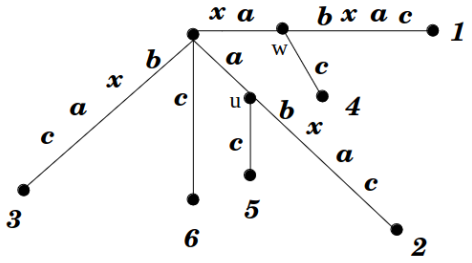
Sufiksno drvo

Sufiksno drvo (eng. suffix tree) je struktura podataka kojom se može pogodno predstaviti interna struktura niske[1]. Preciznije, sufiksno drvo niske s dužine n je usmereno korensko drvo za koje važe naredna svojstva:

- drvo ima tačno n listova koji su numerisani brojevima $1, 2, \ldots, n$;
- svaki unutrašnji čvor različit od korena ima bar dva deteta;
- svaka grana je označena nepraznim segmentom niske s;
- nikoje dva grane koje polaze iz istog čvora nemaju oznake koje počinju istim karakterom;
- nadovezivanjem oznaka grana na putu od korena do lista sa oznakom i dobija se sufiks $S_i = s[i..n]$, pri čemu podrazumevamo da indeksiranje u niski kreće od 1.

Primer 1. Na slici 1 prikazano je sufiksno drvo niske $s =$ "xabxac". Sufiksno drvo sadrži $|s| = 6$ listova, kolika je i dužina niske s. Od korena do lista koji odgovara sufiksu $S_4 =$ "xac" dolazimo preko unutrašnjeg čvora w, a sam sufiks dobijamo nadovezivanjem niski "xa" i "c" kojima su označene odgovarajuće grane. Primetimo da sufiksi S_1 i S_4 počinju istim prefiksom "xa", te na putu od korena do listova numerisanih vrednostima 1 i 4 pratimo istu granu, kodiranu niskom "xa", nakon čega se putanje do tih listova razdvajaju.



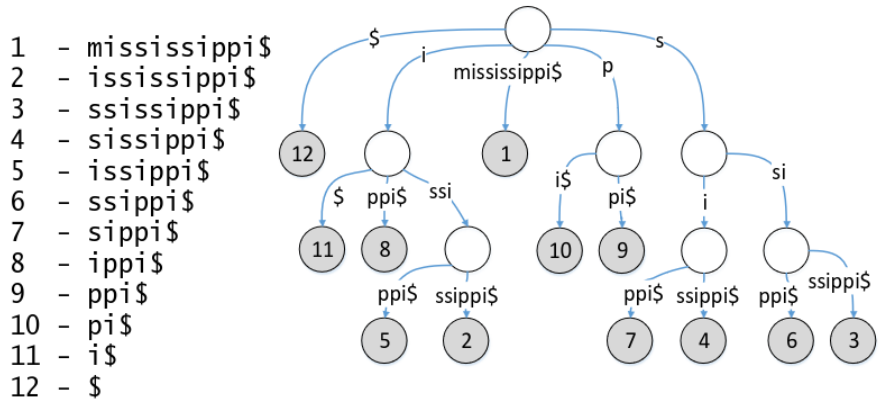
Slika 1: Sufiksno drvo niske $s =$ "xabxac". Čvorovi sa oznakama u i w su jedini unutrašnji čvorovi drveta.

Iz prethodne definicije sufiksnog drveta ne sledi nužno postojanje sufiksnog drveta za proizvoljnu nisku s. Problem se, naime, javlja onda kada je neki sufiks S_i jednak prefiksu nekog drugog sufiksa S_j, jer se tada put koji odgovara sufiksu S_i završava unutar puta koji odgovara sufiksu S_j, tj. ne završava se u listu drveta. Na primer, ne postoji sufiksno drvo za nisku "cxabxa", jer se put koji

[1] Pretpostavljamo da je azbuka nad kojom je definisana niska konačna i poznata.

odgovara sufiksu S_5 = ”xa”, kao prefiks sufiksa S_2 = ”xabxa”, ne bi završavao u listu. Primetimo da ako se poslednji karakter niske s ne nalazi nigde drugde unutar s, ovaj problem se ne javlja (takav je bio slučaj sa niskom ”xabxac” u prethodnom primeru). Uobičajen način za rešavanje ovog problema je da se na kraj niske s doda karakter koji se ne javlja nigde unutar nje (obično je to karakter \$; pretpostavlja se da karakter \$ leksikografski prethodi svim ostalim karakterima).

Primer 2. Sufiksno drvo niske s = ”mississippi\$” prikazano je na slici 2. Pošto je $|s| = 12$, sufiksno drvo ima 12 listova. Pored toga, ovo drvo ima koren i 6 unutrašnjih čvorova. Primetimo da se recimo nadovezivanjem oznaka grana na putu od korena do lista 5 dobija sufiks S_7 = ”i − ssi − ppi\$”.



Slika 2: Sufiksno drvo niske s = ”mississippi”.

Po definiciji sufiksno drvo niske dužine n ima n listova, po jedan za svaki sufiks niske. S obzirom na to da svaki unutrašnji čvor sufiksnog drveta ima bar dva deteta, može postojati najviše $n - 1$ unutrašnjih čvorova. Sumiranjem broja listova, unutrašnjih čvorova i korena dobijamo da je ukupan broj čvorova u sufiksnom drvetu manji ili jednak od $n + (n - 1) + 1 = 2n$.

Slično, važi da je broj grana u sufiksnom drvetu koje vode ka listovima n, a broj grana koje vode ka unutrašnjim čvorovima najviše $n - 1$, te je ukupan broj grana u sufiksnom drvetu manji ili jednak $2n - 1$.

Čvorovi sufiksnog drveta mogu imati različit broj dece, ali je maksimalni broj dece određen veličinom azbuke. Svaki čvor sufiksnog drveta može čuvati listu svoje dece ili (uređenu ili neuređenu) heš mapu koja rečima koje kreću iz tog čvora pridružuje grane koje iz tog čvora kreću ka njegovoj deci. Na ovaj način operacije za rad sa jednim čvorom sežu od $O(\sigma)$, gde je σ veličina azbuke do $O(1)$ amortizovano.

Ako bi se u sufiksnom drvetu uz svaku granu čuvala njena kompletna oznaka, onda bi prostorna složenost drveta za nisku dužine n bila u najgorem slučaju

$O(n^2)$. Međutim, proizvoljni segment $s[i..j]$ poznate niske s u potpunosti određen vrednostima indeksa i i j. Pošto je broj grana u sufiksnom drvetu niske dužine n najviše $O(n)$, zamenom oznaka na granama sa po dva indeksa koja ograničavaju odgovarajući segment niske postiže se da je prostorna složenost sufiksnog drveta $O(n)$.

Prvi algoritam za konstrukciju sufiksnih drveta linearne vremenske složenosti predložio je Piter Vajner (Peter Weiner) 1973. godine[2]. Nakon toga su formulisani i neki drugi algoritmi linearne složenosti, a najpoznatiji od njih danas su Ukonenov algoritam i Farahov algoritam.

Sufiksna drveta, iako omogućavaju efikasno rešavanje velikog broja složenih algoritama nad tekstom, nisu dostigla očekivanu popularnost i pažnju. Razlog za to je i taj što su prvi algoritmi za njihovu konstrukciju u linearnoj vremenskoj složenosti bili izuzetno složeni i teški za razumevanje.

Danas je najjednostavniji pristup za konstrukciju sufiksnog drveta konstrukcija na osnovu odgovarajućeg sufiksnog niza i niza najdužih zajedničkih prefiksa, koji čuva dužine najdužeg zajedničkog prefiksa dva sufiksa. S obzirom na to da je ove nizove moguće konstruisati u linearnoj složenosti, a da je sufiksni niz na osnovu njih takođe moguće konstruisati u linearnom vremenu, ukupna vremenska složenost konstrukcije sufiksnog drveta je linearna po dužini niske.

Primene sufiksnog drveta

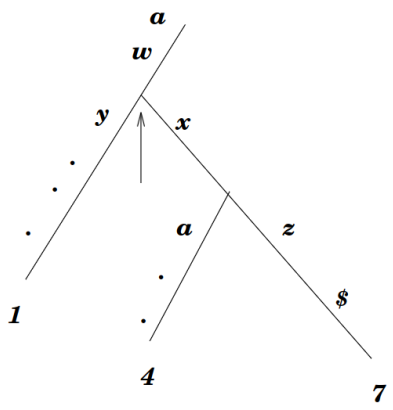
Problemi za koje smo videli da se mogu rešavati primenom sufiksnog niza mogu se još jednostavnije rešavati primenom sufiksnog drveta. Na primer, korišćenjem sufiksnog drveta moguće je rešiti problem traženja reči u tekstu u linearnoj vremenskoj složenosti (postižući na taj način istu složenost najgoreg slučaja kao u KMP algoritmu ili Bojer-Murovom algoritmu), međutim njihova prava vrednost se vidi u mogućnosti rešavanja nekih složenijih problema nad tekstom u linearnoj složenosti.

Traženje reči u tekstu Data je reč p dužine m i tekst t dužine n. Utvrditi da li se p javlja unutar teksta t, tj. da li je p segment niske t, i ako jeste naći sve takve pojave.

Primetimo naredno: p se javlja u tekstu t od pozicije j, ako i samo ako je p prefiks sufiksa $t[j..n]$. To, dalje, odgovara situaciji kada u sufiksnom drvetu postoji putanja od korena označena niskom p. Dakle, nakon konstrukcije sufiksnog drveta niske t u složenosti $O(n)$, možemo pratiti jedinstvenu putanju od korena do odgovarajućeg čvora u drvetu, praćenjem karaktera reči p sve dok je to moguće: ukoliko u nekom čvoru ne postoji grana označena narednim karakterima reči p javljamo da se reč p ne javlja u tekstu t. Inače, ako smo sa uparivanjem karaktera stigli do kraja reči p, svaki list poddrveta sa korenom u tom čvoru označen je indekstom početka pojave reči p u tekstu t.

[2]Donald Knut je ovaj algoritam zbog svoje značajnosti nazvao "algoritmom 1973. godine"

Primer 3. Razmotrimo problem traženja reči p = "aw" u tekstu t = "awyawxawxz". Na slici 3 prikazan je deo sufiksnog drveta za nisku t = "awyawxawxz". Možemo zaključiti da se reč p = "aw" javlja tri puta u tekstu t, počev od pozicija $1, 4$ i 7.



Slika 3: Tri pojavljivanja reči s = "aw" u tekstu t = "awyawxawxz".

S obzirom na to da je azbuka nad kojom je definisana niska konačna, posao koji izvršavamo u svakom čvoru je konstantne složenosti, te je vreme za uparivanje reči p sa putanjom od korena sufiksnog drveta niske t proporcionalna dužini niske p, tj. iznosi $O(m)$.

Ukoliko se reč p javlja u sufiksnom drvetu, algoritam može da pronađe indekse početaka svih pojavljivanja reči p u tekstu t obilaskom poddrveta sa korenom u čvoru koji odgovara kraju poklopljenog puta, pri čemu sakuplja indekse koji se nalaze u listovima. Stoga se sva pojavljivanja p u t mogu naći u vremenu $O(n + m)$. Ovo odgovara složenosti već razmatranih algoritama za traženje uzorka u tekstu, međutim drugačija je raspodela vremena: naime, kod prethodnih algoritama je faza preprocesiranja niske p trajala $O(m)$, a onda je trebalo $O(n)$ vremena za pretragu. Nasuprot tome, kod sufiksnih drveta faza preprocesiranja traje $O(n)$, a onda pretraga traje $O(m + z)$, gde je z broj pojava niske p u t. Naime, dolazak do unutrašnjeg čvora koji odgovara niski p dužine m je složenosti $O(m)$, a onda pretraživanje traje $O(z)$. Dokažimo ovo. Da bi se pronašlo svih z početnih pozicija u niski t, potrebno je obići poddrvo algoritmom linearne vremenske složenosti (npr. pretragom u dubinu). Naime, ako pretraga u dubinu poddrveta pronađe z poklapanja, to znači da je posetila z različitih listova. S obzirom na to da svaki unutrašnji čvor sufiksnog drveta ima bar dva deteta, ukupan broj unutrašnjih čvorova koje smo posetili tokom pretrage u dubinu je najviše $z - 1$. Tokom DFS pretrage ne moramo vršiti uparivanje oznaka na granama, već samo da pratimo grane što je složenosti $O(1)$. Stoga pretraga u dubinu posećuje najviše $O(z)$ čvorova i grana i troši vreme $O(1)$ po čvoru i grani, te je ukupno vreme izvršavanja $O(z)$.

Dakle, vreme preprocesiranja koje odgovara formiranju sufiksnog drveta teksta t proporcionalno je dužini teksta, ali nakon toga se za različite niske dužine m potraga za niskom p može izvesti u vremenu proporcionalnom dužini reči koju tražimo p, nezavisno od dužine teksta t. Primetimo da ovo nije moguće postići korišćenjem KMP algoritma, niti Bojer-Murovog algoritma – naime, ovi algoritmi imaju fazu preprocesiranja koja je linearne složenosti u funkciji dužine uzorka, ali se nakon toga pretraga za datom reči u tekstu izvršava u složenosti koja je proporcionalna dužini teksta, a ne reči koju tražimo. S obzirom na to da se m i n mogu značajno razlikovati po vrednosti, ovo može biti značajno ubrzanje. Na primer, u molekularnoj biologiji ovo može biti veoma značajno kada se traže specifični kratki stringovi u dugačkim nizovima DNK.

Leksikografski najmanja rotacija Data je niska s dužine n. Odrediti njenu leksikografski najmanju rotaciju. Npr. ako je s jednako "alabala", onda su sve njene rotacije redom jednake: "alabala", "labalaa", "abalaal", "balaala", "alaalab", "laalaba", "aalabal" i najmanja među njima je "aalabal".

Problem možemo rešiti korišćenjem sufiksnih drveta tako što formiramo sufiksno drvo za nisku ss\$, gde je \$ karakter koji je leksikografski veći od svih karaktera u niski s. Obilazimo sufiksno drvo i u svakom čvoru biramo granu koja je leksikografski najmanja. Obilazak nastavljamo sve dok niska koju dobijamo nadovezivanjem oznaka na granama ne bude dužine n.

Najduži ponovljeni segment Data je niska s. Pronaći dužinu najdužeg segmenta koji se u niski s javlja bar dva puta. Na primer, u tekstu "to be or not to be", najduži ponovljeni segment je "to be".

Primetimo da se ovaj problem može rešiti korišćenjem sufiksnog drveta tako što ćemo tražiti unutrašnji čvor koji ima najveću dubinu niske, pri čemu pod dubinom niske podrazumevamo broj karaktera na putu od korena do tog čvora. Ovaj čvor je moguće pronaći pokretanjem pretrage u dubinu, praćenjem dubine niske tokom obilaska kako bi se pronašao unutrašnji čvor drveta sa najvećom dubinom niske.

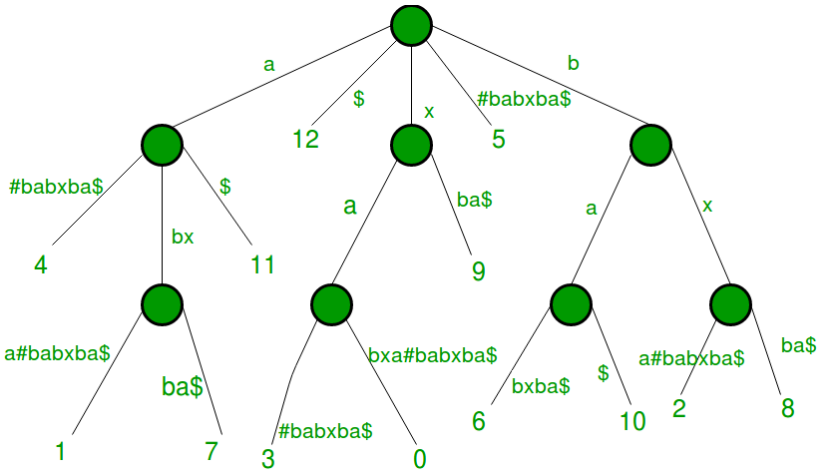
Traženje najdužeg zajedničkog segmenta dve date niske Date su dve niske s_1 i s_2. Odrediti njihov najduži zajednički segment.

Uopšteno sufiksno drvo (eng. generalized suffix tree) je uopštenje klasičnog sufiksnog drveta koje može efikasno da barata većim brojem niski istovremeno. Za razliku od standardnog sufiksnog drveta, koje predstavlja sve sufikse jedne niske, uopšteno sufiksno drvo predstavlja sufikse većeg broja niski istovremeno.

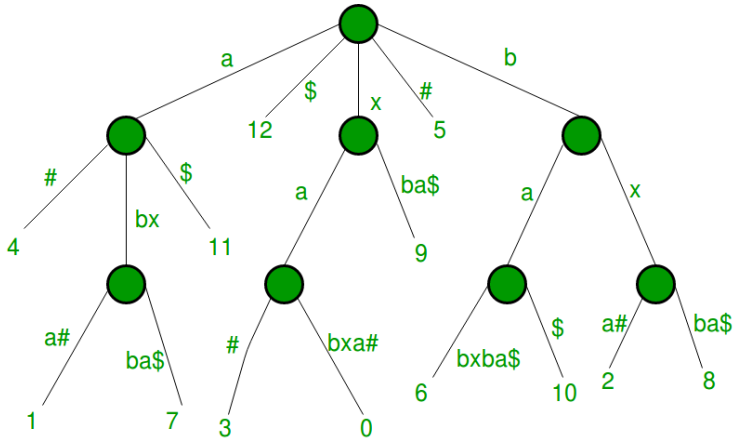
Razmotrimo primer dve niske s_1 i s_2. Uopšteno sufiksno drvo ove dve niske može se konstruisati tako što se kreira sufiksno drvo niske $s_1\#s_2$\$, gde su # i \$ karakteri koji se ne javljaju u niskama s_1 i s_2 (slika 4), a nakon toga se u svakom listu obriše deo desno od prvog terminirajućeg karaktera. Kako bi sufiksno drvo spojene niske moglo da se konstruiše, neophodno je da ovi terminirajući karakteri

budu međusobno različiti. Na taj način svaki od listova predstavlja sufiks tačno jedne od polaznih niski (slika 5). Svaki list je označen jednim od brojeva 1 i 2, na osnovu toga da li je u pitanju sufiks niske s_1 ili niske s_2.

Vratimo se polaznom problemu. Obilaskom drveta u dubinu možemo za svaki unutrašnji čvor sufiksnog drveta utvrditi da li se u njegovom poddrvetu nalaze listovi samo jedne od datih niski. Čvor sufiksnog drveta koji ima najveću vrednost dubine niske i sadrži listove obe date niske biće najduži zajednički segment niski s_1 i s_2. Primetimo da se obilazak drveta i označavanje čvorova mogu izvesti standardnim algoritmom obilaska grafa te je složenost odgovarajućeg algoritma linearna u funkciji zbira dužina niski s_1 i s_2.



Slika 4: Sufiksno drvo niske ”xabxa#babxa$”.

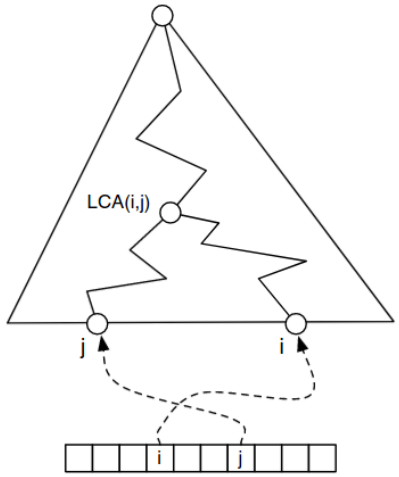


Slika 5: Uopšteno sufiksno drvo niski ”xabxa” i ”babxa”.

Najduži zajednički nastavak niski Date su niske s i t. Potrebno je omogućiti efikasno izvršavanje velikog broja upita kojima se za prosleđene vrednosti indeksa i i j računa dužina najdužeg segmenta niske s koji počinje na poziciji i, a koji se poklapa sa segmentom niske t koji počinje na poziciji j. Ovu vrednost zvaćemo *najdužim zajedničkim nastavkom* (eng. longest common extension) niski s i t.

Označimo ovu vrednost sa $LCE_{S,T}(i, j)$. Na primer, za niske $s =$ "mama" i $t =$ "marama" važi: $LCE_{S,T}(2, 2) = 1$ i $LCE_{S,T}(2, 4) = 3$. Primetimo da je vrednost $LCE_{S,T}(i, j)$ dužina najdužeg zajedničkog prefiksa sufiksa niske s i t koji počinju na pozicijama i i j. Uopšteno sufiksno drvo niski s i t omogućava traženje ovih sufiksa i pamćenje informacija o njihovim zajedničkim prefiksima.

Dakle, potrebno je konstruisati uopšteno sufiksno drvo niski s i t. Pronalazimo listove koji odgovaraju sufiksu S_i i T_j. Tražimo njihovog *najnižeg zajedničkog pretka* (eng. lowest common ancestor, LCA), odnosno čvor drveta koji ima najveću dubinu niske od svih predaka datih čvorova (slika 6). Pokazuje se da je moguće preprocesirati sufiksno drvo u linearnom vremenu tako da se najniži zajednički predak može naći u konstantnom vremenu. Kreiramo niz koji preslikava brojeve sufiksa u listove drveta. Neka je dubina niske čvora koji predstavlja najniži zajednički predak ovih listova jednaka d. Vraćamo kao rezultat broj d koji odgovara poklopljenim segmentima $t[i..i+d-1]$ i $s[j..j+d-1]$. Ukupno je potrebno uložiti $O(m)$ vremena za fazu preprocesiranja, a nakon toga se upiti mogu izvršavati u vremenu $O(1)$.



Slika 6: Ilustracija najnižeg zajedničkog pretka dva lista.

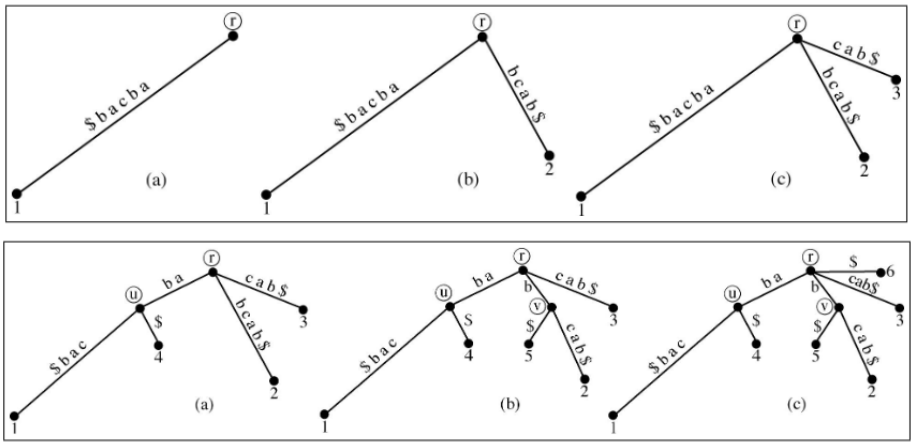
Konstrukcija sufiksnog drveta

Direktan algoritam Da bismo ilustrovali problem formiranja sufiksnog drveta niske s dužine m, opisaćemo najpre rešenje problema najjednostavnijim algo-

ritmom. Algoritam radi na inkrementalan način, obradom jednog po jednog sufiksa $s[i..m]$, od $i = 1$ do $i = m$. U prvom koraku se sufiks $S_1 = s[1..m]$ umeće u prazno drvo i sufiksno drvo se sastoji od jednog čvora. Zatim se za $i = 2, 3, \ldots, m$ u sufiksno drvo redom uključuje sufiks $S_i = s[i..m]$. Induktivna hipoteza glasi: "Umemo da konstruišemo drvo ST_i koje kodira sve sufikse niske s koji počinju na pozicijama od 1 do i".

Bazni slučaj je jednostavan – drvo ST_1 se sastoji od jedne grane označene oznakom $s\$$ koja povezuje koren i list sa oznakom 1. Drvo ST_{i+1} se konstruiše od drveta ST_i na sledeći način: polazeći od korena drveta ST_i pronalazi se najduži put čija se oznaka poklapa sa prefiksom niske $s[i+1..m]\$$; pod oznakom puta podrazumeva se konkatenacija oznaka grana koje čine taj put. Kada se takav put pronađe, on se završava ili u nekom unutrašnjem čvoru w ili unutar neke grane (u, v). Ako je unutar grane, onda se grana (u, v) "razbija" na dve grane umetanjem čvora w tačno nakog poslednjeg karaktera na grani koji se poklopio karakterom u $s[i+1..m]$ i tačno ispred prvog karaktera na grani koji se nije poklopio. Nova grana (u, w) označava se delom oznake grane (u, v) koja se poklopila sa prefiksom niske $s[i+1..m]$, a nova grana (w, v) se označava ostatkom oznake grane (u, v). Zatim, bilo da se novi čvor w formirao ili da je neki čvor već postojao na mestu gde se poklapanje završilo, algoritam kreira novu granu $(w, i+1)$ koja povezuje čvor w i novi list sa oznakom $i+1$ i označava novu granu nepoklopljenim delom sufiksa $s[i+1..m]\$$. Primetimo da zbog terminirajućeg karaktera \$ ne može da bude kompletnog poklapanja na niski.

Opisani postupak konstrukcije sufiksnog drveta niske $s =$ "abcab\$" ilustrovan je na slici 7.



Slika 7: Ilustracija direktnog algoritma za konstrukciju sufiksnog drveta.

Razmotrimo složenost ovog algoritma. U i-tom koraku algoritma traži se najduži zajednički prefiks sufiksa S_i i prethodnih $i-1$ sufiksa $S_1, S_2, \ldots, S_{i-1}$. Neka je taj prefiks jednak $s[i..k]$. Za identifikovanje ovog prefiksa izvedeno je $k - i + 1$

poređenja. Nakon toga, potrebno je pročitati ostatak sufiksa $s[k{+}1..n]$ i pridružiti ga novoj grani. Stoga je ukupan posao u i-tom koraku složenosti $\Theta(n-i)$, a ukupna složenost algoritma je $1+2+\ldots+n=\Theta(n^2)$.

Algoritam linearne vremenske složenosti Postoji veliki broj algoritama za konstrukciju sufiksnog drveta koji su linearne vremenske složenosti. Ako je poznat sufiksni niz SA i niz LCP niske s dužine n, sufiksno drvo te niske može se konstruisati algoritmom složenosti $O(n)$ na sledeći način: polazeći od parcijalnog sufiksnog drveta za leksikografski najmanji sufiks, vrši se umetanje ostalih sufiksa redom kojim se na njih nailazi leksikografskim obilaskom sufiksnog drveta. Na ovom mestu nećemo izvoditi detalje ovog postupka.