More Applications of Suffix Trees

02-714
Slides by Carl Kingsford

Following Gusfield
All Pairs Suffix-Prefix Matches

Problem. Given a set of strings $S = \{s_1, \ldots, s_k\}$ of total length $m$, find all of the longest suffix-prefix exact matches.

- Can represent these matches by $\binom{k}{2}$ integers.
Notation & Main Idea

**Def.** $L(v) := \text{list of strings (indices) from } S \text{ such that } P(v) \text{ is a complete suffix.}$

Generalized suffix tree for strings $S$:

- $P(v) := \text{path from root to } v.$
  - each node on this path represents a prefix of $s_j$
  - If $L(v)$ on $P(s_j)$ contains $i$ then $P(v)$ is a suffix of $s_i$.
  - $\Rightarrow$ deepest node $v$ on $P(s_j)$ s.t. $L(v)$ contains $i$ gives the longest prefix of $s_j$ that matches a suffix of $s_i$.

**Algorithm:** find these deepest nodes for all $s_i$ at once with a single traversal of the tree.
1. Do a DFS on the generalized suffix tree
2. Maintain $k$ stacks during the DFS that you update as follows:

When entering node $v$, push $v$ onto all stacks in $L(v)$.
When backtracking out of $v$, pop all stacks in $L(v)$.

3. When you reach a node corresponding to a full string, output the depth of the nodes at the top of each stack.
Runtime

Thm. Runtime of the preceding algorithm is $O(m + k^2)$

Proof. Building the generalized suffix tree takes $O(m)$ time.

The $L(v)$ sets can be saved as you are building the suffix tree (when you add in a suffix for $i$, add it to $L(v)$).

There are $O(m)$ indices in the union of the $L(v)$ lists, so the total pop/push events take $O(m)$.

There are $k$ nodes at which you will output, and each output takes $O(k)$ time, leading to $O(k^2)$ time for output.
Ziv-Lempel Compression

Let $S$ be a string of length $m$ that we want to compress.

**Notation.**

- $b(i) :=$ the longest string in $S[1...i-1]$ that matches a prefix of $S[i...m]$  
- $e(i) := |b(i)|$  
- $p(i) :=$ the position of $b(i)$ in $S[1...i-1]$  

**Algorithm:**

1. walk down string from left to right.
2. when at position $i$, output $(p(i), e(i))$ instead of $S[i...i+e(i)-1]$
3. skip ahead to $i := i + e(i)$
Computing \( p(i), e(i) \)

Build suffix tree on \( S \), and at every node \( v \), store:

\[ c(v) := \text{minimum suffix } \# \text{ in the subtree rooted at } v \]

Consider the set

\[ V = \{ v \in L(i) : c(v) + \text{depth}(v) < i \} \]

Every \( v \in V \) matches a prefix of \( S[i...m] \) and is contained in \( S[1...i-1] \) (because \( c(v) < i \))

The deepest such node represents \( b(i) \).
Running time

Algorithm for $p(i)$, $e(i)$:

1. To compute $p(i)$: walk down path spelling out $S[i...m]$
2. Stop when $c(v) + \text{depth}(v) = i$.
3. $p(i) = c(v)$
4. $e(i) = \text{depth}(v)$

Total running time to compress a string of length $m$:

- $O(m)$ to built the suffix tree
- $O(m)$ time (bottom up traversal) to compute $c(v)$
- $O(e(i))$ to search at position $i$, but each time you spend $O(e(i))$ time, you skip $e(i)$ letters $\Rightarrow O(m)$ total time.