Text Indexing

Lecture 02: Suffix Trees and Suffix Arrays

Florian Kurpicz
Open Question from Last Lecture

- two-leveled index $O(m + \lg \sigma)$ query time
- requires $O(N)$ words of space
- and caterpillar trees $S = \{\$, a\$, aa\$, \ldots \}$?
two-leveled index $O(m + \lg \sigma)$ query time
requires $O(N)$ words of space
and caterpillar trees $S = \{\$, a\$, aa\$, \ldots\}$?
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

```
abba
bba
abba
abba
```

$\lambda(v) =$ concatenation of labels from $r$ to $v$

$d(v) =$ string-depth of $v$

$\lambda(v) =$ bba

$d(v) =$ 4

$\lambda(v) =$ abba

$d(v) =$ 4

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$</td>
</tr>
</tbody>
</table>

*5/23* 2021-10-25 Florian Kurpicz | Text Indexing | 02 Suffix Trees and Arrays

Institute for Theoretical Computer Science, Algorithmics II
Definition: Suffix Tree [Wei73]

A suffix tree (ST) for a text $T$ of length $n$ is a compact trie over $S = \{T[1..n], T[2..n], \ldots, T[n..n]\}$ such that 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 $v$.

<table>
<thead>
<tr>
<th>$v$</th>
<th>$\lambda(v)$</th>
<th>$d(v)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>$b$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>$b$</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>$a$</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>$a$</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>$b$</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>$b$</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>$a$</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>$$</td>
<td>3</td>
</tr>
</tbody>
</table>

Example:

- $abba$$abba$$abba$$abba$
**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
- for simplicity, we show text

Suffix Information
- label leaves with corresponding suffix
  📌 will be important later on
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)
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
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
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
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
very (most?) powerful text-index
very (most?) powerful text-index
- suffix trees require $\approx 8–20$ bytes per character
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]
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
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
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

next, suffix array construction
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 $i, j \in [1..n]$

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$</td>
</tr>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$LCP$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Suffix Array and LCP-Array
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 $i, j \in [1..n]$

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

**Definition: Longest Common Prefix Array**

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

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

---

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$ $</td>
</tr>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>$LCP$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
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 $i, j \in [1..n]$

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

Definition: Longest Common Prefix Array
Given a text $T$ of length $n$ and its SA, the LCP-array is defined as

$$LCP[i] = \begin{cases} 0 & i = 1 \\ \max\{\ell : T[SA[i]..SA[i] + \ell) = T[SA[i-1]..SA[i-1] + \ell)] \} & i \neq 1 \end{cases}$$
Function `SearchSA(T, SA[1..n], P[1..m])`:

1. $\ell = 1, r = n + 1$
2. while $\ell < r$ do
   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. $s = \ell, \ell = \ell - 1, r = n$
3. while $\ell < r$ do
   4. $i = \lceil (\ell + r) / 2 \rceil$
   5. if $P = T[SA[i]..SA[i] + m]$ then $\ell = i$
   6. else $r = i - 1$
5. return $[s, r]$

pattern $P = abc$
Pattern Matching with the Suffix Array (1/2)

Function `SearchSA(T, SA[1..n], P[1..m])`:

1. \( \ell = 1, r = n + 1 \)
2. while \( \ell < r \) do  
   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  
   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 \( P = \text{abc} \)
Pattern Matching with the Suffix Array (1/2)

**Function** `SearchSA(T, SA[1..n], P[1..m])`:

1. \( \ell = 1, r = n + 1 \)
2. **while** \( \ell < r \) do
   1. 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. \( s = \ell, l = \ell - 1, r = n \)
8. **while** \( \ell < r \) do
   9. Find right border
   10. \( i = \lceil \ell + r/2 \rceil \)
11. if \( P = T[SA[i]..SA[i] + m) \) then \( \ell = i \)
12. else \( r = i - 1 \)
13. return \([s, r]\)

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

1. \( \ell = 1, r = n + 1 \)
2. \text{while } \ell < r \text{ do}
   3. \( i = \lceil (\ell + r)/2 \rceil \)
   4. \text{if } P > T[\text{SA}[i]..\text{SA}[i] + m] \text{ then}
      5. \( \ell = i + 1 \)
   6. \text{else } r = i

7. \( s = \ell, \ell = \ell - 1, r = n \)
8. \text{while } \ell < r \text{ do}
   9. \( i = \lfloor \ell + r/2 \rfloor \)
  10. \text{if } P = T[\text{SA}[i]..\text{SA}[i] + m] \text{ then } \ell = i
  11. \text{else } r = i - 1

12. \text{return } [s, r]

Lemma: Running Time \text{SeachSA}

The \text{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 \text{SA} in \( O(\lg n) \) 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. \( i = \lfloor(\ell + r)/2\rfloor \)
   4. if \( P > T[SA[i]..SA[i] + m] \) then
      5. \( \ell = i + 1 \)
   6. else \( r = i \)
   7. \( s = \ell, \ell = \ell - 1, r = n \)
3. while \( \ell < r \) do
   4. \( i = \lceil(\ell + r/2)\rceil \)
   5. if \( P = T[SA[i]..SA[i] + m] \) then \( \ell = i \)
   6. else \( r = i - 1 \)
5. return \([s, r]\)

Lemma: Running Time SeachSA

The SeachSA answers counting queries in \( O(m \lg n) \) time and reporting queries in \( O(m \lg n + \text{occ}) \) time.

Proof (Sketch)

- Two binary searches on the SA in \( O(\lg n) \) 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.   \( i = \lfloor (\ell + r)/2 \rfloor \)
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
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 \([s, r]\)

Lemma: Running Time SeachSA

The SeachSA answers counting queries in \( O(m \lg n) \) time and reporting queries in \( O(m \lg n + \text{occ}) \) time.

Proof (Sketch)

- two binary searches on the \( SA \) in \( O(\lg n) \) 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 = \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`
3. While `ℓ < r` do
   4. `i = \lceil ℓ + r/2 \rceil`
   5. If `P = T[SA[i]..SA[i] + m]` then `ℓ = i`
   6. Else `r = i - 1`
5. Return `[s, r]`

Lemma: Running Time `SeachSA`

The `SeachSA` answers counting queries in $O(m \lg n)$ time and reporting queries in $O(m \lg n + \text{occ})$ time.

Proof (Sketch)

- Two binary searches on the `SA` in $O(\lg n)$ 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. \( \ell = 1, r = n + 1 \)
2. while \( \ell < r \) do
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
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 \([s, r]\)

Lemma: Running Time SeachSA

The SeachSA answers counting queries in \( O(m \lg n) \) time and reporting queries in \( O(m \lg n + \text{occ}) \) 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
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.     \( i = \lfloor (\ell + r)/2 \rfloor \)
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
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 \([s, r]\)

Lemma: Running Time SeachSA

The SeachSA answers counting queries in \( O(m \log n) \) time and reporting queries in \( O(m \log n + \text{occ}) \) time.

Proof (Sketch)

- two binary searches on the SA in \( O(\log n) \) 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. $\ell = 1, r = n + 1$
2. while $\ell < r$ do
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
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 $[s, r]$

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
Pattern Matching with the Suffix Array (2/2)

Function SearchSA(T, SA[1..n], P[1..m]):

1. \( \ell = 1, r = n + 1 \)
2. While \( \ell < r \) do
   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
   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 \([s, r] \)

Lemma: Running Time SearchSA

The SearchSA answers counting queries in \( O(m \lg n) \) time and reporting queries in \( O(m \lg n + \text{occ}) \) 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 \( O(\text{occ}) \) additional time
Preview: Improving Running Time with LCP-Array

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

- now: how to compute the suffix array directly \textbf{without the suffix tree}
Timeline Sequential Suffix Sorting

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

Prefix Doubling
- 1990 [MM] original
- 1999 [LS] qsort
- 2000
- 2002
- 2003
- 2004
- 2005
- 2006
- 2007
- 2008
- 2009
- 2011
- 2016
- 2017
- 2021

Induced Copying
- 1999 [Sew] 1/2 copy
- 2000 [BW] BWT
- 2001
- 2002 [MF] deep-shallow
- 2003 [SS] bpr
- 2004 [Bai] GSACA
- 2005
- 2006 [MP] cache aware
- 2007
- 2008
- 2009
- 2011
- 2016
- 2017
- 2021

Recursion
- 2000 [IT] A/B copy
- 2002 [BK] diffcover
- 2003 [KA] L/S split
- 2004 [MAN] chains
- 2005
- 2006 [MP] cache aware
- 2007
- 2008 [MP] cache aware
- 2009
- 2011
- 2016
- 2017
- 2021

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

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

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

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>α</td>
<td></td>
<td>β</td>
</tr>
</tbody>
</table>

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**

---

### Example

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$</td>
</tr>
</tbody>
</table>
```

---

**Scan text from right to left**

*do not store types explicitly*

**/info_sign**

Initially, we are only interested in **LMS-suffixes**

---

Florian Kurpicz | Text Indexing | 02 Suffix Trees and Arrays
Definition: Type $L/S$ Suffixes

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

- $T[i] < T[i+1]$ or $i = n$ ⇒ $T[i..n]$ has **type $S$**
- $T[i] > T[i+1]$ ⇒ $T[i..n]$ 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.
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: 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: 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.

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

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>$</td>
</tr>
</tbody>
</table>

$L \quad L \quad S$

\[
\begin{array}{c}
\text{c} \\
\text{b} \\
\text{a} \\
\$ \\
\end{array}
\]
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: Type $L/S$ Suffixes

Given a text $T$ of length $n$ and $i \in [1..n]$, then
- $T[i] < T[i + 1]$ or $i = n \implies T[i..n]$ has **type $S$**
- $T[i] > T[i + 1] \implies T[i..n]$ has **type $L$**
- $T[i] = T[i + 1] \implies T[i..n]$ has $T[i + 1..n]$’s type.
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.

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

\[c\]  \[b\]  \[a\]  \[$\]

\( S \ L \ L \ L \ S \)
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 \( i = n \) ⇒ \( T[i..n] \) has type \( S \)
- \( T[i] > T[i+1] \) ⇒ \( T[i..n] \) has type \( L \)
- \( T[i] = T[i+1] \) ⇒ \( T[i..n] \) 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 $S^*$
### 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 $S^*$
**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 $i = n$ ⇒ $T[i..n]$ has **type S**
- $T[i] > T[i + 1]$ ⇒ $T[i..n]$ has **type L**
- $T[i] = T[i + 1]$ ⇒ $T[i..n]$ 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 $S^*$

- scan text from right to left
- do not store types explicitly initially, we are only interested in LMS-suffixes
Partition suffix array based text’s histogram

Lemma: Order of $L/S$ suffixes

Given a text $T$ of length $n$, a type $L$ suffixes $T[i..n]$ and a type $S$ suffixes $T[j..n]$ with $\alpha = T[i] = T[i+1] = \ldots$, then $T[i..n] < T[j..n]$.

Proof (Sketch)

$T[i..n]$ has type $L$ $T[i..n] = \alpha \alpha \ldots \alpha$ $\ell \geq 0$ times $\beta \ldots$ with $\beta < \alpha$ 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: Classification (2/2)

- partition suffix array based text's histogram

<table>
<thead>
<tr>
<th>$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>...</th>
<th>z</th>
</tr>
</thead>
</table>

Lemma: Order of $L/S$ Suffixes

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

Proof (Sketch)

$T[i..n]$ has type $L$ $T[i..n] = \alpha \alpha ... \alpha$ with $\beta < \alpha$ 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: Classification (2/2)

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

<table>
<thead>
<tr>
<th>SA</th>
<th>$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>...</th>
<th>z</th>
</tr>
</thead>
</table>
Suffix Array Induced Sorting: Classification (2/2)

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

![Suffix Array](image)

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 $\alpha = T[i] = T[i+1]$, then $T[i..n] < T[j..n]$.

Proof (Sketch)

- $T[i..n]$ has type L $T[i..n] = \alpha \beta \gamma \ldots \gamma \ell \geq 0$ times $\beta \gamma \ldots \gamma$

  - with $\beta < \alpha$

- $T[j..n]$ has type S $T[j..n] = \alpha \beta \gamma \ldots \gamma \ell' \geq 0$ times $\gamma \beta \gamma \ldots \beta$

  - 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 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] = \underbrace{\alpha \alpha \ldots \alpha}_{\ell \geq 0} \beta \ldots \$\$
  - with $\beta < \alpha$

- $T[\ldots j..n]$ has type $S$
  - $T[\ldots j..n] = \underbrace{\alpha \alpha \ldots \alpha}_{\ell' \geq 0} \gamma \ldots$
  - with $\alpha < \gamma$

If $\ell < \ell'$ then $\beta < \alpha$ 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

<table>
<thead>
<tr>
<th>SA</th>
<th>$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>...</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>S</td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>S</td>
<td>L</td>
</tr>
</tbody>
</table>

**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 $\alpha = T[i] = T[j]$, then

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

**Proof (Sketch)**

- $T[i..n]$ has type $L$
  - $T[i..n] = \alpha\alpha\ldots\alpha\beta\ldots$\begin{array}{c}$\ell\geq0$ times\end{array}$
  - with $\beta < \alpha$

- $T[j..n]$ has type $S$
  - $T[j..n] = \alpha\alpha\ldots\alpha\gamma\ldots$\begin{array}{c}$\ell'\geq0$ times\end{array}$
  - with $\alpha < \gamma$
## 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 $L$
  - $T[i..n] = \alpha \alpha \ldots \alpha \beta \ldots \$ \quad \ell \geq 0$ times
  - with $\beta < \alpha$
- $T[j..n]$ has type $S$
  - $T[j..n] = \alpha \alpha \ldots \alpha \gamma \ldots \$ \quad \ell' \geq 0$ times
  - with $\alpha < \gamma$
- if $\ell < \ell'$ then $\alpha < \gamma$ and $T[i..n] < T[j..n]$

### Partition Suffix Array Based Text's Histogram

- use types of suffixes to partition suffix array

<table>
<thead>
<tr>
<th>$S$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$\ldots$</th>
<th>$z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>$S$</td>
<td>$L$</td>
<td>$S$</td>
<td>$L$</td>
<td>$S$</td>
</tr>
</tbody>
</table>

$T[i..n]$ has type $L$
Suffix Array Induced Sorting: Classification (2/2)

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

<table>
<thead>
<tr>
<th>SA</th>
<th>$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>...</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>S</td>
</tr>
</tbody>
</table>

**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 $L$
  - $T[i..n] = \alpha \alpha \ldots \alpha \beta \ldots$ 
  - with $\beta < \alpha$

- $T[j..n]$ has type $S$
  - $T[j..n] = \alpha \alpha \ldots \gamma \ldots$
  - 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

<table>
<thead>
<tr>
<th>$</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>...</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>S</td>
<td>L</td>
<td>S</td>
<td>L</td>
<td>S</td>
</tr>
</tbody>
</table>

**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] = \underbrace{\alpha \alpha \ldots \alpha}_\ell \beta \ldots \$ for $\ell \geq 0$ times
  - with $\beta < \alpha$
- $T[j..n]$ has type $S$
  - $T[j..n] = \underbrace{\alpha \alpha \ldots \alpha}_\ell' \gamma \ldots \$ for $\ell' \geq 0$ times
  - 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 $T[i..n] < T[j..n]$
Lemma: Inducing

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

$T[i..n] < T[j..n]$
Lemma: Inducing

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

$T[i..n] < T[j..n]$
Lemma: Inducing

If \( T[i+1..n] < T[j+1..n] \) and \( T[i] = T[j] \) then

\[ T[i..n] < T[j..n] \]
Lemma: Inducing

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

$T[i..n] < T[j..n]$
Lemma: Inducing

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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$</td>
</tr>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>
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
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

```
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>S*</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
**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
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

\[
\begin{array}{cccccccccccc}
  & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\
  S & L & S^* & S & L & S^* & S & L & S^* & L & L & L & S^* \\
  13 & 9 & 6 & 3 & & & & & & & & & \\
\end{array}
\]


**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

---

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$L$</td>
<td>$S^*$</td>
<td>$S$</td>
<td>$L$</td>
<td>$S^*$</td>
<td>$S$</td>
<td>$L$</td>
<td>$S^*$</td>
<td>$L$</td>
<td>$L$</td>
<td>$L$</td>
<td>$S^*$</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$$$</td>
</tr>
</tbody>
</table>
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
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
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

### 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

## Table

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>$$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>L</td>
<td>S*</td>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>S*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>13</th>
<th>12</th>
<th>9</th>
<th>6</th>
<th>3</th>
<th>11</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>13</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>8</td>
</tr>
</tbody>
</table>
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
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)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>(S)</td>
<td>(L)</td>
<td>(S^*)</td>
<td>(S)</td>
<td>(L)</td>
<td>(S^*)</td>
<td>(S)</td>
<td>(L)</td>
<td>(S^*)</td>
<td>(L)</td>
<td>(L)</td>
<td>(L)</td>
<td>(S^*)</td>
<td></td>
</tr>
</tbody>
</table>
| \(SA\) | 13 | 12 | 9 | 6 | 3 | 11 | 8 | 5 |}

---

**Figure:** Induction process in SAIS algorithm.
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
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

### Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>S*</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>12</td>
<td></td>
<td></td>
<td>9</td>
<td>6</td>
<td>3</td>
<td></td>
<td>11</td>
<td></td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

\(T\) sequence: \(a b a b c a b c a b b a\)

\(SA\) sequence: \(13 12 9 6 3 11 8 5\)
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

---

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>$$</td>
<td></td>
</tr>
<tr>
<td>$L$</td>
<td>$S^*$</td>
<td>$S$</td>
<td>$L$</td>
<td>$S^*$</td>
<td>$S$</td>
<td>$L$</td>
<td>$S^*$</td>
<td>$L$</td>
<td>$L$</td>
<td>$L$</td>
<td>$S^*$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **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

---

All suffixes induced?

Now we only need to sort $S^*$ suffixes.
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

![Diagram](image-url)
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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$</td>
</tr>
<tr>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>S</td>
<td>S*</td>
<td>S</td>
<td>L</td>
<td>S*</td>
<td>L</td>
<td>L</td>
<td>S*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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

![Diagram showing SAIS process with suffix array and buckets]
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

<table>
<thead>
<tr>
<th>$S$</th>
<th>$L$</th>
<th>$S^*$</th>
<th>$S$</th>
<th>$L$</th>
<th>$S^*$</th>
<th>$L$</th>
<th>$L$</th>
<th>$S^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$1$</th>
<th>$2$</th>
<th>$3$</th>
<th>$4$</th>
<th>$5$</th>
<th>$6$</th>
<th>$7$</th>
<th>$8$</th>
<th>$9$</th>
<th>$10$</th>
<th>$11$</th>
<th>$12$</th>
<th>$13$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$$</td>
</tr>
</tbody>
</table>
## 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
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
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
### 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
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
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
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
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 Diagram]
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
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 \( S^* \) suffixes
Suffix Array Induced Sorting: LMS-Substrings (1/2)

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

**Definition: LMS-Prefix**

Let $i < j$ or $i = j = n$ be text positions, such that no $k \in (i, j)$ with $T[k..n]$ is LMS, then we call $T[i..j]$ an **LMS-prefix**.
Suffix Array Induced Sorting: LMS-Substrings (1/2)

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

**Definition: LMS-Prefix**

Let $i < j$ or $i = j = n$ be text positions, such that $\not\exists k \in (i, j)$ 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.
how to sort $S^*$ suffixes?
slightly adopt algorithm

**Definition: LMS-Prefix**

Let $i < j$ or $i = j = n$ be text positions, such that $\not\exists k \in (i, j)$ with $T[k..n]$ is LMS, then we call $T[i..j]$ an 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
Lemma: Inducing LMS-Prefixes

The algorithm sorts all LMS-Prefixes correctly.
Lemma: Inducing LMS-Prefixes

The algorithm sorts all LMS-Prefixes correctly

Proof (Sketch)

- initially: only $T[n..n]$ sorted correctly
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
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
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
Lemma: Running Time Computation $T'$

Computing $T'$ requires $O(n)$ time
Lemma: Running Time Computation $T'$

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

Proof (Sketch)

- find LMS-substrings in $O(1)$ time → save $S$-buckets
- scan each LMS-substring twice
- each character is in at most two LMS-substrings
Lemma: Running Time Computation $T'$
Computing $T'$ requires $O(n)$ time

Proof (Sketch)
- find LMS-substrings in $O(1)$ time (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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$$$</td>
</tr>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$$$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>c</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>$$$</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>b</td>
<td>c</td>
<td>$$$</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$$$</td>
<td>a</td>
<td>$$$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

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 ① 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

<table>
<thead>
<tr>
<th>$T$</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td>$&gt;$</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>$&gt;$</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
<td>$&gt;$</td>
</tr>
</tbody>
</table>

$T' = 0122$
Lemma: Running Time Computation $T'$

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

Proof (Sketch)

- find LMS-substrings in $O(1)$ time $\triangleright$ 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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>$</td>
</tr>
<tr>
<td>$SA$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

$T' = 0122$
Lemma: SAIS Time Complexity

Given a text of length $n$, SAIS computes the suffix array in $O(n)$ time using
**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 $S^*$ suffixes is at most $\lceil n/2 \rceil$
- $T(n) = T(\lfloor n/2 \rfloor) + O(n) = O(n)$
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 $S^*$ suffixes is at most $\lfloor n/2 \rfloor$
- $T(n) = T(\lfloor n/2 \rfloor) + O(n) = O(n)$

Space Requirements

- naive: $O(n \lg n)$ bits
- better: $n[\lg n] + 2\sigma[\lg n]$ bits
Conclusion and Outlook

This Lecture

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

Linear Time Construction

- ST
- SA
- LCP
Conclusion and Outlook

This Lecture

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

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

Linear Time Construction

ST  SA  LCP
**This Lecture**

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

- suffix trees require \( \approx 8–20 \) bytes per character
- suffix arrays require 5 bytes per character for up to \( \approx 1 \) TB text
- currently fastest implementation available at [https://github.com/IlyaGrebnov/libsais](https://github.com/IlyaGrebnov/libsais)

**Next Lecture**

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


Bibliography II


Bibliography III
