1 Увод

Суфиксни низ и суфиксно стабло дате ниске су структуре података помоћу којих се ефикасно могу решавати проблеми као што је тражење речи у тексту, тражење понављања или палиндрома у тексту, тражење заједничке подниске за два или више текстова и слично.

Нека је задата ниска s дужине n. Нека је $s[i..j]$ део те ниске са индексима од i до j, $1 \leq i \leq j \leq n$. Суфиксни низ SA ниске s је низ индекса ниске у оквиру сортираног редоследа:

$$s[SA[1]..n] < s[SA[2]..n] < \cdots < s[SA[n]..n].$$

Пример. Нека је $s =$ mississippi. Сортирани низ суфикса ове ниске и њен суфиксни низ приказани су у следећој табели.

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

Суфиксно стабло је друга структура података за представљање интерне структуре ниске. Суфиксно стабло ниске s дужине n је коренско стабло за које важи:

- стабло има тачно n листова који су нумерисани бројевима $1, 2, \ldots, n$;
- сваки унутрашњи чвор има бар два сина;
- свака грана је означена непразном подниском ниске s;
- никоје два гране из истог чвора немају ознаке које почињу истим карактером;
- када се надовежу све ознаке од на путу од корена до листа са ознаком i, добија се суфикс $s[i..n]$ ниске s.

На слици 1 приказано је суфиксно стабло ниске xabxa\$.

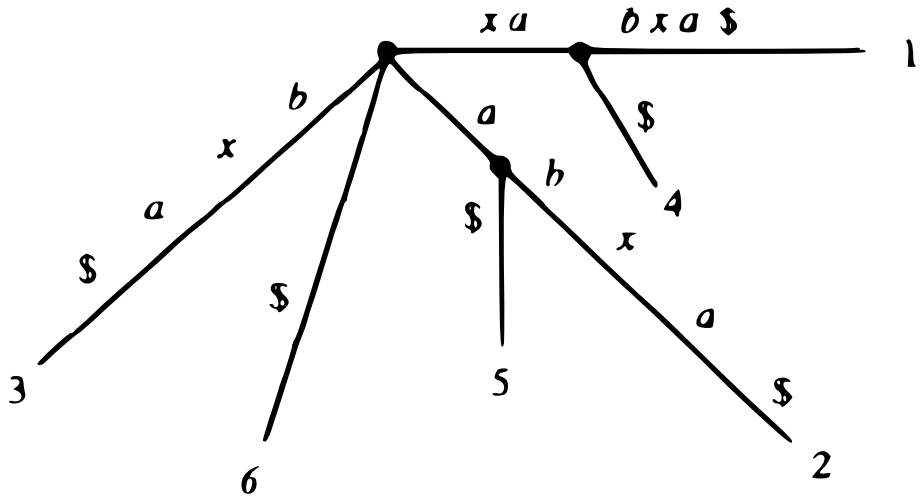

Figure 1: Суфиксно стабло ниске xabxa$.

Из дефиниције не следи постојање суфиксног стабла за произвољну ниску s. Проблем се појављује када се суфикс ниске s поклопи са префиксом другог суфикса ниске s, јер се тада пут који одговара првом суфиксу завршава унутар пута који одговара другом суфиксу, тј. пут који одговара првом суфиксу не завршава се у листу стабла. На пример, за ниску cxabxa пут који одговара суфиксу xabxa завршава се у листу, али не и пут који одговара суфиксу xa. Да би се избегао овај проблем, претпостављамо да се последњи карактер ниске s не појављује нигде више унутар ниске. Један начин да се проблем реши је да се ниска продужи знаком који се не појављује унутар ње (обично је то знак \$; претпоставља се da знак \$ лексикографски претходи свим осталим знацима).

Ако би се у оквиру суфиксног стабла уз сваку грану чувала њена комплетна ознака, онда би просторна сложеност стабла за ниску дужине n била у најгорем случају $O(n^2)$. Међутим, произвољна подниска $s[i..j]$ познате ниске s одређена са два броја, индексима i, j. Пошто је број грана у суфиксном стаблу ниске дужине n највише $O(n)$ (зашто?), заменом ознака на гранама са по два броја постиже се да је простор који заузима суфиксно стабло $O(n)$

2 Алгоритам Каркаинена-Сандерса за конструкцију суфиксног низа

Због компатибилности са C програмом и због тога што се користе остаци индекса при дељењу са 3, у овом одељку се претпоставља да су елементи

улазне ниске нумерисани почевши од индекса 0.

Алгоритам чији су аутори Каркаинен и Сандерс [1] је заснован на специфичном разлагању (divide-and-conquer):

1. Конструише се суфиксни низ за суфиксне од позиција $i \mod 3 \neq 0$ (индекси $3k+1$, $3k+2$). Ово се постиже свођењем на рекурзивну конструкцију суфиксног низа ниске чија је дужина једнака две трећине дужине полазне ниске.

2. Коришћењем овог суфиксног низа сортирају се преостали суфикси који почињу од позиција $i \mod 3 = 0$ (индекси $3k$).

3. Обједињавањем ова два низа суфикса формира се суфиксни низ полазног низа.

Испоставља се да је овакав приступ једноставнији од разлагања суфикса на оне са парним и непарним почетним индексима. Разлог за ово је чињеница да је последњи корак, обједињавање два сортирана подниза суфикса, у овој варијанти скоро тривијалан. На пример, да би се упоредили суфикси који почињу од i и j, при чему је i облика $3k$, а j облика $3k+1$, најпре се упоређују њихови први знаци, па ако су они једнаки упоређују се суфикси који почињу од индекса $i+1$ и $j+1$, чији међусобни однос је познат на основу првог корака, јер су они облика редом $3k+1$, односно $3k+2$.

Користе се ознаке $[a,b] = a,\ldots,b$ и $s[a,b] = [s[a],\ldots,s[b]]$ за ниску или низ s. Слично, $[a,b) = [a,b-1]$ и $s[a,b) = s[a,b-1]$. Оператор $\bullet$ означава конкатенацију ниски. Посматрајмо ниску $s = s[0,n)$ над азбуком $\Sigma = [1,n]$. Суфиксни низ SA ниске s садржи суфиксе $S_i = s[i,n)$ у сортираном редоследу, тј. ако је $SA[i] = j$ онда суфикс S_j има ранг $i+1$ међу нискама $S_0,\ldots,S_{n-1}$. Да би се избегло посебно разматрање специјалних случајева, претпоставља се да је дужина ниске n дељива са 3, као и да за све ниске α важи $\alpha[|\alpha|] = \alpha[|\alpha|+1] = 0$. Програмска реализација алгоритма у наставку садржи преостале детаље алгоритма. На слици 1 приказан је резултат примене алгоритма на ниску mississippi.

[1] Juha Kärkkäinen, Peter Sanders: Simple Linear Work Suffix Array Construction, Lecture Notes in Computer Science 2719: 943–955

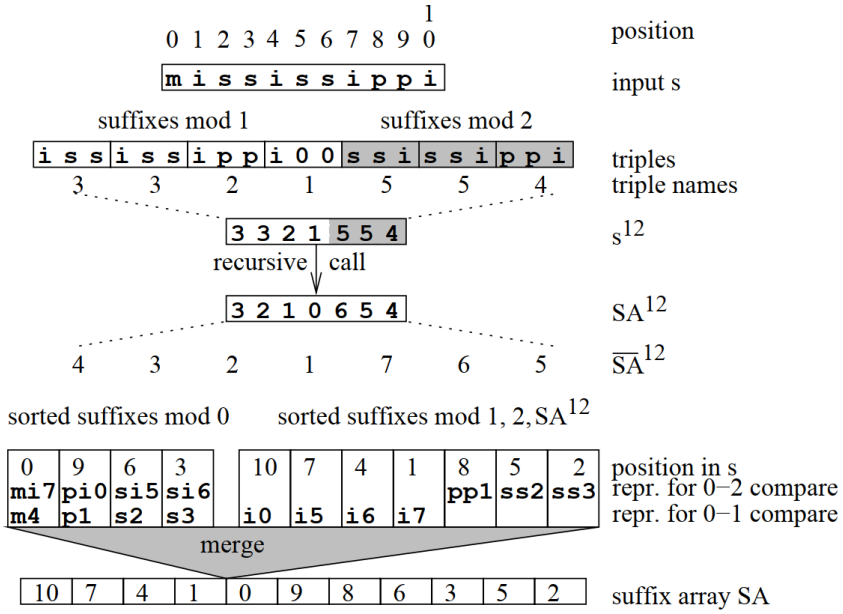

Figure 2: Резултат примене алгоритма Карканина-Сандерса на ниску $s = mississippi$.

Размотримо сада овај пример мало детаљније.

Корак 1. Задата ниска дужине 11 $s = mississippi$ са одговарајућим индексима:

0	1	2	3	4	5	6	7	8	9	10
m	i	s	s	i	s	s	i	p	p	i

Суфикси ове ниске са индексима облика $3k + 1$, $3k + 2$ обухваћени су конкатенацијом $s[1, 12] \bullet s[2, 10]$:

iss	iss	ipp	i00	ssi	ssi	ppi
1	4	7	10	2	5	8

Прва компонента допуњена је са две нуле да би њена дужина била дељива са три и да би од друге компоненте била раздвојена бар једном нулом. Добијена конкатенација раздваја се на тројке слова, које се затим сортирају вишеструким разврставањем (по трећем, другом, и на крају по првом слову).

Разврставање по трећем слову тројке:

група	0	i			p	s	
тројка	i00	ssi	ssi	ppi	ipp	iss	iss
индекс	10	2	5	8	7	1	4

Разврставање по другом (средњем) слову тројке:

група	0	p		s			
тројка	i00	ppi	ipp	ssi	ssi	iss	iss
индекс	10	8	7	2	5	1	4

Разврставање по првом слову тројке, последње:

група	i				p	s	
тројка	i00	ipp	iss	iss	ppi	ssi	ssi
индекс	10	7	1	4	8	2	5

Тиме је завршено сортирање тројки које чине конкатенацију. Пошто постоје једнаке тројке са различитим индексима, за сада се не зна поредак суфикса који почињу таквим тројкама. Због тога се тројке рангирају, тј. додељују им се имена (рангови):

тројка	i00	ipp	iss	iss	ppi	ssi	ssi
индекс	10	7	1	4	8	2	5
име	1	2	3	3	4	5	5

Пошто је међусобни однос тројки исти као однос одговарајућих имена, сортирање суфикса облика $3k{+}1$, $3k{+}2$ своди се на сортирање низа одговарајућих тројки:

	суфикси $3k+1$				суфикси $3k+2$		
група	iss	iss	ipp	i00	ssi	ssi	ppi
име	3	3	2	1	5	5	4
индекс имена	0	1	2	3	4	5	6
индекс тројке	1	4	7	10	2	5	8

Рекурзивним покретањем алгоритма за ниску коју чине имена тројки

0	1	2	3	4	5	6
3	3	2	1	5	5	4

добија се суфиксни низ те ниске

0	1	2	3	4	5	6
3	2	1	0	6	5	4

Овај суфиксни низ директно се преводи у редослед суфикса облика $3k+1$, $3k+2$ полазне ниске

суфикси $3k+1$, $3k+1$	i00	ipp	iss	iss	ppi	ssi	ssi
имена	1	2	3	3	4	5	5
суфиксни низ имена	3	2	1	0	6	5	4
сортирани суфикси $3k+1$, $3k+1$	10	7	4	1	8	5	2

Корак 2. Да би се сортирали суфикси облика $3k$, они се замењују конкатенацијом првог слова и индексом одговарајућег наредног суфикса (облика

$3k+1$): $m2$, $s4$, $s7$, $p10$. Лексикографски поредак суфикса облика $3k+1$ зна се из претходне табеле, што је еквивалентно првом кораку сортирања вишеструким разврставањем (по најдеснијој, тј, другој компоненти): $p10$, $s7$, $s4$, $m2$. Други корак сортирања вишеструким разврставањем (по првој компоненти, првом слову) изводи се разврставањем на групе према првом слову. Постоје три такве групе, за слова p, s, m, односно лексикограским редом m ($m2$), p ($p10$), s ($s7$, $s4$). Према томе, лексикографски редослед суфикса облика $3k$ је $m2, p10, s, s4$, тј.

тројка	mis	sis	sip	pi0
индекс	0	9	6	3

Корак 3. Обједињавање два сортирана низа суфикса добијена у Кораку 1. и Кораку 2.

Да би се суфикс облика $3k$ упоредио са суфиксом облика $3k+1$, ови суфикси се представљају у облику конкатенације првог знака и наредног суфикса, који је облика $3k+1$, односно облика $3k+2$. Да би се суфикс облика $3k$ упоредио са суфиксом облика $3k+2$, ови суфикси се представљају у облику конкатенације прва два знака и наредног суфикса, који је облика $3k+2$, односно облика $3k+4$, тј. $3k+1$.

0	9	6	3		10	7	4	1	8	5	2
mi7	pi0	si5	si6						pp1	ss2	ss3
m4	p1	s2	s3		i0	i5	i6	i7			

Ово омогућује да се ова два сортирана низа суфикса обједине и да се добије суфиксни низ полазна ниске

10	7	4	1	0	9	8	6	3	5	2

Прелазимо на детаљнији опис алгоритма у општем случају.

Корак 1. сортирање суфикса S_i са индексима облика $3k+1$, $3k+2$, троши највише времена (у оквиру анализе сложености алгоритма показаћемо да је то око $2/3$ времена). Корак почиње додељивањем лексикографских имена $s'_i \in [1, 2n/3]$ тројкама $s[i, i+2]$ са индексима облика $3k+1$, $3k+2$, тј. бројева са особином да је $s'_k \leq s'_l$ ако и само ако за одговарајуће индексе важи $s[i, i+2] \leq s[j, j+2]$. Ово се може извршити за линеарно време применом вишеструког разврставања (radix sort) низа тројки знакова — ако је тројка $s[i, i+2]$ k-та различита тројка у сортираном низу тројки, онда јој се додељује име $s'_i = k$. Ако све тројке добију различита лексикографска имена,

онда је тај корак, сортирање суфикса са индексима облика $3k+1$, $3k+2$, завршен. У противном, рекурзивно се одређује суфиксни низ SA^{12} ниске

$$s^{12} = [s'_i : i \mod 3 = 1] \bullet [s'_i : i \mod 3 = 2].$$

Запажа се да број лексикографских имена никад није већи од броја знакова у нисци s^{12}, па величина азбуке у рекурзивном позиву никад није већа од дужине ниске. Рекурзивно израчунати суфиксни низ SA^{12} обезбеђује неопходан редослед суфикса $S_i = s[i, n)$ са индексима облика $3k+1$, $3k+2$. Специјално, запажа се да суфикс $s^{12}\left[\frac{i-1}{3}, \frac{n}{3}\right)$ за $i \mod 3 = 1$ према лексикографском именовању одговара суфиксу $S_i = s[i, n) \bullet [0]$. Знакови 0 на крају s обезбеђују да знак $s^{12}[n/3-1]$ буде јединствен у ниски s^{12}, па није проблем што s^{12} садржи конкатенацију два суфикса s. Слично, $s^{12}\left[\frac{n+i-2}{3}, \frac{2n}{3}\right)$ за $i \mod 3 = 2$ одговара суфиксу $S_i = s[i, n) \bullet [0, 0]$.

Корак 2. је једноставан. Суфикси S_i са индексима облика $3k$ сортирају се сортирањем парова $(s[i], S_{i+1})$. Пошто је поредак суфикса S_{i+1} већ имплицитно одређен у оквиру SA^{12}, довољно је (стабилно, без непотребне промене редоследа) сортирати суфиксе из $SA^{12}[j]$ који одговарају суфиксима S_{i+1} са индексима облика $3k+1$, узимајући у обзир и претходни знак $s[i]$. Ово сортирање може се извршити за линеарно време применом једног пролаза вишеструког разврставања.

Корак 3. је такође једноставан. Потребно је објединити два низа суфикса да би се добио комплетан суфиксни низ SA. Да би се упоредио суфикс S_j за који је j облика $3k$ са суфиксом S_i за који је i облика $3k+1$ или $3k+2$, потребно је разликовати два случаја

- Ако је i облика $3k+1$, онда се суфикс S_i може представити у облику $(s[i], S_{i+1})$, а S_j у облику $(s[j], S_{j+1})$. Пошто је $i+1$ облика $3k+2$, а $j+1$ облика $3k+1$, међусобни однос S_{j+1} и S_{i+1} може се одредити на основу њихових позиција у SA^{12}. Те позиције могу се одредити за константно време ако се претходно израчуна помоћни низ $\overline{SA}^{12}$ такав да је $\overline{SA}^{12}[i] = j+1$ ако је $SA^{12}[j] = i$. Ово је уствари само специјални случај лексикографског именовања ($\overline{SA}^{12}$ је инверзни суфиксни низ за SA^{12}).
- Слично, ако је i облика $3k+2$, упоређују се тројке $(s[i], s[i+1], S_{i+2})$ и $(s[j], s[j+1], S_{j+2})$ замењујући S_{i+2} и S_{j+2} њиховим лексикографским именима у SA^{12}.

Сложеност алгоритма лако је одредити на основу мастер теореме. Сложеност задовољава диференцну једначину $T(n) = O(n) + T(\lfloor 2n/3 \rfloor)$, чије је решење $T(n) = O(n)$.

Ако претпоставимо да је време извршавања алгоритма приближно $T(n) = cn$, онда је време потребно за рекурзивно сортирање индекса облика $3k+1$ и $3k+2$ приближно $T(\lfloor 2n/3 \rfloor)) = c\lfloor 2n/3 \rfloor)$, односно око $2/3$ времена потребног за одређивање суфиксног низа комплетне ниске.

Реализација (C++) овог алгоритма је у прилогу.

3 Одређивање низа LCP на основу суфиксног низа

Најдужи заједнички префикс (НЗП) или LCP (longest common preffix) за две ниске s_1 and s_2 је дужина $lcp(s_1, s_2)$ најдуже ниске која је префикс обе ниске. За фиксирану ниску s нека $lcp(i, j)$ означава НЗП за суфиксе са индексима i, j. Нисци s може се поред њеног суфиксног низа SA придружити и низ lcp, чији је i-ти члан једнак $lcp[i] = lcp(S_{SA[i]}, S_{SA[i+1]})$, $i = 1, 2, \ldots, n-1$. Другим речима, $lcp[i]$ је дужина НЗП i-тог и $(i+1)$-ог суфикса ниске у сортираном редоследу; видети пример lcp ниске

Пример. За ниску mississippi (видети табелу из увода) је $lcp(3) = 4$ Заиста $SA[3] = 5$, $S_5 = issippi$, $SA[4] = 2$, $S_3 = ississippi$. Због тога је $lcp[3] = lcp(issippi, ississippi) = |issi| = 4$. у тачки 3.2.

Низ НЗП (LCP) дате ниске s може се одредити алгоритмом линеарне сложености који је предложило неколико аутора 2001. године.[2] Посматрајмо два суфикса чији су индекси у суфиксном низа узастопни. Нека су њихови индекси у суфиксном низа i_1 и $i_1 + 1$, при чему је $SA[i_1] < n$. Ако је дужина њиховог најдужег заједничког префикса већа од нуле, онда се после брисања њиховог првог знака (истог!) из оба суфикса добијају суфикси чији је међусобни лексикографски однос исти. Поред тога, види се да је дужина њиховог НЗП већа или једнака од дужине претходног НЗП умањене за један.

Пример.

- У нисци mississippi суфикси $S_5 = issippi$ $S_2 = ississippi$ су узастопни у суфиксном низу (трећи и четврти, $SA[4] = 2$, $SA[3] = 5$). Дужина њиховог најдужег заједничког префикса је $lcp[3] = 4$. Када им се

[2] Kasai T., Lee G., Arimura H., Arikawa S., Park K. (2001) Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. In: Amir A. (eds) Combinatorial Pattern Matching. CPM 2001. Lecture Notes in Computer Science, vol 2089. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-48194-X_17

обришу први знаци, добијају се суфикси $S_6 = ssippi$ $S_3 = ssissippi$ који су такође узастопни у суфиксном низу (десети и једанаести, $SA[10] = 6$, $SA[11] = 3$). Због тога је $lcp[10] = lcp[3] - 1$.

- Суфикси $S_{10} = pi$ $S_9 = ppi$ су такође узастопни у суфиксном низу (шести и седми, $SA[6] = 10$, $SA[7] = 9$). Дужина њиховог најдужег заједничког префикса је $lcp[6] = 1$. Када им се обришу први знаци, добијају се суфикси $S_{11} = i$ $S_{10} = pi$ који НИСУ узастопни у суфиксном низу (први и шести, $SA[1] = 11$, $SA[6] = 10$). Због тога је $1 = lcp[1] \geq lcp[6] - 1 = 0$; запажамо да у овом примеру важи строга неједнакост $lcp[1] = 1 > lcp[6] - 1 = 0$

- Суфикси $S_{11} = i$ $S_8 = ippi$ су такође узастопни у суфиксном низу (први и други, $SA[1] = 11$, $SA[2] = 8$). Дужина њиховог најдужег заједничког префикса је $lcp[1] = 1$. Када се суфиксу $S_{11} = i$ обрише први знак, добијају се празан суфикс, кога нема у нашем сортираном редоследу суфикса. Због тога се вредност $lcp[1] = 1$ не може искористити на описани начин за одређивање неке друге вредности $lcp[.]$.

Посматрајмо ниску која се добија од суфикса са индексом i у суфиксном низу брисањем њеног првог знака. Та ниска (сем у случају кад је то ниска S_n) је такође суфикс; нека је индекс тог суфикса у суфиксном низу i_2. Посматрајмо најдужи заједнички префикс суфикса са индексима i_2 и $i_2 + 1$ у суфиксном низу. Види се да је дужина њиховог најдужег заједничког префикса једнака бар $lcp[i] - 1$. Ова чињеница је у вези са особином низа НЗП да је $lcp(i, j) = \min(lcp_i, lcp_{i+1}, \ldots, lcp_{j-1})$ (докажите ову чињеницу).

Следи опис алгоритма за одређивање низа LCP дате ниске s. У оквиру алгоритма користи се помоћни низ $rang[n]$, који садржи индекс суфикса S_i у сортираном редоследу свих суфикса. Најпре се директно (упоређивањем једног по једног знака) израчунава $lcp[i]$ са индексом $i = rang[1]$. Затим се пролазе остали суфикси оним редом којим се појављују у нисци, тј. израчунава се $lcp[rang[i]]$, $i = 2, 3, \ldots, n - 1$ на једноставан начин (упоређивањем једног по једног знака два суфикса) АЛИ полазећи од знака са индексом $lcp[rang[i - 1]] - 1$.

Сложеност овог алгоритма је $O(n)$, јер се у сваком кораку алгоритма тренутна дужина најдужег заједничког префикса смањује највише за 1 (изузев у случају кад је $rang[i] = n - 1$).

Пример. Размотримо ниску mississippi са почетка, за коју је претходно формиран суфиксни низ, пермутација

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

Низ *rang* је инверзна пермутација ове пермутације.

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

Овај низ контролише редослед одређивања елемената низа lcp. Полази се од $lcp[rang[1]] = lcp[5]$; упоређују се суфикси са индексима $SA[5] = 1$ и $SA[5+1] = 10$, тј. суфикси mississippi и pi. Први знаци ова два суфикса се разликују, па је $lcp[5] = 0$. Даље се редом одређују вредности $lcp[rang[i]]$, $i = 2, 3, \ldots, 10$. На пример, $lcp[rang[5]] = lcp[3]$ добија се упоређивањем суфикса issippi и ississippi са индексима редом $SA[3] = 5$ и $SA[3+1] = 2$; упоређујући редом прве, друге, треће, четврте и пете знакове ова два суфикса, добија се $lcp[3] = 4$. Приликом одређивања наредне вредности $lcp[rang[6]] = lcp[10]$ потребно је упоредити суфиксе са индексима $SA[10] = 6$ и $SA[10+1] = 3$, тј. суфиксе ssippi и ssissippi, при чему се са упоређивањем почиње од знака са редним бројем $lcp[3] - 1 = 3$, јер се зна да су прва три знака ова два префикса једнаки.

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

Програм:

```
vector<int> kasai(string s, vector<int> sa)
{
    int n=s.size(),k=0;
    vector<int> lcp(n,0);
    vector<int> rank(n,0);

    for(int i=0; i<n; i++) rank[sa[i]]=i; // одређивање низа rang[]

    for(int i=0; i<n; i++, k?k--:0) // k је почетна вредност за lcp[rank[i]]
    {
        if(rank[i]==n-1) {k=0; continue;} // ако је то најдужи суфикс,
                                          // нема наредног суфикса
        int j=sa[rank[i]+1]; // индекс наредног суфикса
        while(i+k<n && j+k<n && s[i+k]==s[j+k]) k++; // петља почиње од k,
                                                     // tj. од претходног lcp[]
        lcp[rank[i]]=k;
    }
    return lcp;
}
```

4 Примене суфиксног низа

Многе операције са нискама ефикасно се извода ако се претходно формира њихов суфиксни низ и низ lcp. Приказаћемо два таква примера примене: тражење речи у тексту и тражење најдуже заједничке подниске за две дате ниске. На основу суфиксног низа и низа lcp може се алгоритмом линеарне сложености формирати суфиксно стабло ниске, што затим омогућује сличне примене суфиксног стабла.

4.1 Тражење речи у тексту

Потребно је одредити све појаве речи P унутар текста S. Проблем је лако решити ако се одреди суфиксни низ текста S: свака појава P унутар S је почетак неког префикса S. Суфиксни низ текста S одређује сортирани редослед суфикса S, па се прва појава P унутар S може пронаћи бинарном претрагом опсега $[1..|S|]$. Индекси свих осталих појава (ако их има) налазе се на наредним узастопним позицијама у суфиксном низу.

Сложеност алгоритма је $O(n \log n)$, јер је сложеност формирања суфиксног низа $O(n)$, а сложеност сваког од $O(\log n)$ упоређивања је $O(n)$. Упоређивања се могу извршити ефикасније ако се користи низ LCP текста S.

Пример. Пронаћи све појаве речи $P = lednik$ унутар текста $s = prestolonaslednikovica$.

Суфиксни низ текста s (са сортираним редоследом суфикса) приказан је у следећој табели.

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

Бинарна претрага започиње упоређивањем $P = lednik$ са $S[SA[\lceil \frac{1+22}{2} \rceil]] = S[SA[12]] = naslednikovica$. Пошто је $naslednikovica > lednik$, наставља се са бинарном претрагом опсега $[1, 11]$. После неколико корака проналази се да је индекс прве појаве P унутар s једнак 12. Пошто наредни суфикс $s[13] = ednikovica$ не запоиње са P, унутар s нема других појава P.

4.2 Тражење најдуже заједничке подниске за две дате ниске

Потребно је одредити најдужу заједничку подниску две дате ниске s_1 и s_2. Проблем се решава тако што се најпре формира суфиксни низ ниске $s = s_1\$s_2\#$, при чему су знаци \$ и # знаци који се не појављују унутар s_1 или s_2. При томе се сваком суфиксу претходно придружује податак да ли је његов почетак унутар s_1 или s_2. У сортираном низу суфикса ове ниске потребно је пронаћи пар узастопних чланова таквих да

- одговарајући суфикси не припадају истом низу, и
- дужина њиховог најдужег заједничког префикса је највећа у односу на све остале овакве парове индекса (оваквих парова префикса са најдужим заједничким префиксом може бити више).

Ако се користи и низ LCP здружене ниске s (сложеност његовог формирања је $O(n)$; овде је $n = |s_1| + |s_2|$), онда је сложеност алгоритма линеарна, $O(n)$.

Пример. Пронаћи најдужу заједничку подниску ниски

$$s_1 = prestolonaslednikovica$$

и

$$s_2 = kolonizacija.$$

Суфиксни низ SA здружене ниске $s = s_1\$s_2\#$ (са сортираним редоследом суфикса) приказан је у следећој табели.

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

Највећи број у колони $LCP[i]$ је $LCP[26] = 4$. Поред тога, суфикси

$$s[SA[26]] = olonaslednikovica\$kolonizacija\#$$

и

$$s[SA[27]]olonizacija\#$$

са заједничким префиксом *olon* дужине четири потичу из различитих ниски. Према томе, најдужа заједничка подниска ове две ниске је ниска *olon*.

4.3 Проналажење најдужег палиндрома у задатој нисци

Нека је задата ниска s и нека је потребно пронаћи најдужи палиндром унутар ње. Решавање овог проблема своди се на тражење најдуже заједничке подниске ниски s и s', где је s' ниска која се од s добија "читањем уназад".

4.4 Формирање суфиксног стабла

Ако се зна суфиксни низ SA и LCP низ lcp ниске s дужине n, суфиксно стабло те ниске може се конструисати алгоритмом сложености $O(n)$ на основу следеће идеје: полазећи од парцијалног суфиксног стабла за лексикографски најмањи суфикс, уметати у њега остале суфиксе редом којим се на њих наилази лексикографским обиласком суфиксног стабла.

Нека је ST_i парцијално суфиксно стабло добијено уметањем првих i лексикографски најмањих суфикса, $0 \leq i \leq n$. Нека је даље $d(v)$ дужина конкатенације ознака свих грана у стаблу ST_i на путу од корена до чвора v. Полази се од ST_0, стабла које има само корен. Да би се у стабло ST_i убацио суфикс са индексом $SA[i]$, $1 \leq i \leq n$, прелази се пут од последњег уметнутог листа ка корену, до првог чвора v за који је $d(v) \leq lcp[i]$. Могућа су два случаја:

- $d(v) = lcp[i]$ **[уметање нове гране из постојећег чвора]**: tada је дужина конкатенације ознака грана на путу од корена до чвора v једнака $lcp[S_{SA[i-1]}, S_{SA[i]} = lcp[i-1]$. У том случају суфикс са ознаком $SA[i]$ се умеће као нови лист x повезан са чвором v граном (v, x) са ознаком $S_{SA[i]]+lcp[i]}$. Другим речима, ознака додате гране се састоји од преосталих знакова суфикса $SA[i]$ који нису део конкатенације ознака грана на путу од корена до чвора v. На тај начин формирано је парцијално суфиксно стабло ST_i.

- $d(v) < lcp[i]$ **[нова грана из чвора уметнутог у постојећу грану]**: тада је конкатенација ознака грана на путу од корена до чвора v има мање знакова од $lcp[S_{SA[i-1]}, S_{SA[i]} = lcp[i-1]$ и знакови који недостају до чвора v садржани су на почетку ознаке *најдеснији* (последње уметнуте) гране (v, w) која излази из чвора v. Због тога се грана (v, w) мора новим унутрашњим чвором y раздвојити на две гране (v, y) и (y, w). При томе гране (v, y) и (y, w) као ознаке добијају одговарајући почетак, односно завршетак ознаке уклоњене гране (v, w). Прецизније,

  1. Уклонити грану (v, w).
  2. Додати нови унутрашњи чвор y и нову грану (v, y) са ознаком $s[SA[i-1]]+d(v), SA[i-1]+lcp[i]-1]$. Ознака нове гране састоји се од

знакова који недостају на најдужем заједничком префиксу суфикса $S_{SA[i-1]}$ и $S_{SA[i]}$. Према томе, конкатенације ознака грана на путу од корена до чвора y једнака је најдужем заједничком префиксу суфикса $S_{SA[i-1]}$ и $S_{SA[i]}$.

3. Повезати чвор w са новокреираним унутрашњим чвором y граном (y, w) са ознаком $s[SA[i-1]] + lcp[i], SA[i-1] + d(v) - 1]$. Нова ознака састоји се од преосталих знакова обрисане гране (v, w) који нису укључени у ознаку гране (v, y).

4. Уметнути чвор са ознаком $SA[i]$ као нови лист x и повезати га са новим унутрашњим чвором y граном (y, x) са ознаком $S_{[SA[i]]+lcp[i]}$. Према томе, ознака гране (y, x) састоји се од преосталих знакова суфикса $S_{SA[i]}$, који нису укључени у конкатенацију ознака грана на путу од корена до чвора v.

5. На тај начин формирано је парцијално суфиксно стабло ST_i.

Пример. На следећој слици илустрован је овај поступак формирања суфиксног стабла за ниску mississippi. У опису алгоритма претпоставља се да се суфиксно стабло формира слева удесно, тј. да се нови листови додају повезивањем са чвором на тренутно најдеснијој грани. Због једноставнијег цртања стабла, овде се нови листови каче се на најнижу уместо најдеснију грану, односно суфиксно стабло се формира одозго наниже.

i	$SA[i]$	$s[1..SA[i]]$	$LCP[i]$	суфиксно стабло
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

Лако је видети да је амортизована сложеност овог алгоритма $O(n)$. Чворови који се пролазе у кораку i на путу ка корену најдеснијим путем у стаблу ST_i (осим последњег чвора v) уклањају се из најдеснијег пута кад се $A[i]$ дода у стабло као нови лист. Ови чворови се никад не пролазе поново у наредним корацима $j > i$. Према томе, укупан број чворова који се пролазе у току извршења алгоритма је највише $2n$.

5 Примене суфиксног стабла

Исти проблеми sa које смо видели да се могу решавати применом суфиксног низа (тражење речи у тексту, тражење најдуже заједничке подниске две или више ниски, тражење најдужег палиндрома у нисци) могу се (још једноставније) решавати применом суфиксног стабла.

6 Прилог: програмска реализација алгоритма Каркаинена-Сандерса

```
inline bool leq(int a1, int a2, int b1, int b2)
// lexicographic order
{
    return(a1 < b1 || a1 == b1 && a2 <= b2);
} // for pairs

inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3)
{
    return(a1 < b1 || a1 == b1 && leq(a2,a3, b2,b3));
} // and triples

// stably sort a[0..n-1] to b[0..n-1] with keys in 0..K from r
static void radixPass(int* a, int* b, int* r, int n, int K)
{// count occurrences
    int* c = new int[K + 1]; // counter array
    for (int i = 0; i <= K; i++) c[i] = 0; // reset counters
    for (int i = 0; i < n; i++) c[r[a[i]]]++; // count occurrences
    for (int i = 0, sum = 0; i <= K; i++) // exclusive prefix sums
    {
        int t = c[i]; c[i] = sum; sum += t;
    }
    for (int i = 0; i < n; i++) b[c[r[a[i]]]++] = a[i]; // sort
    delete [] c;
}
```

```
// find the suffix array SA of s[0..n-1] in {1..K}^n
// require s[n]=s[n+1]=s[n+2]=0, n>=2
void suffixArray(int* s, int* SA, int n, int K)
{
    int n0=(n+2)/3, n1=(n+1)/3, n2=n/3, n02=n0+n2;
    int* s12 = new int[n02 + 3]; s12[n02]= s12[n02+1]= s12[n02+2]=0;
    int* SA12 = new int[n02 + 3]; SA12[n02]=SA12[n02+1]=SA12[n02+2]=0;
    int* s0 = new int[n0];
    int* SA0 = new int[n0];
    // generate positions of mod 1 and mod 2 suffixes
    // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1
    for (int i=0, j=0; i < n+(n0-n1); i++)
        if (i%3 != 0)
            s12[j++] = i;
    // lsb radix sort the mod 1 and mod 2 triples
    radixPass(s12 , SA12, s+2, n02, K);
    radixPass(SA12, s12 , s+1, n02, K);
    radixPass(s12 , SA12, s , n02, K);

    // find lexicographic names of triples
    int name = 0, c0 = -1, c1 = -1, c2 = -1;
    for (int i = 0; i < n02; i++)
    {
        if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2)
        {
            name++;
            c0 = s[SA12[i]];
            c1 = s[SA12[i]+1];
            c2 = s[SA12[i]+2];
        }
        if (SA12[i] % 3 == 1)
        {
            s12[SA12[i]/3] = name;
        } // left half
        else
        {
            s12[SA12[i]/3 + n0] = name;
        } // right half
    }
```

```
    // recurse if names are not yet unique
    if (name < n02)
    {
        suffixArray(s12, SA12, n02, name);
        // store unique names in s12 using the suffix array
        for (int i = 0; i < n02; i++)
            s12[SA12[i]] = i + 1;
    }
    else // generate the suffix array of s12 directly
        for (int i = 0; i < n02; i++)
            SA12[s12[i] - 1] = i;

    // stably sort the mod 0 suffixes from SA12 by their first character
    for (int i=0, j=0; i < n02; i++)
        if (SA12[i] < n0)
            s0[j++] = 3*SA12[i];
    radixPass(s0, SA0, s, n0, K);

    // merge sorted SA0 suffixes and sorted SA12 suffixes
    for (int p=0, t=n0-n1, k=0; k < n; k++)
    {
#define GetI() (SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2)
        int i = GetI(); // pos of current offset 12 suffix
        int j = SA0[p]; // pos of current offset 0 suffix
        if (SA12[t] < n0 ? // different compares for mod 1 and mod 2 suffixes
            leq(s[i], s12[SA12[t] + n0], s[j], s12[j/3]) :
            leq(s[i],s[i+1],s12[SA12[t]-n0+1], s[j],s[j+1],s12[j/3+n0]))
        {// suffix from SA12 is smaller
            SA[k] = i; t++;
            if (t == n02) // done --- only SA0 suffixes left
                for (k++; p < n0; p++, k++)
                    SA[k] = SA0[p];
        }
            else
        {// suffix from SA0 is smaller
            SA[k] = j; p++;
            if (p == n0) // done --- only SA12 suffixes left
                for (k++; t < n02; t++, k++)
                    SA[k] = GetI();
        }
    }
        delete [] s12; delete [] SA12; delete [] SA0; delete [] s0;
}
```