Suffix trees

Ukkonen algorithm

What are suffix trees?

- A tree-like data structure for a large string (the text $T[1..n]$), which can be built in time $O(n)$
  - it is a compact representation of all suffixes of text T.
- It allows to find all occurrences of a pattern $P[1..m]$ in T in time $O(m + k)$ where k is the number of occurrences of P in T.

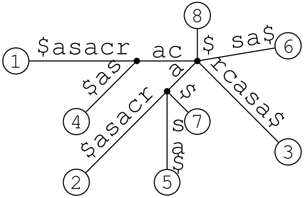
Remarks

1. The algorithm which builds the suffix tree of $T[1..n]$ in linear time $O(n)$ was discovered by Wiener in 1973.
   - Donald Knuth called it “the algorithm of 1973” – he thought the suffix tree can not be built in linear time.
2. Suffix trees have many other interesting applications.

The suffix tree of a string $S[1..n]$ is a tree with the following properties:

1. It has exactly n leaf nodes, labeled with numbers $1, 2, \ldots, n$.
2. Except for the root, every internal node has at lest two children.
3. Every edge is labeled with a nonempty substring of S.
4. Edges from same node to different children are labeled with substrings that start with different characters.
5. The string produced by concatenating the labels of the edges from the root node to a leaf node i is the suffix $S[i..n]$.

$S =$ `carcasa$` has length 8, thus 8 suffixes.
The suffix tree of S is



Remarks

1. Some strings have no suffix trees.
2. If the last character of S occurs only once in S, then S has a suffix tree.

   From now on, we will assume S satisfies this condition.

Auxiliary notions

Let $\mathcal{T}$ be the suffix tree of a string $S[1..n]$, and $\alpha = S[i..j]$ a substring of S.

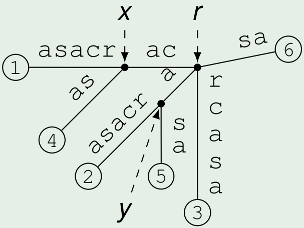
- The label $\mathcal{L}(x)$ of a node x of $\mathcal{T}$ is the string produced by concatenating the labels of edges from root to x.
- The position $pos_{\mathcal{T}}(\alpha)$ of α in $\mathcal{T}$ is defined as follows: Let x be the node of $\mathcal{T}$ such that $\mathcal{L}(x)$ is the shortest node label with prefix α. (Note: x can be foud in $|\alpha|$ steps)
  1. If $\mathcal{L}(x) = \alpha$, then $pos_{\mathcal{T}}(\alpha) := x$
  2. Otherwise, let y be the parent node of x in $\mathcal{T}$ and β the substring such that $\alpha = \mathcal{L}(y)\beta$. In this case, $pos_{\mathcal{T}}(\alpha)$ is the triple $\langle y, x, \beta\rangle$.
     - Intuition: The position of α în $\mathcal{T}$ is between nodes y and x of $\mathcal{T}$.

Auxiliary notions

Positions in a suffix tree

Example

String positions in the suffix tree of string $S = \texttt{carcasa}$

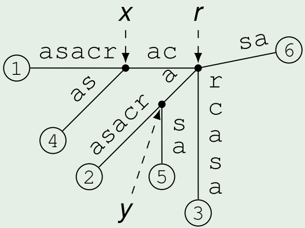


$$pos_T(\lambda) = r$$
$$pos_T(\texttt{c}) = \langle r, x, \texttt{c} \rangle$$
$$pos_T(\texttt{ca}) = x$$
$$pos_T(\texttt{car}) = \langle x, \textcircled{1}, \texttt{r} \rangle$$
$$pos_T(\texttt{carcasa}) = \textcircled{1}$$
$$pos_T(\texttt{arc}) = \langle y, \textcircled{2}, \texttt{rc} \rangle$$
$$pos_T(\texttt{sa}) = \textcircled{6}$$

The node depth $d_{\mathcal{T}}(\alpha)$ of substring α of S in the suffix tree $\mathcal{T}$ of S is:

1. if $pos_{\mathcal{T}}(\alpha)$ is a node y, then $d_{\mathcal{T}}(\alpha)$ is the number of nodes from root of $\mathcal{T}$ to y. The root and node y are counted as well.
2. $pos_{\mathcal{T}}(\alpha) = \langle y, x, \beta \rangle$ then $d_{\mathcal{T}}(\alpha)$ is the number of nodes from root of $\mathcal{T}$ to y, except y. The root is counted, but node y is not.

Example



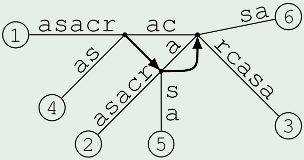
$d_{\mathcal{T}}(\texttt{ca}) = 1$
$d_{\mathcal{T}}(\texttt{carc}) = 2$
$d_{\mathcal{T}}(\texttt{carcasa}) = 2$

Suffix trees have a remarkable property:

> For every interior node x different from root, there is another interior node y such that $\mathcal{L}(y)$ is obtained from $\mathcal{L}(x)$ by dropping its first character.

y is called the suffix link of x, and is denoted by $suf(x)$.

Example (Suffix links in the suffix tree of `carcasa`)

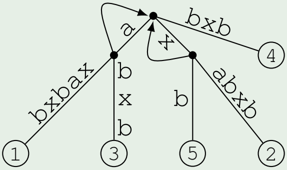


Suffix trees
A compact representation

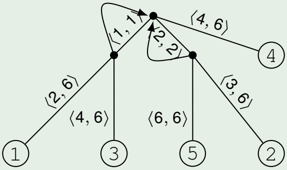
Main idea: Instead of labeling the edges with substrings $S[i..j]$, we can label them with pairs of integers $\langle i, j \rangle$

$\Rightarrow$ edge labels of variable size (substrings) are replaced by edge labels of constant size (pair of integer indices in S)

Example (Suffix tree for the string `axabxb`)



is replaced with



The suffix tree $\mathcal{T}$ of a string $S[1..n]$ has

- n leaf nodes
- except for the root, every internal node has at least 2 children
- the root node may have 1 child.

Therefore:

- $\mathcal{T}$ has at most n internal nodes.
- $\mathcal{T}$ has at most $2 \cdot n$ edges

$\Rightarrow$ the size of $\mathcal{T}$ is $O(n)$.

Fact: The suffix tree and suffix links of a text $S[1..n]$ can be constructed in time $O(n)$

1. Such an algorithm was first described by Wiener, in 1973.
2. A simpler linear-time algorithm was proposed by Ukkonen; it is described in Chapter 6 of the book

   Dan Gusfield, *Algorithms of Strings, trees, and sequences.* Cambridge University Press, 1997.

Generalized suffix trees

What are they?

Let $\mathcal{S} = \{S_1, \ldots, S_p\}$ a set of p non-empty strings.

- We assume w.l.o.g. that every string S_j ends with a specific character z_j which occurs nowhere else.

The generalized suffix tree of $\mathcal{S}$ is a tree with the following properties:

1. It has $|S_1| + \ldots + |S_p|$ leaves, with labels from the set $\{j{:}i \mid 1 \leq j \leq p, 1 \leq i \leq |S_j|\}$
2. All internal nodes, except the root, have ar least 2 children.
3. Every edge is labeled with a nonempty substring of strings from $\mathcal{S}$.
4. Edges from same node to different children are labeled with substrings that start with different characters.
5. $\mathcal{L}(j{:}i) = S_j[i..n_j]$ where $n_j = |S_j|$.

Like for suffix tree, we define a compact representation of generalized suffix trees:

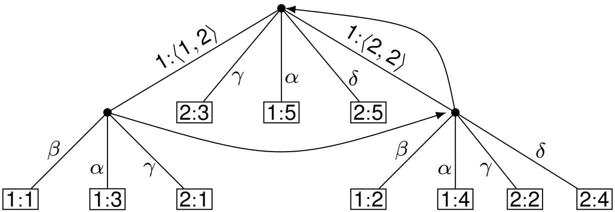
We replace every edge label $S_j[k..\ell]$ with the constant-size label $j{:}\langle k, \ell\rangle$

1. We build suffix tree $\mathcal{G}_1$ of S_1 with Ukkonen alg. in $O(|S_1|)$ time
   - we label edges with $1{:}\langle k, \ell \rangle$ instead of $\langle k, \ell \rangle$, and leaves with $1{:}i$ instead of i.
2. For $m := 2$ to p, we build the generalized suffix tree $\mathcal{G}_m$ of set of strings $\{S_1, \ldots, S_m\}$ as follows:
   - Traverse $\mathcal{G}_{m-1}$ from root, to find longest prefix $S_m[1..j]$ which has a position in $\mathcal{G}_{m-1}$.

     $S_m[1..j]$ is longest prefix of S_m which is prefix of a suffix of a string from $\{S_1, \ldots, S_{m-1}\}$
   - Start extending G_{m-1} from that position, until we produce $\mathcal{G}_m$

$\Rightarrow \mathcal{G}_p$ is a suffix tree of $\mathcal{S} = \{S_1, \ldots, S_p\}$, built in $O(n)$ time, where $n = |S_1| + \ldots + |S_p|$

The generalized suffix tree of $\mathcal{S} = \{\texttt{cocos}, \texttt{comod}\}$ is



where $\alpha = \langle 1, 5, 5\rangle$, $\beta = 1{:}\langle 3, 5\rangle$, $\gamma = 2{:}\langle 3, 5\rangle$, $\delta = 2{:}\langle 5, 5\rangle$.

Given text $S[1..n]$ and pattern $P[1..m]$, find all occurrences of P in S.

1. Construct the suffix tree $\mathcal{T}$ of S in time $O(n)$
2. Find $pos_P(\mathcal{T})$ in time $O(m)$. Suppose $pos_P(\mathcal{T})$ is y or $\langle x, y, \beta \rangle$.
3. Find all leaf nodes of $\mathcal{T}$ below node y.
   - Every occurrence of P in S is a prefix of a suffix $P[j..n]$ of S, where j is the label of such a leaf node.
   - If there are k occurrences of P in S, there are k such leaf nodes. These leaf nodes can be found in $O(k)$ time.

Properties of string matching with (generalized) suffix trees:

1. Finding all occurrences of $P[1..m]$ in a text $S[1..n]$ takes $O(n+m+k)$ time
   - If the suffix tree of S is precomputed, then finding all occurrences of P in S takes $O(m+k)$ time
   - This method is useful if we search often in the same text S (representation of a large database)
2. Finding all occurrences of $P[1..m]$ in all texts of a set $\mathcal{S} = \{S_1, \ldots, S_p\}$ takes $O(n+m+k)$ time where $n = |S_1| + \ldots + |S_p|$

Given two texts S_1 and S_2,

Find the longest substrings common to S_1 and S_2.

Answer:

1. Build the generalized suffix tree $\mathcal{G}$ of $\{S_1, S_2\}$ and mark its internal nodes that have leaf descendants for suffixes of both S_1 and S_2

   Can be done in time $O(n)$ where $n = |S_1| + |S_2|$
2. Traverse the internal nodes of $\mathcal{G}$, and compute the character depth of those which are marked.
   - Note: their character depth is the length of a common substring of S_1 and S_2

Overall computation time: $O(n)$