(Some) suffix tree applications

Pål Sætrom

Topics
- Common substrings
- Matching patterns against text
- Matching suffixes and prefixes
- Combining suffix tree and additional data structures to accelerate search

Longest common substring of two strings

Find longest substring common to S and R

Input: Strings $S = s_1 \ldots s_n$ and $R = r_1 \ldots r_m$
Output: String $T = t_1 \ldots t_p$, such that $t_1 \ldots t_p = s_1 \ldots s_{p-1} = r_1 \ldots r_{p-1}$ and $\max_{i,j} |T|$

Ex:
$S = \text{"gtgca"}, R = \text{"atgcgg"}$

Generalized suffix trees solve longest common substring

$T = (\text{"gtgca"}, \text{"tgca"})$

- Time: $O(\ )$
- Space: $O(\ )$

Longest common substring of several strings

Find longest substring common to $k$ of $K$ strings, where $2 \leq k \leq K$

Input: Set of strings $S$, $|S| = K$
Output: $l(k)$ for all $k$, $2 \leq k \leq K$, such that $l(k)$ is the longest substring common for at least $k$ of the strings in $S$.

Ex: $S = \{\text{sandollar, sandlot, handler, grand, pantry}\}$

Finding $l(k)$

1. Build generalized suffix tree $T$ of $S$ $O(\ )$
2. For each internal node $v$, find the number of distinct string identifiers $C(v)$ $O(\ )$
3. Traverse $T$ and update vector $V$, where $V(k)$ is deepest string depth of node $v$ such that $C(v) = k$ $O(\ )$
4. Compute $l(k)$ by scanning $V$ from largest to smallest $k$ and set $V(k) = \max \{V(k), V(k+1)\}$ $O(\ )$
Searching with suffix trees

Find all occurrences of \( P \) “gca”, “c”, and “gct” in \( T \) “gctgca”

What if \( P \) is known and \( T \) is unknown?

Computing matching statistics

Given pattern \( P \) and text \( T \), compute matching statistics \( ms(i) \)

Input: Strings \( P = p_1 \ldots p_n \) and \( T = t_1 \ldots t_m \)
Output: \( ms(i) \) for all \( 1 \leq i \leq m \), such that \( ms(i) \) is the length of the longest substring starting at position \( i \) of \( T \) that matches a substring somewhere in \( P \)

Ex:
\( T = \) “gctgca”, \( P = \) “atgcgg”
\( ms(1) = 2, ms(3) = __, ms(5) = __ \)

Relation to exact matching problem?

Computing matching statistics in O(m)

- Build suffix tree for \( P \)
- Compute \( ms(1) \) by scanning from root
Ex:
\( P = \) “gctgc”
\( T = \) “ctcgca”

- Compute \( ms(i+1) \)?

Following suffix links to compute \( ms(i + 1) \)

- \( ms(i) \) ends up in b labeled x\( \alpha \gamma \)
  - \( x\alpha \) is prefix of \( T[i..m] \) so \( \alpha \gamma \) is prefix of \( T[i+1..m] \)
  - b is internal node: follow suffix link to s(v) labeled \( \alpha \gamma \)
  - b is on edge: back up, follow suffix link, skip-count \( \gamma \)
  - Continue matching until b’ where mismatch or leaf
  - \( ms(i+1) \) = string depth of b’

Computing \( ms \) is O(m)

- Back up and link traversals is O(1)
- Total time traversing \( y \) is bounded by node depth (similar to construction algorithm) \( O(m) \)
- Each letter in \( T \) scanned at most twice \( O(m) \)

Reporting locations of matching substrings in O(m)

\( p(i) \) is position in \( P \) of substring that matches substring in \( T \) starting at position \( i \) for exactly \( ms(i) \) places

Ex:
\( T = \) “ctcgca”, \( p(1) = __, p(3) = __ \)
All pairs of suffix-prefix matches

For two strings $S = s_1 \ldots s_n$ and $R = r_1 \ldots r_m$, any suffix of $S$ that matches a prefix of $R$ is a suffix-prefix match of $S$ and $R$.

Given set of strings $S$, find all pairs of max-length suffix-prefix matches.

Input: Set of strings $S = \{S_1, \ldots, S_k\}$ of total length $m$.

Output: For each ordered pair $S_i$ and $S_j$, $1 \leq i, j \leq k$, the length $l(i,j)$ of the longest suffix-prefix match of $S_i$ and $S_j$.

Ex: $S = \{"gctgca", "atgcgg", "tcggc"\}$

$l(1,1) = 6$, $l(1,2) = 1$, $l(1,3) = \_\_$, $l(3,1) = \_\_$

Solving all-pairs suffix-prefix problem

- Best possible solution is $O(m + k^2)$
- Can generalized suffix tree be used?

Terminal edges in generalized suffix tree are essential

- Terminal edge: edge labeled with only terminal symbol $\$.$
- Represents complete suffix
- Relation to prefixes?

Keeping track of terminal edges solves problem

- Label each internal node $v$ with indexes of its terminal edges, $L(v)$
- Traverse tree depth-first and keep track of terminal edges we have passed
  - Use $k$ stacks
  - Push $v$ on stack $i$ if $i \in L(v)$
  - If at leaf $S_i$, record for each $i$ the path-label length of node at top of stack $i$
  - When backing up from $v$, pop all stacks $i \in L(v)$