Biosequence Algorithms, Spring 2005

Lecture 7: Linear-Time Construction of Suffix Trees

Pekka Kilpeläinen

University of Kuopio

Department of Computer Science

Ukkonen's Method

Ukkonen's algorithm constructs a suffix tree for $S[1 \dots m]$ in linear time

We begin with a high-level description of the method, and then describe how to implement it to run in linear time

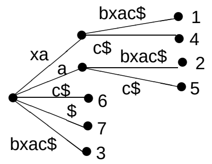
The method builds, as intermediate results, for each *prefix* $S[1 \dots 1], S[1 \dots 2], \dots, S[1 \dots m]$ an *implicit suffix tree*

The **implicit suffix tree** of a string is what results by applying suffix tree construction to the string *without an added end marker $*

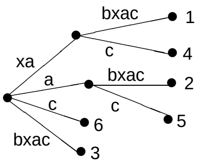
⇝ all suffixes are included, but not necessarily as labels of complete paths leading to the leaves

Example of Implicit Suffix Trees (1)

Suffix tree for $xabxac\$$:



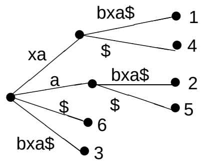
Implicit suffix tree for $xabxac$:



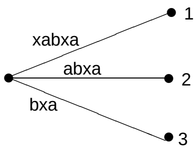
Obs 1: If the last char is unique, the implicit suffix tree is essentially the same as the (true) suffix tree (only without $'s)

Example of Implicit Suffix Trees (2)

Suffix tree for $xabxa\$$:



Implicit suffix tree for $xabxa$:



Obs 2: If the last char is not unique, some suffixes occur as labels of incomplete paths (not leading to a leaf, or even to an internal node)

Implicit Suffix Trees of Prefixes

Denote the implicit suffix tree of the prefix $S[1 \dots i]$ by $\mathcal{I}_i$

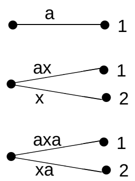
$\mathcal{I}_1$ is just a single edge labeled by $S[1]$ leading to leaf 1

Example:
Implicit suffix trees for the first three prefixes of $axabxb$:

$\mathcal{I}_1$:

$\mathcal{I}_2$:

$\mathcal{I}_3$:



String Paths of $\mathcal{I}_i$

So, $\mathcal{I}_i$ contains each suffix $S[1 \dots i], S[2 \dots i], \dots, S[i]$ of $S[1 \dots i]$ as a label of some path (possibly ending at the middle of an edge)

Let's call such labels of (partial) paths *(string) paths*

That is, a **string path** is

- a string that can be matched along the edges, starting from the root, or equivalently
- a prefix of any node label

Ukkonen's Algorithm on a High Level

Start with $\mathcal{T} := \mathcal{I}_1$

Then update $\mathcal{T}$ to trees $\mathcal{I}_2, \dots, \mathcal{I}_{m+1}$ in m **phases** (*vaihe*)

Let $S[m+1]$ be $ ⇝ the final value of $\mathcal{T}$ is a true suffix tree, which contains all suffixes of S (extended with $)

- **Phase** $i+1$ updates $\mathcal{T}$ from $\mathcal{I}_i$ (with all suffixes of $S[1 \dots i]$) to $\mathcal{I}_{i+1}$ (with all suffixes of $S[1 \dots i+1]$)

Each phase $i+1$ consists of **extensions** (*lisäysaskel*) $j = 1, \dots, i+1$;

Extension j ensures that suffix $S[j \dots i+1]$ is in $\mathcal{I}_{i+1}$

Extension j of Phase $i+1$

Phase $j+1$ starts with $\mathcal{T} = \mathcal{I}_i$

Q: How to ensure that suffix $S[j \dots i+1]$ is in the tree?

A: Extend path $S[j \dots i]$ of $\mathcal{T}$ by character $S[i+1]$ (if it isn't already there)

Q: How to extend the path?

Suffix Extension Rules

Extension rules for the three possible cases:

Rule 1 If path $S[j \ldots i]$ ends at a leaf, catenate $S[i+1]$ to its edge label

Rule 2 If path $S[j \ldots i]$ ends before a leaf, and doesn't continue by $S[i+1]$: Connect the end of the path to a *new leaf* j by an edge labeled by char $S[i+1]$. (If the path ended at the middle of an edge, split the edge and insert a new node as a parent of leaf j)

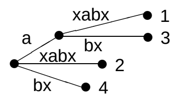
Rule 3 If the path could be continued by $S[i+1]$, do nothing. (Suffix $S[j \ldots i+1]$ is already in the tree)

Let's call an extension that applies Rule 3 **void**

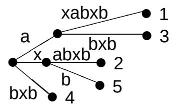
(and applications of Rules 1 and 2 **non-void**)

Example of Extensions

Consider phase 6 with string $S = axabxb$;
Now $\mathcal{T} = \mathcal{I}_5$ contains all suffixes of $S[1 \ldots 5] = axabx$:



Extensions 1–4: $S[1 \ldots 6] = axabxb, \ldots, S[4 \ldots 6] = bxb$ entered by Rule 1
Extension 5: $S[5 \ldots 6] = xb$ entered by Rule 2, creating leaf 5 and its parent
Extension 6: $S[6 \ldots 6] = b$ entered by Rule 3 $\rightsquigarrow \mathcal{T} = \mathcal{I}_6$:



Complexity of a Naive Implementation

Consider first a single phase $i+1$

Each of the extension rules can be applied in constant time

$\rightsquigarrow$ applying them once in each extension takes time $\theta(i)$

Locating the ends of the paths $S[1 \ldots i], \ldots S[i+1 \ldots i]$ by traversing them explicitly takes time $\Theta(\sum_{l=0}^{i} l) = \Theta(i^2)$

$\rightsquigarrow$ Total time for all phases $i = 2, \ldots, m+1$ is

$$\Theta(\sum_{i=2}^{m+1} i^2) = \Theta(m^3)$$

Q: How to improve this?

Reducing the Complexity

To get total time down to $O(m^2)$ we need to avoid or speed up path traversals

Spending even constant time for each extension requires time

$$\Theta(\sum_{i=2}^{m+1} i) = \Theta(m^2)$$

$\rightsquigarrow$ To get total time down to $O(m)$ we also need to avoid performing some extensions at all

Let's first consider speeding up the path traversals

Locating Ends of Paths

The extensions of phase $i+1$ need to locate the ends of all the $i+1$ suffixes of $S[1 \ldots i]$

How to do this efficiently?

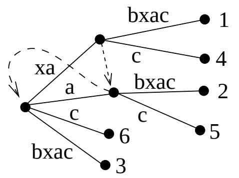
For each *internal node* v of $\mathcal{T}$ labeled by $x\alpha$, where $x \in \Sigma$ and $\alpha \in \Sigma^*$, define $s(v)$ to be the node labeled by α

We'll show that these exist, in a moment

Then a pointer from v to $s(v)$ is the **suffix link** of v

NB: If node v is labeled by a single char (x), then $\alpha = \epsilon$ and $s(v)$ is the root

Example of Suffix Links



What are suffix links good for?

Intuitive Motivation for Suffix Links

Extension j (of phase $i+1$) finds the end of the path $S[j \ldots i]$ in the tree (and extends it with char $S[i+1]$)

Extension $j+1$ similarly finds the end of the path $S[j+1 \ldots i]$

Assume that v is an internal node labeled by $S[j]\alpha$ on the path $S[j \ldots i]$. Then we can avoid traversing path α when locating the end of path $S[j+1 \ldots i]$, by starting from node $s(v)$

Q: Do suffix links always exist?

A: Yes, and each suffix link $(v, s(v))$ is easy to set:

Computation of Suffix Links

Observation: If an internal node v is created during extension j (of phase $i+1$), then extension $j+1$ will find out the node $s(v)$

Why?

Let v be labeled by $x\alpha$

Node v can only be created by extension rule 2

That is, v is inserted at the end of path $S[j \ldots i]$, which continued by some character $c \neq S[i+1]$

$\Rightarrow$ paths $x\alpha c$ and αc have been entered before phase $i+1$

$\Rightarrow$ in extension $j+1$, node $s(v)$ is either found or created at the end of path $\alpha = S[j+1 \ldots i]$

Speeding up Path Traversals

Consider extensions of phase $i + 1$

Extension 1 extends path $S[1 \dots i]$ with char $S[i + 1]$

That is easy: Path $S[1 \dots i]$ always ends at leaf 1, and is thus extended by Rule 1

We can perform extension 1 in constant time, if we maintain a pointer to the edge at the end of $S[1 \dots i]$

What about subsequent extensions $j + 1$ (for $j = 1, \dots, i$)?

Locating Subsequent Paths

Extension j has located the end of the path $S[j \dots i]$

Starting from there, walk up at most one node either

(a) to the root, or

(b) to a node v with a suffix link

In case (a), traverse path $S[j + 1 \dots i]$ explicitly down-wards from the root

Short-cutting Traversals

In case (b), let $x\alpha$ be the label of v
$\Rightarrow S[j \dots i] = x\alpha\beta$ for some $\beta \in \Sigma^*$

(Draw a picture of the paths!)

Then follow the suffix link of v, and continue by matching β down-wards from node $s(v)$ (which is now labeled by α)

Having found the end of path $\alpha\beta = S[j + 1 \dots i]$, apply extension rules to ensure that it extends with $S[i + 1]$

Finally, if a new internal node w was created in extension j, set its suffix link to point to the end node of path $S[j + 1 \dots i]$

Speeding up Explicit Traversals

(**Trick 1 (skip/count)** in Gusfield)

Each path $S[j \dots i]$, which is followed in extension j, is known to exist in the tree

$\rightsquigarrow$ the path can be followed by choosing the correct edges, instead of examining each character

Let $S[k]$ be the next char to be matched on path $S[j \dots i]$

Now an edge labeled by $S[p \dots q]$ can be traversed simply by checking that $S[p] = S[k]$, and skipping the next $q - p$ chars of $S[j \dots i]$

$\rightsquigarrow$ time to traverse a path is proportional to the *node-length* on the path (instead of its string-length)

Bounding the Time of Tree Traversals

Lemma 6.1.2 For any node v with a suffix link to $s(v)$,

$$\text{depth}(s(v)) \geq \text{depth}(v) - 1$$

That is, following a suffix link leads at most one level closer to the root

Proof. (Idea) The suffix links for any ancestor of v lead to distinct ancestors of $s(v)$ □

Now we can argue a linear time bound for any phase by considering how the **current node depth** can change

- (= the depth of the most recently visited node)

Linear Bound for any Single Phase

Theorem 6.1.1 Using suffix links and the skip/count trick, a single phase takes time $O(m)$

Proof

There are $i + 1 \leq m + 1$ extensions in phase $i + 1$

In any extension, other work except tree-traversals takes constant time only

How to bound the work for traversing the tree?

To find the end of the next path, an extension first moves at most one level up. Then a suffix link may be followed, which is followed by a down traversal to match the rest of the path

Bounding the Edge Traversals

The possible up movement and suffix link traversal *decrement* current node depth at most twice

$\rightsquigarrow$ the current node depth is decremented at most $2m$ times during the entire phase

On the other hand, the current node depth cannot exceed $m \rightsquigarrow$ it is incremented (by following downward edges) at most $3m$ times

$\rightsquigarrow$ total time of a phase is $O(m)$ □

NB: Since there are m phases, total time is $O(m^2)$

A few more tricks are needed to get total time linear

Final Improvements

Some extensions can be found unnecessary to compute explicitly

Obs 1: Rule 3 is a “show-stopper”:

- If path $S[j \dots i + 1]$ is already in the tree, so are paths $S[j + 1 \dots i + 1], \dots, S[i + 1]$, too

$\rightsquigarrow$ Phase $i + 1$ can be finished at the first extension j that applies Rule 3

Final Improvements (2)

Obs 2: A node created as a leaf remains a leaf thereafter

- because no extension rule adds children to a leaf
- If extension j created a leaf (numbered j), extension j of any later phase $i+1$ applies Rule 1 (appending the next char $S[i+1]$ to the edge label of j)

Explicit applications of Rule 1 can be eliminated as follows:

Use compressed edge representation
(i.e., indices p and q instead of substring $S[p \dots q]$), and

represent the end position of each terminal edge by a global value e for "the current end position"

Eliminating Extensions

Denote by j_i the *last non-void extension of phase* i
(that is, application of Rule 1 or 2)

Obs 1 $\rightsquigarrow$ extensions $1, \dots, j_i$ of phase i are non-void

$\rightsquigarrow$ leaves $1, \dots j_i$ have been created at the end of phase i

Obs 2 $\rightsquigarrow$ extensions $1, \dots j_i$ of any subsequent phase all apply Rule 1

$\Rightarrow j_{i+1} \geq j_i$

$\rightsquigarrow$ Execute only extensions $j_i+1, j_i+2, \dots$ explicitly in phase $i+1$

Single Phase Algorithm

Algorithm for phase $i+1$ with unnecessary extensions eliminated

1. Set $e := i+1$;
   (implements extensions $1, \dots j_i$ implicitly)
2. Compute extensions $j_i+1, \dots, j^*$ until $j^* > i+1$ or Rule 3 was applied in extension j^*;
3. Set $j_{i+1} := j^* - 1$; (for the next phase)

All these tricks together can be shown to lead to linear time

Analysis of the Tuned Implementation

Theorem 6.1.2 Ukkonen's algorithm builds the suffix tree for $S[1 \dots m]$ in time $O(m)$, when implemented using the above tricks

Proof

The extensions computed explicitly in any two phases i and $i+1$ are disjoint except for extension j^*, which may be computed anew in extension $i+1$

The second computation of extension j^* can be done in constant time by remembering the end of the path entered in the previous computation

Analysis of the Tuned Implementation (2)

Let $\bar{j} = 1, \dots, m+1$ denote the index of the current extension

Over all phases $2, \dots, m+1$ index $\bar{j}$ never decreases, but it can remain the same at the start of phases $3, \dots, m+1$
$\rightsquigarrow$ at most $2m$ extensions are computed explicitly

Similarly to the proof of Th. 6.1.1, the current node depth can be decremented at most $4m$ times, and thus the total length of all downward traversals is bounded by $5m$ □

A Final Touch

Finally, $\mathcal{I}_{m+1}$ can be converted to the *true* suffix tree of $S[1 \dots m]\$$ as follows:

All occurrences of the "current end position" marker e on edge labels can be replaced by $m+1$ (with a simple tree traversal, in time $O(m)$)

Remark: Ukkonen's method is "*on-line*", by processing S left-to-right and having a suffix tree ready for the scanned part