Theory I
Algorithm Design and Analysis

(12 - Text search: suffix trees)

Prof. Dr. Th. Ottmann
Text search

Different scenarios:

**Dynamic texts**
- Text editors
- Symbol manipulators

**Static texts**
- Literature databases
- Library systems
- Gene databases
- World Wide Web
Properties of suffix trees

Search index
for a text $\sigma$ in order to search for patterns $\alpha$

Properties:

1. **Substring search** in time $O(|\alpha|)$.

2. **Queries to $\sigma$ itself**, e.g.:
   Longest substring in $\sigma$ occurring at least twice.

3. **Prefix search**: all positions in $\sigma$ with prefix $\alpha$. 
Properties of suffix trees

4. **Range search:** all positions in $\sigma$ in interval $[\alpha, \beta]$ with $\alpha \leq_{\text{lex}} \beta$, e.g.

   - `abracadabra`, `acacia` $\in$ `[abc, acc]`,
   - `abacus` $\notin$ `[abc, acc]`.

5. **Linear complexity:**
   Required space and time for construction in $O(|\sigma|)$
Tries

Trie: tree for representing keys.

alphabet $\Sigma$, set $S$ of keys, $S \subset \Sigma^*$

**Key** $\triangleq$ String $\in \Sigma^*$

**Edge** of a trie $T$: label with a single character from $\Sigma$

**Neighboring edges:** different characters
Example:
Tries

Each leaf represents a key:

corresponds to the labeling of the edges on the path from the root to the leaf

! Keys are not stored in nodes !
Suffix tries

Trie for all suffixes of a text

Example: $\sigma = \text{ababc}$

suffixes:

- $\text{ababc} = \text{suf}_1$
- $\text{babc} = \text{suf}_2$
- $\text{abc} = \text{suf}_3$
- $\text{bc} = \text{suf}_4$
- $\text{c} = \text{suf}_5$
Internal nodes of a suffix trie = substrings of $\sigma$.

Each proper substring of $\sigma$ is represented as an internal node.

Let $\sigma = a^n b^n : \exists \ n^2 + 2n + 1$ different substrings = internal nodes

$\implies$ Space requirement in $O(n^2)$. 
A suffix trie $T$ fulfills some of the required properties:

1. String matching for $\alpha$: follow the path with edge labels $\alpha$ in $T$ in time $O(|\alpha|)$.

   #leaves of the subtree $\cong$ #occurrences of $\alpha$

2. Longest repeated substring: internal node with the greatest depth which has at least two children.

3. Prefix search: all occurrences of strings with prefix $\alpha$ can be found in the subtree below the internal node corresponding to $\alpha$ in $T$. 
A suffix tree is created from a suffix trie by contraction of unary nodes:

suffix tree = contracted suffix trie
Internal representation of suffix trees

Child-sibling representation

Substring: pair of numbers \((i,j)\)

Example: \(\sigma = ababc\)
Internal representation of suffix trees

Example: $\sigma = \text{ababc}$

Node $v = (v.w, v.o, v.sn, v.br)$

Further pointers (suffix pointers) are added later
Properties of suffix trees

(S1) No suffix is prefix of another suffix; this holds if (last character of $\sigma$) = $\not\in \Sigma$

Search:

(T1) edge $\triangleq$ non-empty substring of $\sigma$.

(T2) neighboring edges: corresponding substrings start with different characters.
Properties of suffix trees

Size

(T3) each internal node (≠ root) has at least two children

(T4) leaf \( \triangleq \) (non-empty) suffix of \( \sigma \).

Let \( n = |\sigma| \neq 1 \)

\( \Rightarrow \) number of leaves \( n \)

\( \Rightarrow \) number of internal nodes \( \leq n - 1 \)

\( \Rightarrow \) Space requirement \( \in O(n) \)
Construction of suffix trees

Definition:

**Partial path**: path from the root to a node in $T$

**Path**: a partial path ending in a leaf

**Location** of a string $\alpha$: node at the end of the partial path labeled with $\alpha$ (if it exists).
Construction of suffix trees

**Extension** of a string $\alpha$: string with prefix $\alpha$

**Extended location** of a string $\alpha$: place of the shortest extension of $\alpha$, whose place is defined.

**Contracted location** of a string $\alpha$: place of the longest prefix of $\alpha$, whose place is defined.
Construction of suffix trees

Definitions:

$suf_i$: suffix of $\sigma$ starting at position $i$, e.g.
$suf_1 = \sigma$, $suf_n = \$. 

$head_i$: longest prefix of $suf_i$ which is also a prefix of $suf_j$ for a $j < i$.

Example: $\sigma = bbabaabc$ $\alpha = baa$ (has no location)
$suf_4 = baabc$
$head_4 = ba$
Construction of suffix trees

\( \sigma = \text{babaabc} \)
Naive suffix-tree construction

Begin with the empty tree $T_0$
Tree $T_{i+1}$ is created from $T_i$ by inserting suffix $suf_{i+1}$.

Algorithm suffix tree
Input: a text $\sigma$
Output: the suffix tree $T$ for $\sigma$

1. $n := |\sigma|; T_0 := \emptyset$
2. for $i := 0$ to $n - 1$ do
3. \hspace{1em} insert $suf_{i+1}$ in $T_i$, resulting in $T_{i+1}$
4. end for
Naive suffix-tree construction

In $T_i$ all suffixes $suf_j$ ($j < i$) already have a location.

$\Rightarrow head_i = \text{longest prefix of } suf_i \text{ whose extended location in } T_{i-1} \text{ exists.}$

Definition:

$tail_i := suf_i - head_i$, i.e. $suf_i = head_i tail_i$.

(S1)

$\Rightarrow tail_i \neq \varepsilon.$
Naive suffix-tree construction

Example: $\sigma = \text{ababc}$

$suf_3 = \text{abc}$
$head_3 = \text{ab}$
$tail_3 = \text{c}$

$T_0 = \begin{cases} \text{node} \\ \end{cases}$

$T_1 = \begin{cases} \text{node} \\ \text{ababc} \\ \end{cases}$

$T_2 = \begin{cases} \text{node} \\ \text{ababc} \\ \text{node} \\ \text{babc} \\ \end{cases}$
$$T_{i+1}$$ can be constructed from $$T_i$$ as follows:

1. Determine the extended location of $$head_{i+1}$$ in $$T_i$$ and split the last edge leading to this location into two new edges by inserting a new node.

2. Create a new leaf as location for $$suf_{i+1}$$
Naive suffix-tree construction

Example: $\sigma = ababc$

$T_2$

$T_3$

$head_3 = ab$

$tail_3 = c$
Naive suffix-tree construction

Algorithm suffix insertion
Input: tree $T_i$ and suffix $suf_{i+1}$
Output: tree $T_{i+1}$

1. $v :=$ root of $T_i$
2. $j := i$
3. repeat
4. find child $w$ of $v$ with $\sigma_{w.u} = \sigma_{j+1}$
5. $k := w.u - 1;$
6. while $k < w.o$ and $\sigma_{k+1} = \sigma_{j+1}$ do
    7. $k := k + 1; j := j + 1$
8. end while
Naive suffix-tree construction

9    \textbf{if} \ k = w.o \ \textbf{then} \ v := w \\
10   \textbf{until} \ k < w.o \ \textbf{or} \ w = \text{nil} \\
11   /* \ v \ is \ the \ contracted \ location \ of \ head_{i+1} */ \\
12   \text{insert} \ \text{the} \ \text{location} \ \text{of} \ head_{i+1} \ \text{and} \ tail_{i+1} \ \text{in} \ T_i \ \text{below} \ \nu \\

Running time for suffix insertion: O(  )
Total time for naive suffix-tree construction: O(  )
The algorithm $M$

(McCreight, 1976)

When the extended location of $\text{head}_{i+1}$ in $T_i$ has been found: creation of a new node and edge splitting in $O(1)$ time.

Idea: Extended location of $\text{head}_{i+1}$ is determined in constant amortized time in $T_i$. (Additional information is required!)
Theorem 1

Algorithm $M$ constructs a suffix tree for $\sigma$ with $|\sigma|$ leaves and at most $|\sigma| - 1$ internal nodes in time $O(|\sigma|)$.

Remark:
Ukkonen (1992) found an $O(n)$ on-line algorithm for the construction of suffix trees, i.e. after each step $i$, the resulting structure is a correct suffix tree for $t_1\ldots t_i$ (where $\sigma = t_1\ldots t_n$).
Suffix tree: application

Usage of suffix tree $T$:

1. Search for string $\alpha$: follow the path with edge labeling $\alpha$ in $T$ in time $O(|\alpha|)$. Leaves of the subtree $\triangleq$ occurrences of $\alpha$

2. Search for longest repeated substring:
   Find the location of a substring with the greatest weighted depth that is an internal node

3. Prefix search: All occurrences of strings with prefix $\alpha$ can be found in the subtree below the „location“ of $\alpha$ in $T$. 
Suffix tree: application

4 Range query for $[\alpha, \beta]$ :
Suffix tree: example

$T_0 = \square$

$suf_1 = bbabaabc$

$T_1 = \bigcirc$

$suf_2 = babaabc$

$head_2 = b$
Suffix tree: example

\[ T_2 = \]

\[ T_3 = \]

\[ \text{suf}_3 = \text{abaabc} \]
\[ \text{head}_3 = \varepsilon \]

\[ \text{suf}_4 = \text{baabc} \]
\[ \text{head}_4 = \text{ba} \]
**Suffix tree: example**

\[ T_4 = \]

- abaabc
- b
- a

**suf_5 = aabc**

**head_5 = a**

**location of head_4**
Suffix tree: example

\[ T_5 = \]

\begin{center}
  \begin{tikzpicture}
    \node [circle] (root) {a}
    child {node [circle] (abc) {abc}
      child {node [circle] (baabc) {baabc}
        child {node [circle] (head5) {head_5}}}
    }
    child {node [circle] (babaabc) {babaabc}
      child {node [circle] (a) {a}
        child {node [circle] (abc) {abc}}}
    }
  \end{tikzpicture}
\end{center}

location of \( \text{head}_5 \)

\[ \text{suf}_6 = \text{abc} \\
\text{head}_6 = \text{ab} \]
Suffix tree: example

$T_6 = \text{abc \ babaabbc}$

location of $\text{head}_6$

$suf_7 = \text{bc}$

$head_7 = \text{b}$
Suffix tree: example

\[ T_7 = \]

\[
\begin{align*}
abc & \quad \text{b} \quad \text{aabc} \quad \text{abc} \\
\quad \text{c} & \quad \text{aabc} \quad \text{baabc}
\end{align*}
\]

\[ \text{suf}_8 = c \]
Suffix tree: example

\[ T_8 = \]

```
  a
 /\  
|  b  |
| \   |
|  aabc|
|     |
|  c   |
|     |
| abc  |
|     |
|  b   |
|     |
| abc  |
|     |
| c    |
|     |
| c    |
```