Karlsruhe Institute of Technology

# Text Indexing

## Lecture 02: Suffix Trees and Suffix Arrays

Florian Kurpicz



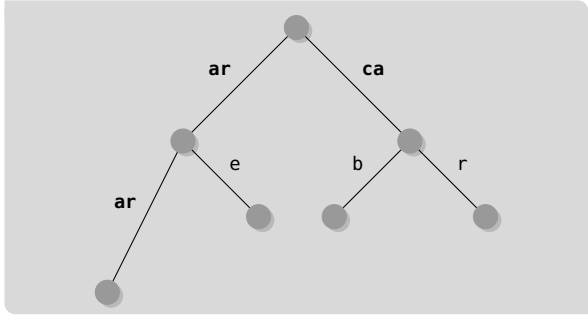
# PINGO

https://pingo.scc.kit.edu/289240

# Recap: Compact Trie

## Definition: Compact Trie

- A compact trie is a trie where all branchless paths are replaced by a single edge.
- The label of the new edge is the concatenation of the replaced edges' labels.




## Next

A full-text index for a text $T$ is

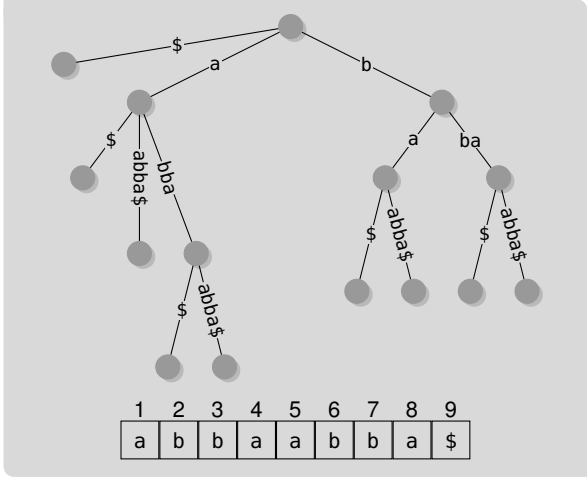
- a data structure that
- allows to answer queries on $T$ faster than naive
- we are interested in *pattern matching* queries
- how to use tries to create full-text index

# Suffix Tree (1/4)

## Definition: Suffix Tree [Wei73]

A suffix tree ($ST$) for a text $T$ of length $n$ is a

- compact trie
- over $S = \{T[1..n], T[2..n], \ldots, T[n..n]\}$
  - suffixes are prefix-free due to sentinel



# Suffix Tree (1/4)

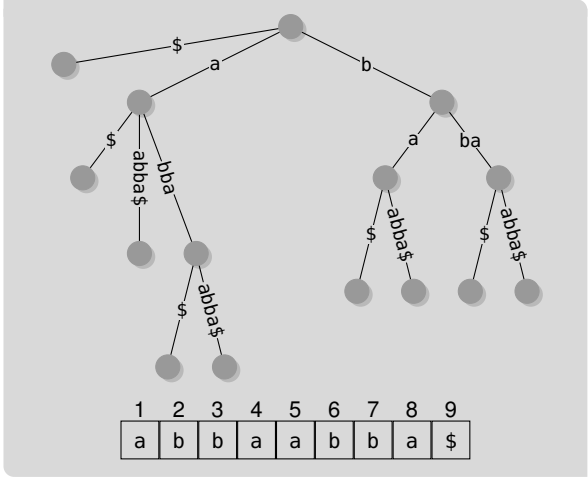
## Definition: Suffix Tree [Wei73]

A suffix tree ($ST$) for a text $T$ of length $n$ is a

- compact trie
- over $S = \{T[1..n], T[2..n], \ldots, T[n..n]\}$
  - suffixes are prefix-free due to sentinel

Let $G = (V, E)$ be a compact trie with root $r$ and a node $v \in V$, then

- $\lambda(v)$ is the concatenation of labels from $r$ to $v$
- $d(v) = |\lambda(v)|$ is the string-depth of $v$
  - string depth $\neq$ depth



# Suffix Tree (1/4)

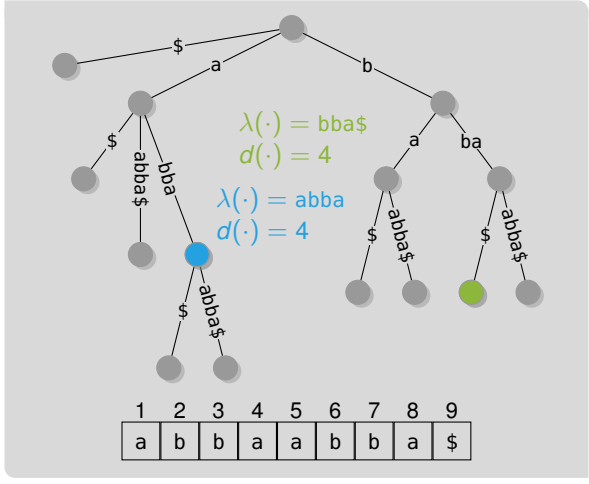
## Definition: Suffix Tree [Wei73]

A suffix tree ($ST$) for a text $T$ of length $n$ is a

- compact trie
- over $S = \{T[1..n], T[2..n], \ldots, T[n..n]\}$
  - suffixes are prefix-free due to sentinel

Let $G = (V, E)$ be a compact trie with root $r$ and a node $v \in V$, then

- $\lambda(v)$ is the concatenation of labels from $r$ to $v$
- $d(v) = |\lambda(v)|$ is the string-depth of $v$
  - string depth $\neq$ depth



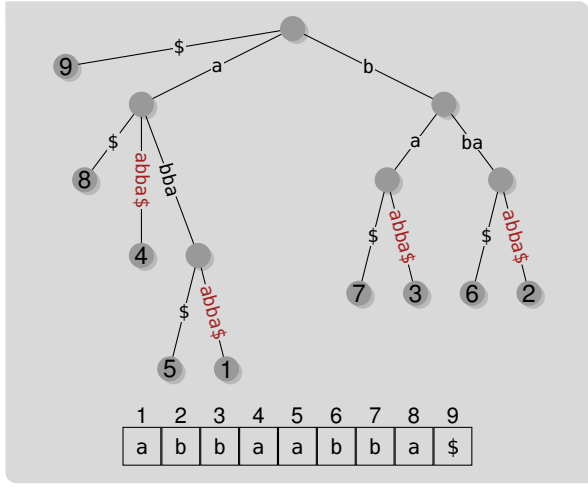
# Suffix Tree (2/4)

## Representing Labels

- explicit edge labels require $O(n^2)$ words space

## Suffix Information

- label leaves with corresponding suffix
  - will be important later on



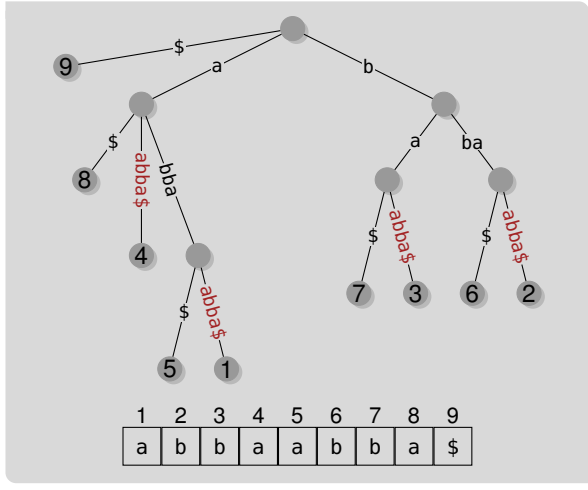
# Suffix Tree (2/4)

## Representing Labels

- explicit edge labels require $O(n^2)$ words space
- references require only $O(n)$ words space

## Suffix Information

- label leaves with corresponding suffix
  - will be important later on



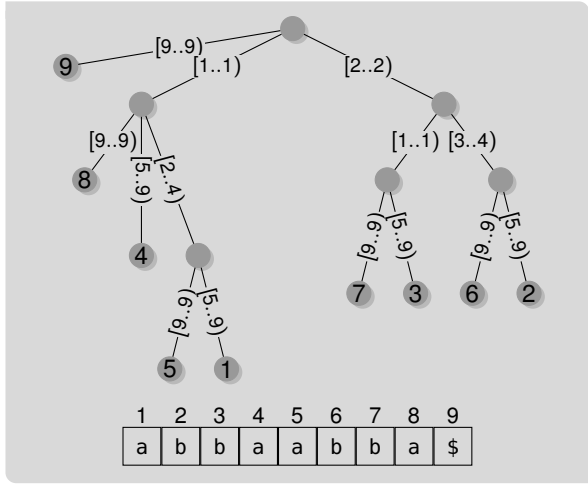
# Suffix Tree (2/4)

## Representing Labels

- explicit edge labels require $O(n^2)$ words space
- references require only $O(n)$ words space

## Suffix Information

- label leaves with corresponding suffix
  - will be important later on



# Suffix Tree (2/4)

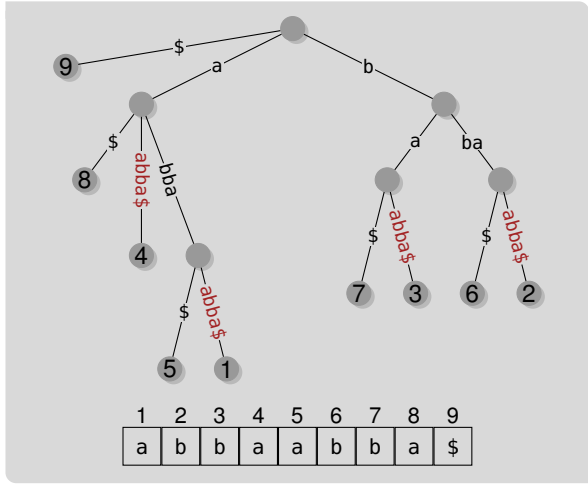
## Representing Labels

- explicit edge labels require $O(n^2)$ words space
- references require only $O(n)$ words space

- for simplicity, we show text

## Suffix Information

- label leaves with corresponding suffix
  - will be important later on

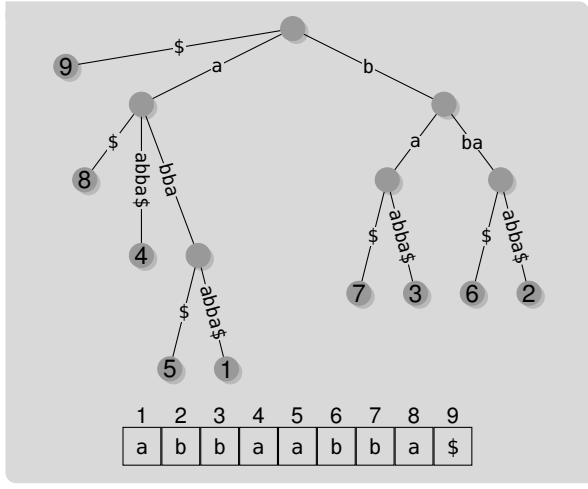


# Suffix Tree (3/4)

## Pattern Matching using Suffix Trees

- Pattern $P[1..m]$
- start at the root and follow edges
- query time depends on representation of children

- $O(m)$ time using $O(n\sigma)$ words space

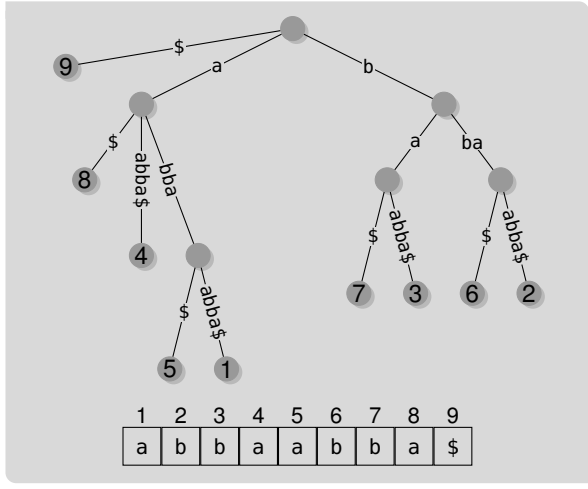


# Suffix Tree (3/4)

## Pattern Matching using Suffix Trees

- Pattern $P[1..m]$
- start at the root and follow edges
- query time depends on representation of children

- $O(m)$ time using $O(n\sigma)$ words space
- $O(m \cdot \lg \sigma)$ time with $O(n)$ words space

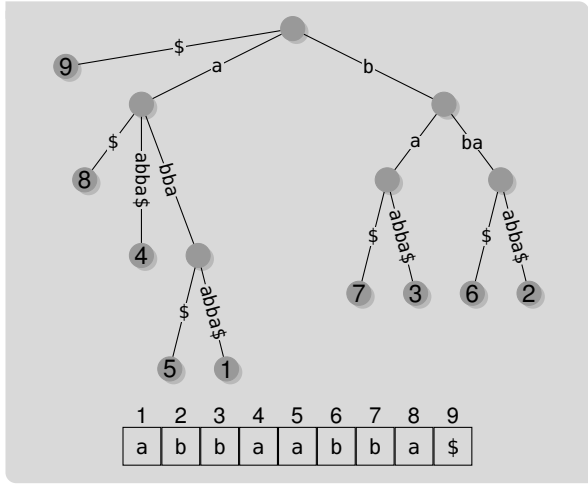


# Suffix Tree (3/4)

## Pattern Matching using Suffix Trees

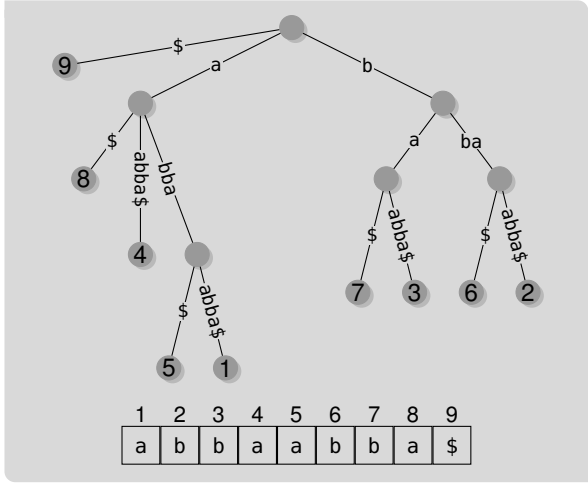
- Pattern $P[1..m]$
- start at the root and follow edges
- query time depends on representation of children

- $O(m)$ time using $O(n\sigma)$ words space
- $O(m \cdot \lg \sigma)$ time with $O(n)$ words space
- $O(m + \lg \sigma)$ time with $O(n)$ words space



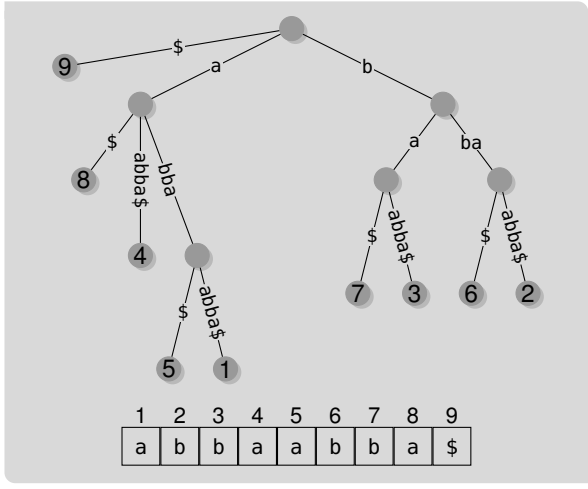
# Suffix Tree (4/4)

- very (most?) powerful text-index



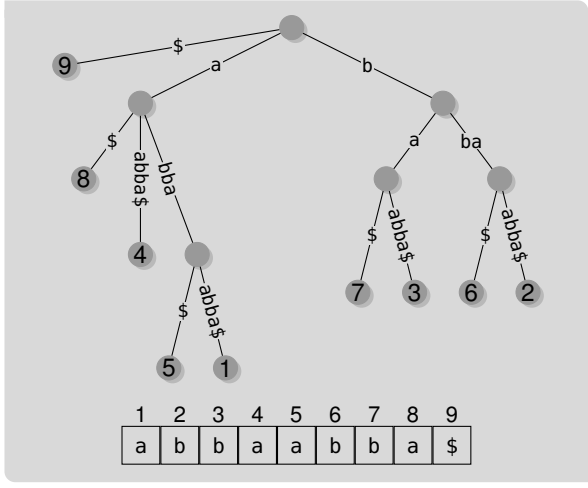
# Suffix Tree (4/4)

- very (most?) powerful text-index
- suffix trees require $\approx$ 8–20 bytes per character



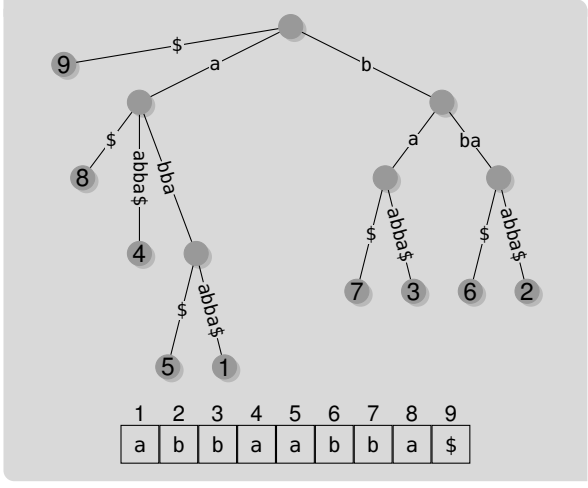
# Suffix Tree (4/4)

- very (most?) powerful text-index
- suffix trees require $\approx$ 8–20 bytes per character
- efficient direct construction in $O(n)$ time [Ukk95]
- also possible for integer alphabets [Far97]



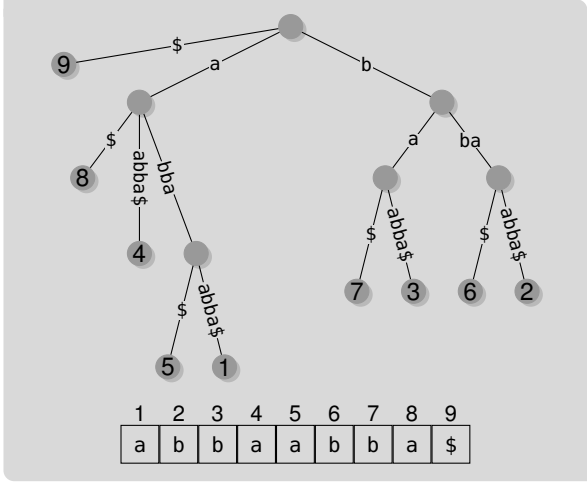
# Suffix Tree (4/4)

- very (most?) powerful text-index
- suffix trees require $\approx$ 8–20 bytes per character
- efficient direct construction in $O(n)$ time [Ukk95]
- also possible for integer alphabets [Far97]
- *SA* and *LCP*-array can replace suffix tree
- can answer all queries in the same time



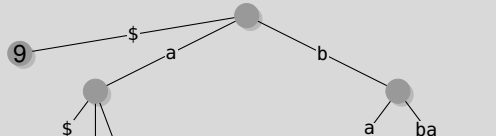
# Suffix Tree (4/4)

- very (most?) powerful text-index
- suffix trees require $\approx$ 8–20 bytes per character
- efficient direct construction in $O(n)$ time [Ukk95]
- also possible for integer alphabets [Far97]
- *SA* and *LCP*-array can replace suffix tree
- can answer all queries in the same time
- Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. “Replacing Suffix Trees with Enhanced Suffix Arrays”. In: *J. Discrete Algorithms* 2.1 (2004), pages 53–86. DOI: 10.1016/S1570-8667(03)00065-0



# Suffix Tree (4/4)

- very (most?) powerful text-index
- suffix trees require $\approx$ 8–20 bytes per character
- efficient direct construction in $O(n)$ time [Ukk95]
- also possible for integer alphabets [Far97]
- *SA* and *LCP*-array can replace suffix tree
- can answer all queries in the same time
- Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. “Replacing Suffix Trees with Enhanced Suffix Arrays”. In: *J. Discrete Algorithms* 2.1 (2004), pages 53–86. DOI: 10.1016/S1570-8667(03)00065-0



next, suffix array construction

# 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$
abaabcabcabba$
abba$
abcabba$
abcabcabba$
ba$
babcabcabba$
bba$
bcabba$
bcabbcabba$
cabba$
cabcabba$

# 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

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

| | 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$ | ababcabba$ | abba$ | abcabba$ | abcabcabba$ | ba$ | babcabcabba$ | bba$ | bcabba$ | bcabcabba$ | cabba$ | cabcabba$ |

# 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

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

| | 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 |

# Pattern Matching with the Suffix Array (1/2)

**Function** SeachSA($T, SA[1..n], P[1..m]$)**:**

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

- pattern $P =$ abc

| | 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 |
| | $ | a$ | ababcabcabba$ | abba$ | abcabba$ | abcabcabba$ | ba$ | babcabcabba$ | bba$ | bcabba$ | bcabcabba$ | cabba$ | cabcabba$ |

# Pattern Matching with the Suffix Array (1/2)

**Function** SeachSA($T, SA[1..n], P[1..m]$)**:**

```
1  | ℓ = 1, r = n + 1
2  | while ℓ < r do  ℹ Find left border
3  |   | i = ⌊(ℓ + r)/2⌋
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 = ⌈ℓ + r/2⌉
10 |   | if P = T[SA[i]..SA[i] + m) then ℓ = i
11 |   | else r = i − 1
12 | return [s, r]
```

- pattern $P =$ abc

| | 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 |
| | $ | a$ | ababcabcabba$ | abba$ | abcabba$ | abcabcabba$ | ba$ | babcabcabba$ | bba$ | bcabba$ | bcabcabba$ | cabba$ | cabcabba$ |

# Pattern Matching with the Suffix Array (1/2)

**Function** SeachSA($T, SA[1..n], P[1..m]$)**:**

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

- pattern $P =$ abc

| | 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 |
| | $ | a$ | ababcabcabba$ | abba$ | abcabba$ | abcabcabba$ | ba$ | babcabcabba$ | bba$ | bcabba$ | bcabcabba$ | cabba$ | cabcabba$ |

# Pattern Matching with the Suffix Array (2/2)

```
Function SeachSA(T, SA[1..n], P[1..m]):
1   ℓ = 1, r = n + 1
2   while ℓ < r do
3       i = ⌊(ℓ + r)/2⌋
4       if P > T[SA[i]..SA[i] + m) then
            ℓ = i + 1
5       else r = i
6   s = ℓ, ℓ = ℓ − 1, r = n
7   while ℓ < r do
8       i = ⌈ℓ + r/2⌉
9       if P = T[SA[i]..SA[i] + m) then ℓ = i
10      else r = i − 1
11  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 $O(lgn)$ time

# Pattern Matching with the Suffix Array (2/2)

```
Function SeachSA(T, SA[1..n], P[1..m]):
1   ℓ = 1, r = n + 1
2   while ℓ < r do
3       i = ⌊(ℓ + r)/2⌋
4       if P > T[SA[i]..SA[i] + m) then
            ℓ = i + 1
5       else r = i
6   s = ℓ, ℓ = ℓ − 1, r = n
7   while ℓ < r do
8       i = ⌈ℓ + r/2⌉
9       if P = T[SA[i]..SA[i] + m) then ℓ = i
10      else r = i − 1
11  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 $O(lgn)$ time

# Pattern Matching with the Suffix Array (2/2)

```
Function SeachSA(T, SA[1..n], P[1..m]):
1   ℓ = 1, r = n + 1
2   while ℓ < r do
3       i = ⌊(ℓ + r)/2⌋
4       if P > T[SA[i]..SA[i] + m) then
            ℓ = i + 1
5       else r = i
6   s = ℓ, ℓ = ℓ − 1, r = n
7   while ℓ < r do
8       i = ⌈ℓ + r/2⌉
9       if P = T[SA[i]..SA[i] + m) then ℓ = i
10      else r = i − 1
11  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 $O(lgn)$ time
- each comparison requires $O(m)$ time

# Pattern Matching with the Suffix Array (2/2)

**Function** SeachSA($T$, $SA[1..n]$, $P[1..m]$)**:**

```
1  | ℓ = 1, r = n + 1
2  | while ℓ < r do
3  |   | i = ⌊(ℓ + r)/2⌋
4  |   | if P > T[SA[i]..SA[i] + m) then
   |   |   ℓ = i + 1
5  |   | else r = i
6  | s = ℓ, ℓ = ℓ − 1, r = n
7  | while ℓ < r do
8  |   | i = ⌈ℓ + r/2⌉
9  |   | if P = T[SA[i]..SA[i] + m) then ℓ = i
10 |   | else r = i − 1
11 | 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 $O(lgn)$ time
- each comparison requires $O(m)$ time

# Pattern Matching with the Suffix Array (2/2)

```
Function SeachSA(T, SA[1..n], P[1..m]):
1   ℓ = 1, r = n + 1
2   while ℓ < r do
3       i = ⌊(ℓ + r)/2⌋
4       if P > T[SA[i]..SA[i] + m) then
            ℓ = i + 1
5       else r = i
6   s = ℓ, ℓ = ℓ − 1, r = n
7   while ℓ < r do
8       i = ⌈ℓ + r/2⌉
9       if P = T[SA[i]..SA[i] + m) then ℓ = i
10      else r = i − 1
11  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 $O(lgn)$ time
- each comparison requires $O(m)$ time
  - counting in $O(1)$ additional time

# Pattern Matching with the Suffix Array (2/2)

**Function** SeachSA($T$, $SA[1..n]$, $P[1..m]$)**:**

```
1  | ℓ = 1, r = n + 1
2  | while ℓ < r do
3  |   | i = ⌊(ℓ + r)/2⌋
4  |   | if P > T[SA[i]..SA[i] + m) then
   |   |   ℓ = i + 1
5  |   | else r = i
6  | s = ℓ, ℓ = ℓ − 1, r = n
7  | while ℓ < r do
8  |   | i = ⌈ℓ + r/2⌉
9  |   | if P = T[SA[i]..SA[i] + m) then ℓ = i
10 |   | else r = i − 1
11 | 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

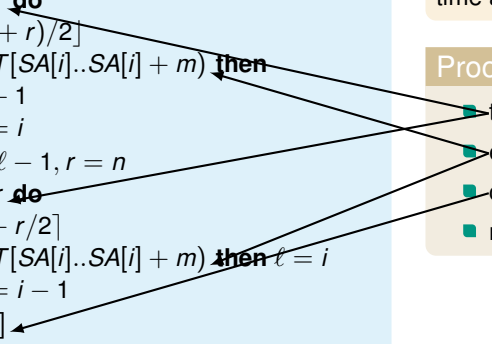
## Proof (Sketch)

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

# Pattern Matching with the Suffix Array (2/2)

**Function** SeachSA($T$, $SA[1..n]$, $P[1..m]$)**:**

```
1  | ℓ = 1, r = n + 1
2  | while ℓ < r do
3  |   | i = ⌊(ℓ + r)/2⌋
4  |   | if P > T[SA[i]..SA[i] + m) then
   |   |   ℓ = i + 1
5  |   | else r = i
6  | s = ℓ, ℓ = ℓ − 1, r = n
7  | while ℓ < r do
8  |   | i = ⌈ℓ + r/2⌉
9  |   | if P = T[SA[i]..SA[i] + m) then ℓ = i
10 |   | else r = i − 1
11 | 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 $O(lgn)$ time
- each comparison requires $O(m)$ time
  - counting in $O(1)$ additional time
  - reporting in $O(occ)$ additional time

# Pattern Matching with the Suffix Array (2/2)

**Function** SeachSA($T$, $SA[1..n]$, $P[1..m]$)**:**

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

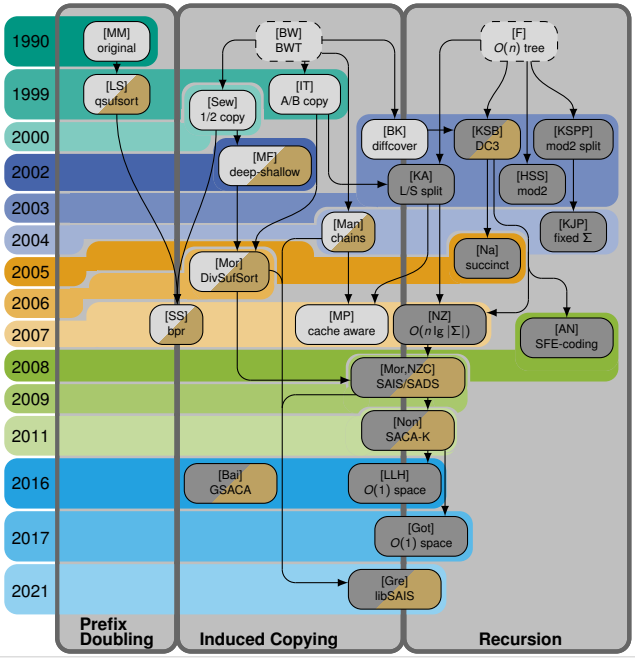
## Proof (Sketch)

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

# Preview: Improving Running Time with LCP-Array

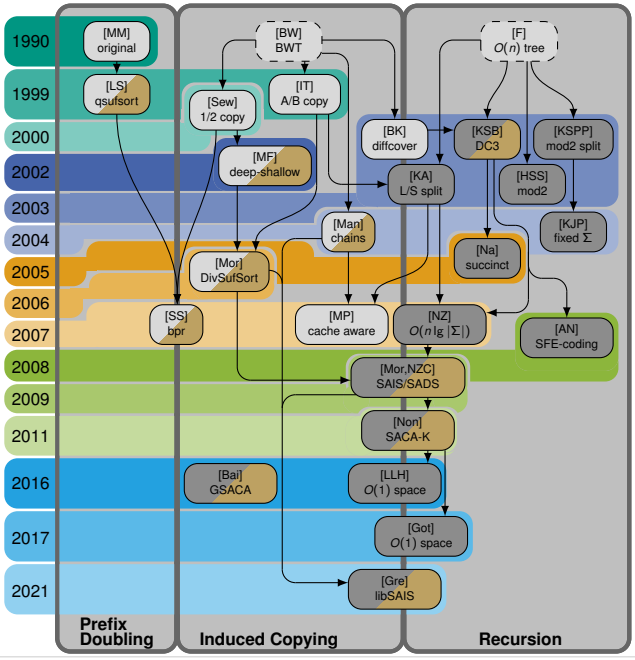
- next lecture: $O(m + \lg n)$ and $O(m + \lg n + occ)$ time
- requires additional indices on LCP-array

- now: how to compute the suffix array directly ℹ without the suffix tree




## Timeline Sequential Suffix Sorting

- based on [Bah+19; Bin18; Kur20; PST07]
- **darker grey**: linear running time
- **brown**: available implementation

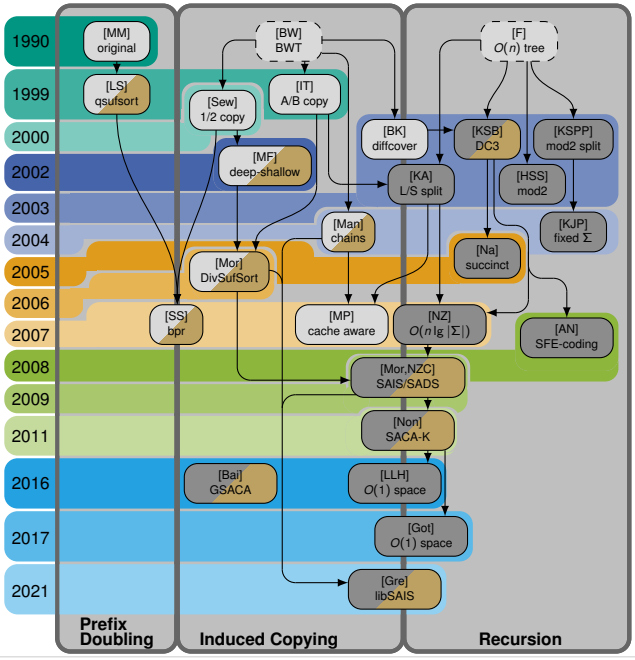



## Timeline Sequential Suffix Sorting

- based on [Bah+19; Bin18; Kur20; PST07]
- **darker grey**: linear running time
- **brown**: available implementation

## Special Mentions

- DC3 first $O(n)$ algorithm
- $O(n)$ running time and $O(1)$ space for integer alphabets possible

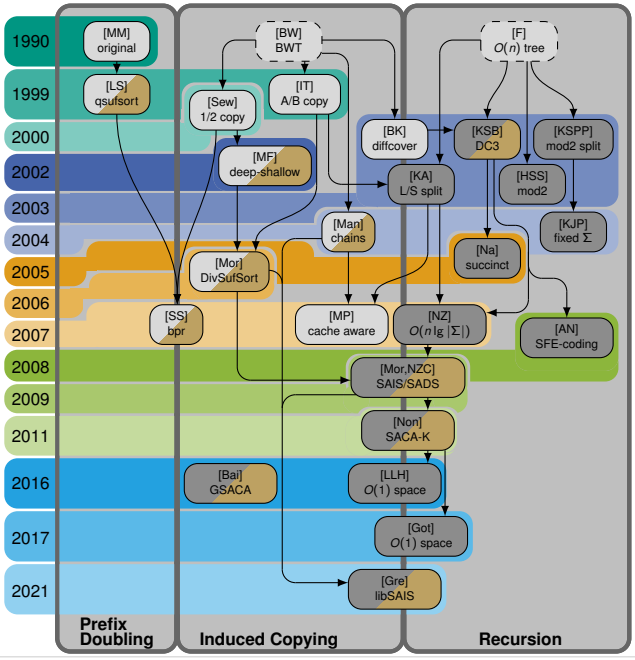



## Timeline Sequential Suffix Sorting

- based on [Bah+19; Bin18; Kur20; PST07]
- **darker grey**: linear running time
- **brown**: available implementation

## Special Mentions

- DC3 first $O(n)$ algorithm
- $O(n)$ running time and $O(1)$ space for integer alphabets possible
- until 2021: DivSufSort fastest in practice with $O(n \lg n)$ running time




## Timeline Sequential Suffix Sorting

- based on [Bah+19; Bin18; Kur20; PST07]
- **darker grey**: linear running time
- **brown**: available implementation

## Special Mentions

- DC3 first $O(n)$ algorithm
- $O(n)$ running time and $O(1)$ space for integer alphabets possible
- until 2021: DivSufSort fastest in practice with $O(n \lg n)$ running time
- since 2021: libSAIS fastest in practice with $O(n)$ running time

# Suffix Array Induced Sorting: Overview

## The Idea: Inducing

Given a text $T$ of length $n$ and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

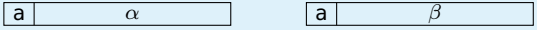
$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$

# Suffix Array Induced Sorting: Overview

## The Idea: Inducing

Given a text $T$ of length $n$ and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$

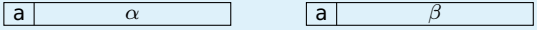


# Suffix Array Induced Sorting: Overview

## The Idea: Inducing

Given a text $T$ of length $n$ and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$




## The Algorithm: SAIS

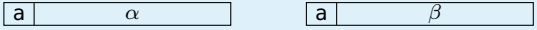
- using inducing for everything
- described in [NZC11]

# Suffix Array Induced Sorting: Overview

## The Idea: Inducing

Given a text $T$ of length $n$ and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$




## The Algorithm: SAIS

- using inducing for everything
- described in [NZC11]

## Suffix Array Construction in 3 Phases

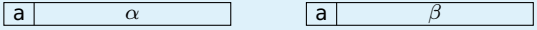
- classification
- sort special substrings/suffixes recursively
- induce all non-sorted suffixes

# Suffix Array Induced Sorting: Overview

## The Idea: Inducing

Given a text $T$ of length $n$ and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$




## The Algorithm: SAIS

- using inducing for everything
- described in [NZC11]

## Suffix Array Construction in 3 Phases

- classification
- sort special substrings/suffixes recursively
- induce all non-sorted suffixes

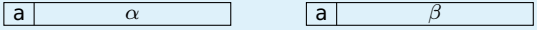
- classification helps identifying special suffixes
- everything in linear time

# Suffix Array Induced Sorting: Overview

## The Idea: Inducing

Given a text $T$ of length $n$ and two positions $i, j \in [1..n]$ with $T[i] = T[j]$, then

$$T[i..n] < T[j..n] \iff T[i+1..n] < T[j+1..n]$$




## The Algorithm: SAIS

- using inducing for everything
- described in [NZC11]

## Suffix Array Construction in 3 Phases

- classification
- sort special substrings/suffixes recursively
- induce all non-sorted suffixes

- classification helps identifying special suffixes
- everything in linear time

## Roadmap

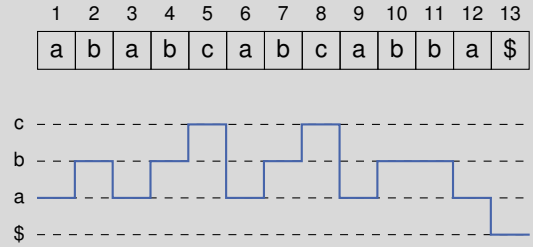
- classification
- inducing
- sorting special suffixes

# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$

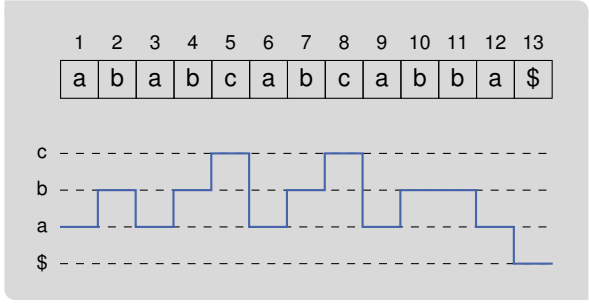


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$

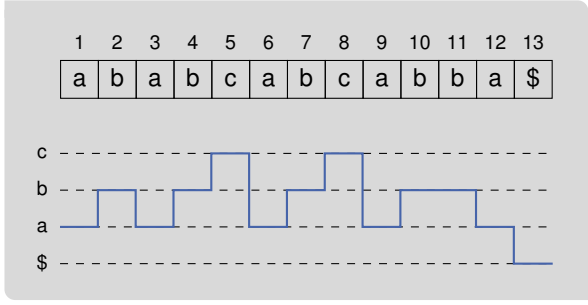


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

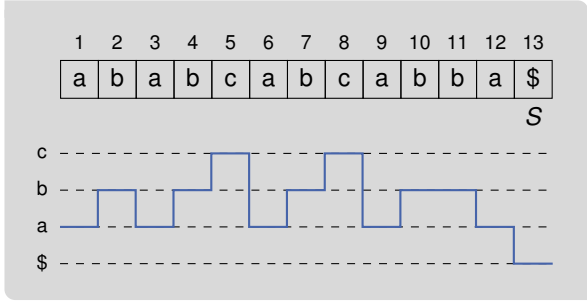


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

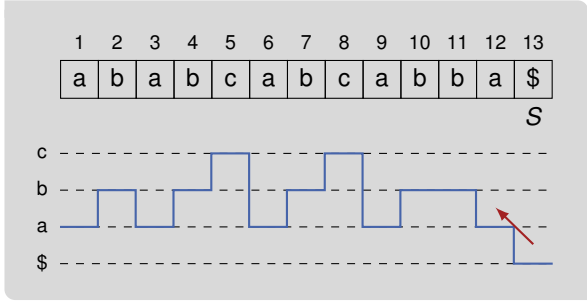


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

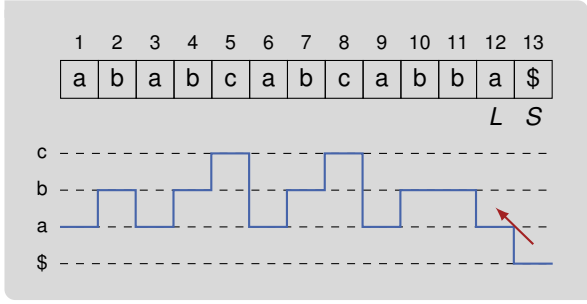


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

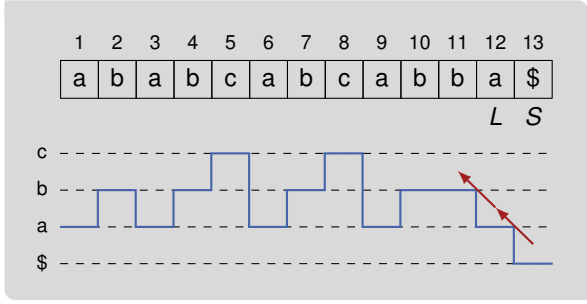


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

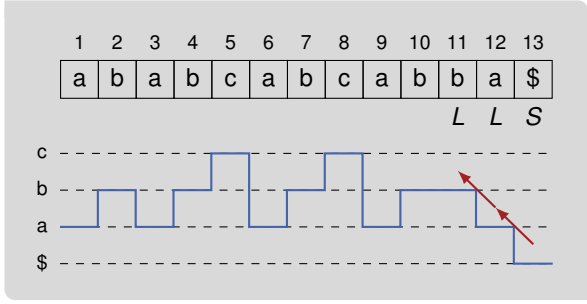


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

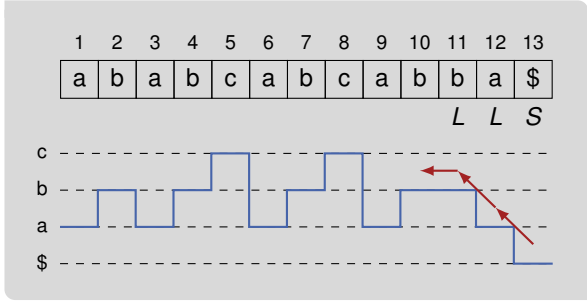


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

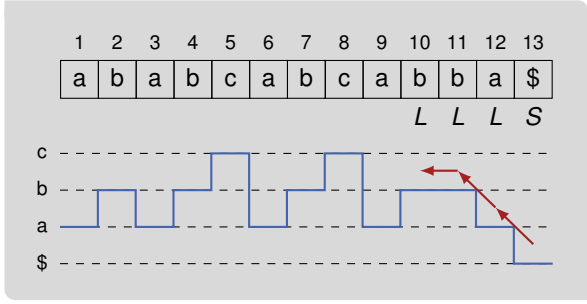


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

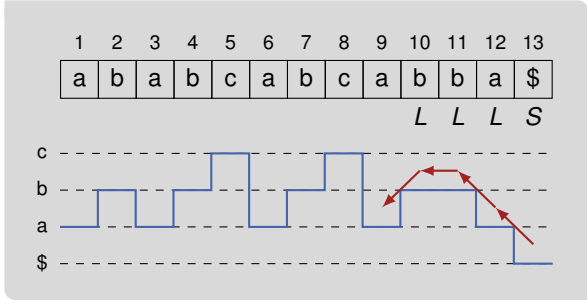


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

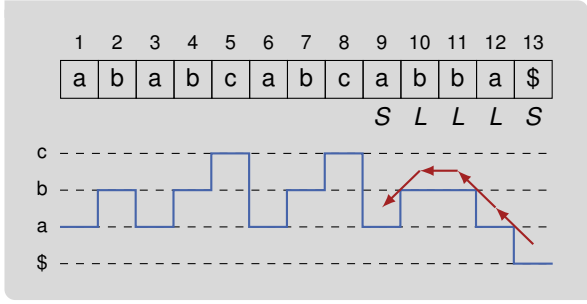


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

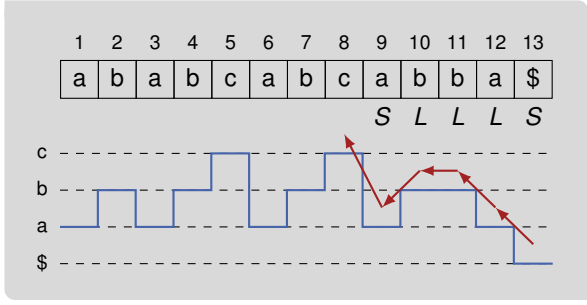


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

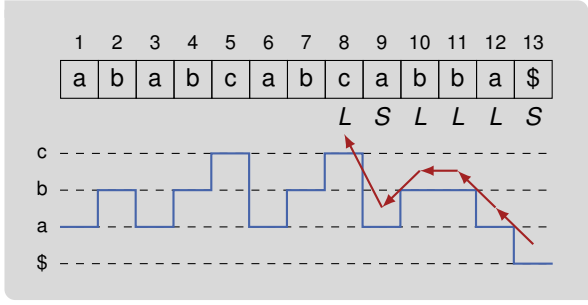


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

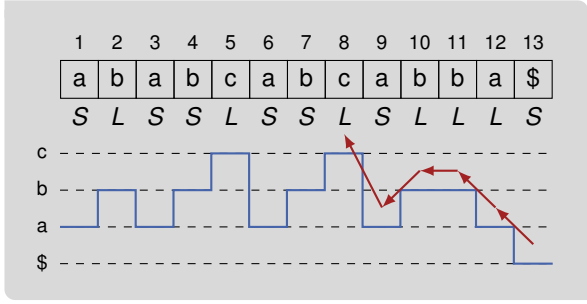


# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type** $S$
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type** $L$
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.



# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

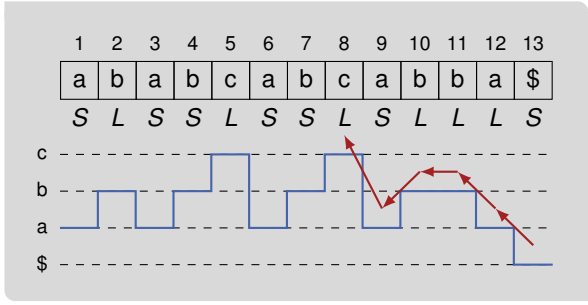
Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type $S$**
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type $L$**
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

## Definition: Leftmost $S$ Suffixes

Given a text $T$ of length $n$, $i \in [2..n]$ such that $T[i..n]$ has type $S$ and $T[i-1..n]$ has type $L$, then $T[i..n]$ is called **leftmost $S$ suffix (LMS)**.

- denoted by $S^\star$



# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

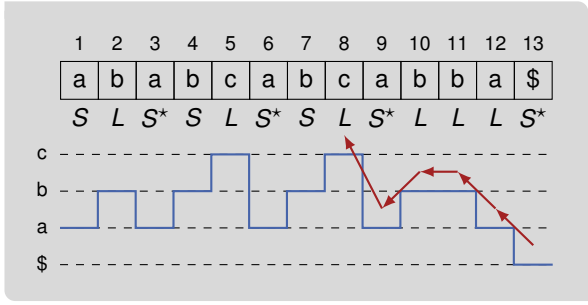
Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type $S$**
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type $L$**
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

## Definition: Leftmost $S$ Suffixes

Given a text $T$ of length $n$, $i \in [2..n]$ such that $T[i..n]$ has type $S$ and $T[i-1..n]$ has type $L$, then $T[i..n]$ is called **leftmost $S$ suffix (LMS)**.

- denoted by $S^\star$



# Suffix Array Induced Sorting: Classification (1/2)

## Definition: Type $L/S$ Suffixes

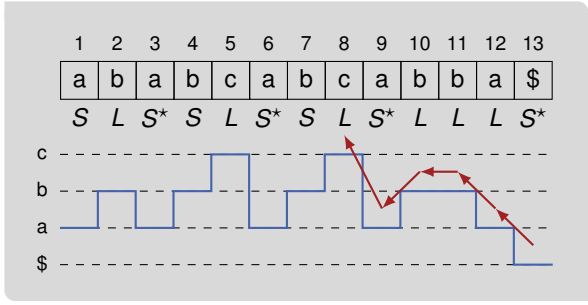
Given a text $T$ of length $n$ and $i \in [1..n]$, then

- $T[i] < T[i+1]$ or $i = n \Rightarrow T[i..n]$ has **type $S$**
- $T[i] > T[i+1] \Rightarrow T[i..n]$ has **type $L$**
- $T[i] = T[i+1] \Rightarrow T[i..n]$ has $T[i+1..n]$'s type.

## Definition: Leftmost $S$ Suffixes

Given a text $T$ of length $n$, $i \in [2..n]$ such that $T[i..n]$ has type $S$ and $T[i-1..n]$ has type $L$, then $T[i..n]$ is called **leftmost $S$ suffix (LMS)**.

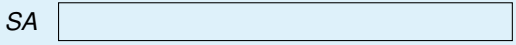
- denoted by $S^\star$




- scan text from right to left
- do not store types explicitly ℹ initially, we are only interested in LMS-suffixes

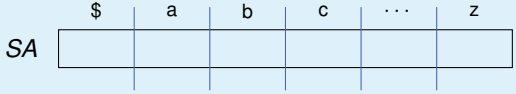
# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram



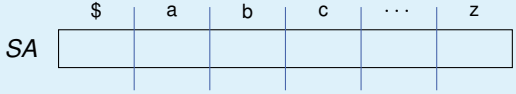
# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram



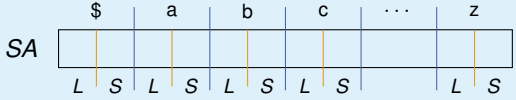
# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram
- use types of suffixes to partition suffix array



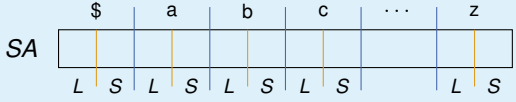
# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram
- use types of suffixes to partition suffix array



# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram
- use types of suffixes to partition suffix array



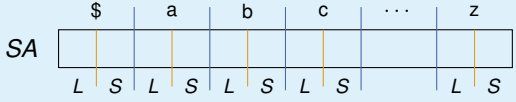

## Lemma: Order of $L/S$ Suffixes

Given a text $T$ of length $n$, a type $L$ suffixes $T[i..n]$ and a type $S$ $T[j..n]$ with $\alpha = T[i] = T[i]$, then

$$T[i..n] < T[j..n]$$

# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram
- use types of suffixes to partition suffix array




## Lemma: Order of $L/S$ Suffixes

Given a text $T$ of length $n$, a type $L$ suffixes $T[i..n]$ and a type $S$ $T[j..n]$ with $\alpha = T[i] = T[i]$, then

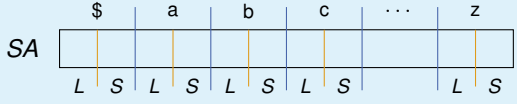
$$T[i..n] < T[j..n]$$

## Proof (Sketch)

- $T[i..n]$ has type $L$
  - $T[i..n] = \alpha \underbrace{\alpha \dots \alpha}_{\ell \geq 0 \text{ times}} \beta \dots \$$
  - with $\beta < \alpha$

# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram
- use types of suffixes to partition suffix array




## Lemma: Order of $L/S$ Suffixes

Given a text $T$ of length $n$, a type $L$ suffixes $T[i..n]$ and a type $S$ $T[j..n]$ with $\alpha = T[i] = T[i]$, then

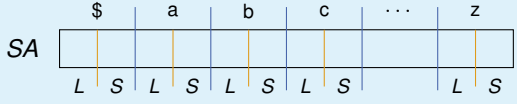
$$T[i..n] < T[j..n]$$

## Proof (Sketch)

- $T[i..n]$ has type $L$
  - $T[i..n] = \alpha \underbrace{\alpha \dots \alpha}_{\ell \geq 0 \text{ times}} \beta \dots \$$
  - with $\beta < \alpha$
- $T[j..n]$ has type $S$
  - $T[j..n] = \alpha \underbrace{\alpha \dots \alpha}_{\ell' \geq 0 \text{ times}} \gamma \dots \$$
  - with $\alpha < \gamma$

# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram
- use types of suffixes to partition suffix array




## Lemma: Order of $L/S$ Suffixes

Given a text $T$ of length $n$, a type $L$ suffixes $T[i..n]$ and a type $S$ $T[j..n]$ with $\alpha = T[i] = T[i]$, then

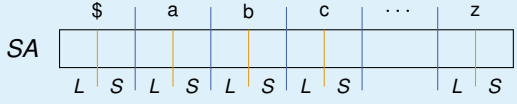
$$T[i..n] < T[j..n]$$

## Proof (Sketch)

- $T[i..n]$ has type $L$
  - $T[i..n] = \alpha \underbrace{\alpha \dots \alpha}_{\ell \geq 0 \text{ times}} \beta \dots \$$
  - with $\beta < \alpha$
- $T[j..n]$ has type $S$
  - $T[j..n] = \alpha \underbrace{\alpha \dots \alpha}_{\ell' \geq 0 \text{ times}} \gamma \dots \$$
  - with $\alpha < \gamma$
- if $\ell < \ell'$ then $\alpha < \gamma$ and $T[i..n] < T[j..n]$

# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram
- use types of suffixes to partition suffix array




## Lemma: Order of $L/S$ Suffixes

Given a text $T$ of length $n$, a type $L$ suffixes $T[i..n]$ and a type $S$ $T[j..n]$ with $\alpha = T[i] = T[i]$, then

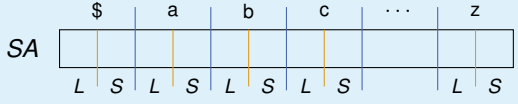
$$T[i..n] < T[j..n]$$

## Proof (Sketch)

- $T[i..n]$ has type $L$
  - $T[i..n] = \alpha \underbrace{\alpha \dots \alpha}_{\ell \geq 0 \text{ times}} \beta \dots \$$
  - with $\beta < \alpha$
- $T[j..n]$ has type $S$
  - $T[j..n] = \alpha \underbrace{\alpha \dots \alpha}_{\ell' \geq 0 \text{ times}} \gamma \dots \$$
  - with $\alpha < \gamma$
- if $\ell < \ell'$ then $\alpha < \gamma$ and $T[i..n] < T[j..n]$
- if $\ell = \ell'$ then $\beta < \gamma$ and $T[i..n] < T[j..n]$

# Suffix Array Induced Sorting: Classification (2/2)

- partition suffix array based text's histogram
- use types of suffixes to partition suffix array




## Lemma: Order of $L/S$ Suffixes

Given a text $T$ of length $n$, a type $L$ suffixes $T[i..n]$ and a type $S$ $T[j..n]$ with $\alpha = T[i] = T[i]$, then

$$T[i..n] < T[j..n]$$

## Proof (Sketch)

- $T[i..n]$ has type $L$
  - $T[i..n] = \alpha \underbrace{\alpha \dots \alpha}_{\ell \geq 0 \text{ times}} \beta \dots \$$
  - with $\beta < \alpha$
- $T[j..n]$ has type $S$
  - $T[j..n] = \alpha \underbrace{\alpha \dots \alpha}_{\ell' \geq 0 \text{ times}} \gamma \dots \$$
  - with $\alpha < \gamma$
- if $\ell < \ell'$ then $\alpha < \gamma$ and $T[i..n] < T[j..n]$
- if $\ell = \ell'$ then $\beta < \gamma$ and $T[i..n] < T[j..n]$
- if $\ell > \ell'$ then $\beta < \alpha$ and $T[i..n] < T[j..n]$

# Suffix Array Induced Sorting: Inducing (1/2)

## Lemma: Inducing

If $T[i+1..n] < T[j+1..n]$ and $T[i] = T[j]$ then

$$T[i..n] < T[j..n]$$

| | 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 |
| | $ | a $ | a b a b c a b c a b b a $ | a b b a $ | a b c a b b a $ | a b c a b c a b b a $ | b a $ | b a b c a b c a b b a $ | b b a $ | b c a b b a $ | b c a b c a b b a $ | c a b b a $ | c a b c a b b a $ |

# Suffix Array Induced Sorting: Inducing (1/2)

## Lemma: Inducing

If $T[i+1..n] < T[j+1..n]$ and $T[i] = T[j]$ then

$$T[i..n] < T[j..n]$$

| | 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 |
| | $ | a$ | abacabcabba$ | abba$ | abcabba$ | abcabcabba$ | ba$ | babcabcabba$ | bba$ | bcabba$ | bcabcabba$ | cabba$ | cabcabba$ |

# Suffix Array Induced Sorting: Inducing (1/2)

## Lemma: Inducing

If $T[i+1..n] < T[j+1..n]$ and $T[i] = T[j]$ then

$$T[i..n] < T[j..n]$$

| | 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 |
| | $ | a$ | abacabcabba$ | abba$ | abcabba$ | abcabcabba$ | ba$ | babcabcabba$ | bba$ | bcabba$ | bcabcabba$ | cabba$ | cabcabba$ |

# Suffix Array Induced Sorting: Inducing (1/2)

## Lemma: Inducing

If $T[i+1..n] < T[j+1..n]$ and $T[i] = T[j]$ then

$$T[i..n] < T[j..n]$$

| | 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 |
| | $ | a$ | abacabcabba$ | abba$ | abcabba$ | abcabba$ | ba$ | babcabcabba$ | bba$ | bcabba$ | bcabcabba$ | caba$ | cabcabba$ |

# Suffix Array Induced Sorting: Inducing (1/2)

## Lemma: Inducing

If $T[i+1..n] < T[j+1..n]$ and $T[i] = T[j]$ then

$$T[i..n] < T[j..n]$$

| | 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 |
| | $ | a$ | abacabcabba$ | abba$ | abcabba$ | abcabbcabba$ | ba$ | babcabcabba$ | bba$ | bcabba$ | bcabcabba$ | caba$ | cabcabba$ |

# Suffix Array Induced Sorting: Inducing (1/2)

## Lemma: Inducing

If $T[i+1..n] < T[j+1..n]$ and $T[i] = T[j]$ then

$$T[i..n] < T[j..n]$$

## Proof (Sketch)

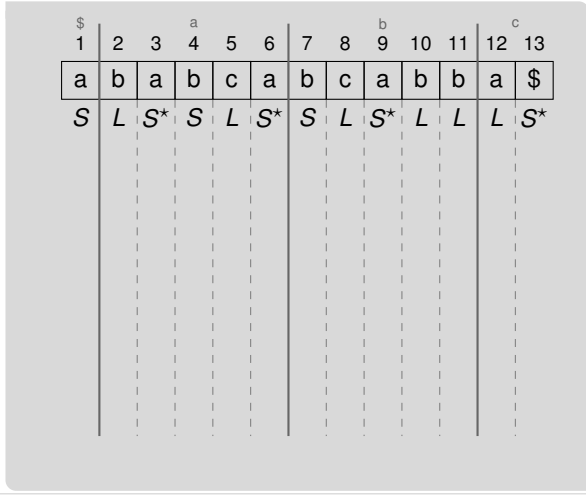
- similar to order of $L/S$ suffixes
- there is a leftmost character where $T[i+1..n]$ and $T[j+1..n]$ differ
- $T[i..n]$ and $T[j..n]$ differ at the same character

| | 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 |
| | $ | a$ | abacabcabba$ | abba$ | abcabba$ | abcabcabba$ | ba$ | babcabcabba$ | bba$ | bcabba$ | bcabcabba$ | caba$ | cabcabba$ |

# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

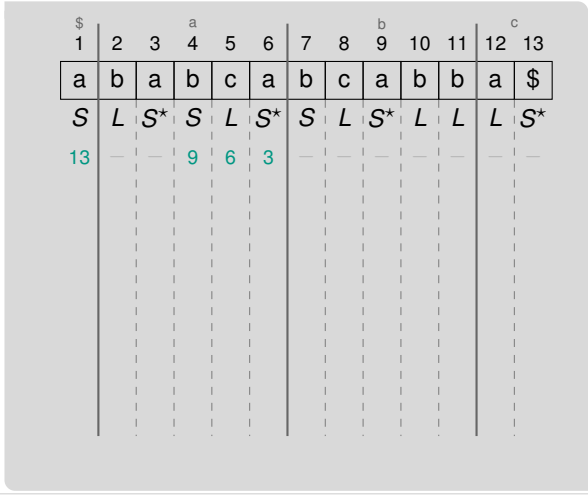
- Initialization
  - initialize each entry in SA with "—"
  - put sorted *LMS*-suffixes at the end of buckets



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

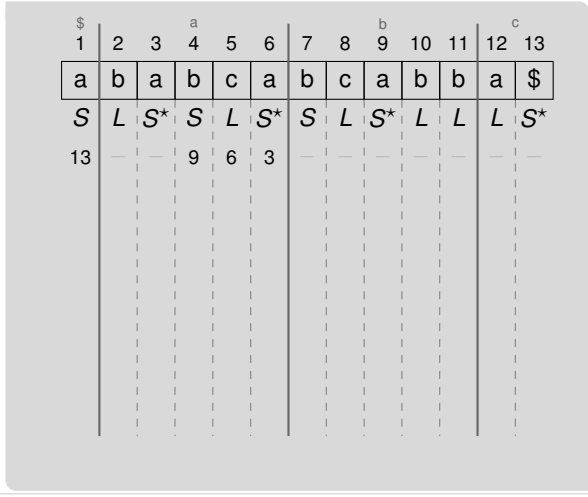
- Initialization
  - initialize each entry in SA with “—”
  - put sorted *LMS*-suffixes at the end of buckets



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

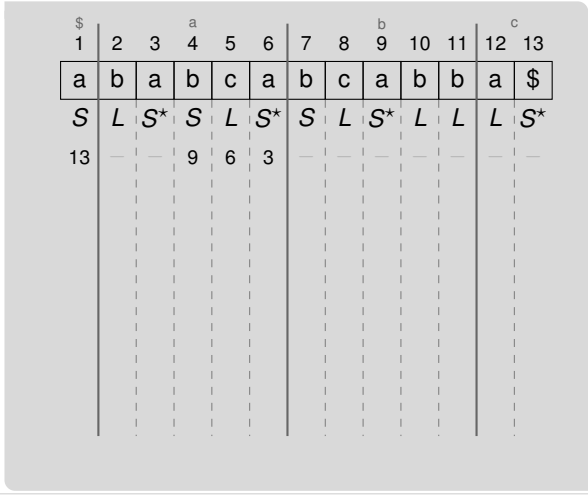
- Initialization
  - initialize each entry in SA with “—”
  - put sorted *LMS*-suffixes at the end of buckets



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

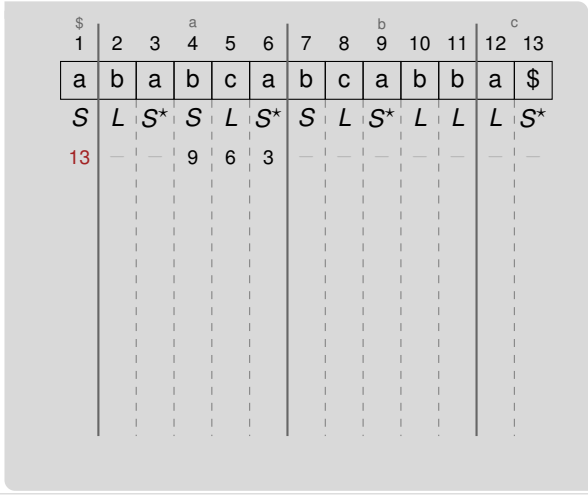
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

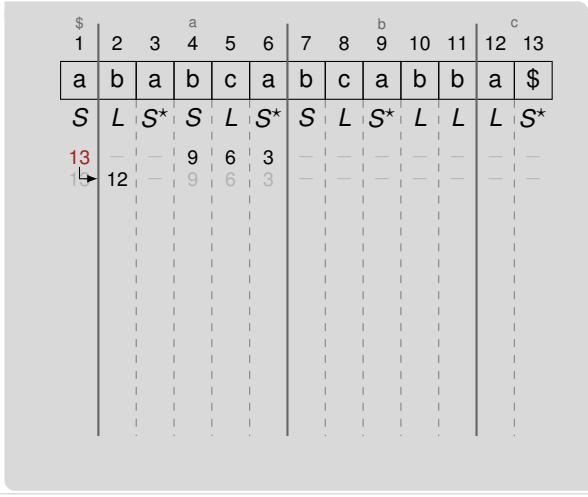
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

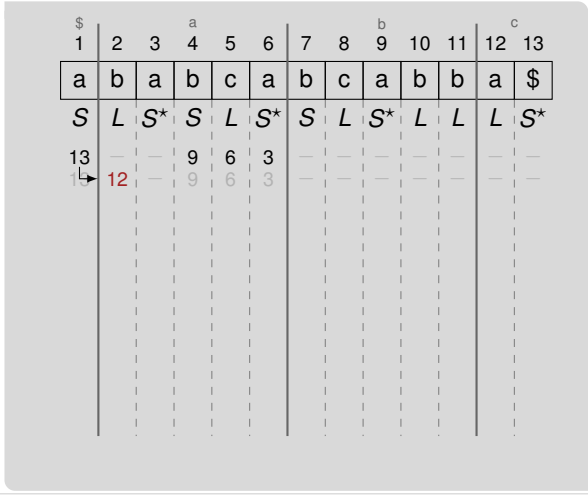
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

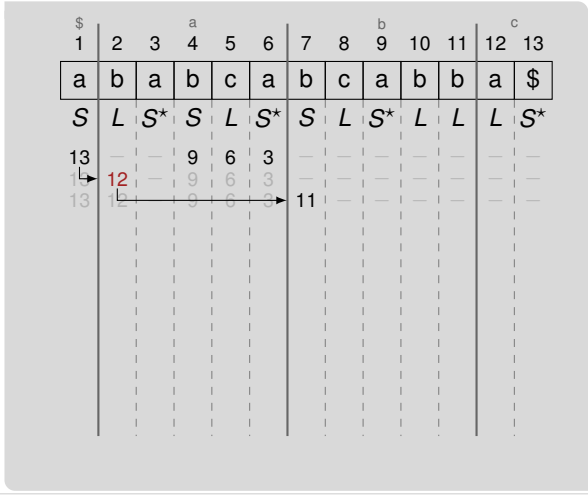
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

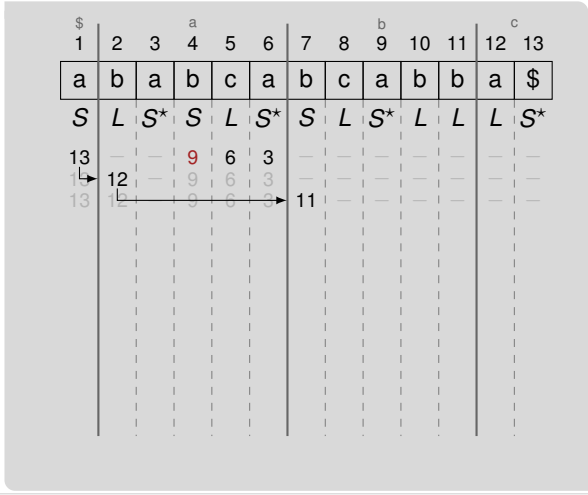
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

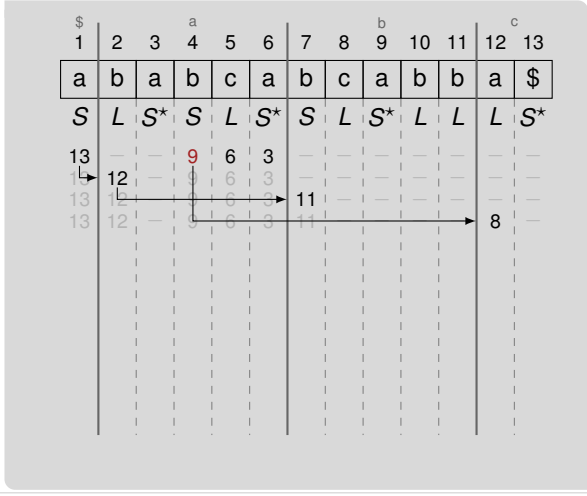
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

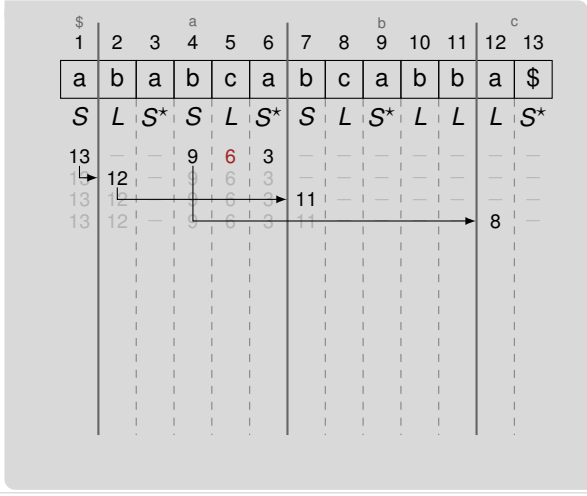
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

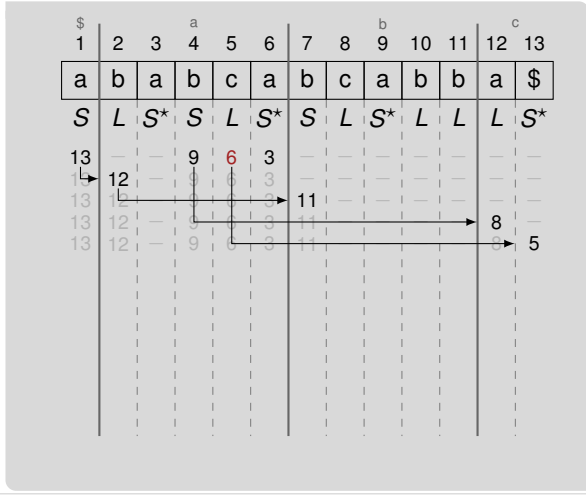
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

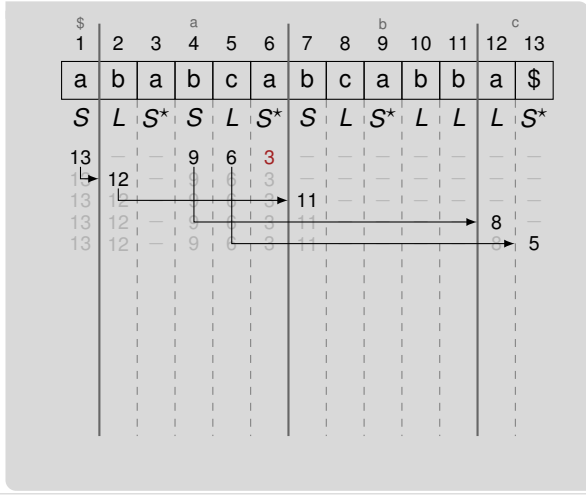
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

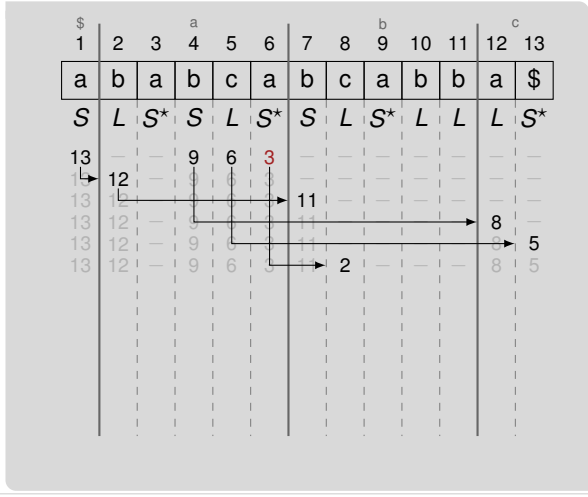
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

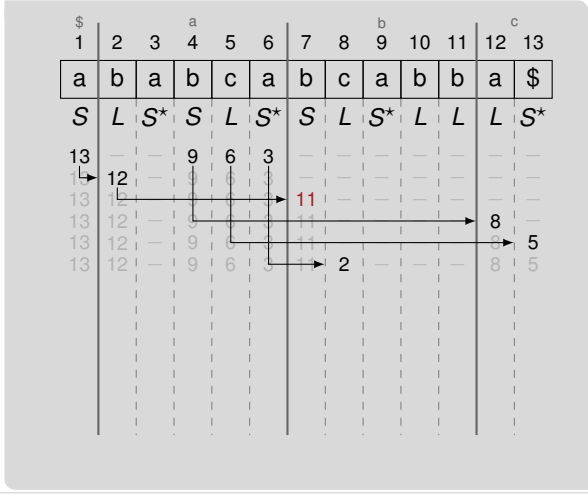
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

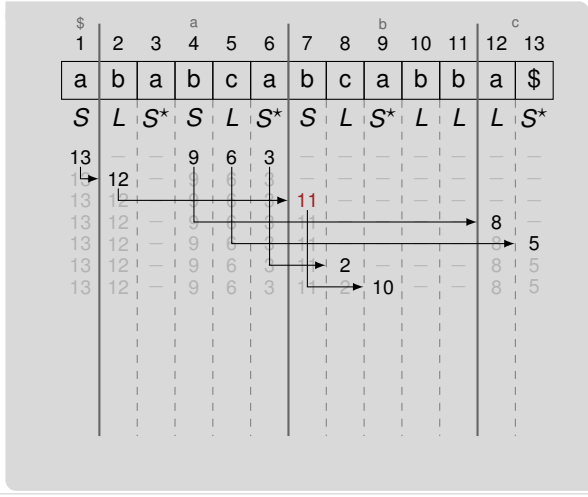
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

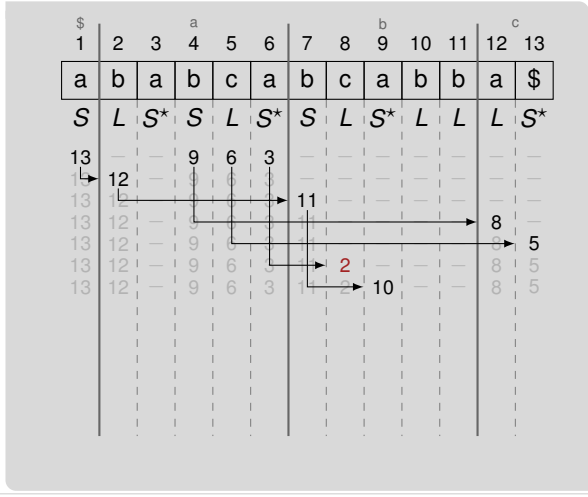
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

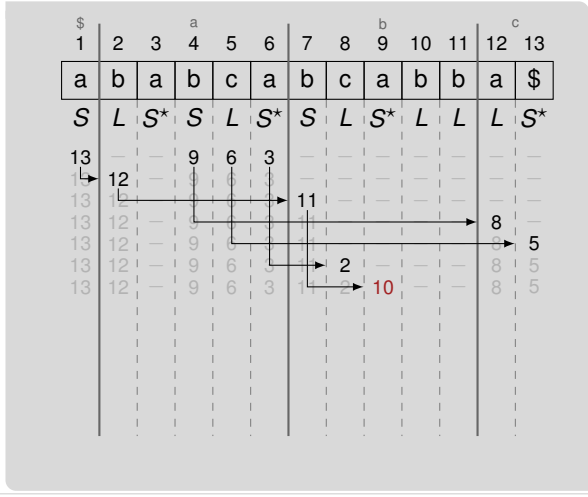
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

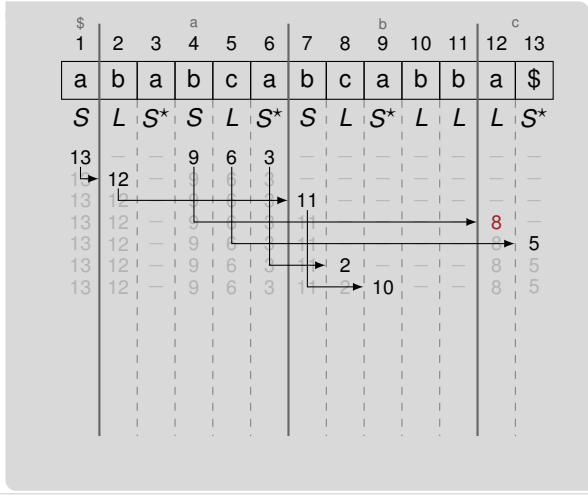
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

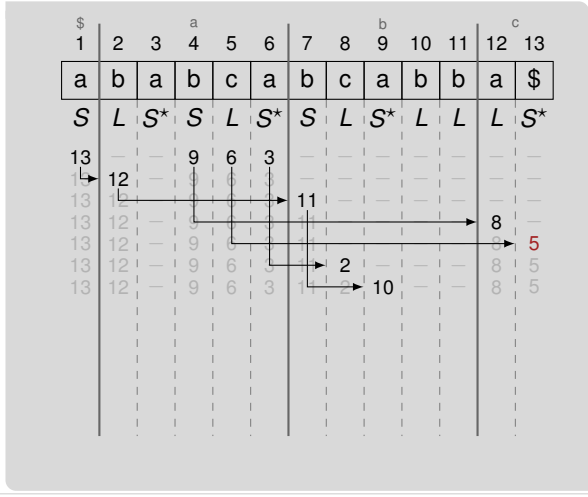
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

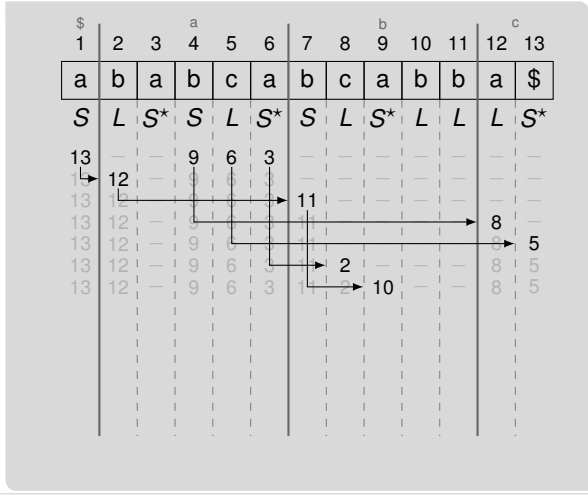
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

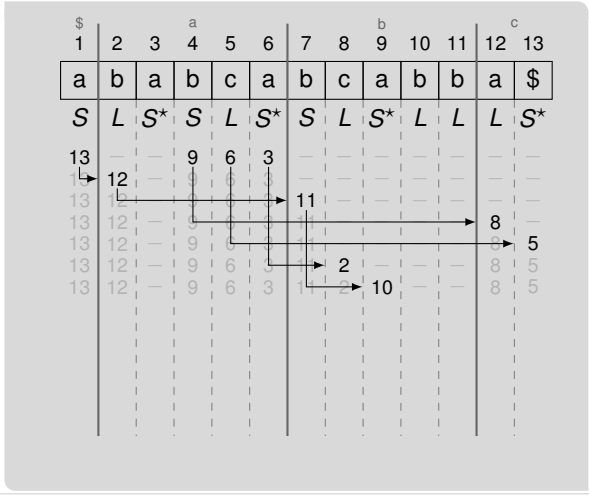
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

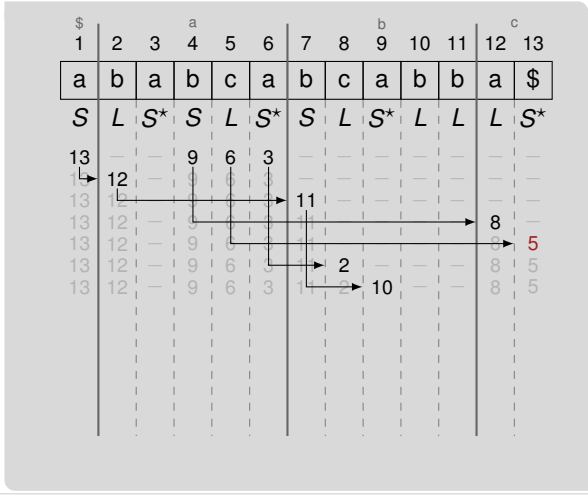
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

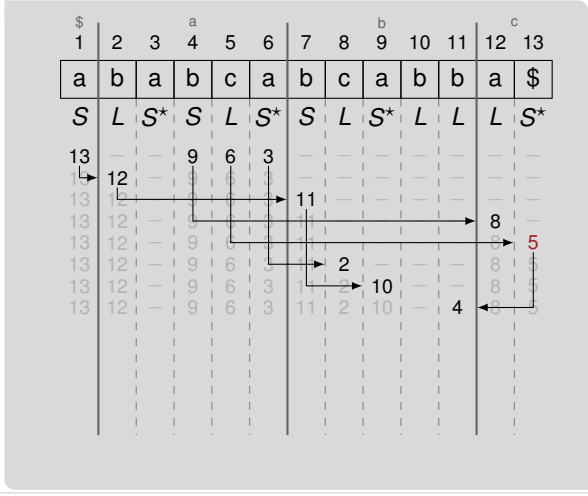
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

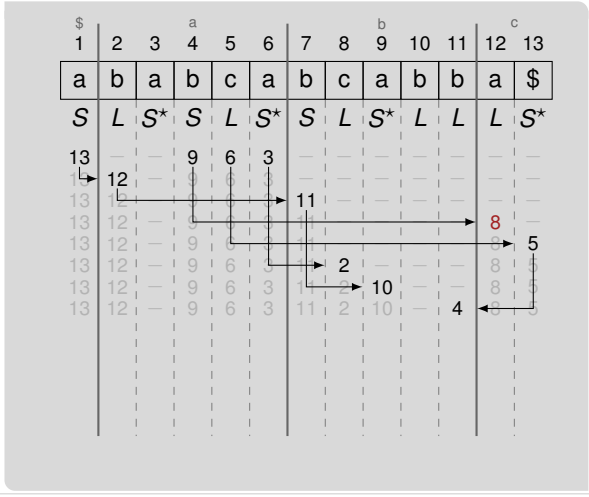
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

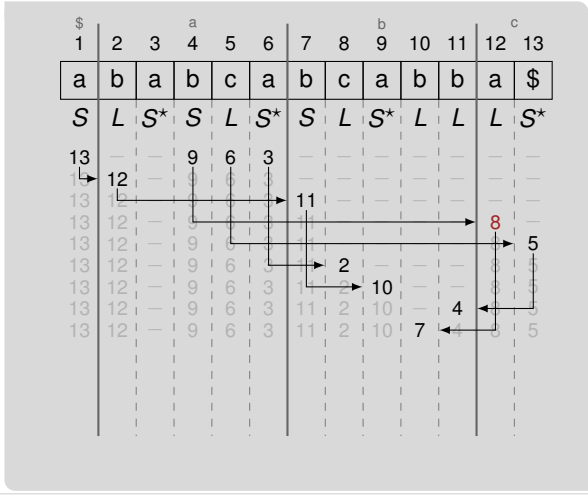
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

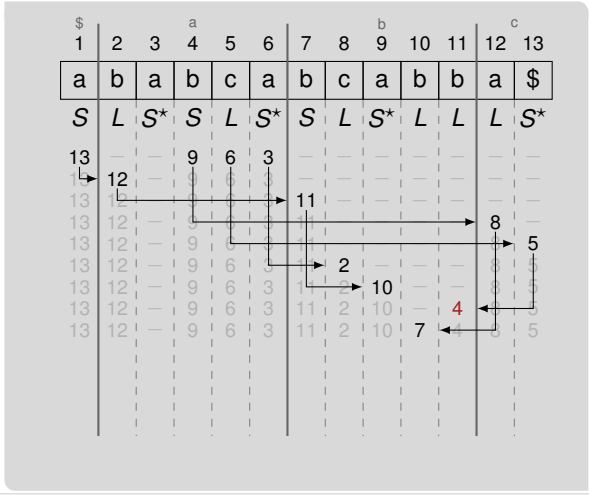
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

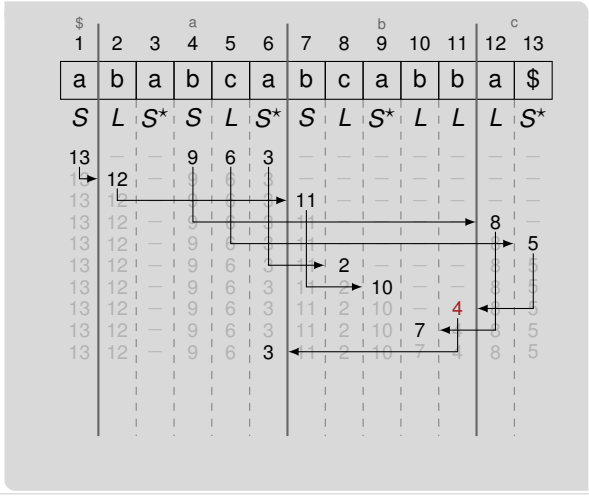
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

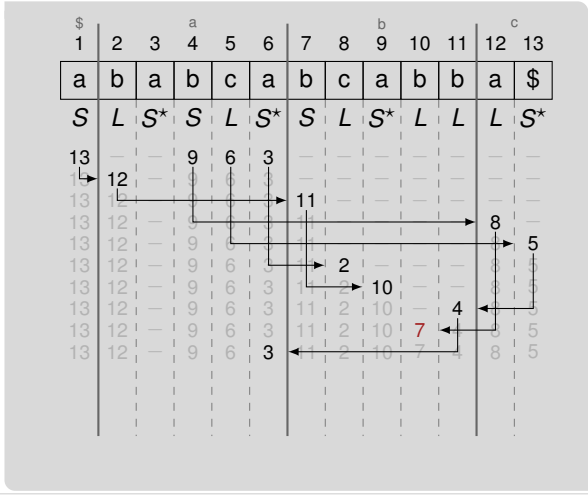
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

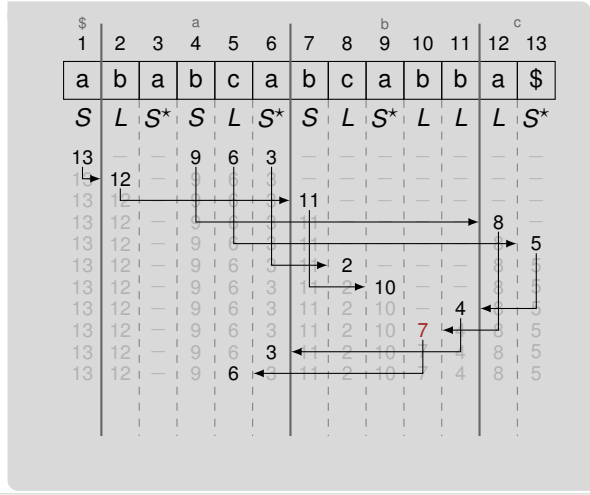
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

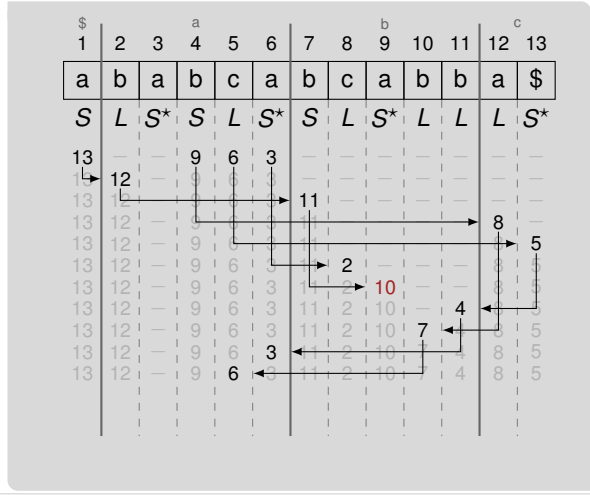
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

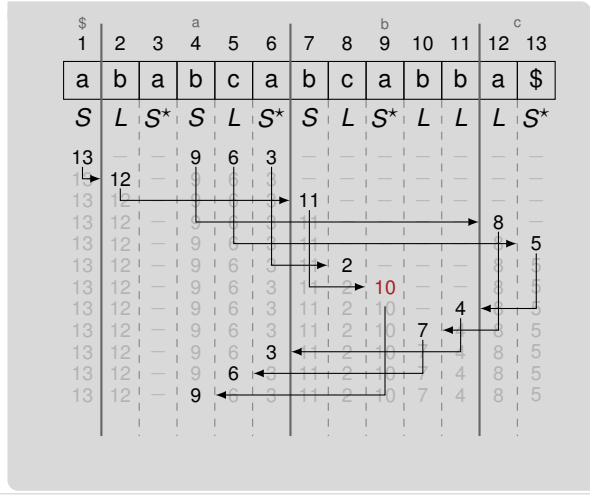
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

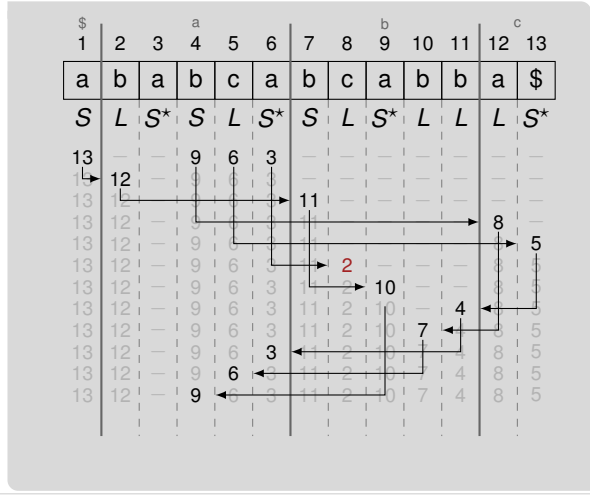
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

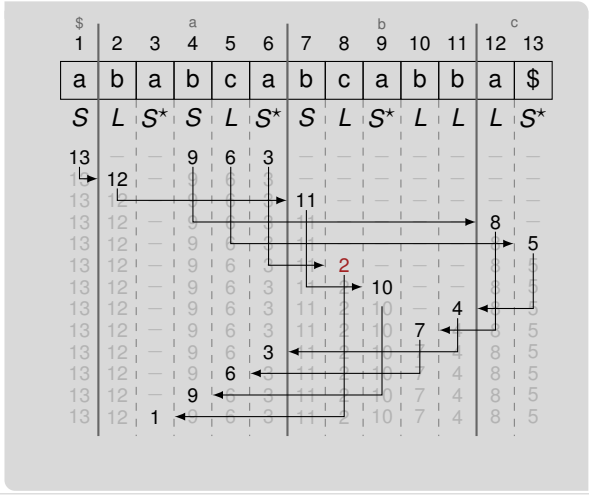
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

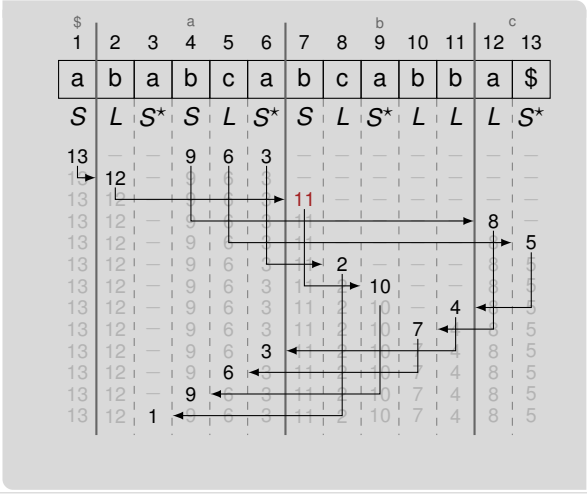
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

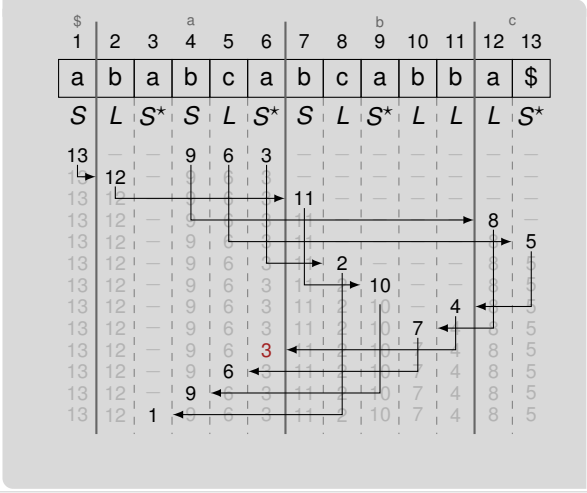
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

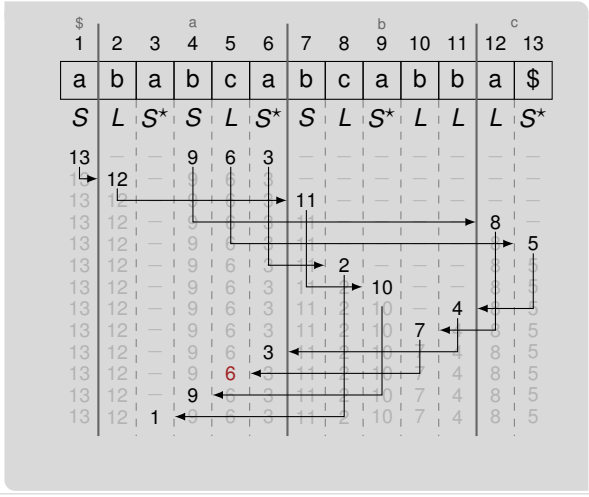
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

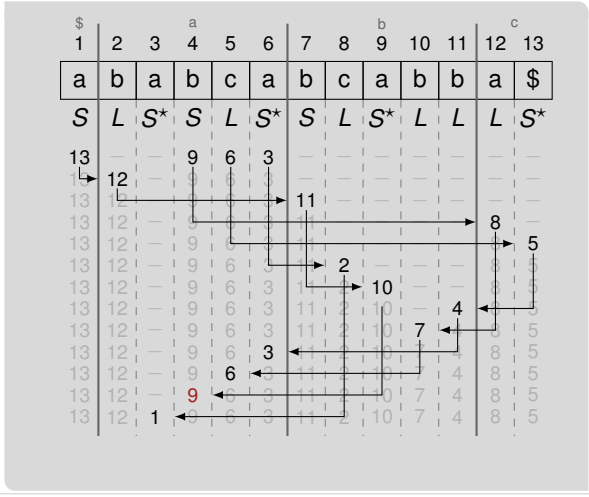
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

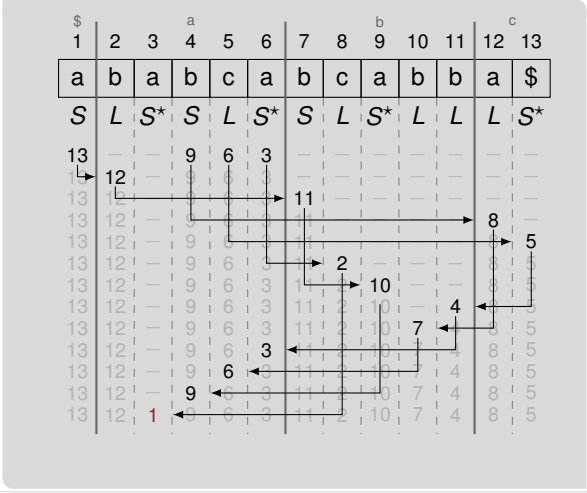
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

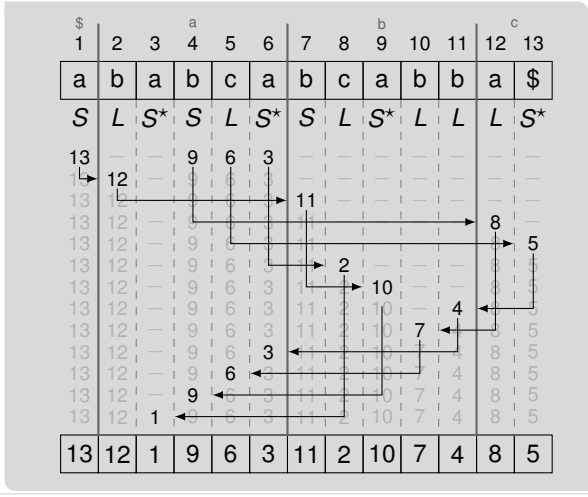
- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket



# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

- Initialization
  - initialize each entry in SA with "$-$"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *L*-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is *S*-type
  - then put $SA[i]-1$ at end of bucket

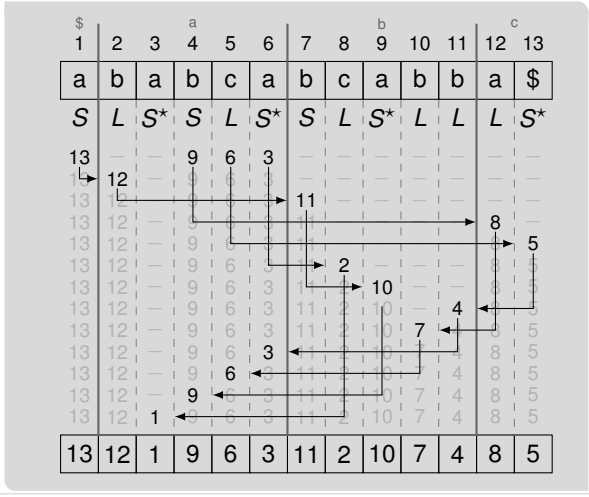


# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

- Initialization
  - initialize each entry in SA with "–"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket

- are all suffixes induced?

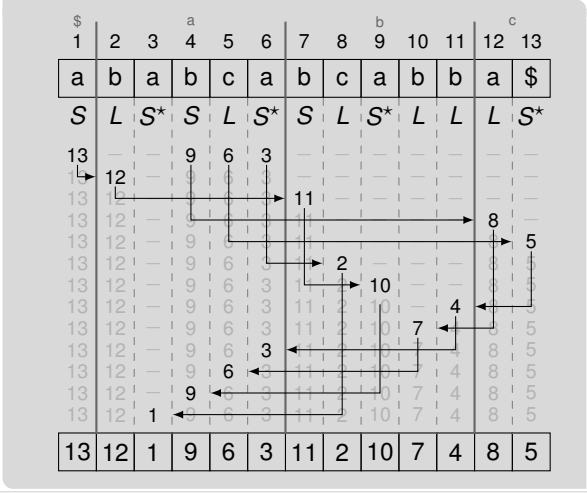


# Suffix Array Induced Sorting: Inducing (2/2)

## Inducing in SAIS

- Initialization
  - initialize each entry in SA with "–"
  - put sorted *LMS*-suffixes at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $L$-type
  - then put $SA[i]-1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i]-1..n]$ is $S$-type
  - then put $SA[i]-1$ at end of bucket

- are all suffixes induced?
- now we only need to sort $S^\star$ suffixes



# Suffix Array Induced Sorting: LMS-Substrings (1/2)

- how to sort $S^\star$ suffixes?
- slightly adopt algorithm

# Suffix Array Induced Sorting: LMS-Substrings (1/2)

- how to sort $S^\star$ suffixes?
- slightly adopt algorithm

**Definition: LMS-Prefix**

Let $i < j$ or $i = j = n$ be text positions, such that $T[j..n]$ is LMS and $\nexists k \in (i, j)$ with $T[k..n]$ is LMS, then we call $T[i..j]$ **LMS-prefix**

# Suffix Array Induced Sorting: LMS-Substrings (1/2)

- how to sort $S^\star$ suffixes?
- slightly adopt algorithm

## Definition: LMS-Prefix

Let $i < j$ or $i = j = n$ be text positions, such that $T[j..n]$ is LMS and $\nexists k \in (i, j)$ with $T[k..n]$ is LMS, then we call $T[i..j]$ **LMS-prefix**

## Definition: LMS-Substring

Let $T[i..j]$ be an LMS-prefix and $T[i..n]$ be LMS, then $T[i..j]$ is an **LMS-substring**

# Suffix Array Induced Sorting: LMS-Substrings (1/2)

- how to sort $S^\star$ suffixes?
- slightly adopt algorithm

## Definition: LMS-Prefix

Let $i < j$ or $i = j = n$ be text positions, such that $T[j..n]$ is LMS and $\nexists k \in (i, j)$ with $T[k..n]$ is LMS, then we call $T[i..j]$ **LMS-prefix**

## Definition: LMS-Substring

Let $T[i..j]$ be an LMS-prefix and $T[i..n]$ be LMS, then $T[i..j]$ is an **LMS-substring**

## Inducing LMS-Prefixes

- Initialization
  - initialize each entry in SA with "$-$"
  - put *LMS*-suffixes in text order at the end of buckets
- Scan Left to Right ($i = 1, 2, \ldots, n$)
  - if $SA[i] \neq -$ and $T[SA[i] - 1..n]$ is $L$-type
  - then put $SA[i] - 1$ at beginning of bucket
- Scan Right to Left ($i = n, n-1, \ldots, 1$)
  - if $SA[i] \neq -$ and $T[SA[i] - 1..n]$ is $S$-type
  - then put $SA[i] - 1$ at end of bucket

# Suffix Array Induced Sorting: LMS-Substrings (2/2)

**Lemma: Inducing LMS-Prefixes**

The algorithm sorts all LMS-Prefixes correctly

# Suffix Array Induced Sorting: LMS-Substrings (2/2)

Karlsruhe Institute of Technology

## Lemma: Inducing LMS-Prefixes

The algorithm sorts all LMS-Prefixes correctly

## Proof (Sketch)

- initially: only $T[n..n]$ sorted correctly

# Suffix Array Induced Sorting: LMS-Substrings (2/2)

Karlsruhe Institute of Technology

## Lemma: Inducing LMS-Prefixes

The algorithm sorts all LMS-Prefixes correctly

## Proof (Sketch)

- initially: only $T[n..n]$ sorted correctly
- L2R: $L$-type LMS-prefixes sorted correctly
  - only care for first character of next LMS
  - LMS in correct bucket
  - sorted correctly for first character

# Suffix Array Induced Sorting: LMS-Substrings (2/2)

## Lemma: Inducing LMS-Prefixes

The algorithm sorts all LMS-Prefixes correctly

## Proof (Sketch)

- initially: only $T[n..n]$ sorted correctly
- L2R: $L$-type LMS-prefixes sorted correctly
  - only care for first character of next LMS
  - LMS in correct bucket
  - sorted correctly for first character
- R2L: $S$-type LMS-prefixes sorted correctly
  - only care for first character of next LMS
  - LMS in correct bucket
  - sorted correct for first character

# Suffix Array Induced Sorting: LMS-Substrings (2/2)

## Lemma: Inducing LMS-Prefixes

The algorithm sorts all LMS-Prefixes correctly

## Proof (Sketch)

- initially: only $T[n..n]$ sorted correctly
- L2R: *L*-type LMS-prefixes sorted correctly
  - only care for first character of next LMS
  - LMS in correct bucket
  - sorted correctly for first character
- R2L: *S*-type LMS-prefixes sorted correctly
  - only care for first character of next LMS
  - LMS in correct bucket
  - sorted correct for first character

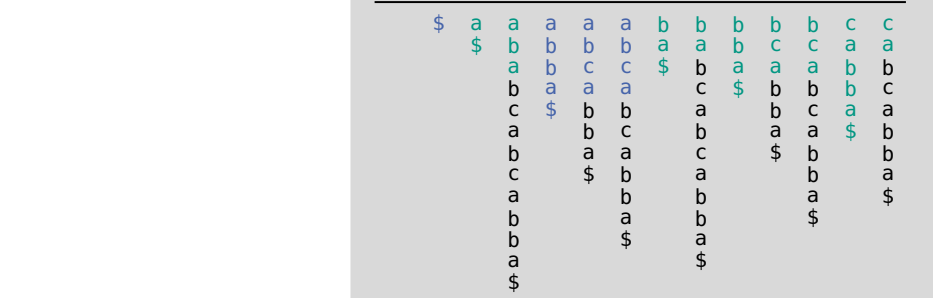
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *T* | a | b | a | b | c | a | b | c | a | b | b | a | $ |
| *SA* | 13 | 12 | 1 | 9 | 6 | 3 | 11 | 2 | 10 | 7 | 4 | 8 | 5 |
| | $ | a$ | abacabba$ | abba$ | abcabba$ | abcabba$ | ba$ | bacabba$ | bba$ | bcabba$ | bcabba$ | cabba$ | cabcabba$ |

# Suffix Array Induced Sorting: Recursion

## Lemma: Running Time Computation $T'$

Computing $T'$ requires $O(n)$ time

| | 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 |



# Suffix Array Induced Sorting: Recursion

## Lemma: Running Time Computation $T'$

Computing $T'$ requires $O(n)$ time

## Proof (Sketch)

- find LMS-substrings in $O(1)$ time ℹ save $S$-buckets
- scan each LMS-substring twice
- each character is in at most two LMS-substrings

| | 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 |
| | $ | a $ | a b a b c a b c a b b a $ | a b b a $ | a b c a b b a $ | a b c a b c a b b a $ | b a $ | b a b c a b c a b b a $ | b b a $ | b c a b b a $ | b c a b c a b b a $ | c a b b a $ | c a b c a b b a $ |

# Suffix Array Induced Sorting: Recursion

## Lemma: Running Time Computation $T'$

Computing $T'$ requires $O(n)$ time

## Proof (Sketch)

- find LMS-substrings in $O(1)$ time ℹ save $S$-buckets
- scan each LMS-substring twice
- each character is in at most two LMS-substrings

- construct text $T'$ using ranks of LMS-substrings
- compare LMS-substrings character-wise

| | 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 |

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

# Suffix Array Induced Sorting: Recursion

## Lemma: Running Time Computation $T'$

Computing $T'$ requires $O(n)$ time

## Proof (Sketch)

- find LMS-substrings in $O(1)$ time ℹ save $S$-buckets
- scan each LMS-substring twice
- each character is in at most two LMS-substrings

- construct text $T'$ using ranks of LMS-substrings
- compare LMS-substrings character-wise

| | 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 |

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

# Suffix Array Induced Sorting: Recursion

## Lemma: Running Time Computation $T'$

Computing $T'$ requires $O(n)$ time

## Proof (Sketch)

- find LMS-substrings in $O(1)$ time ℹ save $S$-buckets
- scan each LMS-substring twice
- each character is in at most two LMS-substrings

- construct text $T'$ using ranks of LMS-substrings
- compare LMS-substrings character-wise

| | 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 |

$ a$ abacabcabba$ abba$ abcabba$ abcabbcabba$ ba$ babcabcabba$ babca$ bcabcabba$ cabba$ cabcabba$

- $T' = 0122\$$

# Suffix Array Induced Sorting: Running Time

## Lemma: SAIS Time Complexity

Given a text of length $n$, SAIS computes the suffix array in $O(n)$ time using

# Suffix Array Induced Sorting: Running Time

## Lemma: SAIS Time Complexity

Given a text of length $n$, SAIS computes the suffix array in $O(n)$ time using

## Proof (Sketch)

- classification, sorting of special suffixes, and inducing in $O(n)$ time
- the number of $S^\star$ suffixes is at most $\lfloor n/2 \rfloor$
- $\mathcal{T}(n) = \mathcal{T}(\lfloor n/2 \rfloor) + O(n) = O(n)$

# Suffix Array Induced Sorting: Running Time

Karlsruhe Institute of Technology

## Lemma: SAIS Time Complexity

Given a text of length $n$, SAIS computes the suffix array in $O(n)$ time using

## Proof (Sketch)

- classification, sorting of special suffixes, and inducing in $O(n)$ time
- the number of $S^\star$ suffixes is at most $\lfloor n/2 \rfloor$
- $\mathcal{T}(n) = \mathcal{T}(\lfloor n/2 \rfloor) + O(n) = O(n)$

## Space Requirements

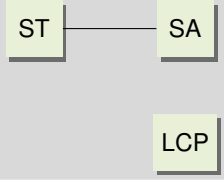
- naive: $O(n \lg n)$ bits
- better: $n\lceil \lg n \rceil + 2\sigma\lceil \lg n \rceil$ bits 

# Conclusion and Outlook

## This Lecture

- suffix trees and suffix arrays
- linear time suffix array construction

## Linear Time Construction



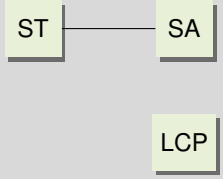
# Conclusion and Outlook

## This Lecture

- suffix trees and suffix arrays
- linear time suffix array construction

- suffix trees require ≈ 8–20 bytes per character
- suffix arrays require 5 bytes per character ℹ for up to ≈ 1 TB text
- currently fastest implementation available at `https://github.com/IlyaGrebnov/libsais`

## Linear Time Construction



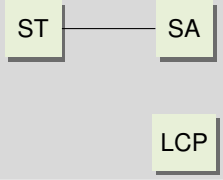
# Conclusion and Outlook

## This Lecture

- suffix trees and suffix arrays
- linear time suffix array construction

- suffix trees require ≈ 8–20 bytes per character
- suffix arrays require 5 bytes per character ℹ for up to ≈ 1 TB text
- currently fastest implementation available at `https://github.com/IlyaGrebnov/libsais`

## Linear Time Construction




## Next Lecture

- linear time LCP-array construction
- interesting properties of LCP-array
- computing suffix trees using suffix array and LCP-array

# Bibliography I


[AKO04] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. “Replacing Suffix Trees with Enhanced Suffix Arrays”. In: *J. Discrete Algorithms* 2.1 (2004), pages 53–86. DOI: 10.1016/S1570-8667(03)00065-0.

[Bah+19] Johannes Bahne, Nico Bertram, Marvin Böcker, Jonas Bode, Johannes Fischer, Hermann Foot, Florian Grieskamp, Florian Kurpicz, Marvin Löbel, Oliver Magiera, Rosa Pink, David Piper, and Christopher Poeplau. “SACABench: Benchmarking Suffix Array Construction”. In: *SPIRE*. Volume 11811. Lecture Notes in Computer Science. Springer, 2019, pages 407–416. DOI: 10.1007/978-3-030-32686-9_29.

[Bin18] Timo Bingmann. “Scalable String and Suffix Sorting: Algorithms, Techniques, and Tools”. PhD thesis. Karlsruhe Institute of Technology, Germany, 2018. DOI: 10.5445/IR/1000085031.

[Far97] Martin Farach. “Optimal Suffix Tree Construction with Large Alphabets”. In: *FOCS*. IEEE Computer Society, 1997, pages 137–143. DOI: 10.1109/SFCS.1997.646102.

# Bibliography II


[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.

[Kur20] Florian Kurpicz. “Parallel Text Index Construction”. PhD thesis. Technical University of Dortmund, Germany, 2020. DOI: 10.17877/DE290R-21114.

[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.

[NZC11] Ge Nong, Sen Zhang, and Wai Hong Chan. “Two Efficient Algorithms for Linear Time Suffix Array Construction”. In: *IEEE Trans. Computers* 60.10 (2011), pages 1471–1484. DOI: 10.1109/TC.2010.188.

[PST07] Simon J. Puglisi, William F. Smyth, and Andrew Turpin. “A Taxonomy of Suffix Array Construction Algorithms”. In: *ACM Comput. Surv.* 39.2 (2007), page 4. DOI: 10.1145/1242471.1242472.

# Bibliography III

[Ukk95] Esko Ukkonen. “On-Line Construction of Suffix Trees”. In: *Algorithmica* 14.3 (1995), pages 249–260. DOI: 10.1007/BF01206331.

[Wei73] Peter Weiner. “Linear Pattern Matching Algorithms”. In: *SWAT (FOCS)*. IEEE Computer Society, 1973, pages 1–11. DOI: 10.1109/SWAT.1973.13.