

Bioinformatika 1

Sufiksno polje

Mirjana Domazet-Lošo
FER, 2021./2022.



Creative Commons Imenovanje-Nekomercijalno-Dijeli pod istim uvjetima 3.0

Sufiksno stablo - problem

- najučinkovitije implementacije sufiksni stabala zahtijevaju oko 10 okteta za svaki ulazni znak, a često 15-20 okteta (Kurtz, 1999)
- Što napraviti?
 - korištenje drugih struktura podataka kako bi se uskladili vremenski i memorijski zahtjevi:**sufiksna polja**
- sufiksna stabla ostaju kao konceptualni alat

Sufiksna polja (1)

- Manber i Myers su 1990. predložili sufiksno polje (engl. *suffix array*)

"Suffix arrays: a new method for on-line string searches"
(SODA, 1990)

- Sufiksno polje *SA* za niz znakova *S*:
polje cijelih brojeva
koje sadrži početna mjesta (indekse) abecedno sortiranih
sufiksa niza *S*

Sufiksna polja (2)

Primjer:

$S = \text{ACCA}\$$ ($\$$ abecedno veći od znakova abecede nad S)

Sufiksi: $s_1 = \text{ACCA}\$, s_2 = \text{CCA}\$, s_3 = \text{CA}\$, s_4 = \text{A}\$, s_5 = \$$

Abecedno poredani sufiksi niza S :

$s_1 = \text{ACCA}\$$

$s_4 = \text{A}\$$

$s_3 = \text{CA}\$$

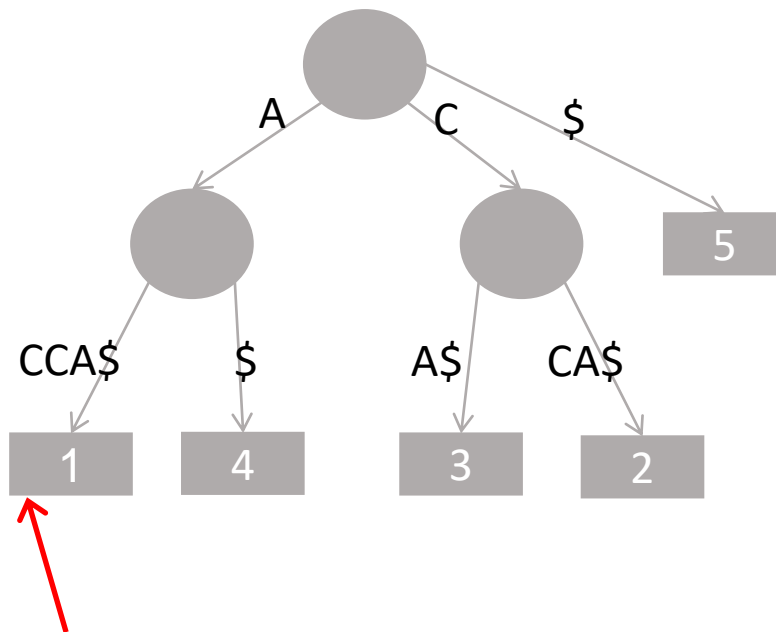
$s_2 = \text{CCA}\$$

$s_5 = \$$

$\rightarrow \text{SA} = [1, 4, 3, 2, 5]$

Sufiksno polje i sufiksno stablo

$S = \text{ACCA}\$$



i	s_i	$SA[i]$
1	ACCA\$	1
2	CCA\$	4
3	CA\$	3
4	A\$	2
5	\$	5

Preorder obilazak sufiksnog stabla $\rightarrow SA$

Sortiranje sufiksa

Kolika je vremenska složenost sortiranja svih sufiksa nekog znakovnog niza, ako koristimo *quicksort*?

Sufiksna polja (3)

- postoje algoritmi koji konstruiraju sufiksno polje (engl. *Suffix Array Construction Algorithm; SACA*) u linearnom vremenu
- postoje algoritmi koji imaju supralinearno vrijeme izvođenja za najgori slučaj, ali u praksi mogu biti brži od onih koji imaju linearno vrijeme izvođenja (Shrestha *et al.*, 2014; Puglisi *et al.*, 2007)
- svaki problem, koji se može riješiti korištenjem sufiksni stabala, može se riješiti i korištenjem sufiksnog polja s istom asimptotskom složenošću (Puglisi *et al.*, 2007)

Usporedba SACA (1)

	Algorithm	Worst Case	Time	Memory
Puglisi <i>et al.</i> , 2007	Prefix-Doubling			
	MM [Manber and Myers 1993]	$O(n \log n)$	30	$8n$
	LS [Larsson and Sadakane 1999]	$O(n \log n)$	3	$8n$
	Recursive			
	KA [Ko and Aluru 2003]	$O(n)$	2.5	$7 - 10n$
	KS [Kärkkäinen and Sanders 2003]	$O(n)$	4.7	$10-13n$
	KSPP [Kim et al. 2003]	$O(n)$	—	—
	HSS [Hon et al. 2003]	$O(n)$	—	—
	KJP [Kim et al. 2004]	$O(n \log \log n)$	3.5	$13-16n$
	N [Na 2005]	$O(n)$	—	—
	Induced Copying			
	IT [Itoh and Tanaka 1999]	$O(n^2 \log n)$	6.5	$5n$
	S [Seward 2000]	$O(n^2 \log n)$	3.5	$5n$
	BK [Burkhardt and Kärkkäinen 2003]	$O(n \log n)$	3.5	$5-6n$
	MF [Manzini and Ferragina 2004]	$O(n^2 \log n)$	1.7	$5n$
	SS [Schürmann and Stoye 2005]	$O(n^2)$	1.8	$9-10n$
	BB [Baron and Bresler 2005]	$O(n\sqrt{\log n})$	2.1	$18n$
	M [Maniscalco and Puglisi 2006a]	$O(n^2 \log n)$	1.3	$5-6n$
	MP [Maniscalco and Puglisi 2006b]	$O(n^2 \log n)$	1	$5-6n$
	Hybrid			
	IT+KA	$O(n^2 \log n)$	4.8	$5n$
	BK+IT+KA	$O(n \log n)$	2.3	$5-6n$
	BK+S	$O(n \log n)$	2.8	$5-6n$
	Suffix Tree			
	K [Kurtz 1999]	$O(n \log \sigma)$	6.3	$13-15n$

Usporedba SACA (2)

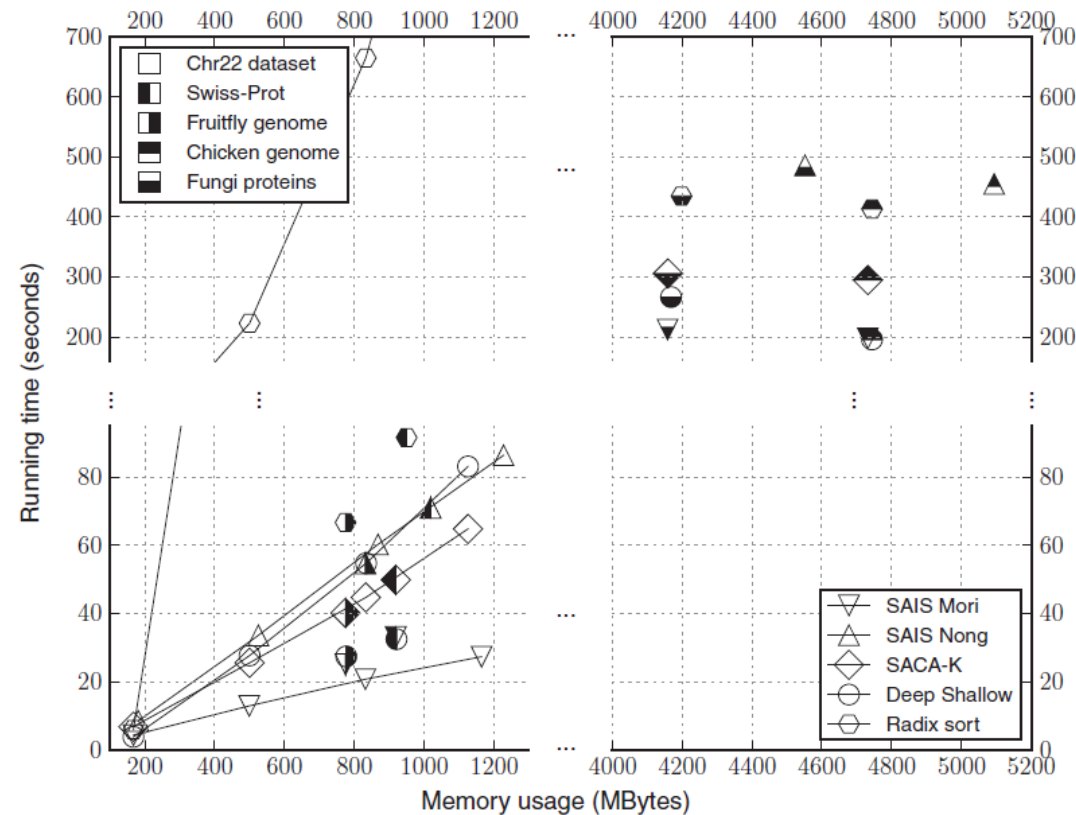


Figure 8: Time and memory performance of implementations of select suffix sorting algorithms. The shapes of the markers distinguish the different programs, and the fill-styles distinguish the data sets. The nonfilled markers connected by lines correspond to the performance for the four data sets constructed from increasingly many polymorphic copies of human chromosome 22.

Shrestha *et al.* 2014

Prošireno sufiksno polje

- prošireno sufiksno polje (engl. *enhanced suffix array*; ESA) sastoji se od sufiksnog polja i dodatnih polja:
 - polje LCP (*Longest Common Prefix*)
 - polje ISA (*Inverse Suffix Array*; inverzno sufiksno polje)
 - polje BWT (*Burrows-Wheeler Transform(ation)*)
- prošireno sufiksno polje može zamijeniti sufiksno stablo (Abouelhoda *et al.* 2004)
 - ista vremenska složenost; u praksi brže
 - memorijski učinkovitija izvedba
 - jednostavnija izvedba

Inverzno sufiksno polje

- inverzno sufiksno polje, ISA :
 - $ISA[SA[i]] := i$
 - $ISA[i] = j \leftrightarrow SA[j] = i$
 - određuje se u $O(n)$ vremenu iz SA
 - memorijski zahtjev: (obično) $4n$ okteta

$S = \text{ACCA\$}$

i	Abecedno poredani sufiksi od S	$SA[i]$	s_i	$ISA[i]$
1	ACCA\$	1	ACCA\$	1
2	A\$	4	CCA\$	4
3	CA\$	3	CA\$	3
4	CCA\$	2	A\$	2
5	\$	5	\$	5

Polje najduljih zajedničkih prefiksa

- polje najduljih zajedničkih prefiksa, *LCP*:
 - $LCP[1] := 0$
 - $LCP[i] := \text{lcp}(SA[i], SA[i - 1]), 2 \leq i \leq n$
 - određuje se u $O(n)$ vremenu iz *SA* i *ISA* (Kasai *et al.*, 2004; Fischer, 2011)
 - memorijski zahtjev: (obično) $4n$ ili $(1 + \varepsilon)n$ okteta

***S* = ACCA\$**

<i>i</i>	<i>SA</i> [<i>i</i>]	<i>LCP</i> [<i>i</i>]	Abecedno poredani sufiksi
1	1	0	ACCA\$
2	4	1	A\$
3	3	0	CA\$
4	2	1	CCA\$
5	5	0	\$

Polja SA , ISA i LCP - primjer

$S = ACCA\$$

i	$SA[i]$	$ISA[i]$	$LCP[i]$	Abecedno poredani sufiksi od S
1	1	1	0	ACCA\$
2	4	4	1	A\$
3	3	3	0	CA\$
4	2	2	1	CCA\$
5	5	5	0	\$



$ISA[SA[i]] := i$



$LCP[i] := \text{lcp}(SA[i], SA[i - 1])$

h -sortiranje (1)

- **h -sortiranje** (engl. *h -sort*): parcijalno sortiranje
 - sortiramo prefikse sufiksa koji su duljine h ,
gdje h može biti ≥ 1
 - rezultat je **h -poredak** sufiksa (engl. *h -order*), tj. poredak sufiksa prema prvih h znakova
- **h -rang** (engl. *h -rank*)
 - ako su neka dva sufiksa jednaka prema h -poretku (tj. imaju jednake prefikse duljine h), kažemo da imaju isti h -rang, tj. da su h -jednaki (engl. *h -equal*)

h -sortiranje (2)

Primjer:

za niz $S = \text{ACCA\$}$ odrediti ISA_1 i SA_1 ($\$$ je abecedno najveći znak)

- 1-sortiranje sufiksa (tj. h -sortiranje za $h = 1$):
ACCA\$, A\$, CCA\$, CA\$, \$
- npr. sufiksi ACCA\$ i A\$ su 1-jednaki; također CCA\$ i CA\$
- SA_1 je sufiksno polje za sufikse sortirane 1-sortiranjem:
 $SA_1 = [\underline{1}, 4, \underline{2}, 3, 5]$
- SA_1 ne sadrži konačan abecedni poredak sufiksa
(nije konačno sufiksno polje SA)!

h -sortiranje (3)

Primjer:

za niz $S = \text{ACCA}\$$ odrediti ISA_1 i SA_1 ($\$$ je abecedno najveći znak)

- $SA_1 = [\underline{1}, 4, \underline{2}, 3, 5] \rightarrow$ nije konačan poredak sufiksa
(nije konačno sufiksno polje SA)
- ISA_1 – inverzno sufiksno polje za 1-sortirane sufikse
(nije konačno inverzno sufiksno polje ISA)
 - općenito: $ISA[i] = j \leftrightarrow SA[j] = i$, tj. i -ti sufiks ima j -ti rang (abecedno)
ovdje : $ISA_1[i] = j \leftrightarrow SA_1[j] = i$
- $ISA_1 = [1, 2, 2, 1, 3]$ ili
 $ISA_1 = [1, 3, 3, 1, 5]$

h -sortiranje (4)

- $(h+1)$ -sortiranje je proširenje h -sortiranja

Primjer (prema Puglisi *et al.*, 2007; \$ abecedno najmanji znak):

	1	2	3	4	5	6	7	8	9	10	11	12
S	A	B	E	A	C	A	D	A	B	E	A	\$
SA₁	12	(1	4	6	8	11)	(2	9)	5	7	(3	10)
ISA₁	2	7	11	2	9	2	10	2	7	11	2	1
SA₂	12	(11	(1	8)	4	6)	(2	9)	5	7	(3	10)
ISA₂	3	7	11	5	9	6	10	3	7	11	2	1
SA₃	12	11	(1	8)	4	6	(2	9)	5	7	10	3
ISA₃	3	7	12	5	9	6	10	3	7	11	2	1

Udvostručavanje prefiksa

- Karp *et al.* (1972)
- Neka su s_i i s_j sufiksi koji pripadaju istom pretincu nakon h -tog koraka, tj. $s_i =_h s_j$.
- u koraku $2h$ uspoređujemo **sljedećih h** znakova, tj. prvih h znakova sufiksa $s_{i'}$ i $s_{j'}$, gdje $s_{i'} = s_{i+h}$ i $s_{j'} = s_{j+h}$
- **Uočiti:** prvih h znakova sufiksa $s_{i'}$ i $s_{j'}$ su već uspoređeni u h -tom koraku te znamo njihov međusobni poredak
- međusobni odnos sufiksa $s_{i'}$ i $s_{j'}$ određujemo iz h -ranga, tj. iz ISA_h

Izgradnja SA u linearnom vremenu

- rekurzivni algoritmi:
 - Kärkkäinen-Sandersov (2003)
 - Kim *et al.* (2003)
 - Ko-Alurov (2003)
 - Nong-Zhang-Chanov (2009; 2011)

Kärkkäinen-Sandersov algoritam - ideja

- Ideja:
 - iz ulaznog niza S , tvore se nizovi S' i S''
 - zatim se pokaže da, ako je određen $SA(S')$, onda se može generirati $SA(S'')$, a zatim i $SA(S)$ iz $SA(S')$ i $SA(S'')$ u linearnom vremenu
 - S' se odabire tako da je $|S'| < 2 \cdot |S| / 3$.
 - ukupno vrijeme izvođenja: $\Theta(n)$

Kärkkäinen-Sandersov algoritam

Neka je S niz duljine n nad abecedom Σ .

Algoritam:

1. Rekurzivno sortirati $2n/3$ sufiksa s_i ($i \bmod 3 \neq 0$, tj. $i = 1, 2$).
2. Sortirati $n/3$ sufiksa s_i ($i \bmod 3 = 0$) koristeći rezultat koraka (1).
3. Napraviti spajanje (engl. *merge*) sortiranih polja dobivenih u koracima (1) i (2).

Nong-Zhang-Chanov algoritam: SA-IS

SA-IS(S, SA)

- ▷ S is the input string;
- ▷ SA is the output suffix array of S ;
- t : array $[0..n - 1]$ of boolean;
- P_1, S_1 : array $[0..n_1 - 1]$ of integer; ▷ $n_1 = \|S_1\|$
- B : array $[0..\|\Sigma(S)\| - 1]$ of integer;
- 1 Scan S once to classify all the characters as L- or S-type into t ;
- 2 Scan t once to find all the LMS-substrings in S into P ;
- 3 Induced sort all the LMS-substrings using P_1 and B ;
- 4 Name each LMS-substring in S by its bucket index to get a new shortened string S_1 ;
- 5 **if** Each character in S_1 is unique
- 6 **then**
- 7 Directly compute SA_1 from S_1 ;
- 8 **else**
- 9 SA-IS(S_1, SA_1); ▷ Fire a recursive call
- 10 Induce SA from SA_1 ;
- 11 **return**

Nong, Zhang & Chan (2009; 2011)

SA-IS: poveznica s Ko-Alurovim algoritmom

- koristi osnovnu ideju i pojmove koje su Ko i Aluru uveli u svojem algoritmu (2003) za izgradnju sufiksnog polja
 - L-tip i S-tip: znak, sufiks, podniz
 - princip *podijeli, pa vladaj* (engl. *divide and conquer*)
 - sufiksi S-tipa su sortirani kao potproblem
 - inducirano sortiranje → sufiksi L-tipa se sortiraju inducirano temeljem sufiksa S-tipa
- ново:
 - uvođenje LMS-podnizova
 - potproblem: izgradnja suf. polja izgrađenog nad LMS-podnizovima (SA_1)
 - inducirano određivanje sufiksnog polja originalnog niza (SA) iz SA_1

SA-IS: S-tip i L-tip sufiksa

- $S[i, |S|]$ je S-tip sufiksa (engl. *S-type*):
 - ako je $S[i, |S|] < S[i + 1, |S|]$
ili
 - zadnji sufiks, tj. $S[|S|, |S|] = \$$
pretpostavka: $\$$ je abecedno manji od ostalih znakova iz Σ
- $S[i, |S|]$ je L-tip sufiksa (engl. *L-type*):
 - ako je $S[i, |S|] > S[i + 1, |S|]$

SA-IS: S-tip i L-tip znaka

- $S[i]$ je S-tip znaka:
 - ako je $S[i] < S[i + 1]$ ili
 - ako je $S[i] = S[i + 1]$ i $S[i + 1, |S|]$ je S-tip
- $S[i]$ je L-tip znaka:
 - ako je $S[i] > S[i + 1]$ ili
 - ako je $S[i] = S[i + 1]$ i $S[i + 1, |S|]$ je L-tip
- Primjer:


 $S = \text{ATTAGCGAGCG\$}$

	0	1	2	3	4	5	6	7	8	9	10	11
S	A	T	T	A	G	C	G	A	G	C	G	\$
t	S	L	L	S	L	S	L	S	L	S	L	S

SA-IS: LMS

- $S[i]$ je **LMS-znak** (engl. *leftmost S-type*)
 - ako je $S[i]$ S-tip i $S[i - 1]$ L-tip ($i \geq 1$)
- $S[i, |S|]$ je **LMS-sufiks**
 - ako je $S[i]$ LMS-znak
- $S[i, j]$ je **LMS-podniz**
 - ako je $S[i, j]$ znak za kraj niza ($i = j$)ili
 - ako su $S[i]$ i $S[j]$ LMS-znakovi, a između njih nema drugih LMS-znakova, $i \neq j$

SA-IS: Usporedba LMS-podnizova

- dva LMS-podniza uspoređuju se slijeva nadesno:
 - za svaki par znakova promatra se prvo njihov leksikografski poredak
 - ako imaju istu leksikografsku vrijednost, onda se promatra jesu li znakovi S-tipa ili L-tipa, gdje S-tip ima viši prioritet od L-tipa (tj. S-tip je leksikografski nakon L-tipa)
- jednakost dva LMS-podniza
 - dva su LMS-podniza jednaka, ako su jednake duljine i ako su im svi znakovi leksikografski jednaki te istog tipa

Nong-Zhang-Chanov algoritam: SA-IS

SA-IS(S, SA)

t : polje oznaka za
S-tip/L-tip \rightarrow 1/0

P_1 : polje pokazivača
na LMS-podnizove

B : polje pokazivača
na pretince unutar SA

$\triangleright S$ is the input string;
 $\triangleright SA$ is the output suffix array of S ;
 t : array $[0..n - 1]$ of boolean;
 P_1, S_1 : array $[0..n_1 - 1]$ of integer; $\triangleright n_1 = \|S_1\|$
 B : array $[0..\|\Sigma(S)\| - 1]$ of integer;
1 Scan S once to classify all the characters as
L- or S-type into t ;
2 Scan t once to find all the LMS-substrings in S into P_1 ;
3 Induced sort all the LMS-substrings using P_1 and B ;
4 Name each LMS-substring in S by its bucket
index to get a new shortened string S_1 ;
5 if Each character in S_1 is unique
6 then
7 Directly compute SA_1 from S_1 ;
8 else
9 SA-IS(S_1, SA_1); \triangleright Fire a recursive call
10 Induce SA from SA_1 ;
11 return

Nong, Zhang & Chan (2009; 2011)

SA-IS: polje pokazivača na LMS-podnizove

- Primjer: $S = \text{ATTAGCGAGCG}\$$
- P_1 je polje pokazivača na LMS-podnizove iz S
- Određivanje P_1 :
proći po S slijeva udesno i svaki LMS-znak dodati u polje P_1
 - $S[i]$ je LMS-znak:
 - (i) ako je $S[i]$ S-tip znaka i ako je $S[i - 1]$ L-tip znaka; ili
 - (ii) ako je $S[i] = \$$

	0	1	2	3	4	5	6	7	8	9	10	11
S	A	T	T	A	G	C	G	A	G	C	G	\$
t	S	L	L	S	L	S	L	S	L	S	L	S
P_1		3	5	7	9	11						

Nong-Zhang-Chanov algoritam: SA-IS

SA-IS(S, SA)

- ▷ S is the input string;
- ▷ SA is the output suffix array of S ;
- t : array $[0..n - 1]$ of boolean;
- P_1, S_1 : array $[0..n_1 - 1]$ of integer; ▷ $n_1 = \|S_1\|$
- B : array $[0..\|\Sigma(S)\| - 1]$ of integer;
- 1 Scan S once to classify all the characters as L- or S-type into t ;
- 2 Scan t once to find all the LMS-substrings in S into P ;
- 3 Induced sort all the LMS-substrings using P_1 and B ;
- 4 Name each LMS-substring in S by its bucket index to get a new shortened string S_1 ;
- 5 **if** Each character in S_1 is unique
- 6 **then**
- 7 Directly compute SA_1 from S_1 ;
- 8 **else**
- 9 SA-IS(S_1, SA_1); ▷ Fire a recursive call
- 10 Induce SA from SA_1 ;
- 11 **return**

Nong, Zhang & Chan (2009; 2011)

SA-IS: polje B

- Izgraditi polje pretinaca B (engl. *bucket*):
 - $B[k]$ je pokazivač na pretinac unutar SA za znak abecede $\Sigma[k]$, gdje je Σ abeceda nad znakovima iz S , $k = 0, \dots, |\Sigma|$.
 - $B[k]$ sadrži pokazivač na početak ili kraj pretinca za sufikse koji počinju znakom $\Sigma[k]$
 - omogućuje promatranje sufiksnog polja SA kao niza potpolja, gdje svako potpolje sadrži indekse sortiranih sufiksa koji počinju istim početnim znakom
- Primjer: $S = \text{ATTAGCGAGCG}\$$

\$ A C G T

$SA = \{ _ \} \{ _, _, _ \} \{ _, _ \} \{ _, _, _, _ \} \{ _, _ \}$

SA-IS: Algoritam za inducirano sortiranje LMS-podnizova korištenjem polja P_1 i B (korak 1)

Algoritam 1

1. Postaviti sve članove polja SA na -1.

Postaviti pokazivač $B[k]$ na **kraj** k -tog potpolja od SA za svaki $k = 0, \dots, |\Sigma|$.

Proći kroz polje **S slijeva nadesno**:

- dodavati indekse LMS-sufiksa iz S u pripadajuće potpolje od **kraja** potpolja prema početku
- nakon svakog dodavanja, pomaknuti pokazivač $B[k]$ za jedno mjesto ulijevo

SA-IS: Algoritam za inducirano sortiranje LMS-podnizova korištenjem polja P_1 i B (koraci 2 i 3)

2. Postaviti pokazivač $B[k]$ na početak k -tog potpolja od SA za svaki $k = \Sigma[0], \dots, \Sigma[|\Sigma|]$. Proći kroz polje **SA slijeva nadesno**:
 - za svaki $SA[i] > 0$:
ako je znak $k = S[SA[i] - 1]$ L-tip, onda dodati $SA[i] - 1$ na početak k -tog potpolja i pomaknuti $B[k]$ jedno mjesto udesno
3. Postaviti pokazivač $B[k]$ na kraj k -tog potpolja od SA za svaki $k = 0, \dots, |\Sigma|$. Proći kroz polje **SA zdesna nalijevo**:
 - za svaki $SA[i] > 0$:
ako je znak $k = S[SA[i] - 1]$ S-tip, onda dodati $SA[i] - 1$ na kraj k -tog potpolja i pomaknuti $B[k]$ jedno mjesto ulijevo

SA-IS: Inducirano sortiranje LMS-podnizova korištenjem polja P_1 i B – primjer (korak 1)

	0	1	2	3	4	5	6	7	8	9	10	11
S	A	T	T	A	G	C	G	A	G	C	G	\$
t	S	L	L	S	L	S	L	S	L	S	L	S

P_1	3	5	7	9	11
-------	---	---	---	---	----

$B['\$'] = 0; B['A'] = 3; B['C'] = 5; B['G'] = 9; B['T'] = 11;$

1. korak: $S = \text{ATTAGCGAGCG\$}$

\$ A C G T

Dodati indekse LMS-sufiksa iz S u pripadajuće potpolje od kraja potpolja prema početku

$SA = \{-1\} \{-1, -1, -1\} \{-1, -1\} \{-1, -1, -1, -1\} \{-1, -1\}$

$SA = \{11\} \{-1, 7, 3\} \{9, 5\} \{-1, -1, -1, -1\} \{-1, -1\}$

$B['\$'] = -1; B['A'] = 1; B['C'] = 3; B['G'] = 9; B['T'] = 11;$

SA-IS: Inducirano sortiranje LMS-podnizova korištenjem polja P_1 i B – primjer (korak 2)

	0	1	2	3	4	5	6	7	8	9	10	11
s	A	T	T	A	G	C	G	A	G	C	G	\$
t	S	L	L	S	L	S	L	S	L	S	L	S

Postaviti $B[k]$ na početak k -tog potpolja za svaki k

za svaki $SA[i] > 0$:

ako je $k = S[SA[i] - 1]$ L-tip,
onda dodati $SA[i] - 1$ na početak
 k -tog potpolja; $B[k] ++$

2. korak: $S = \text{ATTAGCGAGCG\$}$

$B['\$'] = 0$; $B['A'] = 1$; $B['C'] = 4$; $B['G'] = 6$; $B['T'] = 10$

\$	A			C			G			T		
0	1	2	3	4	5	6	7	8	9	10	11	

$SA = \{11\} \{-1, 7, 3\} \{9, 5\} \{-1, -1, -1, -1\} \{-1, -1\}$

$SA = \{\underline{11}\} \{-1, 7, 3\} \{9, 5\} \{10, -1, -1, -1\} \{-1, -1\}$ $i = 0 \rightarrow$ dodati 10, $B['G'] = 7$

$SA = \{11\} \{-\underline{1}, \underline{7}, 3\} \{9, 5\} \{10, 6, -1, -1\} \{-1, -1\}$ $i = 2 \rightarrow$ dodati 6, $B['G'] = 8$

$SA = \{11\} \{-1, 7, \underline{3}\} \{9, 5\} \{10, 6, -1, -1\} \{\underline{2}, -1\}$ $i = 3 \rightarrow$ dodati 2, $B['T'] = 11$

$SA = \{11\} \{-1, 7, 3\} \{\underline{9}, 5\} \{10, 6, 8, -1\} \{2, -1\}$ $i = 4 \rightarrow$ dodati 8, $B['G'] = 9$

$SA = \{11\} \{-1, 7, 3\} \{9, \underline{5}\} \{\underline{10}, \underline{6}, \underline{8}, \underline{4}\} \{2, -1\}$ $i = 5 \rightarrow$ dodati 4; $B['G'] = 10$;

$SA = \{11\} \{-1, 7, 3\} \{9, 5\} \{10, 6, 8, 4\} \{\underline{2}, \underline{1}\}$ $i = 10 \rightarrow$ dodati 1; $B['T'] = 12$;

SA-IS: Inducirano sortiranje LMS-podnizova korištenjem polja P_1 i B – primjer (korak 3)

	0	1	2	3	4	5	6	7	8	9	10	11
S	A	T	T	A	G	C	G	A	G	C	G	\$
t	S	L	L	S	L	S	L	S	L	S	L	S

Postaviti $B[k]$ na kraj k -tog potpolja za svaki k

3. korak: $S = \text{ATTAGCGAGCG\$}$

$B['\$'] = 0$; $B['A'] = 3$; $B['C'] = 5$; $B['G'] = 9$; $B['T'] = 11$

\$	A	C	G	T
0	1 2 3	4 5	6 7 8 9	10 11

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

$SA = \{11\} \{-1, 7, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\}$ $i = 11 \rightarrow$ dodati 0; $B['A'] = 2$;

$SA = \{11\} \{-1, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\}$ $i = 9 \rightarrow$ dodati 3; $B['A'] = 1$;

$SA = \{11\} \{7, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\}$ $i = 8 \rightarrow$ dodati 7; $B['A'] = 0$;

$SA = \{11\} \{7, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\}$ $i = 7 \rightarrow$ dodati 5; $B['C'] = 4$;

$SA = \{11\} \{7, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\}$ $i = 6 \rightarrow$ dodati 9; $B['C'] = 3$

za svaki $SA[i] > 0$:

ako je $k = S[SA[i] - 1]$ S-tip, onda dodati $SA[i] - 1$ na kraj k -tog potpolja; $B[k] --$;

Nong-Zhang-Chanov algoritam: SA-IS

SA-IS(S, SA)

- ▷ S is the input string;
- ▷ SA is the output suffix array of S ;
- t : array $[0..n - 1]$ of boolean;
- P_1, S_1 : array $[0..n_1 - 1]$ of integer; ▷ $n_1 = \|S_1\|$
- B : array $[0..\|\Sigma(S)\| - 1]$ of integer;
- 1 Scan S once to classify all the characters as L- or S-type into t ;
- 2 Scan t once to find all the LMS-substrings in S into P ;
- 3 Induced sort all the LMS-substrings using P_1 and B ;
- 4 Name each LMS-substring in S by its bucket index to get a new shortened string S_1 ;
- 5 **if** Each character in S_1 is unique
- 6 **then**
- 7 Directly compute SA_1 from S_1 ;
- 8 **else**
- 9 SA-IS(S_1, SA_1); ▷ Fire a recursive call
- 10 Induce SA from SA_1 ;
- 11 **return**


Nong, Zhang & Chan (2009; 2011)

SA-IS: Imenovanje LMS-podnizova

	0	1	2	3	4	5	6	7	8	9	10	11
<i>s</i>	A	T	T	A	G	C	G	A	G	C	G	\$
<i>t</i>	S	L	L	S	L	S	L	S	L	S	L	S

<i>P</i> ₁	3	5	7	9	11
<i>S</i> ₁	1	3	1	2	0

0	1	2	3	4
SA = { 11 }	{ 7 , 3 , 0}	{ 9 , 5 }	{10, 6, 8, 4}	{2, 1}



Izgraditi novi niz *S*₁

- svakom LMS-podnizu pridijeliti novo leksikografsko ime (cijeli broj)
- dva LMS-podniza imaju ista pridijeljena imena, ako su LMS-podnizovi jednaki, tj. ako su na svim mjestima u oba LMS-podniza jednaki znakovi i ti su znakovi istog tipa
- za $i = 0$ do $n - 1$:
ako je $SA[i]$ indeks početka LMS-podniza, pridijeli mu odgovarajuće ime: ili novo ili isto kao prethodniku (ako su međusobno jednaki) $\rightarrow S_1$

Nong-Zhang-Chanov algoritam: SA-IS

SA-IS(S, SA)

- ▷ S is the input string;
- ▷ SA is the output suffix array of S ;
- t : array $[0..n - 1]$ of boolean;
- P_1, S_1 : array $[0..n_1 - 1]$ of integer; ▷ $n_1 = \|S_1\|$
- B : array $[0..\|\Sigma(S)\| - 1]$ of integer;
- 1 Scan S once to classify all the characters as L- or S-type into t ;
- 2 Scan t once to find all the LMS-substrings in S into P ;
- 3 Induced sort all the LMS-substrings using P_1 and B ;
- 4 Name each LMS-substring in S by its bucket index to get a new shortened string S_1 ;
- 5 if Each character in S_1 is unique
- 6 **then**
- 7 Directly compute SA_1 from S_1 ;
- 8 **else**
- 9 SA-IS(S_1, SA_1); ▷ Fire a recursive call
- 10 Induce SA from SA_1 ;
- 11 **return**

Nong, Zhang & Chan (2009; 2011)

SA-IS: Rekurzivni poziv SA-IS (1)

	0	1	2	3	4	5	6	7	8	9	10	11
S	A	T	T	A	G	C	G	A	G	C	G	\$
t	S	L	L	S	L	S	L	S	L	S	L	S

P_1	3	5	7	9	11
S_1	1	3	1	2	0

- Ako svaki LMS-podniz ima jedinstveno ime (cijeli broj), onda se SA_1 može izgraditi izravno iz S_1
- Inače, pozvati rekurzivno SA-IS(S_1 , SA_1)
Implementacijski: S_1 i SA_1 se pohranjuju unutar originalnog SA .
(zato jer je S_1 sigurno najviše pola duljine od S)
- Ovdje: pozivamo rekurzivno SA-IS, gdje je $S_1 = "13120"$
- Rekurzivnim pozivom određujemo $SA_1 = \{4, 2, 0, 3, 1\}$

SA-IS: Rekurzivni poziv SA-IS (2)

	0	1	2	3	4
S_1	1	3	1	2	0
t_1	S	L	S	L	S

P_2	2	4
S_2	1	0

	0	1	2	3
$SA_1 = \{ _ \} \{ _, _ \} \{ _ \} \{ _ \}$				
$B_1[0] = 0; B_1[1] = 2; B_1[2] = 3; B_1[3] = 4;$				

- izgraditi polja t_1 i P_2
- izgraditi polje B_1
- inducirano sortiranje LMS-podnizova korištenjem P_2 i B_1
 - korak 1 (dodati indekse LMS-sufiksa iz S_1 u pripadajuće potpolje od kraja potpolja prema početku): $SA_1 = \{4\} \{-1, 2\} \{-1\} \{-1\}$
 - korak 2 (*induceSA*): $SA_1 = \{4\} \{-1, 2\} \{3\} \{1\}$
 - korak 3 (*induceSAs*): $SA_1 = \{4\} \{2, 0\} \{3\} \{1\}$
- izgraditi S_2 (pridjeljivanje imena LMS-podnizovima) – proći kroz SA_1 :
 $SA_1 = \{\underline{4}, \underline{2}, 0, 3, 1\} \rightarrow S_2 = "10"$

SA-IS: Rekurzivni poziv SA-IS (3)

	0	1	2	3	4	P_2	2	4
S_1	1	3	1	2	0	S_2	1	0
T_1	S	L	S	L	S	SA_2	1	0

- iz $S_2 = "10"$ je moguće izravno odrediti $\underline{SA_2}$: $SA_2[S_2[i]] = i$; $i = 0, 1$
 $SA_2 = \{1, 0\}$

Inducirati SA_1 iz SA_2 : Algoritam 2

- 1. korak:

Postaviti sve članove SA_1 na -1.

Pronaći kraj pretinca $B_1[k]$ za svaki k iz abecede nad S_1 .

Prolazeći kroz SA_2 od kraja prema početku, staviti $P_2[SA_2[i]]$ na kraj pretinca za pripadajući sufiks i pomaknuti $B_1[k]$ za jedno mjesto ulijevo

- 2. korak: isti kao 2. korak kod Algoritma 1 (*induceSAI*)
- 3. korak: isti kao 3. korak kod Algoritma 1 (*induceSAs*)

SA-IS: Odrediti SA_1 iz SA_2

	0	1	2	3	4		P_2	2	4
S_1	1	3	1	2	0	S_2	1	0	
T_1	S	L	S	L	S	SA_2	1	0	

	0	1	2	3
SA_1	$\{-1\}$	$\{-1, -1\}$	$\{-1\}$	$\{-1\}$
B_1	$B_1[0] = 0$	$B_1[1] = 2$	$B_1[2] = 3$	$B_1[3] = 4$

- 1. korak:

Postaviti sve članove SA_1 na -1.

Pronaći kraj pretinca $B_1[k]$ za svaki k iz abecede nad S_1 .

Prolazeći kroz SA_2 od kraja prema početku, staviti $P_2[SA_2[i]]$ na kraj pretinca za pripadajući sufiks i pomaknuti $B_1[k]$ za jedno mjesto ulijevo

$SA_1 = \{4\} \{-1, 2\} \{-1\} \{-1\}$; $B_1[0] = -1$; $B_1[1] = 1$; $B_1[2] = 3$; $B_1[3] = 4$;

- 2. korak (*induceSAI*): $SA_1 = \{4\} \{-1, 2\} \{3\} \{1\}$
- 3. korak (*induceSAs*): $SA_1 = \{4\} \{2, 0\} \{3\} \{1\}$

SA-IS: Odrediti SA iz SA_1

	0	1	2	3	4	5	6	7	8	9	10	11	P_1	3	5	7	9	11
s	A	T	T	A	G	C	G	A	G	C	G	\$	s_1	1	3	1	2	0
t	S	L	L	S	L	S	L	S	L	S	L	S	SA_1	4	2	0	3	1

- 1. korak: svi članovi SA na -1: $SA = \{-1\} \{-1, -1, -1\} \{-1, -1\} \{-1, -1, -1, -1\} \{-1, -1\}$

- Postaviti $B[k]$: $B['\$'] = 0$; $B['A'] = 3$; $B['C'] = 5$; $B['G'] = 9$; $B['T'] = 11$

Prolazeći kroz SA_1 od kraja prema početku, staviti $P_1[SA_1[i]]$ na kraj pretinca za pripadajući sufiks i pomaknuti $B[k]$ za jedno mjesto ulijevo

$SA = \{11\} \{-1, 7, 3\} \{9, 5\} \{-1, -1, -1, -1\} \{-1, -1\}$

- $B['\$'] = -1$; $B['A'] = 1$; $B['C'] = 3$; $B['G'] = 9$; $B['T'] = 11$

- 2. korak (*induceSAI*): $SA = \{11\} \{-1, 7, 3\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\}$ (kao na str. 35)

- 3. korak (*induceSAs*): $SA = \{11\} \{7, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\}$ (kao na str. 36)

Primjena sufiksnih polja i stabala u biologiji

- A bioinformatician's guide to the forefront of suffix array construction algorithms (Shrestha et al. 2014)
- MUMMER (Marçais et al. 2018; Kurtz *et al.*, 2004)
- MAVID - a multiple alignment program for large genomic sequences (Bray & Pachter, 2004).
- essaMEM: finding maximal exact matches using enhanced sparse suffix arrays (Vyverman *et al.* 2013)

Popis literature

- Manber, U., Myers, G. 1990. Suffix arrays: a new method for on-line string searches. *In Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms* 90(319): 327
- Abouelhoda, M. I., Kurtz, S., Ohlebusch, E. 2004. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms* **2**: 53.
- Shrestha *et al.* 2014. A bioinformatician's guide to the forefront of suffix array construction algorithms. *Briefings in Bioinformatics*
- S. J. Puglisi, W. F. Smyth, A. Turpin. 2007. A taxonomy of suffix array construction algorithms, *ACM Computing Surveys*, Vol. 39 (2)
- Kärkkäinen, J. , Sanders, P. 2003. Simple linear work suffix array construction. *In Proceedings of the 30th International Colloquium Automata, Languages and Programming*. Lecture Notes in Computer Science, vol. 2971. Springer-Verlag, Berlin, 943–955.
- Nong, G., Zhang, S., Chan, W. H. 2009. Linear Suffix Array Construction by Almost Pure Induced-Sorting, *Proceedings of 19th IEEE Data Compression Conference (IEEE DCC) 2009*