

# 1 Суфиксни низ

Суфиксни низ дате ниске је структура података помоћу које 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:

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

**Пример.** Нека је  $s = \text{mississippi}$ . 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	

## 2 Алгоритам Каркаинен-Сандерс 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 <sup>1</sup> је заснован на специфичном разлагању (divide-and-conquer):

1. Конструира се суфиксни низ за суфиксне од позиција  $i \bmod 3 \neq 0$  (индекси  $3k + 1, 3k + 2$ ). Ово се постиже свођењем на рекурзивну

---

<sup>1</sup>Juha Kärkkäinen, Peter Sanders: Simple Linear Work Suffix Array Construction, Lecture Notes in Computer Science 2719: 943–955

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  $i \bmod 3 = 0$  (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  $[a, b] = a, \dots, b$  и  $s[a, b] = [s[a], \dots, s[b]]$  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  $S_0, \dots, S_{n-1}$ . Да би се избегло посебно разматрање специјалних случајева, претпоставља се да је дужина ниске  $n$  дељива са 3, као и да за све ниске  $\alpha$  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.

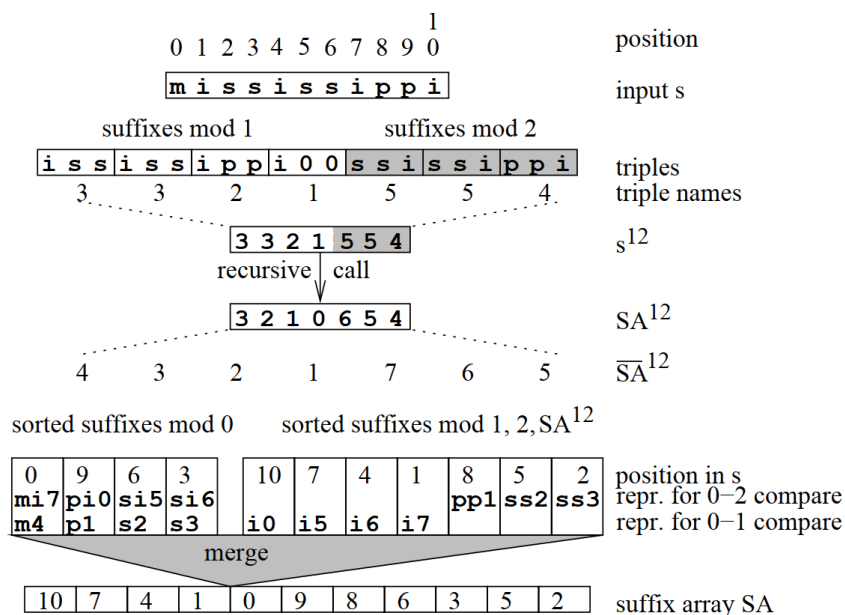


Figure 1: Резултат примене алгоритма Карканина-Сандерса на ниску  $s = mississippi$ .

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:

група име индекс имена индекс тројке	суфикси $3k+1$				суфикси $3k+2$		
	iss	iss	ipp	i00	ssi	ssi	ppi
	3	3	2	1	5	5	4
	0	1	2	3	4	5	6
	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 $3k+1$ , $3k+1$ имена	i00	ipp	iss	iss	ppi	ssi	ssi
	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 лексикографским 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. 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

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

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  $i \bmod 3 = 1$  према лексикографском именовану 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  $i \bmod 3 = 2$  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 диференцијалну једначину  $T(n) = O(n) + T(\lfloor 2n/3 \rfloor)$ , чије је решење  $T(n) = O(n)$ .

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

### 3 Одређивање низа LCP на основу суфиксног низа

Најдужи заједнички префикс (НЗП) или LCP (longest common prefix) за две ниске  $s_1$  and  $s_2$  је дужина  $LCP(s_1, s_2)$  најдуже ниске која је префикс обе ниске. Специјално, нека  $lcp(i, j)$  означава НЗП за суфиксе са индексима  $i, j$  фиксираних ниске  $s$ . Свакој нисци  $s$  може се поред њеног суфиксног низа  $SA$  придружити и низ  $lcp$ , такав да је  $lcp[i] = LCP(s[SA[i]], s[SA[i+1]])$ ,  $i = 1, 2, \dots, n-1$ . Другим речима,  $lcp[i]$  је дужина НЗП два узастопна суфикса ниске у сортираном редоследу; видети пример  $lcp$  ниске у тачки 3.2.

Низ LCP дате ниске  $s$  може се одредити алгоритмом линеарне сложености који је предложило неколико аутора 2001. године.<sup>2</sup>

Посматрајмо два суфикса чији су индекси у суфиксном низу узастопни. Нека су њихови индекси у суфиксном низу  $i_1$  и  $i_1 + 1$ . Ако је њихов НЗП већи од нуле, онда се види да после брисања првог знака (истог!) из оба суфикса добијају суфикси чији је међусобни однос исти. Поред тога, види се да је њихов НЗП тачно за један мањи од претходног НЗП.

Посматрајмо ниску која се добија од суфикса са индексом  $i$  (у суфиксном низу) брисањем његовог првог знака. Та ниска је такође суфикс; нека је индекс тог суфикса (такође у суфиксном низу)  $i_2$ . Посматрајмо НЗП суфикса са индексима  $i_2$  и  $i_2 + 1$ . Види се да је дужина њиховог НЗП једнака бар  $lcp[i] - 1$ . Ова чињеница је у вези са особином низа НЗП да је  $lcp(i, j) = \min(lcp_i, lcp_{i+1}, \dots, lcp_{j-1})$  (доказ ове чињенице је једноставан).

Следи опис алгоритма за одређивање низа LCP дате ниске  $s$ . У оквиру алгоритма користи се помоћни низ  $rang[n]$ , који садржи индекс (ранг) у оквиру суфиксног низа суфикса који почиње од индекса  $i$ . Најпре се израчунава НЗП суфикса са индексом  $rang[0]$ . Затим се пролазе остали суфикси оним редом којим се појављују у нисци и израчунава се  $lcp[rang[i]]$  на једноставан начин (упоређивањем једног по једног знака два суфикса) АЛИ полазећи од знака са индексом  $lcp[rang[i - 1]] - 1$ .

<sup>2</sup>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](https://doi.org/10.1007/3-540-48194-X_17)

Сложеност овог алгоритма је  $O(n)$ , јер се у сваком кораку алгоритма тренутни НЗП 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  $lcp[rang[i]]$ ,  $i = 2, 3, \dots, 10$ . 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 sissippi, 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	issippi	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;

    for(int i=0; i<n; i++, k?k--:0)
    {
        if(rank[i]==n-1) {k=0; continue;}
        int j=sa[rank[i]+1];
        while(i+k<n && j+k<n && s[i+k]==s[j+k]) k++;
        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  $P = \text{lednik}$  унутар текста  $s = \text{prestolonaslednikovica}$ .

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	tonaslednikovica	5	13	ednikovica	0
6	lonaslednikovica	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	lonaslednikovica	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	tonaslednikovica	0
22	a	22	19	vica	0

Бинарна претрага започиње упоређивањем  $P = \text{lednik}$  са  $S[SA[\lceil \frac{1+22}{2} \rceil]] = S[SA[12]] = \text{naslednikovica}$ . Пошто је  $\text{naslednikovica} > \text{lednik}$ , 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  $s[13] = \text{ednikovica}$  не започиње 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 поднису ниски

$$s_1 = \textit{prestolonaslednikovica}$$

и

$$s_2 = \textit{kolonoskopija}.$$

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	14	#	0
1	restolonaslednikovica\$ko...	2	1	23	\$ko...	0
1	estolonaslednikovica\$ko...	3	2	13	a#	1
1	stolonaslednikovica\$ko...	4	1	22	a\$ko...	1
1	tolonaslednikovica\$ko...	5	1	10	aslednikovica\$ko...	0
1	olonaslednikovica\$ko...	6	1	21	ca\$ko...	0
1	lonaslednikovica\$ko...	7	1	14	dnikovica\$ko...	0
1	onaslednikovica\$ko...	8	1	13	ednikovica\$ko...	1
1	naslednikovica\$ko...	9	1	3	estolonaslednikovica\$ko...	0
1	aslednikovica\$ko...	10	1	20	ica\$ko...	1
1	slednikovica\$ko...	11	2	11	ija#	1
1	lednikovica\$ko...	12	1	16	ikovica\$ko...	0
1	ednikovica\$ko...	13	2	12	ja#	0
1	dnikovica\$ko...	14	2	1	kolonoskopija#	2
1	nikovica\$ko...	15	2	8	kopija#	2
1	ikovica\$ko...	16	1	17	kovica\$ko...	0
1	kovica\$ko...	17	1	12	lednikovica\$ko...	1
1	ovica\$ko...	18	1	7	lonaslednikovica\$ko...	3
1	vica\$ko...	19	2	3	lonoskopija#	0
1	ica\$ko...	20	1	9	naslednikovica\$ko...	1
1	ca\$ko...	21	1	15	nikovica\$ko...	1
1	a\$ko...	22	2	5	noskopija#	0
1	\$ko...	23	1	6	olonaslednikovica\$ko...	4
2	kolonoskopija#	24	2	2	olonoskopija#	2
2	olonoskopija#	25	1	8	onaslednikovica\$ko...	2
2	lonoskopija#	26	2	4	onoskopija#	1
2	onoskopija#	27	2	9	opija#	1
2	noskopija#	28	2	6	oskopija#	1
2	oskopija#	29	1	18	ovica\$ko...	0
2	skopija#	30	2	10	pija#	1
2	kopija#	31	1	1	prestolonaslednikovica\$ko...	0
2	opija#	32	1	2	restolonaslednikovica\$ko...	0
2	pija#	33	2	7	skopija#	1
2	ija#	34	1	11	slednikovica\$ko...	1
2	ja#	35	1	4	stolonaslednikovica\$ko...	0
2	a#	36	1	5	tolonaslednikovica\$ko...	0
2	#	37	1	19	vica\$ko...	0

Највећи број у колони  $LCP[i]$  је  $LCP[23] = 4$ . Поред тога, суфикси

$$s[SA[23]] = \text{olonaslednikovica\$kolonoskopija\#}$$

и

$$s[SA[24]]\text{olonoskopija\#}$$

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

### 4.3 Формирање суфиксног стабла

Ако се зна суфиксни низ  $A$  и LCP низ  $H$  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,  $0 \leq i \leq n$ . Нека је даље  $d(v)$  дужина конкатенације ознака свих грана  $ST_i$  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  $A[i+1]$ , прелази се пут од последњег уметнутог листа ка корену, до првог чвора  $v$  за који је  $d(v) \leq H[i+1]$ .

- $d(v) = H[i+1]$ : то значи да је конкатенација ознака грана на путу од корена до чвора  $v$  једнака дужини НЗП суфикса  $A[i]$  и  $A[i+1]$ .

У том случају суфикс  $A[i+1]$  се умеће као нови лист  $x$  повезан са чвором  $v$  граном  $(v, x)$  са ознаком  $S[A[i+1]] + H[i+1], n$ . Другим речима, ознака гране се састоји од преосталих знакова суфикса  $A[i+1]$  који нису део конкатенације ознака грана на путу од корена до чвора  $v$ . На тај начин формирано је парцијално суфиксно стабло  $ST_{i+1}$ .

- $d(v) < H[i+1]$ : то значи да је конкатенација ознака грана на путу од корена до чвора  $v$  има мање знакова од дужине НЗП суфикса  $A[i]$  и  $A[i+1]$ , и да су знакови који недостају садржани на почетку ознаке *најдесније* гране (са лексикографски наредним наставком) која излази из  $v$ . Због тога се грана мора новим унутрашњим чвором раздвојити на две гране на следећи начин.

Нека је  $w$  син чвора  $v$  на најдеснијем путу из  $ST_i$ .

1. Уклонити грану  $(v, w)$ .
2. Додати нови унутрашњи чвор  $y$  и нову грану  $(v, y)$  са ознаком  $S[A[i]] + d(v), A[i] + H[i+1] - 1$ . Ознака нове гране састоји се од знакова који недостају на НЗП суфикса  $A[i]$  и  $A[i+1]$ . Према томе, конкатенације ознака грана на путу од корена до чвора  $y$  једнака је НЗП суфикса  $A[i]$  и  $A[i+1]$ .
3. Повезати чвор  $w$  са новокреираним унутрашњим чвором  $y$  граном  $(y, w)$  са ознаком  $S[A[i]] + H[i+1], A[i] + d(v) - 1$ . Нова ознака састоји се од преосталих знакова обрисане гране  $(v, w)$  који нису укључени у ознаку гране  $(v, y)$ .

4. Уметнути  $A[i+1]$  као нови лист  $x$  и повезати га са новим унутрашњим чвором  $y$  граном  $(y, x)$  са ознаком  $S[A[i + 1]] + H[i + 1], n$ . Према томе, ознака гране састоји се од преосталих знакова суфикса  $A[i+1]$ , који нису укључени у конкатенације ознака грана на путу од корена до чвора  $v$ .
5. На тај начин формирано је парцијално суфиксно стабло  $ST_{i+1}$ .

**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, 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 7
9	4	sissippi	1	ssippi 4
10	6	ssippi	3	si ppi 6
11	3	ssissippi		ssippi 3

sip

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  $A[i + 1]$  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 Прилог: програмска реализација алгорита 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;
}

```