Exact String Matching Part II

Suffix Trees

See Gusfield, Chapter 5
Outline for Today

• What are suffix trees
• Application to exact matching
• Building a suffix tree in linear time, part I: Ukkonen’s algorithm
Suffix Tree for xabxac
A suffix tree for string $S[1..m]$ is a rooted directed tree with the following properties:

- Each internal node, other than the root, has at least two children
- Each edge is labeled with a nonempty substring of $S$
- No two edges out of a node can have edge labels beginning with the same character
- The tree has exactly $m$ leaves numbered 1 to $m$
- For each $i$, the concatenation of the edge labels on the path from the root to leaf $i$ is $S[i..m]$
More examples

Exercise: build suffix trees for

• ababx
• xabxa

• Try adding suffixes in decreasing order of length
More examples

Exercise: build suffix trees for

• ababx
• xabxa

• Problem: xabxa does not have a suffix tree, since the tree would both need to have a leaf for xa AND extend the path labeled xa to xabxa, and there's no way to do both.

• More generally, when one suffix is a prefix of another suffix, we're in trouble. We can easily avoid this by adding a special $ symbol at the end of a string
Suffix Tree Notation

- **Edge label**: the string labeling an edge
- **Path**: starts at root and follows edges; may end in the middle of an edge label
- **Path label**: concatenation, in order, of strings from root to the end of the path
- **Node label**: label of the path from the root to the node
- **Node depth**: the number of nodes on the path to the node
- **Node string-depth**: the number of characters in the node’s label
Using Suffix Trees for Exact Matching
Using Suffix Trees for Exact Matching

Given a suffix tree for a text $T[1,...,m]$, we can find all occurrences of a pattern $P[1,...,n]$ in $T$ in $O(n+k)$ time, where $k$ is the number of occurrences. How?
Using Suffix Trees for Exact Matching

Given a suffix tree for a text \( T[1,\ldots,m] \), we can find all occurrences of a pattern \( P[1,\ldots,n] \) in \( T \) in \( O(n+k) \) time, where \( k \) is the number of occurrences. How?

1. Traverse a path from the root as long as it matches \( P \)

Two cases (2 and 2'): Not all characters of \( P \) can be matched, or all characters of \( P \) are matched
Using Suffix Trees for Exact Matching

Given a suffix tree for a text $T[1,\ldots,m]$, we can find all occurrences of a pattern $P[1,\ldots,n]$ in $T$ in $O(n+k)$ time, where $k$ is the number of occurrences. How?

1. Traverse a path from the root as long as it matches $P$
2. If not all characters of $P$ can be matched, there is no occurrence of $P$ in $T$
Using Suffix Trees for Exact Matching

Given a suffix tree for a text $T[1,...,m]$, we can find all occurrences of a pattern $P[1,...,n]$ in $T$ in $O(n+k)$ time, where $k$ is the number of occurrences. How?

1. Traverse a path from the root as long as it matches $P$
2’ All characters of $P$ are matched. Then traverse the subtree from the point where the last character was matched and return the leaf labels
Using Suffix Trees for Exact Matching

Given a suffix tree for a text $T[1,\ldots,m]$, we can find all occurrences of a pattern $P[1,\ldots,n]$ in $T$ in $O(n+k)$ time, where $k$ is the number of occurrences. How?

1. Traverse a path from the root as long as it matches $P$

2′ All characters of $P$ are matched. Then traverse the subtree from the point where the last character was matched and return the leaf labels

Step 1 takes $\Theta(n)$ time and Step 2′ takes $\Theta(k)$ time
Using Suffix Trees for Exact Matching

Given a suffix tree for a text $T[1,...,m]$, we can find all occurrences of a pattern $P[1,...,n]$ in $T$ in $O(n+k)$ time, where $k$ is the number of occurrences. How?

1. Traverse a path from the root as long as it matches $P$
2’ All characters of $P$ are matched. Then traverse the subtree from the point where the last character was matched and return the leaf labels

Step 1 takes $\Theta(n)$ time and Step 2’ takes $\Theta(k)$ time...but how to build the suffix tree for $T$?
Ukkonen’s Algorithm

Build a Suffix Tree for S[1..m] in O(m) Time

The algorithm first builds an *implicit* suffix tree, and then converts the implicit suffix tree to a true suffix tree.
Implicit Suffix Tree

Obtained from a suffix tree by

- Removing every copy of $ from the edge labels
- Removing any edge that has no label
- Removing any node that does not have at least two children
Suffix Tree and Implicit Suffix Tree for abxa$
Ukkonen’s Algorithm

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

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 $j$:

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

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

1. Find the end of path $S[j..i]$
2. Apply the appropriate suffix extension rule:
   - **Rule 1**: Path $S[j..i]$ ends at a leaf. Add $S[i+1]$ to the end of the label on that leaf edge.
   - **Rule 2**: At least one path continues from the end of $S[j..i]$, but no such path starts with $S[i+1]$. Create a new leaf edge labeled $S[i+1]$, starting from $S[j,...,i]$. (Create a new node if $S[j..i]$ ends inside an edge.) Number the new leaf "j".
   - **Rule 3**: Some path from the end of $S[j..i]$ starts with character $S[i+1]$. Do nothing
Ukkonen’s Algorithm

Example: awayawxawxz

Diagram showing the application of Ukkonen’s Algorithm with nodes labeled from 0 to 10.
Ukkonen’s Algorithm

Naïve implementation takes $\Theta(m^3)$ time: $m$ phases, $i$th phase has $i$ extensions, taking time proportional to $1, 2, \ldots, i$

A bottleneck in doing an extension $j$ of phase $i+1$ is finding the end of path $S[j..i]$

We need a faster implementation
Ukkonen’s Algorithm: Suffix Links

Let $x$ be a character and $\alpha$ be a string.
Let internal node $v$ have label $x\alpha$.
Let internal node $v'$ have label $\alpha$.

A suffix link is a pointer from $v$ to $v'$. 
Let $x$ be a character and $\alpha$ be a string.
Let internal node $v$ have label $x\alpha$.
Let internal node $v'$ have label $\alpha$.

A *suffix link* is a pointer from $v$ to $v'$.

In addition to maintaining suffix links, the algorithm also maintains a link to leaf node 1, let’s call this the *1-link*.
Ukkonen’s Algorithm: Suffix Links

Example: awayawxawxz

suffix links to the root not shown
Ukkonen’s Algorithm: Phase i+1, Extension j

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

1. Find the end of path $S[j..i]$
   (using the 1-link or suffix links)
2. Apply the appropriate suffix extension rule (as before, to ensure that a path labeled $S[j, i+1]$ is in the tree)
3. If a new internal node $w$ was added to the tree on extension $j-1$, add the suffix link from $w$ now

Next time we’ll see details of steps 1 and 3
Summary so far

• Suffix trees have many applications, including an exact matching algorithm that finds all $k$ occurrences of a pattern $P$ of length $n$ in a given text $T$ in time $O(n+k)$, independent in the length $m$ of $T$, assuming that the suffix tree for $T$ has already been created.

• Ukkonen’s algorithm can create the suffix tree for $T$ in $O(m)$ time.

• We’ve seen the high level description of the algorithm, and introduced suffix links.

• More details next time...