Strukture podataka za rad sa niskama

Algoritmi za rad sa niskama igraju važnu ulogu u različitim oblastima. Veliki broj korisnika svakodnevno pretražuje veb strane koje su pune tekstovnog sadržaja. Rad u oblasti bioinformatike podrazumeva različite operacije nad genomskim bazama podataka koje uključuju konstantan rad sa tekstom. U ovim i drugim sličnim primenama važno je razviti efikasne strukture podataka i efikasne algoritme za rešavanje različitih problema za rad sa tekstom.

Sufiksni niz i sufiksno drvo date niske su strukture podataka pomoću kojih se efikasno mogu rešavati problemi nad niskama kao što je ispitivanje da li se reč javlja u tekstu, traženje ponovljenih segmenata ili palindroma u tekstu, zajedničkih segmenata za dve ili veći broj niski i slično. One imaju primenu u indeksiranju tekstova (koriste se u strukturama podataka za indeksiranje, poput ESA), kompresiji teksta (sufiksni nizovi igraju važnu ulogu u BWT algoritmu kojim se karakteri niske reorganizuju tako da se popravi stepen kompresije), kompresiji podataka (u LZW algoritmu za kodiranje i dekodiranje nizova simbola) i bioinformatici (za sastavljanje genoma poravnanjem i spajanjem manjih fragmenata DNK).

Sufiksne nizove su po prvi put predložili Manber i Majers 1990. godine, kao jednostavnu, prostorno efikasnu alternativu sufiksnim drvetima. Pokazuje se da je do iste strukture podataka nezavisno došao i Gonet 1987. godine.

Sufiksni niz

Neka je data niska s dužine n. Neka je sa $s[i..j]$ označen segment date niske sa indeksima od i do j, pri čemu ćemo indeksiranje vršiti od 1. Specijalno, neka $S_i = s[i..n]$ označava sufiks niske s koji počinje na poziciji i.

Sufiksni niz (eng. suffix array) SA niske s je niz celobrojnih vrednosti, kojim se zadaju indeksi leksikografski sortiranih sufiksa: $S_{SA[1]} < S_{SA[2]} < \cdots < S_{SA[n]}$. Drugim rečima, ako je $SA[i] = j$, onda sufiks S_j ima rang $i = rang[j]$ među niskama $S_1, S_2, \ldots, S_n$. Nizovi SA i $rang$ su međusobno inverzne permutacije skupa $\{1, 2, \ldots, n\}$:

$$SA[rang[j]] = rang[SA[j]] = j, \quad 1 \leq j \leq n$$

Primer 1. Sortirani niz sufiksa niske $s =$ "mississippi" i njen sufiksni niz prikazani su u Tabeli 1. Na primer, niska $S_8 =$ "ippi" odgovara sufiksu $s[8..11]$.

- Leksikografski redosled sufiksa niske s je

  $$S_{11} < S_8 < S_5 < S_2 < S_1 < S_{10} < S_9 < S_7 < S_4 < S_6 < S_3.$$

  odnosno niz SA ima redom elemente:

  $$SA = \{11, 8, 5, 2, 1, 10, 9, 7, 4, 6, 3\}$$

- Koji je po redu u sufiksnom nizu sufiks S_5 = "issippi"? Treći jer je njegov rang jednak 3, odnosno važi $rang[5] = 3$. Preciznije, niz rangova elemenata jednak je

$$rang = \{5, 4, 11, 9, 3, 10, 8, 2, 7, 6, 1, 12\}$$

- Koji sufiks je sedmi po redu u sufiksnom nizu? To je sufiks S_9 = "ppi", jer je $SA[7] = 9$.

i	$SA[i]$	$rang[i]$	$S_{SA[i]}$	$LCP[i]$
1	11	5	i	
2	8	4	ippi	1
3	5	11	issippi	1
4	2	9	ississippi	4
5	1	3	mississippi	0
6	10	10	pi	0
7	9	8	ppi	1
8	7	2	sippi	0
9	4	7	sissippi	2
10	6	6	ssippi	1
11	3	1	ssissippi	3

Tabela 1: Sufiksni niz i niz LCP niske "mississippi".

Primetimo da se sufiksni niz pamti implicitno, tako što se čuvaju indeksi početka sufiksa, umesto samih sufiksa. Stoga je ova struktura podataka prostorno efikasna: za nisku dužine n zahteva $O(n)$ prostora. Pokazuje se da analizom svojstava sufiksnog niza date niske možemo steći značajan uvid u njenu strukturu.

Najduži zajednički prefiks (eng. longest common preffix, skraćeno LCP) dve date niske s_1 i s_2 je dužina $LCP(s_1, s_2)$ najduže niske koja je prefiks i s_1 i s_2. Za fiksiranu nisku s neka je $LCP(i, j) = LCP(S_i, S_j)$. Nisci s može se pored njenog sufiksnog niza SA pridružiti i niz LCP, čiji je i-ti član jednak

$$LCP[i] = LCP(S_{SA[i-1]}, S_{SA[i]}), \quad i = 2, 3, \ldots, n$$

Drugim rečima, $LCP[i]$ niske s je dužina najdužeg zajedničkog prefiksa $(i-1)$-vog i i-tog sufiksa niske s u sortiranom redosledu. Pritom se za vrednost $LCP[1]$ može uzeti 0.

Primer 2. Niz LCP niske s = "mississippi" prikazan je u Tabeli 1 zajedno sa sufiksnim nizom. Za ovu nisku je $LCP[4] = 4$, jer je $SA[3] = 5$, S_5 = "issippi", $SA[4] = 2$, S_2 = "ississippi", i

$$LCP[4] = LCP(\text{"issippi"}, \text{"ississippi"}) = |\text{"issi"}| = 4.$$

Sufiksni niz i niz LCP niske dužine n mogu se formirati u vremenu $O(n)$, što ćemo videti u nastavku materijala.

Primene sufiksnog niza

Mnoge operacije sa niskama efikasno se izvode ako se prethodno formira njihov sufiksni niz i niz LCP. Prikazaćemo nekoliko takvih primera primene. Na osnovu sufiksnog niza i niza LCP može se algoritmom linearne složenosti formirati sufiksno drvo niske, što zatim omogućuje i slične primene sufiksnog drveta.

Traženje reči u tekstu Data je reč P dužine m i tekst s dužine n. Potrebno je odrediti sve pojave reči P unutar teksta s.

Problem je lako rešiti ako se odredi sufiksni niz teksta s: svaka pojava reči P unutar s je prefiks nekog sufiksa niske s. Sufiksni niz teksta s određuje sortirani redosled sufiksa s i sufiksi koji imaju zajednički prefiks se u sufiksnom nizu nalaze jedan do drugog. Stoga se prva pojava reči P unutar teksta s može pronaći binarnom pretragom opsega $[1..|s|] = [1..n]$. Indeksi svih ostalih pojava (ako ih ima) nalaze se na narednim uzastopnim pozicijama u sufiksnom nizu.

Složenost formiranja sufiksnog niza je $O(n)$, a složenost svakog od $O(\log_2 n)$ upoređivanja niske P sa prefiksom dužine m tekućeg sufiksa je $O(m)$. Stoga je složenost algoritma za traženje reči u tekstu, zasnovanog na sufiksnom nizu, jednaka $O(n + m \log n + z)$, gde je sa z označen broj pojava reči P u tekstu.

Primer 3. Pronaći sve pojave reči P = "lednik" unutar teksta s = "prestolonaslednikovica". Sufiksni niz teksta s (sa sortiranim redosledom sufiksa) prikazan je u Tabeli 2.

Binarna pretraga započinje upoređivanjem reči "lednik" sa sufiksom $S_{SA[\lfloor \frac{1+22}{2} \rfloor]} = S_{SA[11]}$ = "lonaslednikovica". Pošto je "lonaslednikovica" > "lednik", nastavlja se sa binarnom pretragom opsega $[1, 10]$. U narednom koraku poređenje se vrši sa sufiksom $S_{SA[\lfloor \frac{1+10}{2} \rfloor]} = S_{SA[5]}$ = "ednikovica". Pošto je "ednikovica" < "lednik", nastavlja se sa binarnom pretragom opsega $[6, 10]$. Posle nekoliko koraka pronalazi se da je indeks jednog pojavljivanja reči P unutar s jednak 12. Pošto ni leksikografski prethodni sufiks S_{17} ni naredni sufiks S_7 ne započinju sa P, unutar s nema drugih pojava reči P.

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

Problem se rešava tako što se najpre formira sufiksni niz niske $s = s_1\$s_2\#$, pri čemu su karakteri \$ i # karakteri koji se ne pojavljuju unutar niske s_1 ili s_2. Pri tome se svakom sufiksu prethodno pridružuje podatak da li je njegov početak unutar s_1 ili s_2. U sortiranom nizu sufiksa ove niske potrebno je pronaći par uzastopnih članova takvih da:

- odgovarajući sufiksi ne pripadaju istom nizu, i
- dužina njihovog najdužeg zajedničkog prefiksa je najveća u odnosu na sve ostale ovakve parove indeksa (primetimo da ovakvih parova prefiksa sa najdužim zajedničkim prefiksom može biti više).

i	S_i	$SA[i]$	$S_{SA[i]}$	$LCP[i]$
1	prestolonaslednikovica	22	a	
2	restolonaslednikovica	10	aslednikovica	1
3	estolonaslednikovica	21	ca	0
4	stolonaslednikovica	14	dnikovica	0
5	tolonaslednikovica	13	ednikovica	0
6	olonaslednikovica	3	estolonaslednikovica	0
7	lonaslednikovica	20	ica	0
8	onaslednikovica	16	ikovica	1
9	naslednikovica	17	kovica	0
10	aslednikovica	12	**lednik**ovica	0
11	slednikovica	7	lonaslednikovica	1
12	lednikovica	9	naslednikovica	0
13	ednikovica	15	nikovica	1
14	dnikovica	6	olonaslednikovica	0
15	nikovica	8	onaslednikovica	1
16	ikovica	18	ovica	1
17	kovica	1	prestolonaslednikovica	0
18	ovica	2	restolonaslednikovica	0
19	vica	11	slednikovica	0
20	ica	4	stolonaslednikovica	1
21	ca	5	tolonaslednikovica	0
22	a	19	vica	

Tabela 2: Sufiksni niz i niz LCP niske "prestolonaslednikovica".

Naglasimo da je važno niske s_1 i s_2 razdvojiti nekim posebnim karakterom, da bismo izbegli situaciju da se segment koji počinje u prvoj niski proteže i kroz nisku s_2. Npr. u slučaju kada je niska s_1 = "abcaa" i s_2 = "bacaabb", mogli bismo kao segment niske s_1s_2 koji se dva puta javlja razmatrati nisku "caab", koja jeste segment niske s_2, ali nije segment niske s_1.

Ako se koristi i niz LCP združene niske s (složenost njegovog formiranja je $O(n)$, gde je $n = |s_1| + |s_2|$), onda je složenost algoritma linearna, tj. iznosi $O(n)$.

Primer 4. Pronaći najduži zajednički segment niski

$$s_1 = \text{"prestolonaslednikovica"}$$

i

$$s_2 = \text{"kolonizacija"}.$$

Sufiksni niz SA združene niske $s = s_1\$s_2\#$ (sa sortiranim redosledom sufiksa) prikazan je u Tabeli 3. Najveći broj u koloni $LCP[i]$ je $LCP[27] = 4$. Pored toga, sufiksi

$$S_{SA[26]]} = \text{"olonaslednikovica\$kolonizacija\#"}$$

i

$$S_{SA[27]]} = \text{"olonizacija\#"}$$

sa zajedničkim prefiksom "olon" dužine 4 potiču iz različitih niski. Prema tome, najduži zajednički segment ove dve niske je niska "olon".

	S_i	i		$SA[i]$	$S_{SA[i]}$	$LCP[i]$
1	prestolonaslednikovica\$ko...#	1	2	37		0
1	restolonaslednikovica\$ko...#	2	2	36	#	0
1	estolonaslednikovica\$ko...#	3	1	23	\$ko...#	0
1	stolonaslednikovica\$ko...#	4	2	35	a#	0
1	tolonaslednikovica\$ko...#	5	1	22	a\$ko...#	1
1	olonaslednikovica\$ko...#	6	2	31	acija#	1
1	lonaslednikovica\$ko...#	7	1	10	aslednikovica\$ko...#	1
1	onaslednikovica\$ko...#	8	1	21	ca\$ko...#	0
1	naslednikovica\$ko...#	9	2	32	cija#	1
1	aslednikovica\$ko...#	10	1	14	dnikovica\$ko...#	0
1	slednikovica\$ko...#	11	1	13	ednikovica\$ko...#	0
1	lednikovica\$ko...#	12	1	3	estolonaslednikovica\$ko...#	1
1	ednikovica\$ko...#	13	1	20	ica\$ko...#	0
1	dnikovica\$ko...#	14	2	33	ija#	1
1	nikovica\$ko...#	15	1	16	ikovica\$ko...#	1
1	ikovica\$ko...#	16	2	29	izacija#	1
1	kovica\$ko...#	17	2	34	ja#	0
1	ovica\$ko...#	18	2	24	kolonizacija#	0
1	vica\$ko...#	19	1	17	kovica\$ko...#	2
1	ica\$ko...#	20	1	12	lednikovica\$ko...#	0
1	ca\$ko...#	21	1	7	lonaslednikovica\$ko...#	1
1	a\$ko...#	22	2	26	lonizacija#	3
1	\$ko...#	23	1	9	naslednikovica\$ko...#	0
2	kolonizacija#	24	1	15	nikovica\$ko...#	1
2	olonizacija#	25	2	28	nizacija#	2
2	lonizacija#	26	**1**	6	**olon**aslednikovica\$ko...#	0
2	onizacija#	27	**2**	25	**olon**izacija#	**4**
2	nizacija#	28	1	8	onaslednikovica\$ko...#	1
2	izacija#S	29	2	27	onizacija#	2
2	zacija#	30	1	18	ovica\$ko...#	1
2	acija#	31	1	1	prestolonaslednikovica\$ko...#	0
2	cija#	32	1	2	restolonaslednikovica\$ko...#	0
2	ija#	33	1	11	slednikovica\$ko...#	0
2	ja#	34	1	4	stolonaslednikovica\$ko...#	1
2	a#	35	1	5	tolonaslednikovica\$ko...#	0
2	#	36	1	19	vica\$ko...#	0
2		37	2	30	zacija#	0

Tabela 3: Sufiksni niz niske "prestolonaslednikovica\$kolonizacija#".

Pronalaženje najdužeg palindroma u zadatoj nisci Neka je data niska s. Pronaći najduži palindrom unutar nje. Npr. za reč "banana" potrebno je vratiti

"anana".

Rešavanje ovog problema svodi se na traženje najdužeg zajedničkog segmenta niski s i s', gde je s' niska koja se od s dobija čitanjem unazad.

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

Ovaj problem ima puno važnih primena, uključujući kompresiju podataka i kriptografiju. Na primer, standardna tehnika koja se koristi u razvoju velikih softverskih sistema je refaktorisanje koda. Naime, ako se program razvija tokom velikog broja godina, neretko se dešava da se u njemu javlja ponovljeni kod. U tim situacijama je od značaja i za razumevanje i za dalje održavanje softvera zameniti duplirani kod pozivima funkcije jedne kopije koda. Slično, u bioinformatičkim primenama može biti od značaja utvrditi da li se isti DNK fragment može naći u datom genomu. U oba slučaja se problem svodi na traženje najdužeg ponovljenog segmenta u niski.

Ovaj problem se može formulisati i drugačije, u terminima koji su pogodni za primenu sufiksnih nizova: pronaći najduži zajednički prefiks između dva proizvoljna sufiksa date niske. Važi da će sufiksi sa najdužim zajedničkim prefiksom biti susedni u sufiksnom nizu. Stoga možemo izračunati elemente niza LCP i u njemu naći maksimalni element: on će odgovarati dužini najdužeg zajedničkog prefiksa dva sufiksa niske s (slika 1). Ukoliko smo za datu nisku s već konstruisali sufiksni niz, onda je složenost ovog algoritma $O(n)$, gde je n dužina niske s.

Broj različitih segmenata niske Neka je data niska s. Izračunati broj različitih segmenata niske s.

Računanje broja različitih segmenata niske s svešćemo na sumiranje broja različitih segmenata koji počinju na nekoj fiksiranoj poziciji. Pritom ćemo voditi računa da brojimo samo nove, do sad neračunate segmente. Za ove potrebe možemo iskoristiti sufiksni niz. S obzirom na to da su sufiksi niske u sufiksnom nizu sortirani, jasno je da ćemo nove segmente koji počinju na poziciji $SA[i]$ dobiti kao prefikse sufiksa niske s koji počinje na poziciji $SA[i]$, pri čemu treba isključiti one koji su prefiksi sufiksa koji počinje na poziciji $SA[i-1]$. Dakle od ukupnog broja prefiksa koji počinju na poziciji $SA[i]$ treba oduzeti njih $LCP[i]$. S obzirom na to da je dužina sufiksa koji počinje na poziciji $SA[i]$ jednaka $n - SA[i] + 1$, ukupno $n - SA[i] + 1 - LCP[i]$ novih prefiksa počinje na poziciji $SA[i]$. Dakle, ukupan broj različitih segmenata biće jednak:

Slika 1: Najduži ponovljeni segment u genomu.

$$\begin{aligned}\sum_{i=1}^{n}(n - SA[i] + 1 - LCP[i]) &= \sum_{i=1}^{n}(n + 1 - SA[i]) - \sum_{i=1}^{n} LCP[i] \\ &= \frac{n(n+1)}{2} - \sum_{i=1}^{n} LCP[i]\end{aligned}$$

Leksikografski najmanja rotacija Neka je data 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 tako što nadovežemo nisku s na samu sebe i za nju odredimo minimalni segment dužine n. S obzirom na to da je njihov redosled

određen redosledom sufiksa ove niske, u te svrhe možemo iskoristiti sufiksne nizove i u sortiranom nizu sufiksa tražiti prvi sufiks koji je dužine bar n.

Konstrukcija sufiksnog niza

Od svog nastanka devedesetih godina prošlog veka pa sve do danas razvijani su različiti algoritmi za konstrukciju sufiksnih nizova. Najbolji poznat algoritam za konstrukciju sufiksnih nizova je složenosti $O(n)$ za nisku dužine n, i postoji nekoliko različitih algoritama linearne složenosti. Za veliki broj regularnih primena, konstrukcija sufiksnog niza u vremenu $O(n \log n)$ je sasvim zadovoljavajuća. Ipak, u nekim primenama je neophodno primeniti neki od algoritama linearne vremenske složenosti.

Naivni algoritam Direktan način da konstruišemo sufiksni niz niske s je smestiti sve sufikse niske s u jedan niz i sortirati ih algoritmom za sortiranje složenosti $O(n \log n)$. S obzirom na to da je složenost poređenja dva sufiksa složenosti $O(n)$, ukupna složenost direktnog algoritma za konstrukciju sufiksnog niza je $O(n^2 \log n)$.

Algoritam složenosti $O(n \log n)$ Direktno poređenje dva sufiksa niske je složenosti $O(n)$; postavlja se pitanje da li je sufikse niske moguće uporediti efikasnije. Bolji algoritam za konstrukciju sufiksnog niza može se dobiti na sledeći način: održavamo sve sufikse datog stringa sortirane na osnovu njihovih prefiksa dužine 2^k. Imaćemo $m = \lfloor \log_2 n \rfloor$ koraka, pri čemu u k-tom koraku računamo poredak prefiksa dužine 2^k; dakle, najpre ih poredimo na osnovu prvog karaktera, nakon toga na osnovu prva dva, nakon toga na osnovu prva četiri itd. Pri prelasku sa k-tog na $(k+1)$-vi korak, vrši se nadovezivanje segmenata dužine 2^k i dobija segment dužine 2^{k+1}. Za efikasno ustanovljavanje poretka sufiksa koriste se informacije dobijene u prethodnom koraku. Primetimo da za neke segmente neće postojati segment dužine 2^k nakon njega, ali to možemo rešiti tako što nisku dopunjujemo karakterima \$, koji je leksikografski manji od svih drugih karaktera. Na ovaj način se složenost prethodnog algoritma može svesti na $O(n \log^2 n)$. Ali možemo i bolje. Za sortiranje sufiksa u svakom od koraka algoritma može se koristiti radix sort algoritam, koji dva puta vrši sortiranje prebrojavanjem. Na ovaj način se sortiranje vrši u linearnom vremenu i dobijamo algoritam složenosti $O(n \log n)$.

Algoritam linearne složenosti Postoji veliki broj algoritama za konstrukciju sufiksnog niza koji su linearne vremenske složenosti. Mi ćemo se ovde upoznati sa algoritmom čiji su autori Karkainen i Sanders i koji je prikazan u četvrtom izdanju znamenite knjige CRLS.

Bez smanjenja opštosti može se pretpostaviti da su karakteri niske $s = s[1..n]$ kodirani prirodnim brojevima iz skupa $\{1, 2, \ldots, n\}$.

Algoritam je zasnovan na specifičnoj primeni dekompozicije:

1. Sortiraju se sufiksi S_i niske s, za $i \mod 3 \neq 0$ ($i = 1, 2, 4, 5, 7, 8, 10, 11, \ldots$; u daljem tekstu: A-sufiksi) svođenjem na rekurzivnu konstrukciju sufiksnog niza posebno formirane niske dužine oko $2n/3$.
2. Na osnovu toga se sortiraju preostali sufiksi S_i, $i \mod 3 = 0$ ($i = 3, 6, 9, \ldots$; u daljem tekstu: B-sufiksi).
3. Objedinjavanjem sortiranih nizova A-sufiksa i B-sufiksa određuje se sufiksni niz niske s.

Prelazimo na detaljniji opis tri osnovna koraka algoritma, ilustrujući postupak na primeru niske $s =$ "mississippi".

1. Polazeći od niske s, formira se niska P nad azbukom sastavljenom od meta karaktera — troslovnih blokova polazne azbuke; pri tome sufiksi niske P imaju isti leksikografski redosled (određen sufiksnim nizom):

   A Formirati niske P_1 i P_2 od meta karaktera (trojke odvajamo zagradama):

   - $P_1 = (s[1..3])(s[4..6])(s[7..9]) \cdots (s[n'..n'+2])$, gde je n' najveći broj manji ili jednak n, koji daje ostatak 1 po modulu 3. Pri tome je niska s produžena specijalnim karakterima $\varnothing$ sa kodom 0. Za nisku $s =$ "mississippi" dobija se $P_1 = (mis)(sis)(sip)(pi\varnothing)$.
   - $P_2 = (s[2..4])(s[5..8])(s[8..10]) \cdots (s[n''..n''+2])$, gde je n'' najveći broj manji ili jednak n, koji daje ostatak 2 po modulu 3. Za nisku s je $P_2 = (iss)(iss)(ipp)(i\varnothing\varnothing)$.

   Ako je n deljivo sa 3, na kraj niske P_1 dodaje se meta karakter $(\varnothing\varnothing\varnothing)$. Time se postiže da se P_1 uvek završava meta karakterom koji sadrži $\varnothing$ (ovo ne mora da važi za nisku P_2).

   B Neka je P konkatenacija niski P_1 i P_2. Činjenica da se P_1 završava meta karakterom koji sadrži $\varnothing$ obezbeđuje da su dva sufiksa P, od kojih jedan počinje unutar P_1, a drugi unutar P_2, uvek različita.

   Od indeksa i' meta karaktera $P[i'] = s[i..i+2]$, odgovarajući indeks $i = g(i')$, $i \equiv 1, 2 \pmod 3$, unutar niske s dobija se na sledeći način:

$$i = g(i') = \begin{cases} 3i' - 2, & i' \leq |P_1| \\ 3i' - 1 - 3|P_1|, & i' > |P_1| \end{cases} \tag{1}$$

   Obrnuto:

$$i' = g^{-1}(i) = \begin{cases} (i+2)/3, & i \equiv 1 \pmod 3 \\ (i+1)/3 + |P_1|, & i \equiv 2 \pmod 3 \end{cases} \tag{2}$$

   pri čemu se dužina niske P_1 računa kao broj metakaraktera. U tabeli 4 prikazana je niska P. Pri tome je na primer

   - $P[3] =$ sip $= s[7..9]$, jer je $7 = g(3) = 3 \cdot 3 - 2$ $(3 \leq |P_1| = 4)$, odnosno $3 = (7+2)/3$ $(7 \equiv 1 \pmod 3)$;

- $P[6] = \text{iss} = s[5..7]$, jer je $5 = g(6) = 3{\cdot}6{-}1{-}3|P_1|$ $(5 > |P_1| = 4)$, odnosno $6 = (5+1)/3 + |P_1|$ $(5 \equiv 2 \pmod 3)$.

Zapaža se da je leksikografski redosled sufiksa niske P isti kao redosled odgovarajućih A-sufiksa niske s.

C Sortirati trostrukim razvrstavanjem i rangirati jedinstvene meta karaktere niske P (dakle, zanemarujući duplikate), računajući rangove od 1. Složenost ovog sortiranja je $O(n)$. U našem primeru P ima 7 jedinstvenih meta karaktera. Njihov sortirani redosled je (i∅∅), (ipp), (iss), (mis), (pi∅), (sip), (sis). Rangovi ovih meta karaktera su redom $1, 2, \ldots, 7$. Meta karakter (iss) se u nisci P jedini pojavljuje dva puta.

D Kao u tabeli 4, formira se nova niska P' kodiranjem meta karaktera njihovim upravo određenim rangovima. Ako P sadrži k jedinstvenih meta karaktera, onda je svaki "karakter" u P' neki prirodni broj od 1 do k. Sufiksni nizovi niski P i P' su identični.

E Odrediti sufiksni niz $SA_{P'}$ niske P'. Ako su svi karakteri u P' jedinstveni, onda se sufiksni niz dobija direktno, jer redosled pojedinih karaktera određuje sufiksni niz. U protivnom, odrediti rekurzivno sufiksni niz P', tretirajući rangove u P' kao karaktere niske. U tabeli 4 je prikazan sufiksni niz $SA_{P'}$ dobijen u našem primeru. Pošto je broj meta karaktera u P', a time i dužina P', otprilike $2n/3$, rekurzivni problem je manji od polaznog.

F Na osnovu $SA_{P'}$ i pozicija A-sufiksa u nisci s, odrediti spisak rangova (pozicija) sortiranih A-sufiksa u s. Preciznije, za $i = 1, 2, \ldots, |P|$ izračunati na osnovu (1) indeks $j = g(SA_{P'}[i])$ i staviti $r_j \leftarrow i$. U tabeli 5 prikazani su rangovi r_j sortiranih A-sufiksa $s[i..i+2]$ u našem primeru. Drugim rečima, rezultat sortiranja A-sufiksa je

$$S_{11} < S_8 < S_5 < S_2 < S_1 < S_{10} < S_7 < S_4$$

2. Broj B-sufiksa je oko 1/3 broja svih sufiksa. Koristeći sortirani redosled A-sufiksa, sortirati B-sufikse primenjujući sledeći postupak.

G Proširivši nisku s sa dva karaktera ∅∅ (posle čega je dužina niske $n+2$), posmatrajmo sufikse $s[i..n+2]$ za $i = 1, 2, \ldots, n+2$. Dodeliti svakom sufiksu $s[i..n+2]$ njegov rang r_i.

- Za dva specijalna karaktera ∅∅ staviti $r_{n+1} = r_{n+2} = 0$.
- Za A-sufikse u nisci s rangovi su dodeljeni u koraku F.
- Rangovi B-sufiksa trenutno nisu definisani, pa za njih staviti $r_i = \square$.

U tabeli 5 prikazani su rangovi za nisku $s = \text{mississippi}$ i $n = 11$.

H Sortirati B-sufikse dvostrukim razvrstavanjem (složenosti $O(n)$), zamenjujući ih (jedinstvenim!) parovima $(s[i], r_{i+1})$. U našem primeru dobija se

$$S_9 < S_6 < S_3,$$

jer je $(p, 6) < (s, 7) < (s, 8)$.

3. Objediniti sortirane nizove A-sufiksa i B-sufiksa. Neka su S_i i S_j neka dva sufiksa koja treba uporediti u toku objedinjavanja, pri čemu je S_i A-sufiks, a S_j B-sufiks. U zavisnosti od toga da li je $i \mod 3$ jednako 1 ili 2, rezultat upoređivanja sufiksa S_i i S_j isti je kao rezultat upoređivanja

   - parova $(s[i], r_{i+1})$ i $(s[j], r_{j+1})$, odnosno
   - trojki $(s[i], s[i+1], r_{i+2})$ i $(s[j], s[j+1], r_{j+2})$.

   U našem primeru prilikom objedinjavanja se vrši sledeći niz upoređivanja A- i B-sufiksa:

   - $S_{11} < S_9$ jer je $(i, \varnothing, 0) < (p, p, 1)$; izlaz S_{11};
   - $S_8 < S_9$ jer je $(i, p, 6) < (p, p, 1)$; izlaz S_8;
   - $S_5 < S_9$ jer je $(i, s, 7) < (p, p, 1)$; izlaz S_5;
   - $S_2 < S_9$ jer je $(i, s, 8) < (p, p, 1)$; izlaz S_2;
   - $S_1 < S_9$ jer je $(m, 4) < (p, 6)$; izlaz S_1;
   - $S_{10} < S_9$ jer je $(p, 1) < (p, 6)$; izlaz S_{10};
   - $S_7 > S_9$ jer je $(s, 2) < (p, 6)$; izlaz S_9;
   - $S_7 < S_6$ jer je $(s, 2) < (s, 7)$; izlaz S_7;
   - $S_4 < S_6$ jer je $(s, 3) < (s, 7)$; izlaz S_4;
   - preostala dva sufiksa S_6, S_3 kopiraju se u izlazni niz.

   Time je određen sortirani redosled svih sufiksa niske s, odnosno njen sufiksni niz, videti tabelu 1. Pošto je složenost upoređivanja $O(1)$, složenost objedinjavanja je $O(n)$.

Indeks i' u P, P'	1	2	3	4	5	6	7	8
Indeks $i = g(i')$ u s	1	4	7	10	2	5	8	11
$P[i'] = s[i..i+2]$	(mis)	(sis)	(sip)	(pi$\varnothing$)	(iss)	(iss)	(ipp)	(i$\varnothing\varnothing$)
$P'[i']$	4	7	6	5	3	3	2	1
$SA_{P'}[i'] = SA_P[i']$	8	7	6	5	1	4	3	2
Indeks i za koji je rang $r_i = i'$	11	8	5	2	1	10	7	4

Tabela 4: Postupak sortiranja A-sufiksa niske s = mississippi u prvoj fazi određivanja sufiksnog niza.

i	1	2	3	4	5	6	7	8	9	10	11	12	13
$s[i]$	m	i	s	s	i	s	s	i	p	p	i	∅	∅
$r_i = i'$	5	4	□	8	3	□	7	2	□	6	1	□	0

Tabela 5: Rangovi r_1 do r_{n+3} za nisku $s =$ mississippi, $n = 11$.

Složenost algoritma zadovoljava diferencnu jednačinu $T(n) = O(n) + T(\lfloor 2n/3 \rfloor)$, čije je rešenje na osnovu master teoreme $T(n) = O(n)$.

Ako pretpostavimo da je vreme izvršavanja algoritma približno $T(n) = cn$, zamenom u diferencnoj jednačini

- $T(n)$ sa cn,
- $O(n)$ sa an, i
- $T(2n/3)$ sa $2cn/3$,

dolazi se do jednačine $cn = an + 2cn/3$, iz koje sledi da je $a = c/3$. Zaključuje se da rekurzivno sortiranje A-sufiksa traje približno oko 2/3 vremena potrebnog za određivanje sufiksnog niza kompletne niske.

Konstrukcija niza LCP

Niz LCP date niske s može se odrediti algoritmom linearne složenosti koji je predložilo nekoliko autora 2001. godine~[**?**]. Podsetimo se da je $LCP[i]$ dužina najdužeg zajedničkog prefiksa $(i-1)$-og i i-tog leksikografski najmanjeg sufiksa $S_{SA[i-1]}$ i $S_{SA[i]}$ niske s. Po definiciji je $LCP[1] = 0$. Prilikom izračunavanja niza LCP koristi se niz $rang$, koji sadrži inverznu permutaciju permutacije SA: ako je $SA[i] = j$, onda je $rang[j] = i$. Drugim rečima, $rang[SA[i]] = i$ za $i = 1, 2, \ldots, n$. Za sufiks S_i broj $rang[i]$ je pozicija tog sufiksa u među leksikografski sortiranim sufiksima.

Primer 5. Videli smo da je sufiksni niz niske mississippi permutacija

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 11 & 8 & 5 & 2 & 1 & 10 & 9 & 7 & 4 & 6 & 3 \end{pmatrix}$$

Niz $rang$ je inverzna permutacija ove permutacije.

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 5 & 4 & 11 & 9 & 3 & 10 & 8 & 2 & 7 & 6 & 1 \end{pmatrix}$$

Na primer, sufiks sissippi je S_4. Njegova pozicija u sortiranom redosledu sufiksa je $rang[4] = 9$.

U toku računanja niza LCP potrebno je odrediti gde se u leksikografskom redosledu nalazi sufiks koji se od tekućeg sufiksa dobija brisanjem prvog karaktera. Za ovo je koristan niz $rang$. Posmatrajmo i-ti najmanji sufiks, $S_{SA[i]}$. Brisanjem njegovog prvog karaktera dobija se sufiks $S_{SA[i]+1}$, kao sufiks koji počinje

od indeksa $SA[i]+1$ niske s. Pozicija tog sufiksa u sortiranom redosledu je $rang[SA[i]+1]$.

Primer 6. Nastavak prethodnog primera. Potrebno je za sufiks $S_4 =$ sissippi odrediti gde se nalazi sufiks issippi (sissippi bez prvog karaktera). Sufiks sissippi je na poziciji $rang[4] = 9$ u sufiksnom nizu i $SA[9] = 4$. Pošto je $rang[SA[9]+1] = rang[5] = 3$, naredni sufiks issippi je na poziciji 3 u sortiranom redosledu.

Niz LCP može se odrediti algoritmom čiji kod je prikazan na sledećoj strani.

$Određivanje LCP(s, SA, n)$
// alocirati prostor za nizove $rang[n]$ i $LCP[n]$
for $i \leftarrow 1$ **to** n **do**
  $rang[SA[i]] \leftarrow i$ // po definiciji
$LCP[1] \leftarrow 0$ // po definiciji
$l \leftarrow 0$ // tekući element niza
for $i \leftarrow 0$ **to** n **do**
  if $rang[i] > 1$ // tada se određuje $LCP[rang[i]]$
    $j \leftarrow SA[rang[i]-1]$ //S_j je leksikografski prethodnik S_i
    $m \leftarrow \max\{i, j\}$
    while $m + l < n$ **and** $s[i+l] = s[j+l]$ **do**
      $l \leftarrow l + 1$ // naredni karakter u zajedničkom prefiksu
    $LCP[rang[i]] \leftarrow l$ // $LCP(S_j, S_i)$
    if $l > 0$
      $l \leftarrow l - 1$ // ukloniti prvi karakter u zajedničkom prefiksu S_i, S_j
return LCP

Korektnost algoritma zasnovana je na narednoj lemi.

Lema 1. Posmatrajmo sufikse S_{i-1} i S_i, čije su pozicije u leksikografskom redosledu sufiksa redom $rang[i-1]$ i $rang[i]$. Ako je $LCP[rang[i-1] = l > 1$, onda za sufiks S_i, koji se od sufiksa S_{i-1} dobija brisanjem prvog karaktera, važi $LCP[rang[i]] \geq l - 1$.

Dokaz: Sufiks S_{i-1} je na poziciji $rang[i-1]$ u sortiranom redosledu sufiksa. Sufiks koji mu neposredno prethodi je u sortiranom redosledu na poziciji $rang[i-1]-1$, i to je sufiks $S_{rang[i-1]-1}$. Po pretpostavci je $LCP(S_{SA[rang[i-1]-1]}, S_{i-1}) = l > 1$. Uklanjanjem prvog karaktera u ovim sufiksima dobijaju se sufiksi $S_{SA[rang[i-1]-1]+1}$ i S_i, za koje je najduži zajednički prefiks dužine $l-1$.

- Ako sufiks $S_{SA[rang[i-1]-1]+1}$ neposredno prethodi sufiksu S_i u sortiranom redosledu, dokaz je završen.
- Pretpostavimo zato da sufiks $S_{SA[rang[i-1]-1]+1}$ ne prethodi neposredno sufiksu S_i u sortiranom redosledu. Sufiks $S_{SA[rang[i-1]-1]}$ neposredno prethodi sufiksu S_{i-1}, i oni imaju jednakih prvih $l > 1$ karaktera, pa sufiks $S_{SA[rang[i-1]-1]+1}$ mora da bude u sortiranom

redosledu pre sufiksa S_i. Između njih stoji nekoliko sufiksa. Svaki takav sufiks počinje sa istih $l-1$ karaktera kao $S_{SA[rang[i-1]-1]+1}$ i S_i. Prema tome, koji god od ovih sufiksa stoji na poziciji $rang[i]-1$ neposredno ispred S_i, on ima bar $l-1$ prvih karaktera istih kao S_i. Prema tome, $LCP[rang[i]] \geq l-1$.

□

U prvoj **for** petlji se izračunava niz $rang$. Prvi element niza LCP je po definiciji $LCP[1] = 0$. Druga **for** petlja izračunava niz LCP prolazeći sufikse prema opadajućim dužinama.

Invarijanta petlje je sledeće tvrđenje:

- U liniji $LCP[rang[i]] = l$ određuju se elementi niza LCP sa indeksima redom $rang[1]$, $rang[2]$, $rang[3]$, ..., $rang[n]$.
- Pre te linije sufiks S_i i sufiks S_j koji mu neposredno prethodi imaju najduži zajednički prefiks dužine bar l.

Ovo tvrđenje je tačno u prvom prolasku kroz **for** petlju, jer je tada $l = 0$. Ako se pretpostavi da se u liniji $LCP[rang[i]] = l$ određuje ispravna vrednost $LCP[rang[i]]$, onda se ta induktivna pretpostavka zbog dekrementiranja l (ako je pozitivno), održava na osnovu dokazane leme. Najduži zajednički prefiks sufiksa S_i i S_j može biti duži od vrednosti l na početku iteracije, pa se tačna vredost l određuje u **while** petlji. Indeks m koristi se da obezbedi da test $s[i+l] = s[j+l]$ ne izađe van granica niske. Posle izlaska iz **while** petlje vrednost l jednaka je dužini najdužeg zajedničkog prefiksa sufiksa S_i i S_j.

Primer 7. Na primeru niske mississippi mogu se videti tri različita slučaja na koje se može naići.

- U nisci mississippi sufiksi $S_5 = issippi$ $S_2 = ississippi$ su uzastopni u sufiksnom nizu (treći i četvrti, $SA[4] = 2$, $SA[3] = 5$). Dužina njihovog najdužeg zajedničkog prefiksa je $LCP[4] = 4$. Kada im se obrišu prvi karakteri, dobijaju se sufiksi $S_6 = ssippi$ $S_3 = ssissippi$ koji su takođe uzastopni u sufiksnom nizu (deseti i jedanaesti, $SA[10] = 6$, $SA[11] = 3$). Ovde je $LCP[11] = lcp[4] - 1$.
- Sufiksi $S_{10} = pi$ i $S_9 = ppi$ su takođe uzastopni u sufiksnom nizu (šesti i sedmi, $SA[6] = 10$, $SA[7] = 9$). Dužina njihovog najdužeg zajedničkog prefiksa je $LCP[7] = 1$. Kada im se obrišu prvi karakteri, dobijaju se sufiksi $S_{11} = i$ $S_{10} = pi$ koji NISU uzastopni u sufiksnom nizu (prvi i šesti, $SA[1] = 11$, $SA[6] = 10$). Zbog toga je $1 = LCP[2] \geq LCP[7] - 1 = 0$; zapažamo da u ovom primeru važi stroga nejednakost $LCP[2] = 1 > LCP[7] - 1 = 0$
- Sufiksi $S_{11} = i$ i $S_8 = ippi$ su takođe uzastopni u sufiksnom nizu (prvi i drugi, $SA[1] = 11$, $SA[2] = 8$). Dužina njihovog najdužeg zajedničkog prefiksa je $LCP[2] = 1$. Kada se sufiksu $S_{11} = i$ obriše prvi karakter,

dobija se prazan sufiks, koga nema u našem sortiranom redosledu sufiksa. Zbog toga se vrednost $LCP[2] = 1$ ne može iskoristiti na opisani način za određivanje neke druge vrednosti $LCP[.]$.

Niz *rang* kontroliše redosled određivanja elemenata niza LCP.

- Polazi se od $LCP[rang[1]] = LCP[5]$; upoređuju se sufiksi sa indeksima $SA[rang[1]] = 1$ i $SA[rang[1] - 1] = 2$, tj. sufiksi mississippi i ississippi. Prvi karakteri ova dva sufiksa se razlikuju, pa je $LCP[5] = 0$.
- Dalje se redom određuju vrednosti $LCP[rang[i]]$, $i = 2, 3, \ldots, 10$, koristeći kao početnu vrednost indeksa za upoređivanje $l + 1 = LCP[rang[i] - 1] + 1$.
- Na primer, pošto je ustanovljeno da je $LCP[rang[2]] = LCP[4] = 4$, prelazi se na određivanje $LCP[rang[3]] = LCP[11] \geq 4 - 1 = 3$. Upoređuju se, počevši od indeksa $l + 1 = 4$ sufiksi ssissippi i ssippi sa indeksima redom $SA[rang[3]] = SA[11] = 3$ i $SA[rang[3] - 1] = SA[10] = 6$; upoređujući (različite) četvrte karaktere ova dva sufiksa, zaključuje se da je $LCP[11] = 3$.

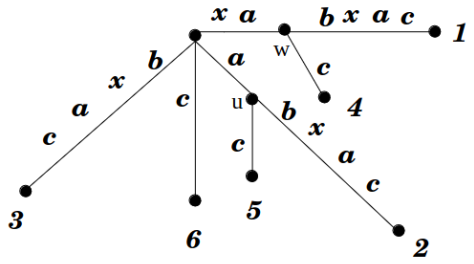
Početna vrednost l je 0; l ni u jednom trenutku ne prelazi n. Pošto je broj dekrementiranja l najviše $n - 1$, broj inkrementiranja l unutar **while** petlje je najviše $2n$. Prema tome, složenost algoritma je $O(n)$.

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 važi:

- 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 $S_i = s[i..n]$.

Primer 8. Na slici 2 prikazano je sufiksno drvo niske $s =$ "xabxac". Sufiksno drvo sadrži $|s| = 6$ listova. 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.



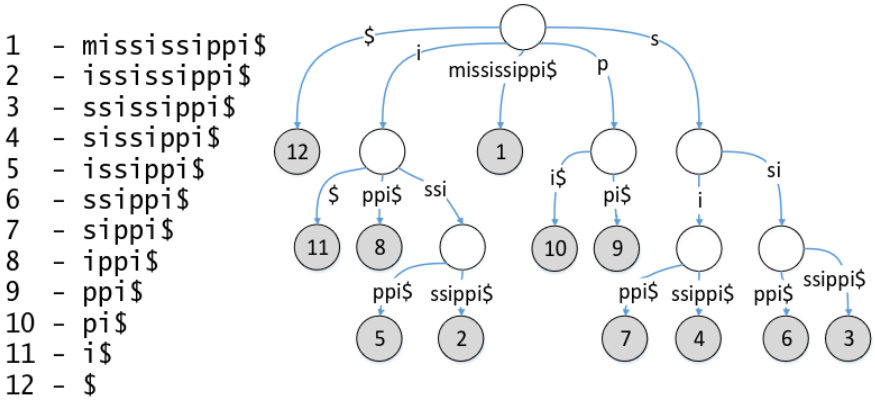
Slika 2: Sufiksno drvo niske $s =$ "xabxac". Čvorovi sa oznakama u i w su unutrašnji čvorovi.

Iz definicije sufiksnog drveta ne sledi 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 odgovara sufiksu "xa", kao prefiks sufiksa "xabxa", ne bi završavao u listu. 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

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

unutar nje (obično je to karakter $; pretpostavlja se da karakter $ leksikografski prethodi svim ostalim karakterima).

Primer 9. Sufiksno drvo niske $s =$ "mississippi$" prikazano je na slici 3. 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 3: 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 biti 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 $2n - 1$.

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 određen je sa dva 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 prostor koji zauzima sufiksno drvo $O(n)$.

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

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

Sufiksna drveta, iako omogućavaju efikasno rešavanje velikog broja složenih algoritama nad tekstom, nisu dosegla 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 LCP, 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 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.

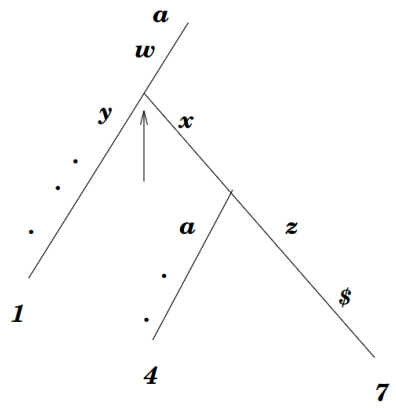
Traženje reči u tekstu Jedna od uobičajenih primena sufiksnih drveta je za rešavanje problema ispitivanja da li se data reč P dužine m nalazi u datom tekstu T dužine n, tj. da li je P segment niske T.

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 brojem početne pozicije pojave reči P u tekstu T.

Razmotrimo primer traženja reči $P =$ "aw" u tekstu $T =$ "awyawxawxz". Na slici 4 prikazan je deo sufiksnog drveta za nisku $T =$ "awyawxawxz". Reč $P =$ "aw" se javlja tri puta u T, počev od pozicija $1, 4$ i 7.

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 P javlja u drvetu, algoritam može da pronađe početnu poziciju svih pojavljivanja P u T obilaskom poddrveta sa korenom u kraju poklopljenog puta, pri čemu sakuplja pozicije koje se nalaze u listovima. Stoga se sva pojavljivanja



Slika 4: Tri pojavljivanja reči $S =$ "aw" u tekstu $T =$ "awyawxawxz".

P u T mogu naći u vremenu $O(n+m)$. Ovo odgovara složenosti već viđenih 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, faza preprociranja kod sufiksnih drveta traje $O(n)$ and onda pretraga traje $O(m+z)$, gde je z broj pojava niske P u T. 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 uparivati oznake 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 S se može izvesti u vremenu proporcionalnom dužini niske S, 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 teksta, 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 Neka je data 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 nisci javlja bar dva puta. Na primer, u tekstu "to be or not to be", najduži ponovljeni segment je "to be".

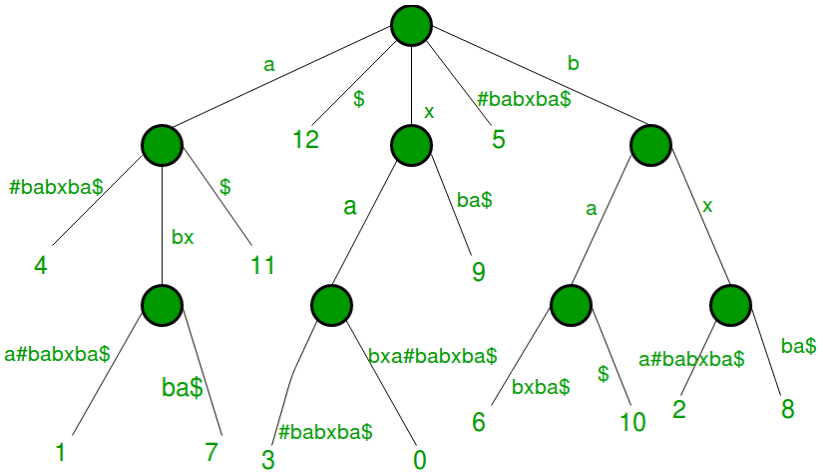
Primetimo da se ovaj problem može rešiti korišćenjem sufiksnog drveta tako što ćemo tražiti unutrašnji čvor koji ima najveću dubinu niske. Njega je moguće pronaći pokretanjem pretrage u dubinu, praćenjem dubine niske tokom obilaska kako bi se pronašao unutrašnji čvor drveta sa najvećom dubinom niske.

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

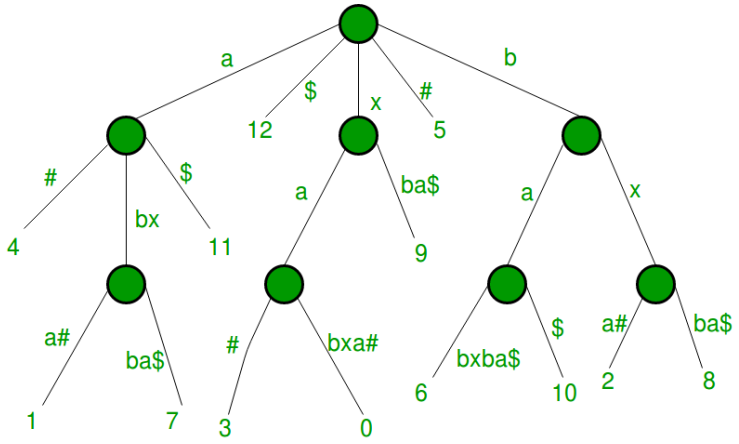
Uopšteno sufiksno drvo je sufiksno nisko za dati skup niski. Razmotrimo primer dve niske s_1 i s_2. Uopšteno sufiksno drvo ove dve niske može se konstruisati tako što se kreira sufiksno drvo niske $s_1\#s_2\$$, gde su # i \$ karakteri koji se ne javljaju u niskama s_1 i s_2 (slika 5), a nakon toga se u svakom listu obriše deo desno od prvog terminirajućeg karaktera. Na taj način svaki od listova predstavlja sufiks tačno jedne od polaznih niski (slika 6). 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. Obilaskom 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 list obe 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.

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.



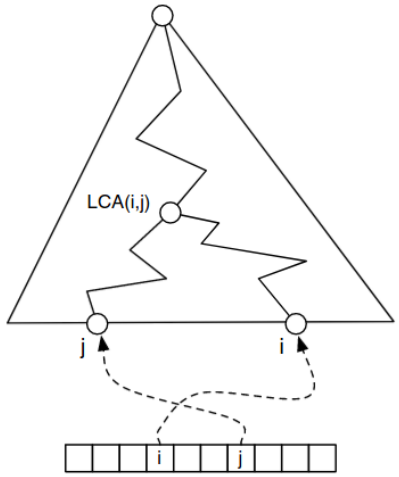
Slika 5: Sufiksno drvo niske ”xabxa#babxa$”.



Slika 6: Uopšteno sufiksno drvo niski ”xabxa” i ”babxa”.

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 7). Potrebno je preprocesirati sufiksno drvo tako da se najniži zajednički predak može naći u konstantnom vremenu. Pokazuje se da je to moguće uraditi u linearnoj vremenskoj složenosti korišćenjem različitih algoritama. 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 7: Ilustracija najnižeg zajedničkog pretka dva lista.

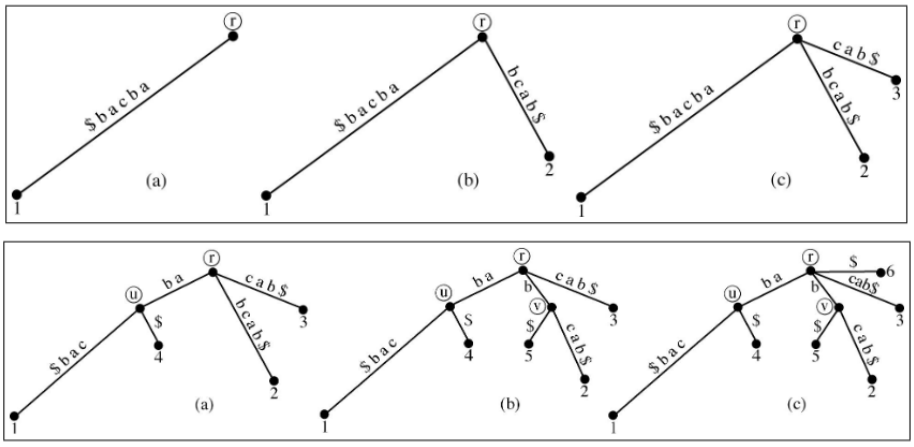
Konstrukcija sufiksnog drveta

Direktan algoritam Da bismo ilustrovali problem formiranja sufiksnog drveta niske s dužine m, opisaćemo najpre rešenje problema najjednostavnijim algoritmom. 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 završnog karaktera ne može da bude kompletnog poklapanja na niski.

Postupak konstrukcije sufiksnog drveta niske $s =$ "abcab$" ilustrovan je na slici 8.



Slika 8: 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 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, umetati u njega ostale sufikse redom kojim se na njih nailazi leksikografskim obilaskom sufiksnog drveta.

Neka je ST_i parcijalno sufiksno drvo dobijeno umetanjem prvih i leksikografski najmanjih sufiksa, $0 \leq i \leq n$. Neka je dalje $d(v)$ dužina nadovezanih oznaka svih grana u drvetu ST_i na putu od korena do čvora v. Polazi se od ST_0, drveta koje ima samo koren. Da bi se u drvo ST_{i-1} ubacio sufiks sa indeksom $SA[i]$, $1 \leq i \leq n$, prelazi se put od poslednjeg umetnutog lista ka korenu, do prvog čvora v za koji je $d(v) \leq LCP[i]$. Moguća su dva slučaja:

- $d(v) = LCP[i]$ [umetanje nove grane iz postojećeg čvora]: tada je dužina nadovezanih oznaka grana na putu od korena do čvora v jednaka $LCP(S_{SA[i-1]}, S_{SA[i]}) = LCP[i]$. U tom slučaju sufiks sa oznakom $SA[i]$ se umeće kao novi list x povezan sa čvorom v granom (v, x) sa oznakom

$S_{SA[i]+LCP[i]}$. Drugim rečima, oznaka dodate grane se sastoji od preostalih znakova sufiksa $SA[i]$ koji nisu deo nadovezanih oznaka grana na putu od korena do čvora v. Na taj način formirano je parcijalno sufiksno drvo ST_i. <!– iscrtaj za primer xabxac –>

- $d(v) < LCP[i]$ [nova grana iz čvora umetnutog u postojeću granu]: tada nadovezana oznaka grana na putu od korena do čvora v ima manje karaktera od $LCP(S_{SA[i-1]}, S_{SA[i]}) = LCP[i]$ i karakteri koji nedostaju do čvora v sadržani su na početku oznake *najdesnije* (poslednje umetnute) grane (v, w) (sa leksikografski narednim nastavkom) koja izlazi iz čvora v. Zbog toga se grana (v, w) mora novim unutrašnjim čvorom y razdvojiti na dve grane (v, y) i (y, w). Pri tome grane (v, y) i (y, w) kao oznake dobijaju odgovarajući početak, odnosno završetak oznake uklonjene grane (v, w). Preciznije, potrebno je:

  1. ukloniti granu (v, w);
  2. dodati novi unutrašnji čvor y i novu granu (v, y) sa oznakom $s[SA[i-1]] + d(v), SA[i-1] + LCP[i] - 1]$. Oznaka nove grane sastoji se od znakova koji nedostaju na najdužem zajedničkom prefiksu sufiksa $S_{SA[i-1]}$ i $S_{SA[i]}$. Prema tome, nadovezivanjem oznaka grana na putu od korena do čvora y dobija se najduži zajednički prefiks sufiksa $S_{SA[i-1]}$ i $S_{SA[i]}$;
  3. Povezati čvor w sa novokreiranim unutrašnjim čvorom y granom (y, w) sa oznakom $s[SA[i-1]] + LCP[i], SA[i-1] + d(v) - 1]$. Nova oznaka sastoji se od preostalih znakova obrisane grane (v, w) koji nisu uključeni u oznaku grane (v, y);
  4. Umetnuti čvor sa oznakom $SA[i]$ kao novi list x i povezati ga sa novim unutrašnjim čvorom y granom (y, x) sa oznakom $S_{SA[i]+LCP[i]}$. Prema tome, oznaka grane (y, x) sastoji se od preostalih karaktera sufiksa $S_{SA[i]}$, koji nisu uključeni u konkatenaciju oznaka grana na putu od korena do čvora v;
  5. Na taj način formirano je parcijalno sufiksno drvo ST_i.

Primer 10. Na sledećoj slici ilustrovan je ovaj postupak formiranja sufiksnog drveta za nisku mississippi. U opisu algoritma pretpostavlja se da se sufiksno drvo formira sleva udesno, tj. da se novi listovi dodaju povezivanjem sa čvorom na trenutno najdesnijoj grani. Zbog jednostavnijeg crtanja drveta, ovde se novi listovi priključuju drvetu ispod najniže grane umesto desno od najdesnije grane. Drugim rečima, sufiksno drvo se formira odozgo naniže.

i	$SA[i]$	$s[SA[i]..n]$	$LCP[i]$	Sufiksno drvo
1	11	i	1	i 11
2	8	ippi	1	ppi 8
3	5	issippi	4	ssi ippi 5
4	2	ississippi	0	ssippi 2
5	1	mississippi	0	mississippi 1
6	10	pi	1	p i 10
7	9	ppi	0	pi 9
8	7	sippi	2	s i ppi 4
9	4	sissippi	1	ssippi 4
10	6	ssippi	3	si ppi 6
11	3	ssissippi		ssippi 3

Lako je videti da je amortizovana složenost ovog algoritma $O(n)$. Čvorovi koji se prolaze u koraku i na putu ka korenu najdesnijim putem u drvetu ST_i (osim poslednjeg čvora v) uklanjaju se iz najdesnijeg puta kad se $A[i]$ doda u drvo kao novi list. Ovi čvorovi se nikad ne prolaze ponovo u narednim koracima $j > i$. Prema tome, ukupan broj čvorova koji se prolaze u toku izvršenja algoritma je najviše $2n$.