Strukture podataka za rad sa tekstom

Algoritmi za rad sa tekstom 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. Zbog ovih i drugih sličnih primena važno je razviti efikasne strukture podataka i efikasne algoritme za rešavanje različitih problema za rad sa tekstom.

Sufiksni niz i *sufiksno drvo* su strukture podataka pomoću kojih se efikasno mogu rešavati problemi nad tekstom kao što su ispitivanje da li se neka reč javlja u širem tekstu, traženje ponovljenih segmenata ili palindroma u tekstu, zajedničkih segmenata za dve ili veći broj niski i slično. Ove strukture podataka imaju širok spektar primena: koriste se u indeksiranju tekstova (u strukturama podataka za indeksiranje, poput ESA[1]), kompresiji teksta (sufiksni nizovi igraju važnu ulogu u BWT algoritmu kojim se karakteri niske reorganizuju tako da se popravi stepen kompresije[2]), kompresiji podataka (u LZW algoritmu za kodiranje i dekodiranje nizova simbola[3]) i bioinformatici (za sastavljanje genoma poravnanjem i spajanjem manjih fragmenata DNK).

Sufiksni niz

Neka je data niska s dužine n. Označimo sa $s[i..j]$ segment niske s od pozicije i do pozicije j, pri čemu indeksiranje elemenata niske počinje od 1. Specijalno, sa $S_i = s[i..n]$ označićemo *sufiks* niske s koji počinje na poziciji i.

Sufiksni niz (eng. suffix array) SA niske s je niz celobrojnih vrednosti, kojim se zadaju indeksi početaka leksikografski sortiranih sufiksa niske s:

$$S_{SA[1]} < S_{SA[2]} < \cdots < S_{SA[n]}.$$

Drugim rečima, ako je $SA[i] = j$, onda je sufiks S_j i-ti po redu sufiks u sortiranom nizu sufiksa $S_1, S_2, \ldots, S_n$. Reći ćemo i da sufiks S_j ima rang $i = rang[j]$, gde se podrazumeva da se u nizu $rang$ na j-toj poziciji nalazi rang sufiksa S_j. Primetimo da su nizovi SA i $rang$ 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 $s[8..11]$.

[1] http://docs.seqan.de/seqan1/1.4.0/SPEC_Index_Esa.html
[2] https://en.wikipedia.org/wiki/Burrows-Wheeler_transform
[3] https://en.wikipedia.org/wiki/Lempel-Ziv-Welch

- 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

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

- 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	0
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, niz rangova i niz najdužih zajedničkih prefiksa niske "mississippi".

Primetimo da se u sufiksnom nizu ne pamte kompletni sufiksni niske, već se sufiksi pamte implicitno, tako što se čuvaju samo indeksi njihovih početaka. Ovo omogućava da izbegnemo kvadratnu prostornu složenost: za nisku dužine n sufiksni niz zauzima $O(n)$ prostora, te je ova struktura prostorno efikasna. 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 se stoga može pored njenog sufiksnog niza SA pridružiti i niz najdužih zajedničkih prefiksa, 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 najdužih zajedničkih prefiksa 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

Mnogi zadaci nad niskama se mogu efikasno izvesti ako se prethodno formira njihov sufiksni niz i niz najdužih zajedničkih prefiksa. Prikazaćemo nekoliko takvih primera primene.

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.

Dati problem se može formulisati i nešto drugačije, u terminima koji su pogodni za primenu sufiksnih nizova: pronaći najduži zajednički prefiks između dva proizvoljna sufiksa date niske. Primetimo 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 Data je 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

Slika 1: Najduži ponovljeni segment u genomu.

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 segmente koji su ujedno i 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:

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

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 tako što nadovežemo nisku s na samu sebe i za nisku ss odredimo leksikografski minimalni segment dužine n. S obzirom na to da je njihov redosled određen redosledom sufiksa ove niske, u te svrhe možemo iskoristiti sufiksni niz u kome ćemo tražiti prvi sufiks dužine bar n.

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, ako se p javlja u s, neka pojava reči p unutar teksta s može pronaći binarnom pretragom opsega $[1, n]$. Indeksi svih ostalih pojava (ako ih ima) nalaze se na susednim 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 u najgorem slučaju $O(m)$. 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. Naime, svako naredno pojavljivanje se može naći u konstantnom vremenu, korišćenjem niza najdužih zajedničkih prefiksa.

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 rečju p (što se može zaključiti analizom niza LCP), unutar s nema drugih pojava reči p.

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 najdužih zajedničkih prefiksa niske "prestolonaslednikovica".

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 niti unutar niske s_1 niti niske s_2. Pritom se svakom sufiksu pridružuje podatak da li je njegov početak unutar s_1 ili s_2.

U sortiranom nizu sufiksa objedinjene niske potrebno je pronaći par uzastopnih elemenata 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).

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 u s_1s_2 razmatrati nisku "caab", koja jeste segment niske s_2, međutim nije segment niske s_1.

Ako se koristi i niz najdužih zajedničkih prefiksa združene niske $s_1\$s_2\#$ (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. Linearnim prolaskom kroz niz LCP može se utvrditi da je najveća vrednost koja se javlja u nizu $LCP[27] = 4$. 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".

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. Ovaj problem smo već rešavali.

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.

	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	$ko...#	23	1	9	nasledniковica$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#".

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. Na osnovu sufiksnog niza i niza LCP može se algoritmom linearne složenosti formirati sufiksno drvo niske, što zatim omogućuje i slične primene sufiksnog drveta.

Naivni algoritam Direktan način da konstruišemo sufiksni niz niske s je smestiti svih n sufiksa 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 u najgorem slučaju 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. Efikasniji algoritam za konstrukciju sufiksnog niza može se dobiti na sledeći način: održavamo sve sufikse niske s 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 odgovarajućim brojem karaktera \$, 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 (eng. counting sort). 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, ali ih zbog njihove kompleksnosti mi ovde nećemo dalje analizirati.