Algorithms for Massive Data Sets

Student name and id: Alexander R. Johansen (s145706)

Collaborator name(s) and id(s): Jonas (s142957)

Hand-in for week: 4

---

1 FCA Problem

The First Covering Ancestor Problem can be solved in linear space using $|\Sigma|$ Y-fast tries totalling $O(n)$ space, and RMQ +-1 with indirection - also linear space. The $FCA(l, \alpha)$ query can be solved in lg lg n time on such data structure.

1.1 Analysis of problem

Using predecessor/successor
Suppose L is a given node. P is the predecessor of that node, PP is the predecessor's predecessor, S is the successor and SS is the successor of the successor. The goal of NCA is to get the ancestor at the maximal depth. Thus using the nearest common ancestor, NCA, the depth relation can be proved to have the following relationship: $NCA(L, P).depth \geq NCA(L, PP).depth$ and $NCA(L, S).depth \geq NCA(L, SS).depth$.

Predecessor/successor for highest depth
Using this description of the depth relationship of successors and predecessors. I can use the predecessor and successor to find the highest depth, and thus solve the *First Covering Ancestor Problem* only computing NCA of L and the Predecessor, and L and Successor.

Several Y-Fast Tries
As FCA(a,L), where a denotes any given letter in the Σ alphabet, needs to provide letter specific predecessor and successor to L. A data structure such as the one proposed by me last week in 3.2 is needed. $|\Sigma|$ different Y-Fast Tries, each storing indexes of a given letter in the alphabet Σ.

Using NCA
After the predecessor, P, and successor, S, have been found, comparing NCA(L,P) and NCA(L,S) for the largest depth can be computed in constant time.

1.2 Specific data structure

Given an empty tree T
1. Compute the depths of all nodes in T, and store it in the nodes.
Space: $O(n)$
2. Use an Euler Tour to fold out the tree T in an array, with depths as values, pointers to the original tree and the leaf values represented.
Space: $O(n)$
3. Compute $|\Sigma|$ Y-Fast Tries using step 2, each of which stores the indexes of a given letter, such that every letter, α, is represented by a Y-Fast trie(similar to 1.2 in last assignment).
Space: $O(n)$
4. Create a hash table pointing to the root for all α Y-Fast Trie – a large alphabet could cause linear time search for the given letter, hash table provides constant time reference to the wanted Y-Fast Trie)
Space: $O(|\Sigma|)$
5. Compute the indirection RMQ data structure on the array from step 2 to support constant time $NCA(node1, node2)$ using linear space.
Space: $O(n)$

1.2.1 Evaluation data structure

Step 1: $O(n)$
Step 2: $O(n)$
Step 3: $O(n)$
Step 4: $O(\Sigma)$
Step 5: $O(n)$
Since $|\Sigma| <= n$, as the alphabet cannot be larger than the total amount of letters.
Total: $O(n)$

1.3 Specific algorithm

Given a leaf L and a letter α
1. Using step 3,4: Find the predecessor, P and successor, S, of the index of L using the Y-Fast Trie of α(both O(lg lg n) operations) the α Y-Fast Trie can be found in constant time with the hash table from step 4.
time: $O(lglgn)$
2. Using step 5: Compute $a1 = NCA(L, P), a2 = NCA(L, S)$, return $max(a1.depth, a2.depth)$
time: $O(1)$

1.3.1 Evaluating algorithm

Step 1: $O(lglgn)$
Step 2: $O(1)$
Total: $O(lglgn)$