

Text Indexing

Lecture 03: Longest Common Prefix Array

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License © ⓘ ⓘ: www.creativecommons.org/licenses/by-sa/4.0 | commit a82b315 compiled at 2021-12-11-14:54



<https://pingo.scc.kit.edu/265548>

Recap: Suffix Array and LCP-Array

Definition: Suffix Array [GBS92; MM93]

Given a text T of length n , the **suffix array** (SA) is a permutation of $[1..n]$, such that for $i \leq j \in [1..n]$

$$T[SA[i]..n] \leq T[SA[j]..n]$$

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3

\$	a	a	a	a	a	b	b	b	b	b	c	c	
	\$	b	b	b	b	a	a	a	a	a	a	a	
		a	a	a	a								
		b	b	b	b								
		c	c	c	c								
		a	a	a	a								
		b	b	b	b								
		c	c	c	c								
		a	a	a	a								
		b	b	b	b								
		a	a	a	a								
		b	b	b	b								
		a	a	a	a								
		\$											

Recap: Suffix Array and LCP-Array

Definition: Suffix Array [GBS92; MM93]

Given a text T of length n , the **suffix array** (SA) is a permutation of $[1..n]$, such that for $i \leq j \in [1..n]$

$$T[SA[i]..n] \leq T[SA[j]..n]$$

Definition: Longest Common Prefix Array

Given a text T of length n and its SA, the **LCP-array** is defined as

$$LCP[i] = \begin{cases} 0 & i = 1 \\ \max\{\ell: T[SA[i]..SA[i] + \ell) = \\ T[SA[i - 1]..SA[i - 1] + \ell)\} & i \neq 1 \end{cases}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3

\$	a	a	a	a	a	b	b	b	b	b	c	c
\$	\$	b	b	b	b	a	a	b	c	c	a	a
		a	b	c	c	\$	b	a	a	a	b	b
		b	a	a	a		c	b	b	b	b	c
		c	\$	b	b		a	a	a	a	a	a
		a		b	a		b	b	\$	b	\$	b
		b		\$	a		c	a		a		a
		c			b		a	b		b		b
		a			a		b	b		a		a
		b			\$		a	a		\$		\$
		b					b	b				
		a					a	a				
		\$					\$	\$				

Recap: Suffix Array and LCP-Array

Definition: Suffix Array [GBS92; MM93]

Given a text T of length n , the **suffix array** (SA) is a permutation of $[1..n]$, such that for $i \leq j \in [1..n]$

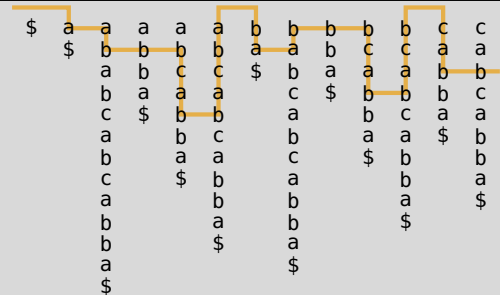
$$T[SA[i]..n] \leq T[SA[j]..n]$$

Definition: Longest Common Prefix Array

Given a text T of length n and its SA, the **LCP-array** is defined as

$$LCP[i] = \begin{cases} 0 & i = 1 \\ \max\{\ell: T[SA[i]..SA[i] + \ell) = \\ T[SA[i - 1]..SA[i - 1] + \ell)\} & i \neq 1 \end{cases}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3



Naive Computation of the LCP-Array

Task

- given: text T of length n and its suffix array
- wanted: longest common prefix array

Naive Construction

- for each pair $(SA[i - 1], SA[i])$
- compare $T[SA[i - 1] + \ell]$ and $T[SA[i] + \ell]$
- until mismatch

Naive Computation of the LCP-Array

Task

- given: text T of length n and its suffix array
- wanted: longest common prefix array

Naive Construction

- for each pair $(SA[i - 1], SA[i])$
- compare $T[SA[i - 1] + \ell]$ and $T[SA[i] + \ell]$
- until mismatch

Running Time

- naive construction requires $O(n^2)$ time
- all-a texts are worst case
- here $LCP[1] = 0$, $LCP[2] = 0$, and $LCP[i] = i - 2$
- only distinguishable character is \$

Properties of the LCP-Array

- do not compare all suffixes naively
- compare only unknown parts

Properties of the LCP-Array

- do not compare all suffixes naively
- compare only unknown parts

Lemma: Values in LCP-array

Given a text T of length n , its suffix array SA and LCP -array LCP , then

$$\exists i \in [1, n): LCP[i] = \ell > 0 \Rightarrow \exists j \in [1, n): LCP[j] = \ell - 1$$

Properties of the LCP-Array


- do not compare all suffixes naively
- compare only unknown parts

Lemma: Values in LCP-array

Given a text T of length n , its suffix array SA and LCP -array LCP , then

$$\exists i \in [1, n): LCP[i] = \ell > 0 \Rightarrow \exists j \in [1, n): LCP[j] = \ell - 1$$

Proof (Sketch)

- let $LCP[i] = k > 0$
- $T[SA[i]..SA[i] + k) = T[SA[i - 1]..SA[i - 1] + k)$
- $T[SA[i] + 1..SA[i] + k) = T[SA[i - 1] + 1..SA[i - 1] + k)$
- not necessarily next to each other in SA 

The Inverse Suffix Array

Definition: Inverse Suffix Array

Given a suffix array SA of length n , the **inverse suffix array** ($ISA = SA^{-1}$) is

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

for $n \in [1..n]$

- inverse permutation ⓘ as hinted by the name
- where i is a suffix in the suffix array

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  |   |   j = SA[ISA[i] - 1]
6  |   |   while T[i + ℓ] = T[j + ℓ] do
7  |   |   |   ℓ = ℓ + 1
8  |   |   |   LCP[ISA[i]] = ℓ
9  |   |   |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  |   |   j = SA[ISA[i] - 1]
6  |   |   while T[i + ℓ] = T[j + ℓ] do
7  |   |   |   ℓ = ℓ + 1
8  |   |   |   LCP[ISA[i]] = ℓ
9  |   |   |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  |   |   j = SA[ISA[i] - 1]
6  |   |   while T[i + ℓ] = T[j + ℓ] do
7  |   |   |   ℓ = ℓ + 1
8  |   |   |   LCP[ISA[i]] = ℓ
9  |   |   |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

```

Function LinearTimeLCP( $T, SA[1..n]$ ):
1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

```

Function LinearTimeLCP( $T, SA[1..n]$ ):
1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

```

Function LinearTimeLCP( $T$ ,  $SA[1..n]$ ):
1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0$ ,  $LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4  |   if  $ISA[i] \neq 1$  then
5  |   |    $j = SA[ISA[i] - 1]$ 
6  |   |   while  $T[i + \ell] = T[j + \ell]$  do
7  |   |   |    $\ell = \ell + 1$ 
8  |   |    $LCP[ISA[i]] = \ell$ 
9  |   |    $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  |   |   j = SA[ISA[i] - 1]
6  |   |   while T[i + ℓ] = T[j + ℓ] do
7  |   |   |   ℓ = ℓ + 1
8  |   |   |   LCP[ISA[i]] = ℓ
9  |   |   |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	1	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  |   |   j = SA[ISA[i] - 1]
6  |   |   while T[i + ℓ] = T[j + ℓ] do
7  |   |   |   ℓ = ℓ + 1
8  |   |   |   LCP[ISA[i]] = ℓ
9  |   |   |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	1	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  | |   j = SA[ISA[i] - 1]
6  | |   while T[i + ℓ] = T[j + ℓ] do
7  | | |   ℓ = ℓ + 1
8  | | |   LCP[ISA[i]] = ℓ
9  | | |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	1	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4      if  $ISA[i] \neq 1$  then
5           $j = SA[ISA[i] - 1]$ 
6          while  $T[i + \ell] = T[j + \ell]$  do
7               $\ell = \ell + 1$ 
8               $LCP[ISA[i]] = \ell$ 
9               $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  | |   j = SA[ISA[i] - 1]
6  | |   while T[i + ℓ] = T[j + ℓ] do
7  | | |   ℓ = ℓ + 1
8  | | |   LCP[ISA[i]] = ℓ
9  | | |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  | |   j = SA[ISA[i] - 1]
6  | |   while T[i + ℓ] = T[j + ℓ] do
7  | | |   ℓ = ℓ + 1
8  | | |   LCP[ISA[i]] = ℓ
9  | | |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4      if  $ISA[i] \neq 1$  then
5           $j = SA[ISA[i] - 1]$ 
6          while  $T[i + \ell] = T[j + \ell]$  do
7               $\ell = \ell + 1$ 
8               $LCP[ISA[i]] = \ell$ 
9               $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

```

Function LinearTimeLCP( $T$ ,  $SA[1..n]$ ):
1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0$ ,  $LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

```

Function LinearTimeLCP( $T$ ,  $SA[1..n]$ ):
1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0$ ,  $LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  |   |   j = SA[ISA[i] - 1]
6  |   |   while T[i + ℓ] = T[j + ℓ] do
7  |   |   |   ℓ = ℓ + 1
8  |   |   |   LCP[ISA[i]] = ℓ
9  |   |   |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	1	⊥	2	⊥	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4      if  $ISA[i] \neq 1$  then
5           $j = SA[ISA[i] - 1]$ 
6          while  $T[i + \ell] = T[j + \ell]$  do
7               $\ell = \ell + 1$ 
8               $LCP[ISA[i]] = \ell$ 
9               $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	⊥	1	⊥	2	5	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

- compute suffixes in text order
- use *ISA* to find lex. smaller suffix

```

Function LinearTimeLCP(T, SA[1..n]):
1  for i = 1, ..., n do ISA[SA[i]] = i
2  ℓ = 0, LCP[1] = 0
3  for i = 1, ..., n do
4  |   if ISA[i] ≠ 1 then
5  |   |   j = SA[ISA[i] - 1]
6  |   |   while T[i + ℓ] = T[j + ℓ] do
7  |   |   |   ℓ = ℓ + 1
8  |   |   |   LCP[ISA[i]] = ℓ
9  |   |   |   ℓ = max{0, ℓ - 1}
10 return LCP
  
```

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	b	a	b	c	a	b	c	a	b	b	a	\$
<i>SA</i>	13	12	1	9	6	3	11	2	10	7	4	8	5
<i>ISA</i>	3	8	6	11	13	5	10	12	4	9	7	2	1
<i>LCP</i>	0	⊥	1	⊥	2	5	⊥	⊥	⊥	⊥	⊥	⊥	⊥

Linear Time Construction [Kas+01]

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4    if  $ISA[i] \neq 1$  then
5       $j = SA[ISA[i] - 1]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7         $\ell = \ell + 1$ 
8       $LCP[ISA[i]] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3

Linear Time Construction [Kas+01]


Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4      if  $ISA[i] \neq 1$  then
5           $j = SA[ISA[i] - 1]$ 
6          while  $T[i + \ell] = T[j + \ell]$  do
7               $\ell = \ell + 1$ 
8               $LCP[ISA[i]] = \ell$ 
9               $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

- compute suffixes in text order
- use ISA to find lex. smaller suffix

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
ISA	3	8	6	11	13	5	10	12	4	9	7	2	1
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3

- correctness and running time 

The Φ -Array

Definition: Φ -Array

Given a text T of length n and its suffix array SA , the Φ -array is defined (for $i > 1$) as

$$\Phi[SA[i]] = SA[i - 1]$$

- $\Phi[i]$ gives suffix that is needed for comparison
- not a permutation of SA

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
Φ	12	11	6	7	8	9	10	4	1	2	3	13	-

Better Linear Time Construction [KMP09]

Function Φ -Algorithm($T, SA[1..n]$):

```

1   $\Phi[n] = SA[n]$   $\bullet$   $SA[1] = n$ ;  $T$  has sentinel
2  for  $i = 2, \dots, n$  do  $\Phi[SA[i]] = SA[i - 1]$ 
3   $\ell = 0$ 
4  for  $i = 1, \dots, n$  do
5  |    $j = \Phi[i]$ 
6  |   while  $T[i + \ell] = T[j + \ell]$  do
7  |   |    $\ell = \ell + 1$ 
8  |    $\Phi[i] = \ell$ 
9  |    $\ell = \max\{0, \ell - 1\}$ 
10 for  $i = 1, \dots, n$  do  $LCP[i] = \Phi[SA[i]]$ 
11 return  $LCP$ 
  
```

- compute LCP -array in text order
- reorder LCP -array as final step

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
Φ	12	11	6	7	8	9	10	4	1	2	3	13	-
LCP	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp	\perp

Better Linear Time Construction [KMP09]


Function Φ -Algorithm($T, SA[1..n]$):

```

1   $\Phi[n] = SA[n]$   $\bullet SA[1] = n$ ;  $T$  has sentinel
2  for  $i = 2, \dots, n$  do  $\Phi[SA[i]] = SA[i - 1]$ 
3   $\ell = 0$ 
4  for  $i = 1, \dots, n$  do
5  |    $j = \Phi[i]$ 
6  |   while  $T[i + \ell] = T[j + \ell]$  do
7  |   |    $\ell = \ell + 1$ 
8  |    $\Phi[i] = \ell$ 
9  |    $\ell = \max\{0, \ell - 1\}$ 
10 for  $i = 1, \dots, n$  do  $LCP[i] = \Phi[SA[i]]$ 
11 return  $LCP$ 
  
```

- compute LCP -array in text order
- reorder LCP -array as final step

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
Φ	12	11	6	7	8	9	10	4	1	2	3	13	-
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3

- example: 
- correctness and running time similar

Better Linear Time Construction [KMP09]


Function Φ -Algorithm($T, SA[1..n]$):

```

1   $\Phi[n] = SA[n]$   $\bullet$   $SA[1] = n$ ;  $T$  has sentinel
2  for  $i = 2, \dots, n$  do  $\Phi[SA[i]] = SA[i - 1]$ 
3   $\ell = 0$ 
4  for  $i = 1, \dots, n$  do
5  |    $j = \Phi[i]$ 
6  |   while  $T[i + \ell] = T[j + \ell]$  do
7  |   |    $\ell = \ell + 1$ 
8  |    $\Phi[i] = \ell$ 
9  |    $\ell = \max\{0, \ell - 1\}$ 
10 for  $i = 1, \dots, n$  do  $LCP[i] = \Phi[SA[i]]$ 
11 return  $LCP$ 
  
```

- compute LCP -array in text order
- reorder LCP -array as final step

	1	2	3	4	5	6	7	8	9	10	11	12	13
T	a	b	a	b	c	a	b	c	a	b	b	a	\$
SA	13	12	1	9	6	3	11	2	10	7	4	8	5
Φ	12	11	6	7	8	9	10	4	1	2	3	13	-
LCP	0	0	1	2	2	5	0	2	1	1	4	0	3

- example: 
- correctness and running time similar

Brief Remainder: Cache & Cache Misses

- cache is small but fast memory
- located on CPU
- cache miss is failure to retrieve data from cache
- instead data has to be loaded from main memory

Cache Sizes (AMD Ryzen 7 PRO 4750U)

- L1: 256 KiB (8 instances)
- L2: 4 MiB (8 instances)
- L3: 8 MiB (2 instances)

Brief Remainder: Cache & Cache Misses

- cache is small but fast memory
- located on CPU
- cache miss is failure to retrieve data from cache
- instead data has to be loaded from main memory

Cache Sizes (AMD Ryzen 7 PRO 4750U)

- L1: 256 KiB (8 instances)
- L2: 4 MiB (8 instances)
- L3: 8 MiB (2 instances)

Latency Numbers

- L1 cache reference ≈ 1 ns
- L2 cache reference ≈ 4 ns
- main memory reference ≈ 100 ns

Better Due to Less Cache Misses

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4      if  $ISA[i] \neq 1$  then
5           $j = SA[ISA[i] - 1]$ 
6          while  $T[i + \ell] = T[j + \ell]$  do
7               $\ell = \ell + 1$ 
8               $LCP[ISA[i]] = \ell$ 
9               $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

Function Φ -Algorithm($T, SA[1..n]$):

```

1   $\Phi[n] = SA[n] \oplus SA[1] = n$ ;  $T$  has sentinel
2  for  $i = 2, \dots, n$  do  $\Phi[SA[i]] = SA[i - 1]$ 
3   $\ell = 0$ 
4  for  $i = 1, \dots, n$  do
5       $j = \Phi[i]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7           $\ell = \ell + 1$ 
8           $\Phi[i] = \ell$ 
9           $\ell = \max\{0, \ell - 1\}$ 
10 for  $i = 1, \dots, n$  do  $LCP[i] = \Phi[SA[i]]$ 
11 return  $LCP$ 
  
```

Better Due to Less Cache Misses

Function LinearTimeLCP($T, SA[1..n]$):

```

1  for  $i = 1, \dots, n$  do  $ISA[SA[i]] = i$ 
2   $\ell = 0, LCP[1] = 0$ 
3  for  $i = 1, \dots, n$  do
4      if  $ISA[i] \neq 1$  then
5           $j = SA[ISA[i] - 1]$ 
6          while  $T[i + \ell] = T[j + \ell]$  do
7               $\ell = \ell + 1$ 
8           $LCP[ISA[i]] = \ell$ 
9           $\ell = \max\{0, \ell - 1\}$ 
10 return  $LCP$ 
  
```

Function Φ -Algorithm($T, SA[1..n]$):

```

1   $\Phi[n] = SA[n] \oplus SA[1] = n$ ;  $T$  has sentinel
2  for  $i = 2, \dots, n$  do  $\Phi[SA[i]] = SA[i - 1]$ 
3   $\ell = 0$ 
4  for  $i = 1, \dots, n$  do
5       $j = \Phi[i]$ 
6      while  $T[i + \ell] = T[j + \ell]$  do
7           $\ell = \ell + 1$ 
8       $\Phi[i] = \ell$ 
9       $\ell = \max\{0, \ell - 1\}$ 
10 for  $i = 1, \dots, n$  do  $LCP[i] = \Phi[SA[i]]$ 
11 return  $LCP$ 
  
```

Practical Comparison of Both Algorithms (1/2)

Pizza & Chili Corpus

- <http://pizzachili.dcc.uchile.cl/>
- de facto standard text corpus

Used in Experiment (50 MB)

- **dblp** XML-Data providing bibliographic information
- **DNA** DNA reads from the Gutenberg Project
- **english** English texts of the Gutenberg Project
- **sources** Source code from the Linux kernel

Experimental Setup

- used text described above
- on T14s with AMD Ryzen 7 PRO 4750U
- times are average of five runs

Practical Comparison of Both Algorithms (2/2)

Text	Naive (ms)	[Kas+01] (ms)	[KMP09] (ms)
dblp	9121.6	3479.0	2567.2
DNA	6763.0	6152.2	4174.6
english	99811.4	4899.8	3316.2
sources	12687.6	3486.4	2536.6

Permuted LCP-Array [KMP09]

Definition: PLCP-Array

- $PLCP[SA[i]] = LCP[i]$
- $PLCP[i] = lcp(i, SA[i - 1]) = lcp(i, \Phi[i])$

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	a	c	a	c	a	c	b	a	a	c	b	\$
<i>SA</i>	13	1	9	2	4	10	6	12	8	3	5	11	7
<i>LCP</i>	0	0	3	1	4	2	3	0	1	0	3	1	2
<i>PLCP</i>	0	1	0	4	3	3	2	1	3	2	1	0	0
Φ	12	8	7	1	2	9	10	11	0	3	4	5	-

Permuted LCP-Array [KMP09]

Definition: PLCP-Array

- $PLCP[SA[i]] = LCP[i]$
- $PLCP[i] = lcp(i, SA[i - 1]) = lcp(i, \Phi[i])$

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	a	c	a	c	a	c	b	a	a	c	b	\$
<i>SA</i>	13	1	9	2	4	10	6	12	8	3	5	11	7
<i>LCP</i>	0	0	3	1	4	2	3	0	1	0	3	1	2
<i>PLCP</i>	0	1	0	4	3	3	2	1	3	2	1	0	0
Φ	12	8	7	1	2	9	10	11	0	3	4	5	-

Permuted LCP-Array [KMP09]

Definition: PLCP-Array

- $PLCP[SA[i]] = LCP[i]$
- $PLCP[i] = lcp(i, SA[i - 1]) = lcp(i, \Phi[i])$

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	a	c	a	c	a	c	b	a	a	c	b	\$
<i>SA</i>	13	1	9	2	4	10	6	12	8	3	5	11	7
<i>LCP</i>	0	0	3	1	4	2	3	0	1	0	3	1	2
<i>PLCP</i>	0	1	0	4	3	3	2	1	3	2	1	0	0
Φ	12	8	7	1	2	9	10	11	0	3	4	5	-

Permuted LCP-Array [KMP09]

Definition: PLCP-Array

- $PLCP[SA[i]] = LCP[i]$
- $PLCP[i] = lcp(i, SA[i - 1]) = lcp(i, \Phi[i])$

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	a	c	a	c	a	c	b	a	a	c	b	\$
<i>SA</i>	13	1	9	2	4	10	6	12	8	3	5	11	7
<i>LCP</i>	0	0	3	1	4	2	3	0	1	0	3	1	2
<i>PLCP</i>	0	1	0	4	3	3	2	1	3	2	1	0	0
Φ	12	8	7	1	2	9	10	11	0	3	4	5	-

Permuted LCP-Array [KMP09]

Definition: PLCP-Array

- $PLCP[SA[i]] = LCP[i]$
- $PLCP[i] = lcp(i, SA[i - 1]) = lcp(i, \Phi[i])$

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	a	c	a	c	a	c	b	a	a	c	b	\$
<i>SA</i>	13	1	9	2	4	10	6	12	8	3	5	11	7
<i>LCP</i>	0	0	3	1	4	2	3	0	1	0	3	1	2
<i>PLCP</i>	0	1	0	4	3	3	2	1	3	2	1	0	0
Φ	12	8	7	1	2	9	10	11	0	3	4	5	-

Permuted LCP-Array [KMP09]

Definition: PLCP-Array

- $PLCP[SA[i]] = LCP[i]$
- $PLCP[i] = lcp(i, SA[i - 1]) = lcp(i, \Phi[i])$

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	a	c	a	c	a	c	b	a	a	c	b	\$
<i>SA</i>	13	1	9	2	4	10	6	12	8	3	5	11	7
<i>LCP</i>	0	0	3	1	4	2	3	0	1	0	3	1	2
<i>PLCP</i>	0	1	0	4	3	3	2	1	3	2	1	0	0
Φ	12	8	7	1	2	9	10	11	0	3	4	5	-



Permuted LCP-Array [KMP09]

Definition: PLCP-Array

- $PLCP[SA[i]] = LCP[i]$
 - $PLCP[i] = lcp(i, SA[i - 1]) = lcp(i, \Phi[i])$
-
- $PLCP[i] \geq PLCP[i - 1] - 1$
 - $T[i - 1] = T[\Phi[i] - 1] \Rightarrow PLCP[i]$ is reducible
 - $PLCP[i]$ is reducible
 $\Rightarrow PLCP[i] = PLCP[i - 1] - 1$

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	a	c	a	c	a	c	b	a	a	c	b	\$
<i>SA</i>	13	1	9	2	4	10	6	12	8	3	5	11	7
<i>LCP</i>	0	0	3	1	4	2	3	0	1	0	3	1	2
<i>PLCP</i>	0	1	0	4	3	3	2	1	3	2	1	0	0
Φ	12	8	7	1	2	9	10	11	0	3	4	5	-



Permuted LCP-Array [KMP09]

Definition: PLCP-Array

- $PLCP[SA[i]] = LCP[i]$
- $PLCP[i] = lcp(i, SA[i - 1]) = lcp(i, \Phi[i])$

- $PLCP[i] \geq PLCP[i - 1] - 1$
- $T[i - 1] = T[\Phi[i] - 1] \Rightarrow PLCP[i]$ is **reducible**
- $PLCP[i]$ is **reducible**
 $\Rightarrow PLCP[i] = PLCP[i - 1] - 1$

- only compute **irreducible** PLCP-values
- sum of all **irreducible** PLCP-values is $\leq n \lg n$

	1	2	3	4	5	6	7	8	9	10	11	12	13
<i>T</i>	a	a	c	a	c	a	c	b	a	a	c	b	\$
<i>SA</i>	13	1	9	2	4	10	6	12	8	3	5	11	7
<i>LCP</i>	0	0	3	1	4	2	3	0	1	0	3	1	2
<i>PLCP</i>	0	1	0	4	3	3	2	1	3	2	1	0	0
Φ	12	8	7	1	2	9	10	11	0	3	4	5	-



Recap: Pattern Matching with the Suffix Array

Function $\text{SeachSA}(T, SA[1..n], P[1..m]):$

```

1   $l = 1, r = n + 1$ 
2  while  $l < r$  do
3     $i = \lfloor (l + r) / 2 \rfloor$ 
4    if  $P > T[SA[i]..SA[i] + m)$  then
5       $l = i + 1$ 
6    else  $r = i$ 
7   $s = l, l = l - 1, r = n$ 
8  while  $l < r$  do
9     $i = \lceil (l + r) / 2 \rceil$ 
10   if  $P = T[SA[i]..SA[i] + m)$  then  $l = i$ 
11   else  $r = i - 1$ 
12  return  $[s, r]$ 
  
```

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

- two binary searches on the SA in $O(\lg n)$ time

Recap: Pattern Matching with the Suffix Array

Function $\text{SeachSA}(T, SA[1..n], P[1..m]):$

```

1   $l = 1, r = n + 1$ 
2  while  $l < r$  do
3     $i = \lfloor (l + r) / 2 \rfloor$ 
4    if  $P > T[SA[i]..SA[i] + m)$  then
5       $l = i + 1$ 
6    else  $r = i$ 
7   $s = l, l = l - 1, r = n$ 
8  while  $l < r$  do
9     $i = \lceil (l + r) / 2 \rceil$ 
10   if  $P = T[SA[i]..SA[i] + m)$  then  $l = i$ 
11   else  $r = i - 1$ 
12  return  $[s, r]$ 
  
```

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

two binary searches on the SA in $O(\lg n)$ time

Recap: Pattern Matching with the Suffix Array

Function $\text{SeachSA}(T, SA[1..n], P[1..m]):$

```

1   $l = 1, r = n + 1$ 
2  while  $l < r$  do
3     $i = \lfloor (l + r) / 2 \rfloor$ 
4    if  $P > T[SA[i]..SA[i] + m]$  then
5       $l = i + 1$ 
6    else  $r = i$ 
7   $s = l, l = l - 1, r = n$ 
8  while  $l < r$  do
9     $i = \lceil (l + r) / 2 \rceil$ 
10   if  $P = T[SA[i]..SA[i] + m]$  then  $l = i$ 
11   else  $r = i - 1$ 
12  return  $[s, r]$ 
  
```

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

- two binary searches on the SA in $O(\lg n)$ time
- each comparison requires $O(m)$ time

Recap: Pattern Matching with the Suffix Array

Function $\text{SeachSA}(T, SA[1..n], P[1..m]):$

```

1   $l = 1, r = n + 1$ 
2  while  $l < r$  do
3     $i = \lfloor (l + r) / 2 \rfloor$ 
4    if  $P > T[SA[i]..SA[i] + m]$  then
5       $l = i + 1$ 
6    else  $r = i$ 
7   $s = l, l = l - 1, r = n$ 
8  while  $l < r$  do
9     $i = \lceil (l + r) / 2 \rceil$ 
10   if  $P = T[SA[i]..SA[i] + m]$  then  $l = i$ 
11   else  $r = i - 1$ 
12  return  $[s, r]$ 
  
```

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

- two binary searches on the SA in $O(\lg n)$ time
- each comparison requires $O(m)$ time

Recap: Pattern Matching with the Suffix Array

Function $\text{SeachSA}(T, SA[1..n], P[1..m]):$

```

1   $l = 1, r = n + 1$ 
2  while  $l < r$  do
3     $i = \lfloor (l + r) / 2 \rfloor$ 
4    if  $P > T[SA[i]..SA[i] + m]$  then
5       $l = i + 1$ 
6    else  $r = i$ 
7   $s = l, l = l - 1, r = n$ 
8  while  $l < r$  do
9     $i = \lceil (l + r) / 2 \rceil$ 
10   if  $P = T[SA[i]..SA[i] + m]$  then  $l = i$ 
11   else  $r = i - 1$ 
12  return  $[s, r]$ 
  
```

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

- two binary searches on the SA in $O(\lg n)$ time
- each comparison requires $O(m)$ time
- counting in $O(1)$ additional time

Recap: Pattern Matching with the Suffix Array

Function $\text{SeachSA}(T, SA[1..n], P[1..m]):$

```

1   $l = 1, r = n + 1$ 
2  while  $l < r$  do
3     $i = \lfloor (l + r) / 2 \rfloor$ 
4    if  $P > T[SA[i]..SA[i] + m]$  then
5       $l = i + 1$ 
6    else  $r = i$ 
7   $s = l, l = l - 1, r = n$ 
8  while  $l < r$  do
9     $i = \lceil (l + r) / 2 \rceil$ 
10   if  $P = T[SA[i]..SA[i] + m]$  then  $l = i$ 
11   else  $r = i - 1$ 
12  return  $[s, r]$ 
  
```

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

- two binary searches on the SA in $O(\lg n)$ time
- each comparison requires $O(m)$ time
- counting in $O(1)$ additional time

Recap: Pattern Matching with the Suffix Array

Function $\text{SeachSA}(T, SA[1..n], P[1..m]):$

```

1   $l = 1, r = n + 1$ 
2  while  $l < r$  do
3     $i = \lfloor (l + r) / 2 \rfloor$ 
4    if  $P > T[SA[i]..SA[i] + m]$  then
5       $l = i + 1$ 
6    else  $r = i$ 
7   $s = l, l = l - 1, r = n$ 
8  while  $l < r$  do
9     $i = \lceil (l + r) / 2 \rceil$ 
10   if  $P = T[SA[i]..SA[i] + m]$  then  $l = i$ 
11   else  $r = i - 1$ 
12  return  $[s, r]$ 
  
```

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

- two binary searches on the SA in $O(\lg n)$ time
- each comparison requires $O(m)$ time
- counting in $O(1)$ additional time
- reporting in $O(occ)$ additional time

Recap: Pattern Matching with the Suffix Array

Function $\text{SeachSA}(T, SA[1..n], P[1..m]):$

```

1   $l = 1, r = n + 1$ 
2  while  $l < r$  do
3     $i = \lfloor (l + r) / 2 \rfloor$ 
4    if  $P > T[SA[i]..SA[i] + m]$  then
5       $l = i + 1$ 
6    else  $r = i$ 
7   $s = l, l = l - 1, r = n$ 
8  while  $l < r$  do
9     $i = \lceil (l + r) / 2 \rceil$ 
10   if  $P = T[SA[i]..SA[i] + m]$  then  $l = i$ 
11   else  $r = i - 1$ 
12  return  $[s, r]$ 
  
```

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

- two binary searches on the SA in $O(\lg n)$ time
- each comparison requires $O(m)$ time
- counting in $O(1)$ additional time
- reporting in $O(occ)$ additional time

Recap: Pattern Matching with the Suffix Array

Function $\text{SeachSA}(T, SA[1..n], P[1..m]):$

```

1   $l = 1, r = n + 1$ 
2  while  $l < r$  do
3     $i = \lfloor (l + r) / 2 \rfloor$ 
4    if  $P > T[SA[i]..SA[i] + m]$  then
5       $l = i + 1$ 
6    else  $r = i$ 
7   $s = l, l = l - 1, r = n$ 
8  while  $l < r$  do
9     $i = \lceil (l + r) / 2 \rceil$ 
10   if  $P = T[SA[i]..SA[i] + m]$  then  $l = i$ 
11   else  $r = i - 1$ 
12  return  $[s, r]$ 
  
```

Lemma: Running Time SeachSA

The SeachSA answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + occ)$ time

Proof (Sketch)

- two binary searches on the SA in $O(\lg n)$ time
 - each comparison requires $O(m)$ time
 - counting in $O(1)$ additional time
 - reporting in $O(occ)$ additional time
- comparison of pattern is expensive

Speeding Up Pattern Matching with the LCP-Array (1/3)

- remember how many characters of the pattern and suffix match
- identify how long the prefix of the old and next suffix is
- do so using the LCP-array and
- **range minimum queries** ⓘ detailed introduction in **Advanced Data Structures**

- $lcp(i, j) = \max\{k: T[i..i+k]$
- $lcp(i, j) = T[j..j+k]\} = LCP[RMQ_{LCP}(i+1, j)]$
- RMQs can be answered in $O(1)$ time and
- require $O(n)$ space

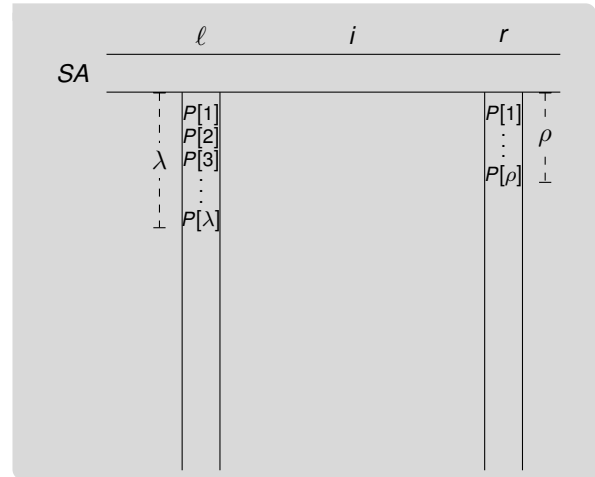
Definition: Range Minimum Queries

Given an array $A[1..m)$, a **range minimum query** for a range $\ell \leq r \in [1, n)$ returns

$$RMQ_A(\ell, r) = \arg \min\{A[k]: k \in [\ell, r]\}$$

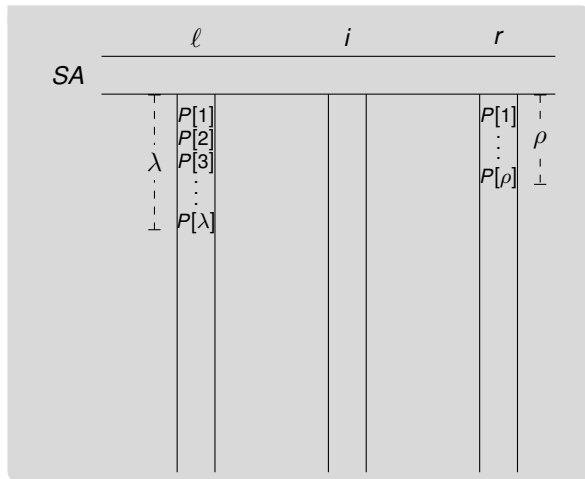
Speeding Up Pattern Matching with the LCP-Array (2/3)

- during binary search matched
 - λ characters with left border ℓ and
 - ρ characters with right border r
 - w.l.o.g. let $\lambda \geq \rho$
-
- middle position i
 - decide if continue in $[\ell, i]$ or $[i, r]$
-
- let $\xi = \text{lcp}(SA[\ell], SA[i])$ $\ominus O(1)$ time with RMQs



Speeding Up Pattern Matching with the LCP-Array (3/3)

- let $\xi = \text{lcp}(SA[\ell], SA[i])$

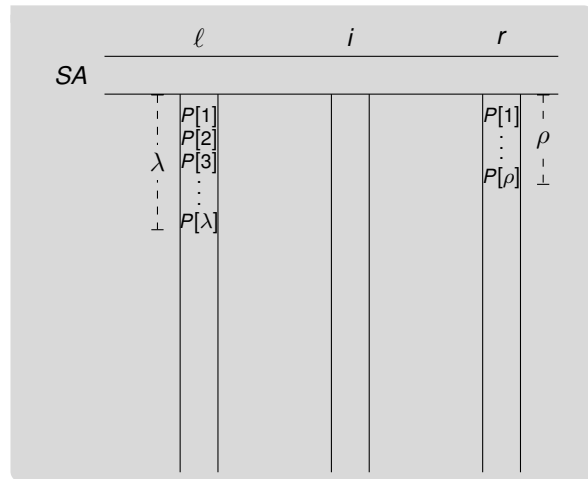


Speeding Up Pattern Matching with the LCP-Array (3/3)

- let $\xi = lcp(SA[\ell], SA[i])$

$\xi > \lambda$

- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison

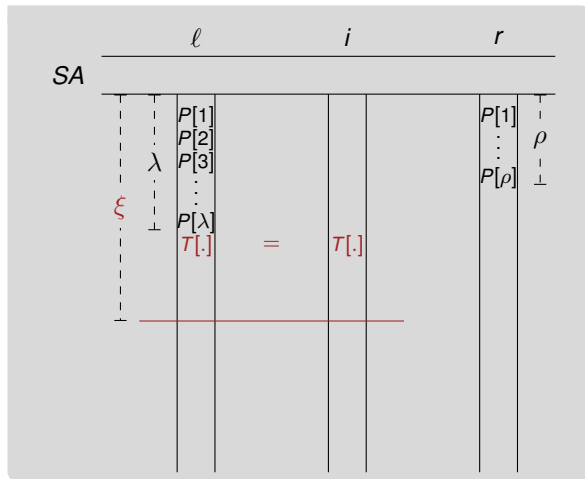


Speeding Up Pattern Matching with the LCP-Array (3/3)

- let $\xi = \text{lcp}(SA[\ell], SA[i])$

$\xi > \lambda$

- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison

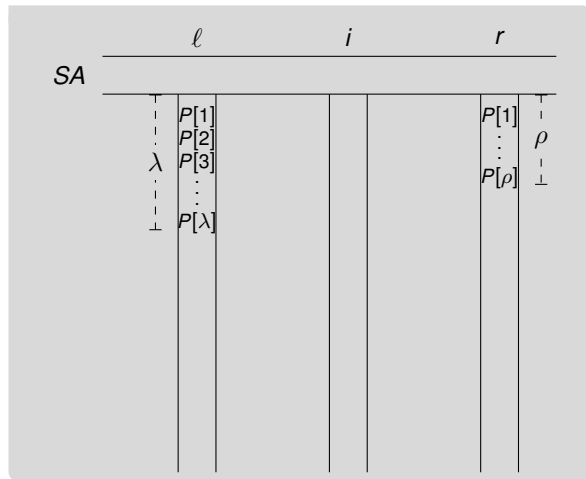


Speeding Up Pattern Matching with the LCP-Array (3/3)

- let $\xi = lcp(SA[\ell], SA[i])$

$\xi > \lambda$

- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison



Speeding Up Pattern Matching with the LCP-Array (3/3)

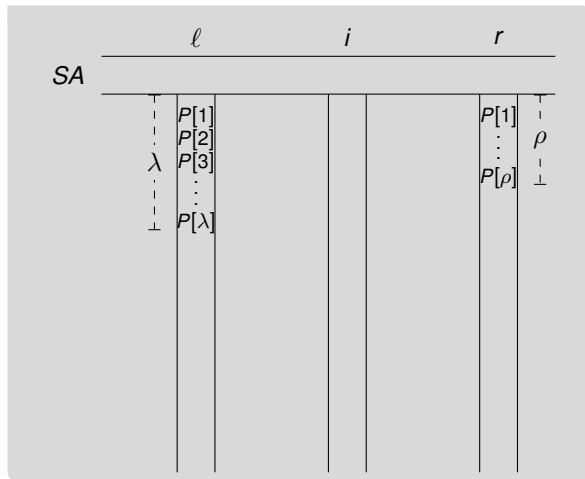
- let $\xi = \text{lcp}(SA[\ell], SA[i])$

$\xi > \lambda$

- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison

$\xi = \lambda$

- compare as before



Speeding Up Pattern Matching with the LCP-Array (3/3)

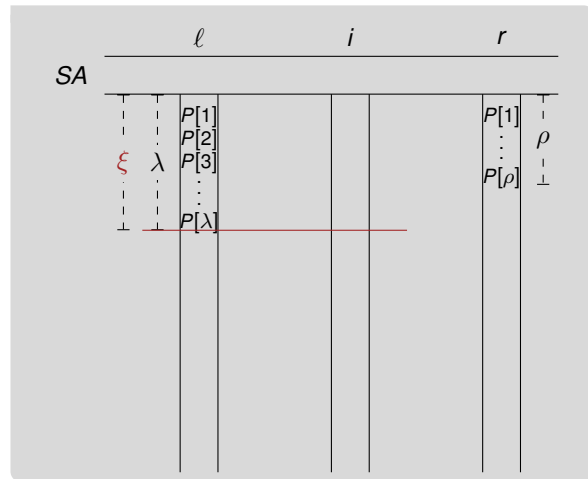
- let $\xi = \text{lcp}(SA[\ell], SA[i])$

$\xi > \lambda$

- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison

$\xi = \lambda$

- compare as before



Speeding Up Pattern Matching with the LCP-Array (3/3)

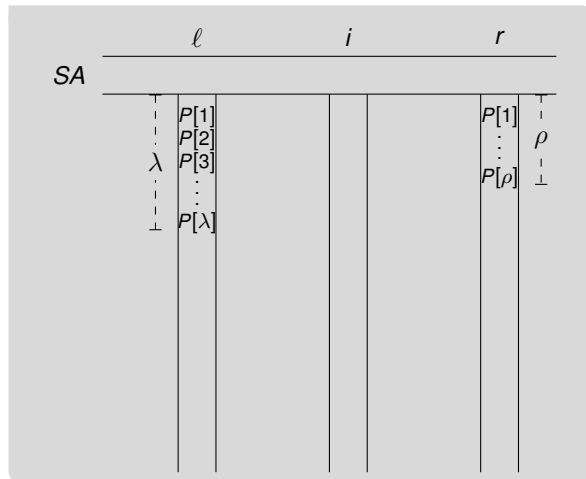
- let $\xi = \text{lcp}(SA[\ell], SA[i])$

$\xi > \lambda$

- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison

$\xi = \lambda$

- compare as before



Speeding Up Pattern Matching with the LCP-Array (3/3)

- let $\xi = \text{lcp}(SA[\ell], SA[i])$

$\xi > \lambda$

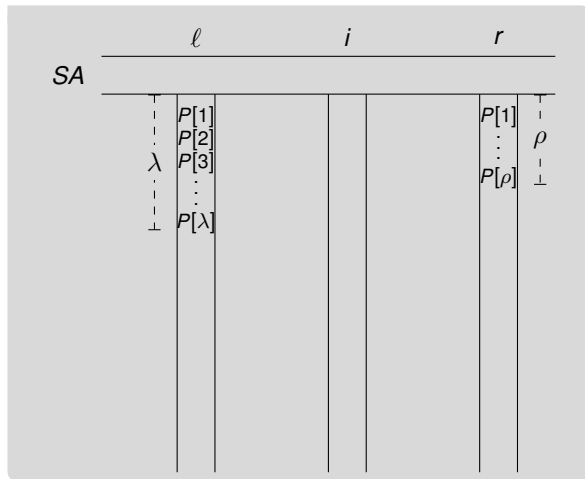
- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison

$\xi = \lambda$

- compare as before

$\xi < \lambda$

- $\xi \geq \rho$ and $P[\xi + 1] < T[SA[i] + \xi]$
- $r = i$ and $\rho = \xi$ without character comparison



Speeding Up Pattern Matching with the LCP-Array (3/3)

- let $\xi = \text{lcp}(SA[\ell], SA[i])$

$\xi > \lambda$

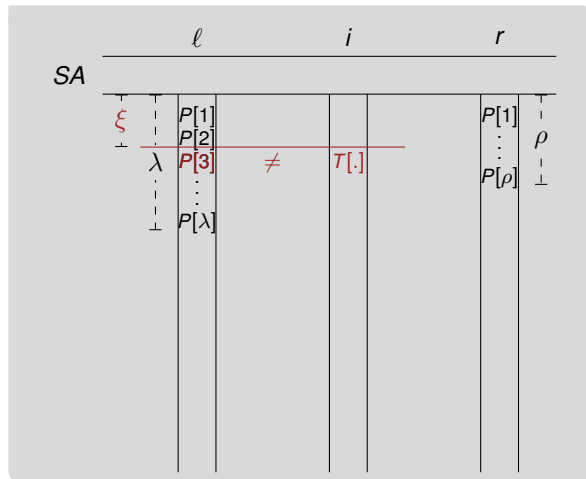
- $P[\lambda + 1] > T[SA[\ell] + \lambda] = T[SA[i] + \lambda]$
- $\ell = i$ without character comparison

$\xi = \lambda$

- compare as before

$\xi < \lambda$

- $\xi \geq \rho$ and $P[\xi + 1] < T[SA[i] + \xi]$
- $r = i$ and $\rho = \xi$ without character comparison



Speeding Up Pattern Matching with the LCP-Array (4/X)

Lemma:

Using RMQs, SeachSA answers counting queries in $O(m + \lg n)$ time and reporting queries in $O(m + \lg n + occ)$ time

Speeding Up Pattern Matching with the LCP-Array (4/X)

Lemma:

Using RMQs, SearchSA answers counting queries in $O(m + \lg n)$ time and reporting queries in $O(m + \lg n + occ)$ time

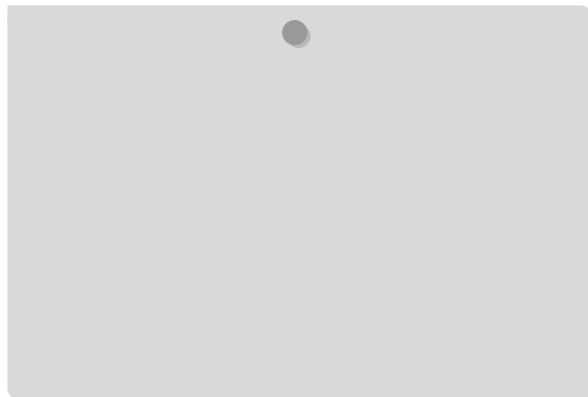
Proof (Sketch)

- either halve the range in the suffix array ($\xi \neq \lambda$)
or
- compare characters of the pattern (at most m)

Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time
- use *SA* and *LCP*
- only look at rightmost path in tree
- find deepest node with string-depth $\leq LCP[i]$
- total $O(n)$ time

	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3



Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time

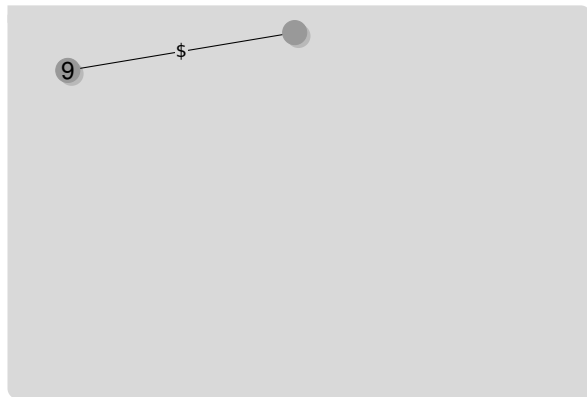
- use *SA* and *LCP*

- only look at rightmost path in tree

- find deepest node with string-depth $\leq LCP[i]$

- total $O(n)$ time

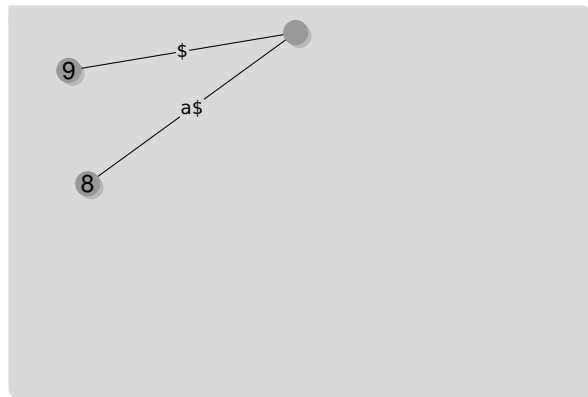
	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3



Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time
- use *SA* and *LCP*
- only look at rightmost path in tree
- find deepest node with string-depth $\leq LCP[i]$
- total $O(n)$ time

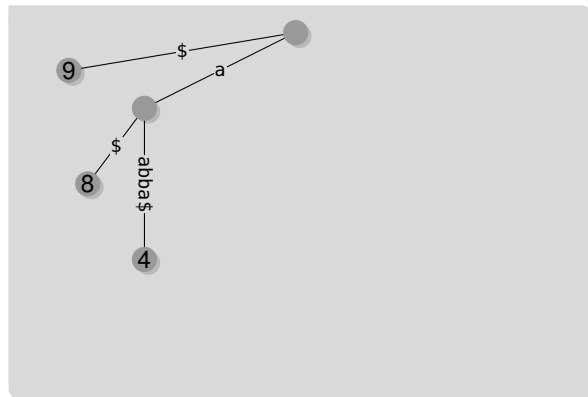
	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3



Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time
- use *SA* and *LCP*
- only look at rightmost path in tree
- find deepest node with string-depth $\leq LCP[i]$
- total $O(n)$ time

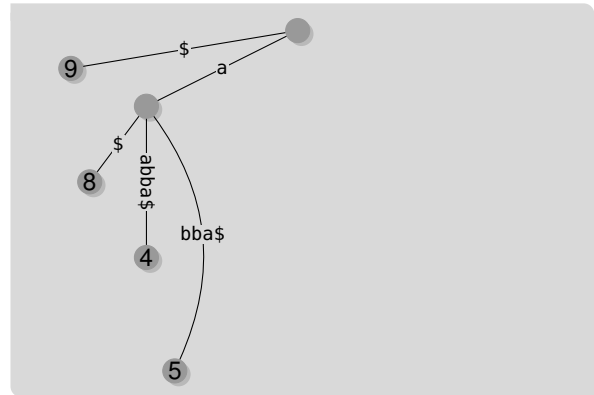
	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3



Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time
- use *SA* and *LCP*
- only look at rightmost path in tree
- find deepest node with string-depth $\leq LCP[i]$
- total $O(n)$ time

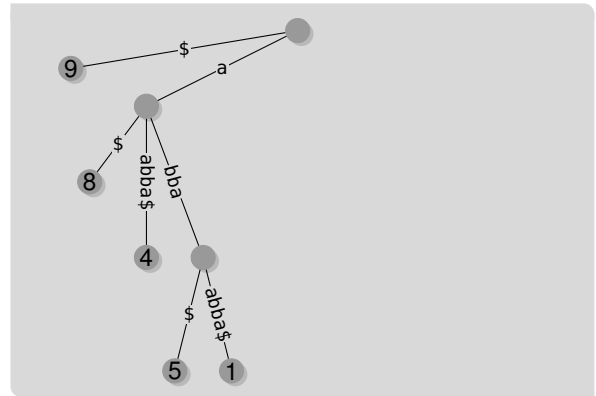
	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3



Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time
- use *SA* and *LCP*
- only look at rightmost path in tree
- find deepest node with string-depth $\leq LCP[i]$
- total $O(n)$ time

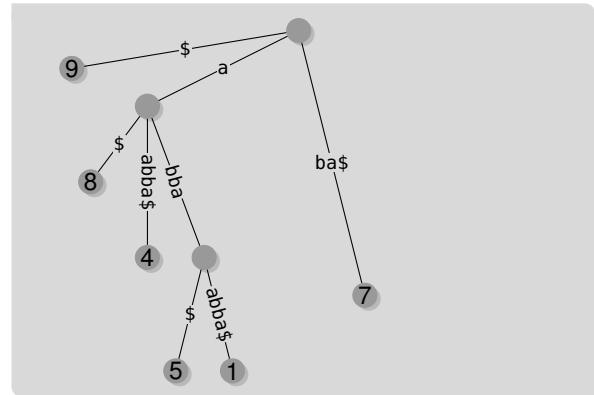
	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3



Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time
- use *SA* and *LCP*
- only look at rightmost path in tree
- find deepest node with string-depth $\leq LCP[i]$
- total $O(n)$ time

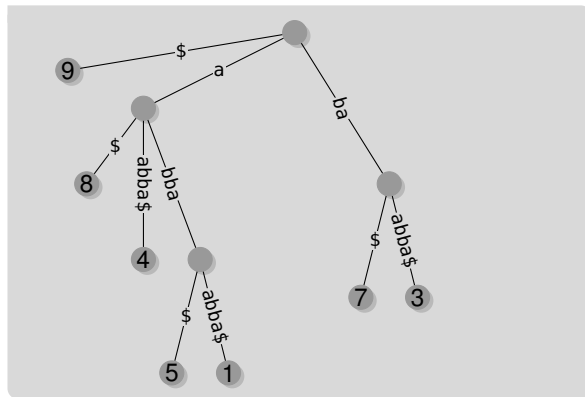
	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3



Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time
- use *SA* and *LCP*
- only look at rightmost path in tree
- find deepest node with string-depth $\leq LCP[i]$
- total $O(n)$ time

	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3



Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time

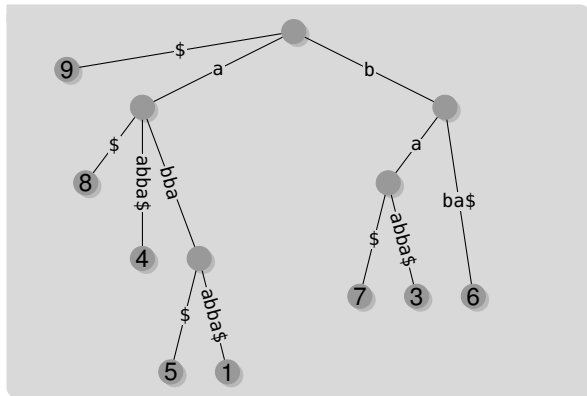
- use *SA* and *LCP*

- only look at rightmost path in tree

- find deepest node with string-depth $\leq LCP[i]$

- total $O(n)$ time

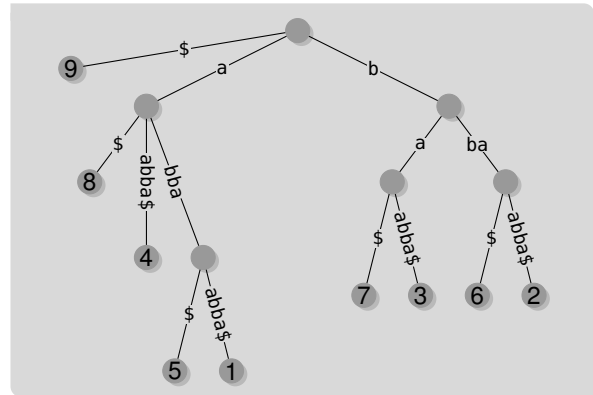
	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3



Back to the Roots: Suffix Tree Construction

- naive in $O(n^2)$ time
- use *SA* and *LCP*
- only look at rightmost path in tree
- find deepest node with string-depth $\leq LCP[i]$
- total $O(n)$ time

	1	2	3	4	5	6	7	8	9
<i>T</i>	a	b	b	a	a	b	b	a	\$
<i>SA</i>	9	8	4	5	1	7	3	6	2
<i>LCP</i>	0	0	1	1	4	0	2	1	3

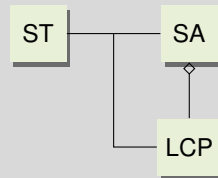


Conclusion and Outlook

This Lecture

- linear time LCP-array construction
- suffix tree construction based on *SA* and *LCP*

Linear Time Construction

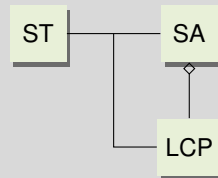


Conclusion and Outlook

This Lecture

- linear time LCP-array construction
- suffix tree construction based on *SA* and *LCP*
- engineered LCP-Array construction algorithms
- cache misses are costly
- interesting properties of the PLCP-array

Linear Time Construction

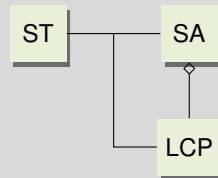


Conclusion and Outlook

This Lecture

- linear time LCP-array construction
- suffix tree construction based on *SA* and *LCP*
- engineered LCP-Array construction algorithms
- cache misses are costly
- interesting properties of the PLCP-array

Linear Time Construction



Next Lecture

- text compression using *SA* and *LCP*

One More Thing: The Project

- programming project including
- experimental evaluation and
- short presentation (5 minutes)

The Task

Implement one (or more) of the text-indices presented in this lecture to answer the following queries:

- which pattern of length ℓ is the k -th frequent one and
 - which is the longest pattern of the form $P = \alpha\alpha$
- exact rules can be found on the website

One More Thing: The Project

- programming project including
- experimental evaluation and
- short presentation (5 minutes)

- small programming contest
- fastest text-index wins
- 25 % construction time
- 75 % query time

The Task

Implement one (or more) of the text-indices presented in this lecture to answer the following queries:

- which pattern of length ℓ is the k -th frequent one and
 - which is the longest pattern of the form $P = \alpha\alpha$
- exact rules can be found on the website

One More Thing: The Project

- programming project including
- experimental evaluation and
- short presentation (5 minutes)

- small programming contest
- fastest text-index wins
- 25 % construction time
- 75 % query time

The Task

Implement one (or more) of the text-indices presented in this lecture to answer the following queries:

- which pattern of length ℓ is the k -th frequent one and
- which is the longest pattern of the form $P = \alpha\alpha$

- exact rules can be found on the website

Grading

- documentation
- evaluation
- presentation
- implementation

Bibliography I

- [GBS92] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. “New Indices for Text: Pat Trees and Pat Arrays”. In: *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, 1992, pages 66–82.
- [Kas+01] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. “Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications”. In: *CPM*. Volume 2089. Lecture Notes in Computer Science. Springer, 2001, pages 181–192. DOI: [10.1007/3-540-48194-X_17](https://doi.org/10.1007/3-540-48194-X_17).
- [KMP09] Juha Kärkkäinen, Giovanni Manzini, and Simon J. Puglisi. “Permuted Longest-Common-Prefix Array”. In: *CPM*. Volume 5577. Lecture Notes in Computer Science. Springer, 2009, pages 181–192. DOI: [10.1007/978-3-642-02441-2_17](https://doi.org/10.1007/978-3-642-02441-2_17).
- [MM93] Udi Manber and Eugene W. Myers. “Suffix Arrays: A New Method for On-Line String Searches”. In: *SIAM J. Comput.* 22.5 (1993), pages 935–948. DOI: [10.1137/0222058](https://doi.org/10.1137/0222058).