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

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 Diagram]

- $\lambda(\cdot) = bba$
  - $d(\cdot) = 4$

- $\lambda(\cdot) = abba$
  - $d(\cdot) = 4$
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
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
- 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
**Suffix Array and LCP-Array**

**Definition: Suffix Array [GBS92; MM93]**

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

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

**Definition: Longest Common Prefix Array**

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

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

---

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

Diagram: Pattern $P = \text{abc}$
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
- reporting in \( O(\text{occ}) \) additional time
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 without the suffix tree
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
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**
- $T[i] > T[i+1] \Rightarrow T[i..n]$ 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 $S^*$

- scan text from right to left
- do not store types explicitly initially, we are only interested in LMS-suffixes
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 $\alpha = T[i] = T[i]$, 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 \alpha \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 $T[i..n] < T[j..n]$
- 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

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

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
**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
how to sort \( S^* \) 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 \( \not\exists k \in (i, j) \) 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**

**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

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

19/22 2021-10-25 Florian Kurpicz | Text Indexing | 02 Suffix Trees and Arrays

Institute for Theoretical Informatics, Algorithm Engineering
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$</td>
<td>13</td>
<td>12</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$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$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$c$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$a$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$c$</td>
<td>$a$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$a$</td>
<td>$b$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$a$</td>
<td>$b$</td>
<td>$c$</td>
<td>$c$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</td>
<td>$a$</td>
<td>$b$</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

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: $n\lceil \lg n \rceil + 2\sigma \lceil \lg n \rceil$ bits
Conclusion and Outlook

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

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
