JASS 2008
Course 1 - Trees
Suffix Trees

Caroline Löbhard

Institut für Mathematik
TU München

St. Petersburg, 9.3. - 19.3. 2008

Contents

Some denotations for strings

Σ	-	fixed alphabet,
$x, y, ... \in \Sigma$	-	single characters,
$P, S, ..., \alpha, \sigma, \tau, ...\Sigma^*$	-	strings over Σ,
$\mathcal{T}, \mathcal{I}...$	-	trees
$u, v, ... \in V$	-	inner nodes of trees
$S[i, j]$	-	substring of S from position i to j
$S[i]$	-	single character at position i

Definition and a first example

Definition

Given a string S of length $|S| = m$ over a fixed alphabet Σ, the *suffix tree* $\mathcal{T}_S$ *of* S is a rooted directed tree with

- edges labelled with nonempty strings and
- exactly m leaves labelled with the integers from 1 to m,

such that

Definition and a first example

Definition

Given a string S of length $|S| = m$ over a fixed alphabet Σ, the *suffix tree* $\mathcal{T}_S$ *of* S is a rooted directed tree with

- edges labelled with nonempty strings and
- exactly m leaves labelled with the integers from 1 to m,

such that

- each internal node other than root has at least two children

Definition and a first example

Definition

Given a string S of length $|S| = m$ over a fixed alphabet Σ, the *suffix tree* $\mathcal{T}_S$ *of* S is a rooted directed tree with

- edges labelled with nonempty strings and
- exactly m leaves labelled with the integers from 1 to m,

such that

- each internal node other than root has at least two children
- no two edges out of one node have edge labels beginning with the same character and

Definition and a first example

Definition

Given a string S of length $|S| = m$ over a fixed alphabet Σ, the *suffix tree* $\mathcal{T}_S$ *of* S is a rooted directed tree with

- edges labelled with nonempty strings and
- exactly m leaves labelled with the integers from 1 to m,

such that

- each internal node other than root has at least two children
- no two edges out of one node have edge labels beginning with the same character and
- for any leaf i, the concatenation of the path-labels from root to leaf i is $S[i, m]$.

A few more notions walking trough trees

- The label of a path in $\mathcal{T}$ is the concatenation of the labels of edges passed when following the path.
- A path does not need to end in a node and it does not need to start at root.
- For a node u in a Tree $\mathcal{T}$ $path(\mathcal{T}, u)$ is the unique path in the tree from root to the node u
- string-depth$(u) = |path(\mathcal{T}, u)|$
- node-depth$(u) =$ number of nodes passed when walking from root to u.

Try to build the suffix tree for $S = banana$

Try to build the suffix tree for $S = banana$

Problem:

Let $S[k, |S|]$ be the suffix of a String S and let $i, j \in \{1, ..., |S| - 1\}$ be Indices such that $S[k, |S|] = S[i, j]$. Then the path $S[i, j]$ does not end in a leaf.

Try to build the suffix tree for $S = banana$

Problem:

Let $S[k, |S|]$ be the suffix of a String S and let $i, j \in \{1, ..., |S| - 1\}$ be Indices such that $S[k, |S|] = S[i, j]$. Then the path $S[i, j]$ does not end in a leaf.

Solution:

Instead of constructing the suffix tree for S we will construct the suffix tree for $S\$$, where $\$ \notin \Sigma$.

Solution of the substring problem

Theorem

Let $\mathcal{T}_P$ be the suffix tree of a string P and let S be another string.

Solution of the substring problem

Theorem

Let $\mathcal{T}_P$ be the suffix tree of a string P and let S be another string.

- *S matches a path in $\mathcal{T}_P$ from root $\Leftrightarrow$ S occurs in P*
- *S occurs in P exactly at the positions numbered with the labels of all leaves of the subtree below the point of the last match in $\mathcal{T}_P$.*

Solution of the substring problem

Theorem

Let $\mathcal{T}_P$ be the suffix tree of a string P and let S be another string.

- *S matches a path in $\mathcal{T}_P$ from root $\Leftrightarrow$ S occurs in P*
- *S occurs in P exactly at the positions numbered with the labels of all leaves of the subtree below the point of the last match in $\mathcal{T}_P$.*

Example

$P = banana\$$
$S = an$

Proof.

- S matches the beginning a path $\in \mathcal{T} \Leftrightarrow S$ is a prefix of all suffixes of P that end in the leaves below the end of that path.
- $\forall\, i \in \{1, ..., |S|\} : P[i, |S|] = path(\mathcal{T}, i)$
- Define M to be the set of all leaf-labels below the end of the path that matches S.
- S is prefix of the suffixes $\{P[i, m] \mid i \in M\}$ of P.
- This means: S occurs in P exactly at the positions M.

In particular, S does not occur in P if and only if it matches no path in $\mathcal{T}_P$ and therefore, $M = \emptyset$. □

suffix links

Theorem

Suppose that:
- $S \in \Sigma^*$ is a string over the alphabet Σ,
- $\mathcal{T}$ is the suffix tree for $S\$$,
- v is a node in the suffix tree
and that there are $x \in \Sigma, \alpha \in \Sigma^*$, such that

$$path(\mathcal{T}, v) = x\alpha$$

$\Rightarrow$ $\exists$ (exactly one) node u with

$$path(\mathcal{T}, u) = \alpha.$$

Proof.

- $\exists\ i, j \in \{1, ..., |S|\}$ with $i \neq j$ and $\sigma, \tau \in \Sigma^*$ such that

$$P[i, m] = x\alpha\sigma,\ P[j, m] = x\alpha\tau \text{ and } \sigma[1] \neq \tau[1] (*).$$

- $\mathcal{T}$ contains paths from root to a leaf for
  $P[i+1, m] = \alpha\sigma$ and
  $P[j+1, m] = \alpha\tau$
- $\overset{(*)}{\Rightarrow}$ We have a node u with $path(\mathcal{T}, u) = \alpha$.

$\square$

Theorem

$path(\mathcal{T}, v) = x\alpha$
$\Rightarrow\ \exists$ *(exactly one) node u with* $path(\mathcal{T}, u) = \alpha$.

V = set of internal nodes in $\mathcal{T}$, the Theorem leads to

Definition

Let $s : V \to V \cup \{root\}$ be the map with

$$\forall\ v \in V : path(\mathcal{T}, s(v)) = path(\mathcal{T}, v)[2, |path(\mathcal{T}, v)|]$$

s is called *suffix link*,
the pairs $(v, s(v))$ are called *suffix links*

Example

edge label compression

Theorem

If the edge labels in a suffix trees $\mathcal{T}$ are written explicitly on the edges, then it is not possible to give a linear-time algorithm that constructs that tree for any String over a fixed or arbitrary alphabet containing at least two characters.

$\rightsquigarrow$ Use a pair of indices (i, j) to represent the edge-labels in the way that the pair (i, j) stands for the substring $P[i, j]$ of P.
For example most of the work done in Ukkonen's algorithm, the explicit characters are not even used.

A historical overview of algorithms

1973 First linear-time algorithm for constructing suffix trees of strings over a fixed alphabet given by Weiner

1976 McCreight suggested a more space-economic version of Weiner's algorithm

1995 Ukkonen gave linear-time on-line algorithm with McCreight's time- and space-efficiency

1995 Delcher and Kosaraju published an algorithm that *seemed* to solve the still open problem of constructing suffix trees of strings P over an arbitrary alphabet in $O(|P|)$ time (time-analysis was incorrect)

1997 Farach closed the gap in giving an algorithm constructing suffix trees of strings P over an arbitrary alphabet in $O(|P|)$ time

Ukkonen's algorithm ...how to use suffix links

Definition

An *implicit suffix tree of String S* is the tree obtained from the suffix tree of $S\$$ by

- removing the \$-symbols from the edge-labels,
- afterwards removing all edges without label
- and at last removing all nodes with less than two children.

Given a string S over a fixed alphabet Σ let $\mathcal{I}_i$ be the *implicit suffix tree for the prefix* $S[1, i]$ *of* S.

To construct the true suffix tree from the implicit one let the algorithm continue with $S\$$ and afterwards give the right labels to the leaves.

High-level Description ...to be improved

Input: $S \in \Sigma^*$

Ukkonen's algorithm runs in

- $m = |S|$ *phases*
- In phase 1, $\mathcal{I}_1$ is constructed
- In phase $i+1$, tree $\mathcal{I}_{i+1}$ is constructed from $\mathcal{I}_i$
- Each phase $i+1$ is further divided into $i+1$ *extensions*, one for each of the $i+1$ suffixes of $P[1..i+1]$.
- In extension j of phase $i+1$, the suffix $P[1..j]$ is fit into the tree.

Procedurally, the algorithm is as follows:

Construct tree $\mathcal{I}_1$
for i from 1 to $m-1$ **do**
  for j from 1 to $i+1$ **do**
    Find the end of path with label $S[j,i]$
    if (a) $\exists$ path with label $S[j,i+1]$ in $\mathcal{I}_i$ **then**
      - nothing to do -
    else if (b) $P[j,i]$ ends at a leaf **then**
      Append $P[i+1]$ to the leaf-edge
    else if (c) $P[i,j]$ ends at $u \in V$ **then**
      Add edge labelled $P[i+1]$ and leaf labelled j to u
    else if (d) $P[i,j]$ ends in the middle of an edge-label **then**
      Add a node to the end-position
      Add an edge labelled $P[i+1]$ and a leaf labelled j to
    end if
  end for
end for

Using suffix links

First extension of each phase:

The path labelled $P[1, i]$ ends in leaf 1,
$\rightsquigarrow$ append $P[i+1]$ to the leaf-edge due to case **b** (constant time)

Extension $j \geq 2$ of phase $i+1$

Assume that extension $j-1$ of phase $i+1$ is just finished, that means:

- The implicate suffix tree $\mathcal{I}_i$ is constructed
- $\forall\ k \in \{1, ..., j-1\}$ $S[k, i+1]$ is already inserted
- Most recently, we inserted $S[j-1, i+1]$
- and now, we want to insert $S[j, i+1]$

Single extension algorithm

1. Take the largest $k \in \{j-1, ..., i\}$ and $v \in V \cup \{root\}$ with $path(\mathcal{I}, v) = S[j-1, k]$ walking up at most one edge.
2. If $v \in V$, traverse suffix link from v to $s(v)$
   If $v = root$, stay there
3. Walk down $S[k, i]$
4. Insert $S[i+1]$ due to the rules ⓐ, ⓑ, ⓒ
5. If a new internal node was created, create the corresponding suffix link

Example

Further improvement of worst case time bound can be achieved with

- skip&count trick: Since $S[k, i]$ has to be included in the current tree, just match the first character, and if the length of the corresponding edge-label is smaller than $i - k$, jump to the next node.
- edge label compression
- once case ⓐ, rest of phase case ⓐ, since in case ⓐ, the path labelled with $S[j.i]$ continues with $S[i+1]$ and so do the paths labelled $S[g, i]$ ($g \in \{j+1, ..., i+1\}$)
- once a leaf, always a leaf (just have a look at the algorithm: It never extends a leaf-edge beyond its current leaf)

High-level Description

Problems for integer alphabet

A lower time-bound for building the suffix tree of a String $S \in \Sigma$ is $\Omega(|S|\log(|\Sigma|))$ since it is at least as large as sorting characters.

The algorithms of Weiner, McCreight and Ukkonen achieve this time bound, but this is not worth anything for large alphabets.
So let's take a look at Farach's algorithm.

High-level Description

Problems for integer alphabet

A lower time-bound for building the suffix tree of a String $S \in \Sigma$ is $\Omega(|S| \log(|\Sigma|))$ since it is at least as large as sorting characters.

The algorithms of Weiner, McCreight and Ukkonen achieve this time bound, but this is not worth anything for large alphabets.
So let's take a look at Farach's algorithm.

- Build the suffix tree $\mathcal{T}_o$ of all suffixes beginning at odd positions in S
- Using $\mathcal{T}_o$, build the suffix tree of all suffixes beginning at even positions in S
- Merge the Trees to get the suffix tree $\mathcal{T}_S$ of S

Substring Problem

- look for information in DNA
- or in a database of DNA-Strings
- ...

Lowest Common Ancestor Problem

lca problem is equivalent to the Longest Common Prefix Problem, which is easily solved with the help of suffix trees.

thanks for listening!