

Biosequence Algorithms, Spring 2005

Lecture 6: Introduction to Suffix Trees

Pekka Kilpeläinen
 University of Kuopio
 Department of Computer Science

II: Suffix Trees and Their Applications (loppuosapuut sekä niiden sovelluksia)

Introduction to Suffix Trees

Suffix tree is an index structure that gives efficient solutions to a wide range of complex string problems

For example, the **substring problem**:

For a text S of length m , after $O(m)$ time preprocessing, given any string P either find an occurrence of P in S , or determine that one does not exist, in time $O(|P|)$

How to solve it?

Solving the Substring Problem

First idea: Build a keyword tree of all substrings of S

No: takes too much time ($\Omega(m^2)$)

Second idea: It is easy to find *prefixes* of strings in a keyword tree. *Each substring $S[i \dots j]$ is a prefix of the suffix $S[i \dots m]$ of S*

~> create a keyword tree of the m non-empty suffixes of S

The rest is just refinement ...

(yet rather complicated to get the construction time down to linear!)

Definition of a Suffix Tree

A **suffix tree** T for $S[1 \dots m]$ is a rooted tree with ...

- m leaves numbered $1, \dots, m$
- at least two children for each internal node (with the root as a possible exception)
- each edge labeled by a nonempty substring of S
- no two edges out of a node beginning with the same character

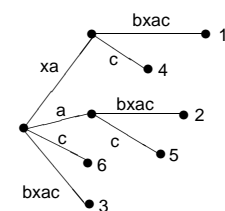
Again, $\mathcal{L}(v)$ denotes the **label of a node** v , i.e., the concatenation of edge labels on the path from the root to v

Key feature:

- $\mathcal{L}(i) = S[i \dots m]$ for each leaf $i = 1, \dots, m$ of T

Example of a Suffix Tree

The suffix tree for string $\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 \\ x & a & b & x & a & c \end{matrix}$:



Does a suffix tree always exist?

Ensuring the Existence of a Suffix Tree

Not necessarily:

If some suffix w appears as a prefix of some other one, the path labeled by w does not lead to a leaf

To avoid this, assume that S has a "termination character" $\$$ that occurs only at the end of S (or insert one, if needed)

~> no suffix can appear as a prefix of any other

~> suffixes label complete paths leading to the leaves

Applying Suffix Trees to Matching

Suffix trees can be used to solve exact matching:

1. Construct the suffix tree T for text $T[1 \dots m]$; (We'll discuss later how to do this in $O(m)$ time)
2. Match characters of P along a path from the root
 - (a) If P can be fully matched^(†), let z be the number of leaves below the path labeled by P . Each of these is a start of an occurrence of P , and they can be collected in time $O(z)$ (Exercise)
 - (b) If doesn't match completely, P doesn't occur in T

Total time: $O(m + n + z)$

(†) P may end at the middle of an edge

Naive Construction of Suffix Trees

Start with a root and a leaf numbered 1, connected by an edge labeled S .

Enter suffixes $S[2 \dots m]$, $S[3 \dots m]$, \dots , $S[m]$ into the tree as follows:

To insert $K_i = S[i \dots m]$, follow the path from the root matching characters of K_i until the first mismatch at character $K_i[j]$ (which is bound to happen)

(a) If the matching cannot continue from a node, denote that node by w

(b) Otherwise the mismatch occurs at the middle of an edge, which has to be split

BSA Lecture 6: Intro to suffix trees - p.9/17

Naive Construction of Suffix Trees (2)

If the mismatch occurs at the middle of an edge $e = (u, v)$, let the label of that edge be $a_1 \dots a_l$

If the mismatch occurred at character a_k , then create a new node w , and replace e by edges (u, w) and (w, v) labeled by $a_1 \dots a_{k-1}$ and $a_k \dots a_l$

Finally, in both cases (a) and (b), create a new leaf numbered i , and connect w to it by an edge labeled with $K_i[j \dots |K_i|]$

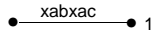
BSA Lecture 6: Intro to suffix trees - p.10/17

Example of the Naive Construction

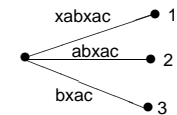
Consider building the suffix tree for the string S :

1 2 3 4 5 6
x a b x a c

Start:



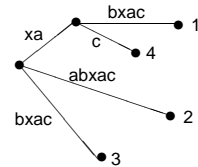
After inserting the second and the third suffix:



BSA Lecture 6: Intro to suffix trees - p.11/17

Example of the Naive Construction (2)

Entering $S[4 \dots 6] = xac$ causes the first edge to split:

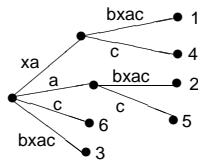


Same happens for the second edge when $S[5 \dots 6] = ac$ is entered

BSA Lecture 6: Intro to suffix trees - p.12/17

Example of the Naive Construction (3)

After entering suffixes $S[5 \dots 6] = ac$ and $S[6] = c$ the suffix tree is complete:



BSA Lecture 6: Intro to suffix trees - p.13/17

Complexity of the Naive Construction

Each suffix $S[i \dots m]$ is entered in the tree in time $\Theta(|S[i \dots m]|) \rightarrow$ total time is $\Theta(\sum_{i=2}^{m+1} i) = \Theta(m^2)$

Observations: Number of edges in a suffix tree T is at most $2m - 1 \rightsquigarrow$ the size of T is $O(m)$ (Exercise)

On the other hand, the *total length of edge labels* can be $\Theta(m^2)$ (Exercise)

As a simple example consider the suffix tree of $abc \dots xyz$, whose total length of edge labels is $\sum_{l=1}^{26} l = 26 \times 27/2$

For linear time we need a **compact representation of edge labels**

BSA Lecture 6: Intro to suffix trees - p.14/17

Compact Representation

Each edge is labeled by a non-empty substring $S[i \dots j]$

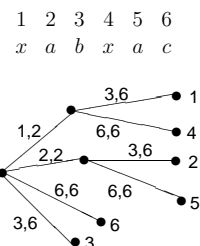
Compression: Represent label $S[i \dots j]$ by two indices i and j to the string S

\rightsquigarrow each edge takes only constant space, and thus $O(m)$ space suffices for the entire suffix tree of $S[1 \dots m]$

BSA Lecture 6: Intro to suffix trees - p.15/17

Example of Compact Representation

The suffix tree for a string with compacted edge labels



BSA Lecture 6: Intro to suffix trees - p.16/17

Short History of Suffix Trees

- 6 Weiner, 1973: the first linear-time construction
- 6 McCreight, 1976: a more space-efficient linear-time method
- 6 Ukkonen, 1995: A simpler linear-time construction, with all advantages of the previous, and more memory-efficient in practice

Next: Ukkonen's linear-time construction (which is rather complex)