Exact Matching Part IV: Ukkonen’s Algorithm

See Gusfield, Chapter 5

Visualizations from http://brenden.github.io/ukkonen-animation/
Outline

- More applications of suffix trees and generalized suffix trees
- Ukkonen’s linear time suffix tree construction: finish proofs of suffix link properties
- How to implement Ukkonen’s algorithm so that it has $O(m)$ running time
A Suffix Tree for Two Strings

Example: $S_1 = xabxa$ and $S_2 = babxba$
A Generalized Suffix Tree for k strings

Generalized suffix trees satisfy suffix tree properties:

• Internal nodes have at least two children, edge labels are nonempty, etc.
• Leaves correspond to suffices of at least one string in the set
• Each suffix of each string corresponds to exactly one leaf
Find the longest common substring of two strings of lengths $m_1$ and $m_2$

(Knuth conjectured that there was no linear time algorithm to do this.)
Applications

Find the longest common substring of two strings of lengths $m_1$ and $m_2$

(Knuth conjectured that there was no linear time algorithm to do this.)

Solution with $O(m_1 + m_2)$ running time:
Build the generalized suffix tree, then find the node with greatest string depth that is on a path to a leaf of both trees
Find all occurrences of a set of patterns $P$ in a text $T$

Do this in time $O(n+m+k)$, where

- $n$ is the total length of the strings in the set $P$
- $k$ is the total number of occurrences of patterns in the text
- $m$ is the length of the text
Applications

Given a set $S$ of strings, find the longest match between a prefix of one string and a suffix of another.

Do this in time $O(m+k^2)$, where

- $m$ is the total length of the strings in the set $S$
- $k$ is the size of set $S$
Ukkonen’s Algorithm Examples

- abbba
Ukkonen’s Algorithm Examples

- aabababbabababaa
Ukkonen’s Algorithm Review

- Last time we saw how Ukkonen's algorithm adds suffix links while building the suffix tree
- The suffix links are created in step 3 of each extension and used in step 1 of each extension; let’s review how they are used
Ukkonen’s Algorithm Review

Build an Implicit Suffix Tree for $S[1..m]$ in $O(m)$ Time

Notation: Let $I_i$ be the implicit suffix tree for prefix $S[1,...,i]$

Construct $I_1$  

// Phase 1

For $i$ from 1 to $m-1$  

// Phase $i+1$: Build $I_{i+1}$ from $I_i$

For $j$ from 1 to $i+1$  

// Extension $j$ of Phase $i+1$

Ensure that a path labeled $S[j..i+1]$ is in the tree
Ukkonen’s Algorithm: Extension j of Phase i+1

Ensure that a path labeled $S[j,i+1]$ is in the tree

1. Find the end of path $S[j..i]$ (using suffix links)
2. Apply the appropriate suffix extension rule (plus some suffix link housekeeping)
3. Create a suffix link if needed; let the suffix link from node $w$ go to $s(w)$
Ukkonen’s Algorithm: Extension j of Phase i+1

Ensure that a path labeled $S[j,i+1]$ is in the tree

1. Find the end of path $S[j..i]$ (using suffix links)
Ukkonen’s Algorithm: Extension j of Phase i+1

Ensure that a path labeled $S[j,i+1]$ is in the tree

1. Find the end of path $S[j..i]$ (using suffix links)
   - If $j = 1$: use the 1-link (path $S[1..i]$ ends at leaf 1)
   - If $j > 1$:
     
     // In extension $j-1$, we located the end of path $S[j-1..i]$
     
     a. Find $w$, the first node at or above the end of path $S[j-1..i]$. Let $\gamma$ denote the string from $w$ to the end of $S[j-1..i]$
     b. If $w$ is not the root, $w \leftarrow s(w)$
     c. Follow the (unique) path from $w$ to $\gamma$
(Step 1a) $w$

(Step 1b) $S(w)$

(Step 1c)
Correctness: Suffix Link Properties

We showed the following:

Claim 1: At the end of phase $i+1$, every internal node of $I_i$ (except for the root) has a suffix link.
We showed the following:

Claim 1: At the end of phase $i+1$, every internal node of $I_i$ (except for the root) has a suffix link. Moreover, at the end of extension $j$, all nodes except that added in extension $j$ (if any) has a suffix link.
Claim 3: Let \((w, w')\) be any suffix link traversed during Ukkonen’s algorithm. At that moment, the node-depth of \(w\) is at most one greater than the node depth of \(w'\).
Correctness: Suffix Link Properties

Claim 3: Let \((w, w')\) be any suffix link traversed during Ukkonen’s algorithm. At that moment, the node-depth of \(w\) is at most one greater than the node depth of \(w'\).

Proof: Consider when \((w, w')\) is traversed.
Claim 3: Let \((w,w')\) be any suffix link traversed during Ukkonen’s algorithm. At that moment, the node-depth of \(w\) is at most one greater than the node depth of \(w'\).

Proof: Consider when \((w,w')\) is traversed.

• Suppose that \((w,w')\) is traversed in extension \(j\) of some phase. At this moment, by Claim 1, the only internal node (if any) that may not have a suffix link is that just inserted in extension \(j-1\) (if \(j > 1\)), and must be a descendant of \(w\). So all ancestors of \(w\) except the root have suffix links.
Claim 3: Let \((w,w')\) be any suffix link traversed during Ukkonen’s algorithm. At that moment, the node-depth of \(w\) is at most one greater than the node depth of \(w'\).

Proof: Consider when \((w,w')\) is traversed.
Correctness: Suffix Link Properties

Claim 3: Let \((w,w')\) be any suffix link traversed during Ukkonen’s algorithm. At that moment, the node-depth of \(w\) is at most one greater than the node depth of \(w'\).

Proof: Consider when \((w,w')\) is traversed.

• First, the suffix link from any ancestor \(u\) of \(w\) goes to an ancestor \(u'\) of \(w'\); this follows since the label of \(u\) is a prefix of \(w\), and also the label of \(u'\) is a prefix of \(w'\).

• Second, links from distinct ancestors of \(w\) must go to distinct ancestors of \(w'\), because edge labels have length at least 1.

• Thus there is a 1-1 correspondence between the internal nodes on the path to \(w\) except for the root, and those on the path to \(w'\). The claim follows.
Efficiency

We'll use a slick representation of partially constructed suffix trees

Example: aababababbabababaa, end of phase 3
Efficiency

We'll use a slick representation of partially constructed suffix trees
Example: aabababbabababaa, end of phase 3
Efficiency

We'll use a slick representation of partially constructed suffix trees

- Replace edge label $S[a..b]$ by $\langle a, b \rangle$
- Then use $#$ in place of $b$ when $b$ is the current phase number
Efficiency

Claim: With the slick representation, each phase takes $O(m)$ time, resulting in $O(m^2)$ time overall.

Why: Recall that Phase $i+1$ has $i+1$ extensions, each with three steps:

1. Navigation: find the end of path $S[j..i]$
2. Suffix extension: three extension rules
3. Suffix link creation

Steps 2 and 3 take $O(1)$ time. Let's focus on Step 1
Ukkonen’s Algorithm: Extension j of Phase i+1

Time for Step 1?

1. Find the end of path S[j..i]
   • If j = 1: use the 1-link (path S[1..i] ends at leaf 1)  O(1)
Ukkonen’s Algorithm: Extension j of Phase i+1

Time for Step 1?

1. Find the end of path $S[j..i]$
   • If $j = 1$: use the 1-link (path $S[1..i]$ ends at leaf 1) $O(1)$
   • If $j > 1$:
     
     // In extension $j-1$, we located the end of path $S[j-1..i]$
     
     a. Find $w$, the first node at or above the end of path $S[j-1..i]$. Let $\gamma$ denote the string from $w$ to the end of $S[j-1..i]$
Ukkonen’s Algorithm: Extension j of Phase i+1

Time for Step 1?

1. Find the end of path $S[j..i]$
   - If $j = 1$: use the 1-link (path $S[1..i]$ ends at leaf 1) $O(1)$
   - If $j > 1$:
     // In extension j-1, we located the end of path $S[j-1..i]$
     a. Find $w$, the first node at or above the end of path $S[j-1..i]$. Let $\gamma$ denote the string from $w$ to the end of $S[j-1..i]$ $O(1)$
     b. If $w$ is not the root, $w \leftarrow s(w)$ $O(1)$
     c. Follow the (unique) path from $w$ to $\gamma$ $O(k)$ time, where $k$ is the number of nodes from $v$ to the end of gamma, if we use "node hopping"
Ukkonen’s Algorithm: Extension j of Phase i+1

Node Hopping:

- Rather than comparing $\gamma$ to edge labels character by character, use edge label lengths to move from node to node until getting to the edge inside which $\gamma$ ends
- Edge label lengths can be computed from the new edge representation $\langle a, b \rangle$
Ukkonen’s Algorithm: Extension j of Phase i+1

Node Hopping:

– Rather than comparing $\gamma$ to edge labels character by character, use edge label lengths to move from node to node until getting to the edge inside which $\gamma$ ends

– Edge label lengths can be computed from the new edge representation $\langle a, b \rangle$
Efficiency

Getting the time down to $O(m)$
Efficiency

*Getting the time down to $O(m)$*

Example: aabababbababaa, phase 3

How does the tree change in phase 4? Phase 5, 6, 7?
Efficiency

*Getting the time down to $O(m)$*

Example: aabababababaa, phase 3

How does the tree change in phase 4? Phase 5, 6, 7?
There are no changes!
Efficiency

Getting the time down to $O(m)$

Example: $aababababababa$, phase 3

The tree never changes when Rules 1 and 3 are applied!
Efficiency

*Getting the time down to $O(m)$*

Example: aabababbababaa, phase 3

Moreover, Rule 2 (adding a new internal node) is only applied $O(m)$ times, *over all phases and extensions*, because the tree can have at most $m$ leaves.
Efficiency

Minor implementation changes ensure that no time is wasted on extensions that do nothing (Rules 1 and 3), and time is only spent on Rule 2 extensions. Each Rule 2 extension takes $O(1)$ time and there are at most $m$ of them.
Converting an Implicit Suffix Tree to a Suffix Tree

- Extend $S[1..m]$ by adding $\$\$ and repeat Ukkonen's algorithm for one last iteration
Summary

• Ukkonen's algorithm builds suffix trees for strings of length $m$ in $O(m)$ time

• Suffix trees, and generalized suffix trees, provide optimal solutions for many exact matching problems

• Additional extensions enable them to be used even for inexact matching problems

• Further refinements, e.g., suffix arrays, have resulted in implementations that are fast and space efficient in practice on very large strings