Sufixové polia

kuko

24.11.2018

Vybrané partie z dátových štruktúr

Suffix array

Array of suffixes in lexicographic order

(assume $\$ < a \quad \forall a \in \Sigma$)

i	0	1	2	3	4	5	6
$S[i]$	b	a	n	a	n	a	$

i	0	1	2	3	4
$S[i]$	a	a	a	a	$

i	$SA[i]$	Suffix
0	**6**	$
1	**5**	a$
2	**3**	ana$
3	**1**	anana$
4	**0**	banana$
5	**4**	na$
6	**2**	nana$

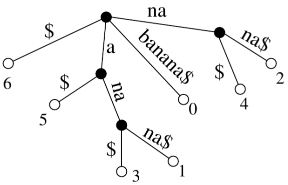
i	$SA[i]$	Suffix
0	**4**	$
1	**3**	a$
2	**2**	aa$
3	**1**	aaa$
4	**0**	aaaa$

32

From suffix array to suffix tree

T =banana$

i	$SA[i]$	$L[i]$	suffix
0	**6**	0	$
1	**5**	1	a$
2	**3**	3	ana$
3	**1**	0	anana$
4	**0**	0	banana$
5	**4**	2	na$
6	**2**	-	nana$



Longest common prefix

- $\mathrm{lcp}(A, B) =$ the length of longest common prefix of strings A and B
- $\mathrm{LCP}(i, j) = \mathrm{lcp}(T[SA[i]..n], T[SA[j]..n])$
  i.e. lcp of two suffixes in a suffix array

i	$SA[i]$	Suffix
0	**6**	$
1	**5**	a$
2	**3**	ana$
3	**1**	anana$
4	**0**	banana$
5	**4**	na$
6	**2**	nana$

LCP(2,3) = lcp(ana$,anana$) = 3

LCP(2,5) = lcp(ana$, na$) = 0

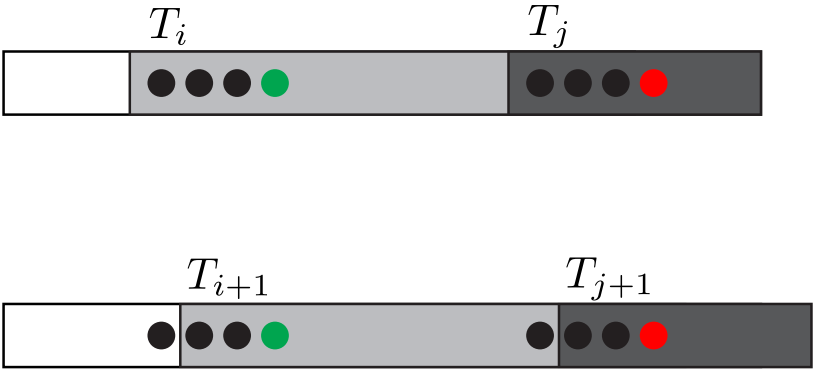
36

LCP

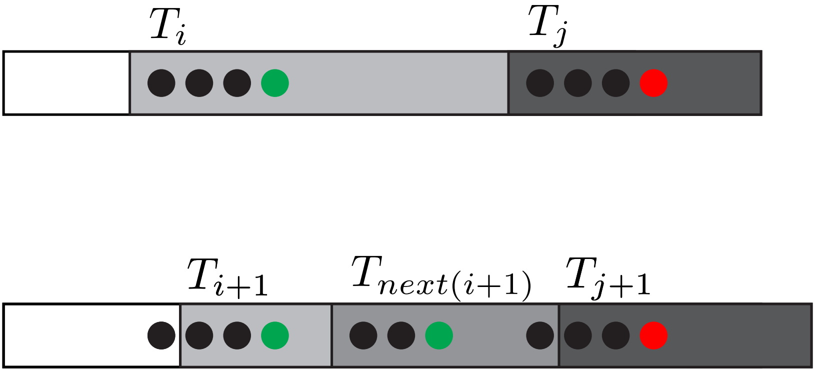
```
h = 0;
for (i=0; i <=n; i++) {
  if (rank[i] < n) {
    j = next(i);    // SA[rank[i] + 1];
    // compare suffixes S[i..] and S[j..]
    h = 0;
    while (S[i+h] == S[j+h]) ++h;
    // we have found first mismatch
    L[i] = h;
  }
}
```

LCP

LCP



LCP



LCP

$rank$	S	SA	LCP	
4	M	7	1	IPPI$
2	I	4	4	ISSIPPI$
10	S	1	1	ISSISSIPPI$
8	S	10	0	I$
1	I	0	0	MISSISSIPPI$
9	S	9	1	PI$
7	S	8	0	PPI$
0	I	6	2	SIPPI$
6	P	3	1	SISSIPPI$
5	P	5	3	SSIPPI$
3	I	2	0	SSISSIPPI$
11	$	11	_	$

LCP

```
h = 0;
for (i=0; i <=n; i++) {
  if (rank[i] < n) {
    j = next(i);    // SA[rank[i] + 1];
    // compare suffixes S[i..] and S[j..]
    h = 0;
    while (S[i+h] == S[j+h]) ++h;
    // we have found first mismatch
    L[i] = h;
  }
}
```

LCP

```
h = 0;
for (i=0; i <=n; i++) {
  if (rank[i] < n) {
    j = next(i);   // SA[rank[i] + 1];
    // compare suffixes S[i..] and S[j..]
    // assume they have >= h characters in common
    while (S[i+h] == S[j+h]) ++h;
    // we have found first mismatch
    L[i] = h;
    if (h > 0) --h;
  }
}
```

Inverse of a suffix array

Array rank such that $rank[i] = x \iff SA[x] = i$

Can be computed in $O(n)$ from SA:

```
for(i = 0; i <= n; i++) {
  rank[SA[i]] = i;
}
```

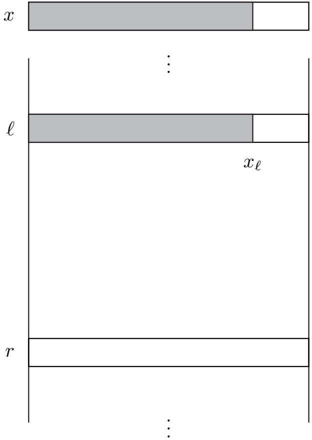
Go from suffix to its position in the suffix array, its neighbors, etc.

Vyhľadávanie

Vyhľadávanie

- jednoduché binárne vyhľadávanie: $O(m \log n)$
- ak poznáme $\mathrm{lcp}(s_i, s_j)$ pre ľubovoľné dva sufixy, vieme zrýchliť na $O(m + \log n)$

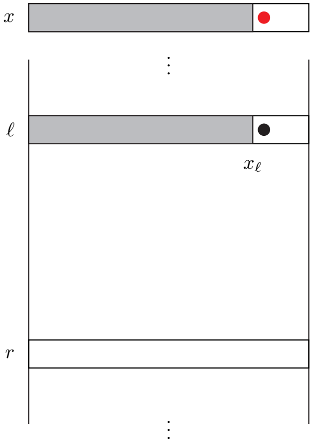
INVARIANTY: $\ell < x \leq r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: Potiaľto sa x a ℓ rovnajú.

Vyhľadávanie

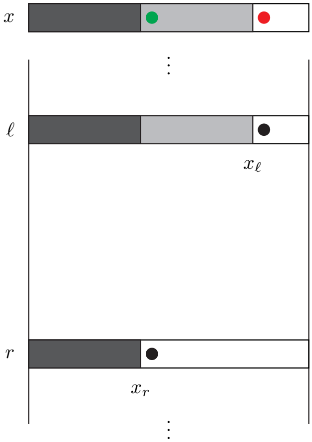
INVARIANTY: $\ell < x \leq r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: $\ell[x_\ell] < x[x_\ell]$.

Vyhľadávanie

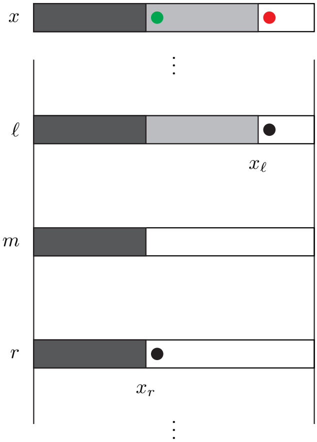
INVARIANTY: $\ell < x \le r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: Potiaľto sa x a r rovnajú. (Predpokladajme $x_\ell \ge x_r$)

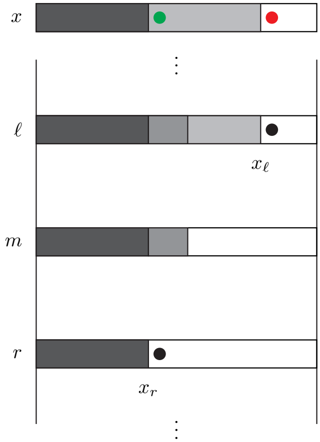
Vyhľadávanie

INVARIANTY: $\ell < x \leq r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: $r[x_r] > x[x_r]$.

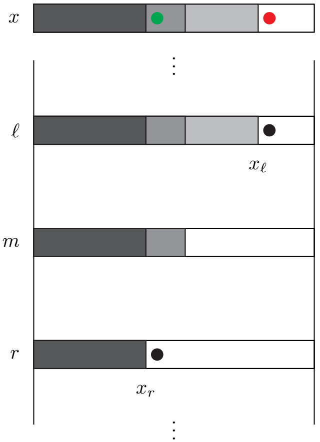
Invarianty: $\ell < x \le r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: Pozrime sa na prostredný sufix; aký je $p = \mathrm{lcp}(\ell, m)$?

Vyhľadávanie

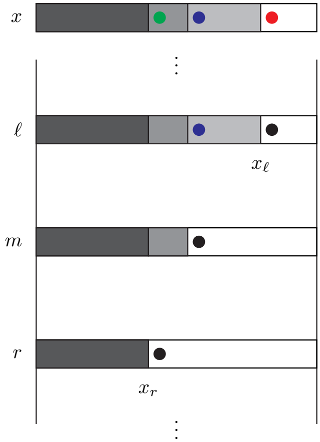
INVARIANTY: $\ell < x \leq r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: Prípad 1: Ak $p < x_\ell \ldots$

Vyhľadávanie

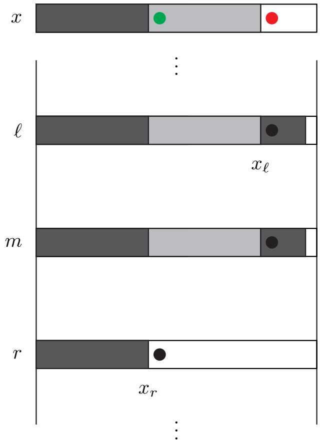
INVARIANTY: $\ell < x \leq r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: Prípad 1: . . . tak $m[p] > \ell[p] = x[p]$, teda $m > x$.

Vyhľadávanie

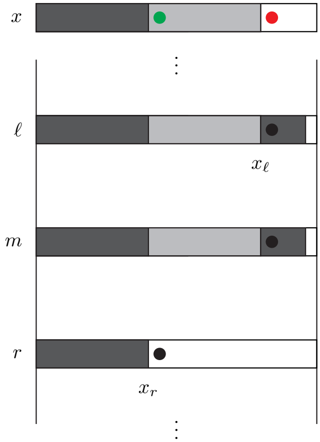
INVARIANTY: $\ell < x \leq r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: Prípad 2: Ak $p > x_\ell, \ldots$

Vyhľadávanie

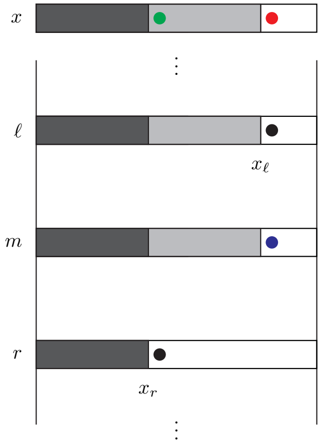
INVARIANTY: $\ell < x \leq r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: Prípad 2: . . . tak $m[x_\ell] = \ell[x_\ell] < x[x_\ell]$, teda $m < x$.

Vyhľadávanie

INVARIANTY: $\ell < x \leq r, \quad x_\ell = \mathrm{lcp}(x, \ell), \quad x_r = \mathrm{lcp}(x, r)$



Obr.: Prípad 3: Ak $p = x_\ell$, začneme porovnávať písmená.

Vyhľadávanie

Invarianty:

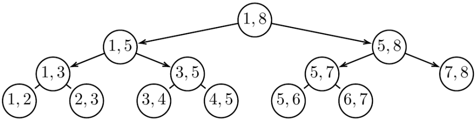
- $\ell < x \leq r$
- $x_\ell = \mathrm{lcp}(x, \ell)$
- $x_r = \mathrm{lcp}(x, r)$

Algoritmus:

- Ak $x_\ell \geq x_r$:
  - nech m je prostredný sufix a $p = lcp(\ell, m)$
  - ak $p < x_\ell$: $r \leftarrow m$, $x_r \leftarrow p$
  - ak $p > x_\ell$: $\ell \leftarrow m$
  - ak $p = x_\ell$:
    - začneme porovnávať písmená x a m od pozície x_ℓ
    - podľa výsledku nastavíme ℓ, x_ℓ alebo r, x_r
- Ak $x_\ell \leq x_r$ – symetricky

LCP values for algorithm 3

Which values are needed? $\mathrm{LCP}(L, k)$ or $\mathrm{LCP}(R, k)$



$2n - 1$ LCP values needed

Let $L[i] = \mathrm{LCP}(i, i+1)$, precompute to an array in $O(n)$ (later)
For $j - i > 1$:

$$\begin{aligned}\mathrm{LCP}(i, j) &= \min\{\mathrm{LCP}(k, k+1) \mid k = i \ldots j-1\} \\ &= \min\{\mathrm{LCP}(i, x), \mathrm{LCP}(x, j)\} \text{ for any } x \in \{i+1, \ldots j-1\}\end{aligned}$$

Konštrukcia SA

Konštrukcia SA

- qsort – $O(n^2 \log n)$
- radix-sort / trie – $O(n^2)$
- lepšie?

Manberov algoritmus

Myšlienka:

- sufix sufixu je sufix
- ak máme pole utriedené podľa prvých K písmen, vieme ho ľahko zotriediť podľa prvých $2K$ písmen
- budeme mať $\log n$ fáz; v k-tej fáze triedime podľa prvých 2^k písmen

Manberov algoritmus

Myšlienka:

- sufix sufixu je sufix
- ak máme pole utriedené podľa prvých K písmen, vieme ho ľahko zotriediť podľa prvých $2K$ písmen
- budeme mať $\log n$ fáz; v k-tej fáze triedime podľa prvých 2^k písmen

Manberov algoritmus

		index sort		inverse	
0	babaaaabcbabaaaaa0	17	0	0	14
1	abaaaabcbabaaaaa0	16	a0	1	9
2	baaaabcbabaaaaa0	15	aa0	2	12
3	aaaabcbabaaaaa0	14	aaa0	3	4
4	aaabcbabaaaaa0	3	aaaabcbabaaaaa0	4	7
5	aabcbabaaaaa0	12	aaaaa0	5	8
6	abcbabaaaaa0	13	aaaa0	6	11
7	bcbabaaaaa0	4	aaabcbabaaaaa0	7	16
8	cbabaaaaa0	5	aabcbabaaaaa0	8	17
9	babaaaaa0	1	abaaaabcbabaaaaa0	9	15
10	abaaaaa0	10	abaaaaa0	10	10
11	baaaaa0	6	abcbabaaaaa0	11	13
12	aaaaa0	2	baaaabcbabaaaaa0	12	5
13	aaaa0	11	baaaaa0	13	6
14	aaa0	0	babaaaabcbabaaaaa0	14	3
15	aa0	9	babaaaaa0	15	2
16	a0	7	bcbabaaaaa0	16	1
17	0	8	cbabaaaaa0	17	0

0 + 4 = 4

9 + 4 = 13

13 < 4 (because 6 < 7) so 9 < 0

Obr.: Triedime podľa prvých 1,2,4,8,16,32,. . . písmen.

Manberov algoritmus

- nech $\mathrm{rank}[i] = j$, ak je sufix s_i v abecednom poradí j-ty (podľa prvých 2^k písmen; ak majú dva sufixy rovnakých prvých 2^k písmen, ranky budú rovnaké)
- 1 fáza:
  - zotriedime trojice $(\mathrm{rank}[i], \mathrm{rank}[i+2^k], i)$

Konštrukcia SA

- qsort – $O(n^2 \log n)$
- radix-sort / trie – $O(n^2)$
- Manberov algoritmus – $O(n \log n)$
- lepšie?

$$\begin{array}{cccccccccccccc} & 1 & 2 & & 4 & 5 & & 7 & 8 & & 10 & 11 & & 13 \\ \mathtt{A} & \mathtt{B} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{B} & \mathtt{A} & \$ & \$ & \$ \end{array}$$

$$\begin{array}{cccccccccccccc} 0 & & & 3 & & & 6 & & & 9 & & & 12 & \\ \mathtt{A} & \mathtt{B} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{B} & \mathtt{A} & \$ & \$ & \$ \end{array}$$

0. S_{10} = A$$$
1. S_5 = ABABBA$$$
2. S_7 = ABBA$$$
3. S_2 = BABABABBA$$$
4. S_4 = BABABBA$$$
5. S_8 = BBA$$$
6. S_1 = BBABABABBA$$$

Obr.: Rozdelíme na sufixy na pozíciach nedeliteľných vs. deliteľných 3.

Konštrukcia SA

	1	2		4	5		7	8		10	11		13
A	B	B	A	B	A	B	A	B	B	A	$	$	$

0			3			6			9			12	
A	B	B	A	B	A	B	A	B	B	A	$	$	$

0. S_{10}
1. S_5
2. S_7
3. S_2
4. S_4
5. S_8
6. S_1

Obr.: Rekurzívne utriedime pozície $\equiv 1, 2 \pmod 3$.
Čo pozície $\equiv 0 \pmod 3$?

Konštrukcia SA

	1	2		4	5		7	8		10	11		13
A	B	B	A	B	A	B	A	B	B	A	\$	\$	\$

0			3			6			9			12	
A	B	B	A	B	A	B	A	B	B	A	\$	\$	\$

0. S_{10}
1. S_5
2. S_7
3. S_2
4. S_4
5. S_8
6. S_1

0. $S_3 = \mathtt{A}S_4$
1. $S_0 = \mathtt{A}S_1$
2. $S_9 = \mathtt{B}S_{10}$
3. $S_6 = \mathtt{B}S_7$

Obr.: Odrolujeme jedno písmeno; zvyšok je poz. $\equiv 1 \pmod 3$.

```
 1 2   4 5   7 8   10 11  13
A B B A B A B A B B A $ $ $
```

0. S_{10}
1. S_5
2. S_7
3. S_2
4. S_4
5. S_8
6. S_1

```
0     3     6     9     12
A B B A B A B A B B A $ $ $
```

0. S_3
1. S_0
2. S_9
3. S_6

Obr.: Ako tieto dve utriedené polia zmerge-ujeme?

Konštrukcia SA

$$\begin{array}{cccccccccccccc} & 1 & 2 & & 4 & 5 & & 7 & 8 & & 10 & 11 & & 13 \\ \mathtt{A} & \mathtt{B} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{B} & \mathtt{A} & \$ & \$ & \$ \end{array}$$

$$\begin{array}{cccccccccccccc} 0 & & & 3 & & & 6 & & & 9 & & & 12 & \\ \mathtt{A} & \mathtt{B} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{A} & \mathtt{B} & \mathtt{B} & \mathtt{A} & \$ & \$ & \$ \end{array}$$

0. $S_{10} = \mathtt{A}S_{11}$
1. $S_5 = \mathtt{AB}S_7$
2. $S_7 = \mathtt{A}S_8$
3. $S_2 = \mathtt{BA}S_4$
4. $S_4 = \mathtt{B}S_5$
5. $S_8 = \mathtt{BB}S_{10}$
6. $S_1 = \mathtt{B}S_2$

0. $S_3 = \mathtt{A}S_4 = \mathtt{AB}S_5$
1. $S_0 = \mathtt{A}S_1 = \mathtt{AB}S_2$
2. $S_9 = \mathtt{B}S_{10} = \mathtt{BA}S_{11}$
3. $S_6 = \mathtt{B}S_7 = \mathtt{BA}S_8$

Obr.: Odrolujeme jedno alebo dve písmená.

Konštrukcia SA

Zložitosť:

- $T(n) =$
  - $T(\frac{2}{3}n)$ – rekurzívne volanie
  - $+O(n)$ – triedenie $\equiv 0 \pmod 3$
    - $+O(n)$ – merge

Moment: ale aké rekurzívne volanie? Je to tá istá úloha?

Konštrukcia SA

Zložitosť:

- $T(n) =$
  - $T(\frac{2}{3}n)$ – rekurzívne volanie
  - $+O(n)$ – triedenie $\equiv 0 \pmod 3$
    - $+O(n)$ – merge

Moment: ale aké rekurzívne volanie? Je to tá istá úloha?

Konštrukcia SA

A B B A B A B A B B A $ $ $

Obr.: Začneme s pôvodným stringom.

Konštrukcia SA

```
1     4     7     10    13
A B B A B A B A B B A $ $ $
```

```
1       4       7       10
B B A | B A B | A B B | A $ $
```

Obr.: Vezmeme jednu kópiu od pozície 1. . .

Konštrukcia SA

	1	2		4	5		7	8		10	11		13
A	B	B	A	B	A	B	A	B	B	A	$	$	$

1	4	7	10	2	5	8
B B A	B A B	A B B	A $ $	B A B	A B A	B B A

Obr.: . . . a pridáme kópiu od pozície 2. . .

Ak budeme každú trojicu znakov považovať za 1 písmeno, tak sufixy tohto stringu zodpovedajú sufixom na poz. $\equiv 0, 1 \pmod 3$ v pôvodnom stringu.
Problém: Veľká abeceda – bude porovnávanie znakov v $O(1)$?

Konštrukcia SA

	1	2		4	5		7	8		10	11		13
A	B	B	A	B	A	B	A	B	B	A	$	$	$

1	4	7	10	2	5	8
B B A	B A B	A B B	A $ $	B A B	A B A	B B A

Obr.: . . . a pridáme kópiu od pozície 2. . .

Ak budeme každú trojicu znakov považovať za 1 písmeno, tak sufixy tohto stringu zodpovedajú sufixom na poz. $\equiv 0, 1 \pmod 3$ v pôvodnom stringu.

Problém: Veľká abeceda – bude porovnávanie znakov v $O(1)$?

Konštrukcia SA

	1	2		4	5		7	8		10	11		13
A	B	B	A	B	A	B	A	B	B	A	$	$	$

1	4	7	10	2	5	8
$\boxed{\texttt{B B A}}$	$\boxed{\texttt{B A B}}$	$\boxed{\texttt{A B B}}$	$\boxed{\texttt{A \$ \$}}$	$\boxed{\texttt{B A B}}$	$\boxed{\texttt{A B A}}$	$\boxed{\texttt{B B A}}$

Obr.: . . . a pridáme kópiu od pozície 2. . .

Ak budeme každú trojicu znakov považovať za 1 písmeno, tak sufixy tohto stringu zodpovedajú sufixom na poz. $\equiv 0, 1 \pmod 3$ v pôvodnom stringu.

Problém: Veľká abeceda – bude porovnávanie znakov v $O(1)$?

Konštrukcia SA

	1	2		4	5		7	8		10	11		13
A	B	B	A	B	A	B	A	B	B	A	$	$	$

1	4	7	10	2	5	8
B B A	B A B	A B B	A $ $	B A B	A B A	B B A

A$$ → a
ABA → b
ABB → c
BAB → d
BBA → e

Obr.: Riešenie: trojíc je málo, takže si ich môžeme prečíslovať.

Konštrukcia SA

```
   1  2     4  5     7  8     10 11    13
A  B  B  A  B  A  B  A  B  B  A  $  $  $
```

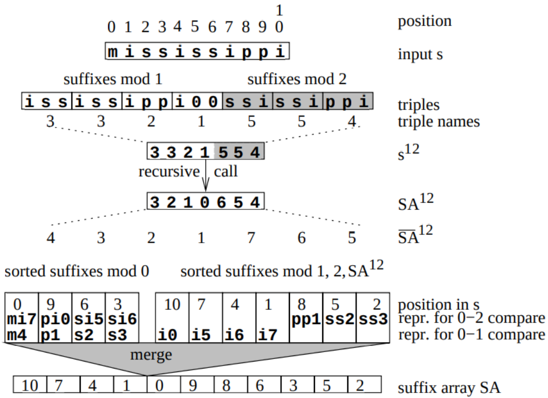
```
1        4        7        10          2        5        8
[B B A]  [B A B]  [A B B]  [A $ $]     [B A B]  [A B A]  [B B A]
```

```
e d c a d b e
```

A$$	$\rightarrow$	a
ABA	$\rightarrow$	b
ABB	$\rightarrow$	c
BAB	$\rightarrow$	d
BBA	$\rightarrow$	e

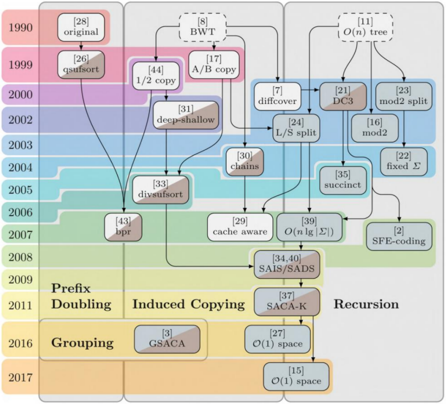
Obr.: Takto ostane abeceda malá. Rekurzívne sa zavoláme na string edcadbe. Výsledné sufixové pole musíme následne prečíslovať: sufixy 0,1,2,3 zodpovedajú pozíciam 1,4,7,10 a sufixy za polovicou: 4,5,6 zodpovedajú pozíciam 2,5,8.

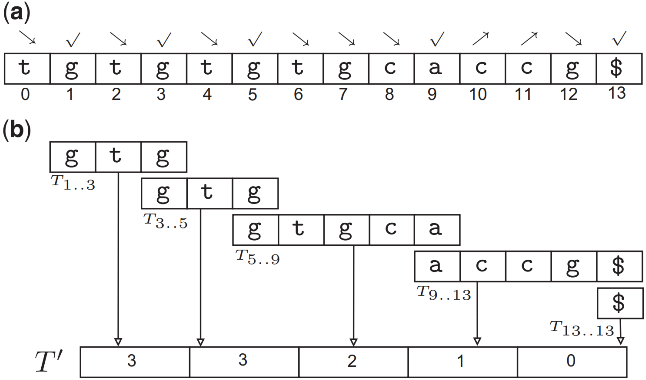
Konštrukcia SA

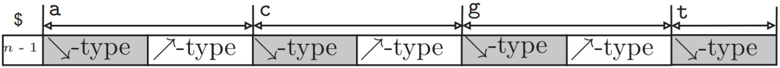


Algoritmus:

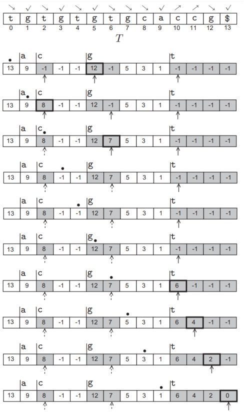
- $SA^{1,2}$ – zotriedime pozície $\equiv 1,2 \pmod 3$:
  - vytvoríme string s' dĺžky $n' = \frac{2}{3}n$:
    - vytvoríme $s^{1,2}$ tak, že zreťazíme $s[1\ldots]$ a $s[2\ldots]$
    - zredukujeme abecedu: radix-sortom utriedime všetky trojice znakov v $s^{1,2}$ a prepíšeme ich na znaky $1,\ldots,k$
  - rekurzívne zavoláme SA(s')
  - vo výsledku prečíslujeme indexy:
    - ak $i < \lceil n'/2 \rceil \longrightarrow 3i+1$
    - ak $i \geq \lceil n'/2 \rceil \longrightarrow 3(i - \lceil n'/2 \rceil) + 2$
- SA^0 – zotriedime pozície $\equiv 0 \pmod 3$:
  - radix-sortom podľa prvého písmena a pozície v $SA^{1,2}$
- zmergujeme $SA^{1,2}$ a SA^0
  - podľa prvého alebo prvých dvoch písmen a pozície v $SA^{1,2}$

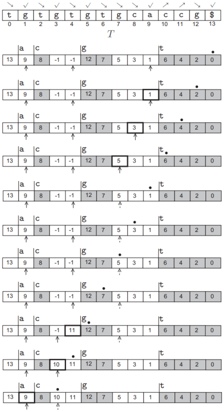

1990
1999
2000
2002
2003
2004
2005
2006
2007
2008
2009
2011
2016
2017
[28] original
[26] qsufsort
[8] BWT
[44] 1/2 copy
[17] A/B copy
[31] deep-shallow
[11] O(n) tree
[7] diffcover
[21] DC3
[23] mod2 split
[24] L/S split
[16] mod2
[30] chains
[22] fixed Σ
[35] succinct
[33] divsufsort
[43] bpr
[29] cache aware
[39] O(n lg |Σ|)
[2] SFE-coding
[34,40] SAIS/SADS
Prefix Doubling
Induced Copying
[37] SACA-K
Recursion
Grouping
[3] GSACA
[27] O(1) space
[15] O(1) space


(a)
t g t g t g t g c a c c g $
0 1 2 3 4 5 6 7 8 9 10 11 12 13
(b)
g t g
T1..3
g t g
T3..5
g t g c a
T5..9
a c c g $
T9..13
$
T13..13
T'
3 3 2 1 0


$
a
c
g
t
n - 1
↘-type
↗-type
↘-type
↗-type
↘-type
↗-type
↘-type

	a	c			g					t			
13	9	-1	-1	-1	-1	-1	5	3	1	-1	-1	-1	-1


t g t g t g t g c a c c g $
0 1 2 3 4 5 6 7 8 9 10 11 12 13
T
a
c
g
t


t g t g t g t g c a c c g $
0 1 2 3 4 5 6 7 8 9 10 11 12 13
T

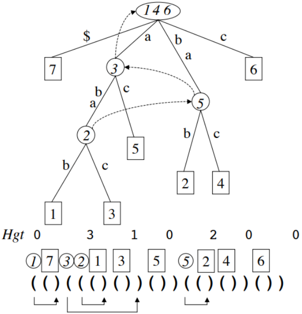
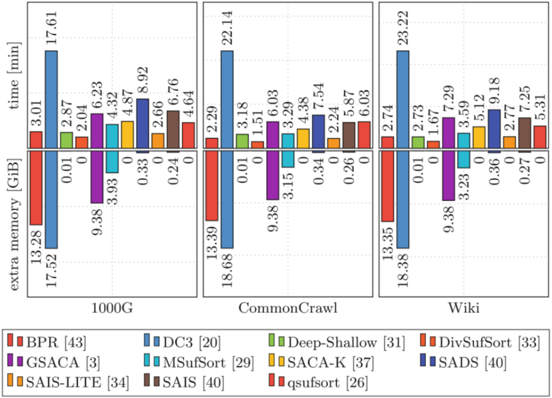
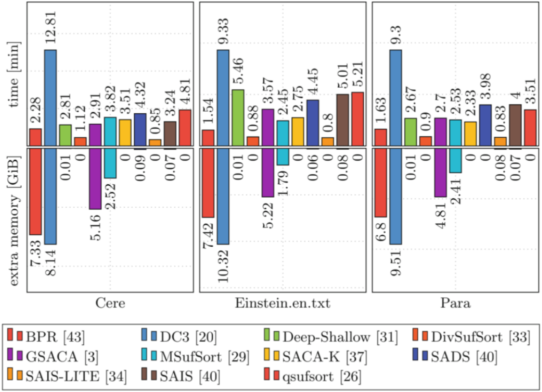
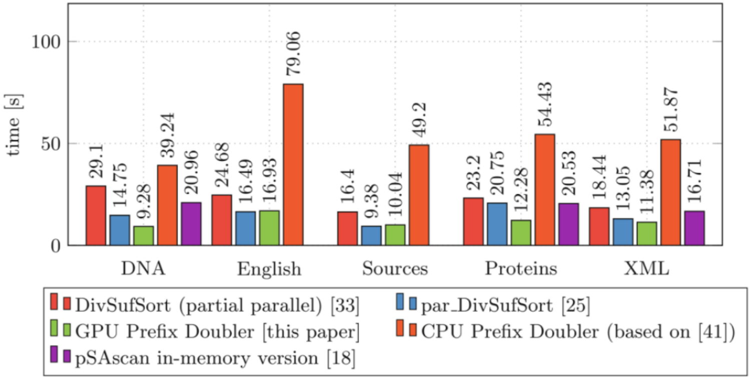


Figure 1: The suffix tree for "ababac$," and its balanced parentheses representation. Suffix links are shown in dotted line.


time [min]
extra memory [GiB]
3.01
17.61
2.87
2.04
6.23
4.32
4.87
8.92
2.66
6.76
4.64
13.28
17.52
0.01
0
9.38
3.93
0
0.33
0
0.24
0
2.29
22.14
3.18
1.51
6.03
3.29
4.38
7.54
2.24
5.87
6.03
13.39
18.68
0.01
0
9.38
3.15
0
0.34
0
0.26
0
2.74
23.22
2.73
1.67
7.29
3.59
5.12
9.18
2.77
7.25
5.31
13.35
18.38
0.01
0
9.38
3.23
0
0.36
0
0.27
0
1000G
CommonCrawl
Wiki
BPR [43]
DC3 [20]
Deep-Shallow [31]
DivSufSort [33]
GSACA [3]
MSufSort [29]
SACA-K [37]
SADS [40]
SAIS-LITE [34]
SAIS [40]
qsufsort [26]


time [min]
extra memory [GiB]
Cere
Einstein.en.txt
Para
BPR [43]
DC3 [20]
Deep-Shallow [31]
DivSufSort [33]
GSACA [3]
MSufSort [29]
SACA-K [37]
SADS [40]
SAIS-LITE [34]
SAIS [40]
qsufsort [26]



time [s]
0
50
100
29.1
14.75
9.28
39.24
20.96
24.68
16.49
16.93
79.06
16.4
9.38
10.04
49.2
23.2
20.75
12.28
54.43
20.53
18.44
13.05
11.38
51.87
16.71
DNA
English
Sources
Proteins
XML
DivSufSort (partial parallel) [33]
par_DivSufSort [25]
GPU Prefix Doubler [this paper]
CPU Prefix Doubler (based on [41])
pSAscan in-memory version [18]

i	1	2	3	4	5	6	7
Hgt	0	3	1	0	2	0	0
SA	7	1	3	5	2	4	6
$SA + Hgt$	7	4	4	5	4	4	6

Figure 2: How to create a sorted sequence from Hgt.