaka puta terminalnog čvora sufiks koji je jednak oznaci puta od korijena stabla. Ako se radi o putu koji predstavlja jedan od znakovnih nizova S_i nad kojima je izgrađeno stablo, onda upravo oznaka tog puta predstavlja sufiks-prefiks preklapanje.

Pogledajmo primjer poopćenog sufiksnog stabla za znakovne nizove „ATCGC“ i „GCATC“ prikazanog na slici 7.3.



Slika 7.3. *Poopćeno sufiksno stablo izgrađeno nad znakovni nizovima „ATCGC“ čiji je put označen plavom bojom i „GCATC“ čiji je put označen crvenom bojom. Terminalni čvorovi označeni su zelenom bojom.*

Na slici 7.3. crvenom i plavom bojom prikazani su putovi za znakovne nizove nad kojima je stablo izrađeno. Kako samo terminalni čvorovi na tim putevima predstavljaju sufiks-prefiks preklapanje potrebno je kretati se od listova prema korijenu i ispisati oznake puta svakog terminalnog čvora. To su sufiks-prefiks preklapanja: „ATC“, za put označen plavom bojom i „GC“ za put označen crvenom bojom.

Algoritam za pronalaženje svih sufiks-prefiks preklapanja prikazan je u nastavku.

```
preklapanja = []
za svaki list koji predstavlja cijeli znakovni niz činiti {
      lista_cvorova = odredi čvorove na putu od lista do korijena
      za svaki čvor u lista_cvorova činiti {
            ako je čvor terminalan onda
                  dodaj oznaku puta čvora u preklapanja
      }
}
```

Algoritam 7.1. Algoritam za pronalaženje sufiks-prefiks preklapanja.

Preostaje nam još komentirati složenost algoritma. Kako za svaki terminalni čvor na putu od lista do korijena stabla (kojih u najgorem slučaju može biti n) računamo oznaku puta, ponovno obrađujući n čvorova u najgorem slučaju, pronalaženje svih preklapanja za pojedini list izvodi se u kvadratnoj složenosti.

Složenost možemo reducirati na linearnu, ako primijenimo dinamičko programiranje tako da održavamo oznaku puta preko sljedeće relacije:

$$oznaka\ puta(roditelj) = oznaka\ puta(dijete)\ /\ oznaka\ brida(roditelj, dijete)$$

Kako sada sva sufiks-prefiks preklapanja pojedinog lista dobivamo u linearnoj složenosti, a algoritam provodimo nad listovima koji predstavljaju znakovne nizove nad kojima je stablo izgrađeno, konačna složenost je: $O(m \cdot n)$, gdje je m broj znakovnih nizovima nad kojima je izgrađeno stablo.

8. Zaključak

Linearnost izgradnje i izvršavanja upita nad sufiksnim stablom, dva su temeljna svojstva sufiksnog stabla zbog kojih upravo ta struktura nalazi brojne primjene u problemima analize velikih znakovnih nizova, od analize teksta do analize reprezentacije genoma u bioinformatici.

Ukkonenov algoritam, kao jedan od algoritama izgradnje stabla u linearnom vremenu, temelji se na izgradnji implicitnih sufiksnih stabala za svaki prefiks znakovnog niza. Algoritam se kroz niz poboljšanja i definiranja pravila na kojima se temelji realizacija, svodi do linearne složenosti. Dodatno, algoritam se može poopćiti za skup znakovnih nizova.

Algoritam izgradnje, koncipiran kroz nekoliko pravila, jednostavan je za implementaciju, a struktura stabla nam omogućuje da nad njom primjenjujemo algoritme teorije grafova i na intuitivan način razvijamo rješenja za brojne zahtjeve nad znakovnim nizom.

U ovom radu opisana je implementacija izgradnje poopćenog sufiksnog stabla te pronalazak svih sufiks-prefiks preklapanja nad znakovnim nizovima preko kojih je stablo izgrađeno. U sklopu rada razvijena je biblioteka koja podržava izgradnju sufiksnog stabla, pretraživanje stabla od korijena prema listovima i u suprotnom smjeru, provjeru podudaranja s nekim drugim znakovnim nizom te pronalaženje svih sufiks-prefiks preklapanja.

Programsko rješenje testirano je na znakovnim nizovima različitih veličina te je bilježena ovisnost brzine izgradnje i obavljanja upita nad stablom o veličini ulaznog niza, te je ustanovljeno da se rezultati testiranja složenosti programskog rješenja podudaraju s teorijskom analizom složenosti.

Literatura

[1] Gusfield, D. Algorithms on string, trees and sequences. University of Cambridge, 1997.

[2] http://stackoverflow.com/questions/9452701/ukkonens-suffix-tree-algorithm-in-plain-english

[3] Ukkonen, E. On-line construction of suffix trees, University of Helsinki

[4] Šikić M., Domazet-Lošo M. Bioinformatika: Sufiksno stablo i sufiksno polje. Fakultet elektrotehnike i računarstva, Zagreb, 2007.

Sufiksno stablo

Sažetak

Sufiksno stablo je struktura podataka koja omogućuje razne manipulacije nad znakovnim nizom u linearnom vremenu ovisnom samo o duljini podniza nad kojima se ostvaruje upit, a moguće ga je izgraditi u linearnom vremenu. U ovom radu opisana je izgradnja stabla Ukkonenovim algoritmom koji se temelji na izgradnji implicitnih sufiksnih stabala za svaki prefiks znakovnog niza kroz implicitna i eksplicitna proširenja, a svoju linearnost postiže kroz eliminaciju implicitnih proširenja, uvođena strukture aktivne pozicije te sufiksnim vezama.

U ovom radu opisana je implementacija Ukkonenovog algoritma za izgradnju poopćenog sufiksnog stabla, što je zapravo generalizacija algoritma za skup znakovnih nizova. Kao primjena poopćenog stabla, prikazan je implementacija pronalaska svih sufiks-prefiks preklapanja između znakovnih nizova nad kojima je stablo izgrađeno. Rezultati testiranja složenosti programskog rješenja nad znakovnim nizovima različitih duljina podudaraju se s teorijskom analizom složenosti.

Ključne riječi: eksplicitno proširenje, implicitno proširenje, implicitno sufiksno stablo, poopćeno sufiksno stablo, sufiks-prefiks preklapanje, sufiksna veza, sufiksno stablo, Ukkonenov algoritam

Suffix tree

Abstract

Suffix tree is a data structure that allows various string operations to be executed in linear time dependent only on the length of the substring for which the operation is executed, and its construction is possible in linear time. Ukkonen's algorithm for suffix tree construction is based on construction of implicit suffix trees for each string prefix through implicit and explicit extensions. Linearity of Ukkonen's algorithm is achieved through elimination of implicit extension check, active position structure and suffix links.

In this thesis implementation of Ukkonen's algorithm for construction of generalised suffix tree, which in fact is generalisation of algorithm over set of strings, is shown. As an example of suffix tree, implementation of finding all suffix-prefix matches between strings over which the tree was built is demonstrated. Complexity calculated by running several tests over different sized strings matches the complexity obtained by theoretical analysis of the algorithm.

Keywords: explicit extension, implicit extension, implicit suffix tree, generalised suffix tree, suffix-prefix match, suffix link, suffix tree, Ukkonen's algorithm