BME 4800/CSE 3800/CSE 5800: Two applications of suffix tree/suffix array

Yufeng Wu
September 10, 2009

1 Find common substrings of more than two strings

This is an extension to the LCS (longest common substring) problem. Say we have K strings (whose total length is n). We define l(k) be the length of longest substring common to at least k of strings. Here 2 ≤ k ≤ K. Surprisingly, this problem can be solved in O(n) time! We will only give a O(nK) algorithm.

Similar to the LCS problem, we build a concatenated string for S1, S2, S3... as S = S1$S2$S3...

Now, we define, for each internal node v, C(v) = number of distinct string identifiers (S1, S2...) appear in the leaves of the subtree. We also collect string depth of each node. Then we simply traverse the suffix tree to find, for each k, the deepest node v, whose C(v) = k. More detail, we initialize an array V(k) = 0 for each k. During the traversal the tree, for each node v, if C(v) = k and V(k) < depth of v, then V(k) = depth of v. Note: V(k) means this substring appears in exactly k strings out of K strings. This is slightly different from the definition l(k). But l(k) is easily derived from V(k): scan V from k being the largest to smallest; if V(k) < V(k+1), then set V(k) = V(k+1). Then the V(k) value is what we need for l(k). If we assume all C(v) values are known, this step takes O(n) time.

Now the question is how to compute C(v). We take a simple approach: for each ancestral node v, store a binary vector bv with one bit for each string. Here bv[i] = 1 means the presence of some suffix of string si within the subtree under the node, and 0 otherwise. Then for each internal node v, suppose it has two children, v1 and v2, we simply generate the b vector by:

\[ b_v = b_{v1} \cup b_{v2} \]

For a leaf node v, we set bv[i] to contain 1 if that leaf suffix is from Si.

Then, a simple tree traversal and we are done. The total running time is O(Kn) time. This is because we need to compute the b vector for each node in the suffix tree. BTW, this problem can be solved to O(n) time. But we will not discuss it here.

2 Finding tandem repeat using suffix tree

A tandem repeat occurs when there is a substring α that appear twice and at adjacent positions. That is, αα is a substring of the text. Now the problem is, give string S (of n symbols), find tandem repeat.
repeats in S. A naive method is, try all possible starting point x and y and direct compare from the two starting positions. This will take $O(n^3)$. Here is the key lemma.

**Lemma 2.1.** There is a tandem repeat starting at position i (and j that is l distance from i) iff i and j are located in the subtree of some internal node of the suffix tree, and depth of this node is longer than l.

Now think about it to see why this lemma holds. I suggest you draw some pictures and think about in two directions to convince yourself the lemma works.

Here is a simple algorithm: collect depth for each node, which takes $O(n)$ time. Then for each internal node, test all pairs of leaves x and y to see if $|x - y| \leq \text{depth}(v)$. This still sounds $O(n^3)$ since there are $O(n)$ nodes and each node, we may test $O(n^2)$ pairs of nodes. But a better analysis gives $O(n^2)$ time: for node v, we only test the nodes x in its left subtree, and y in its right subtree. That is, we only test x and y whose lowest common ancestor is v. This way, each pair of nodes is tested for just once, and each test takes $O(1)$ time. That is, the running time is $O(n^2)$. BTW, there is $O(n\log(n)+k)$ algorithm for this problem, where k is the number of tandem repeats in the string. Again, we will not discuss it here.