Algorithms and programming
École normale supérieure - Département d'informatique
TD 6: Disjoint sets

---

1. Give a sequence of m MAKE_SET, UNION, and FIND_SET operations, $n < m/2$ of which are MAKE_SET operations, that takes $\Omega(m \log n)$ time when we use union by rank only.
2. **(Zombie invasion)** In the post apocalyptic zombie world you and a small group of survivors have barricaded yourself in a small building. The only thing keeping the brutal zombies from eating you is a strong fortification. The fortification consists of a $k \times k$ grid of walls. In the top of the grid the zombies are waiting to come in, and you and your group is located in the bottom. Unfortunately, the walls are weak and collapse regularly. If a path of walls between the top and the bottom of the grid is collapsed the zombies can get in and eat you. In order to start evacuation you want to monitor if there currently is a path through the fortification (from top to bottom). Give a data structure that can efficiently keep track of this while the walls are collapsing one by one.
3. **(Tarjan's off-line least-common-ancestors algorithm)** The least common ancestor of two nodes u and v in a rooted tree T is the node w that is an ancestor of both u and v and that has the greatest depth in T. In the off-line least-common-ancestors problem, we are given a rooted tree T and an arbitrary set $P = \{(u, v)\}$ of unordered pairs of nodes in T, and we wish to determine the least common ancestor of each pair in P. To solve the off-line least-common-ancestors problem, the following procedure performs a tree walk of T with the initial call $LCA(root[T])$. Each node is assumed to be colored WHITE prior to the walk.

Algorithm 1 $LCA(u)$

```
1: $MAKE\_SET(u)$
2: $ancestor[FIND\_SET(u)] \leftarrow u$
3: for each child $v$ of $u$ in $T$ do
4:     $LCA(v)$
5:     $UNION(u, v)$
6:     $ancestor[FIND\_SET(u)] \leftarrow u$
7: end for
8: $color[u] \leftarrow BLACK$
9: for each node $v$ such that $(u, v) \in P$ do
10:     if $color[v] == BLACK$ then
11:         print "The least common ancestor of $u$ and $v$ is $ancestor[FIND\_SET(v)]$"
12:     end if
13: end for
```

(a) Argue that line 11 is executed exactly once for each pair $(u, v) \in P$.

(b) Prove that LCA correctly prints the least common ancestor of u and v for each pair $(u, v) \in P$.

   (c) Analyze the running time of LCA.

4. **(Depth determination)** In the depth-determination problem, we maintain a forest $F = \{T_i\}$ of rooted trees under three operations : MAKE-TREE(v) creates a tree whose only node is v. FIND-DEPTH(v) returns the depth of node v within its tree. GRAFT(r, v) makes node r, which is assumed to be the root of a tree, become the child of node v, which is assumed to be in a different tree than r but may or may not itself be a root.

   (a) Suppose that we use a tree representation similar to a disjoint-set forest : $p[v]$ is the parent of node v, except that $p[v] = v$ if v is a root. If we implement $GRAFT(r, v)$ by setting $p[r] \leftarrow v$ and FIND-DEPTH(v) by following the find path up to the root, returning a count of all nodes other than v encountered, show that the worst-case running time of a sequence of m MAKE-TREE, FIND-DEPTH, and GRAFT operations is $\Theta(m^2)$.

   By using the union-by-rank and path-compression heuristics, we can reduce the worst-case running time. We use the disjoint-set forest $S = \{S_i\}$, where each set S_i (which is itself a tree) corresponds to a tree T_i in the forest F. The tree structure within a set S_i, however, does not necessarily correspond to that of T_i. In fact, the implementation of S does not record the exact parent-child relationships but nevertheless allows us to determine any node's depth in T_i. The key idea is to maintain in each node v a “pseudodistance" $d[v]$, which is defined so that the sum of the pseudodistances along the path from v to the root of its set S, and equals to the depth of v in T_i. That is, if the path from v to its root in S, is $v_0, v_1, \ldots, v_k$, where $v_0 = v$ and v_k is S_j's root, then the depth of v in T, is $\sum_{j=0}^{j=k} d[v_j]$.

   (b) Give an implementation of MAKE-TREE.

   (c) Show how to modify FIND-SET to implement FIND-DEPTH. Your implementation should perform path compression, and its running time should be linear in the length of the find path. Make sure that your implementation updates pseudodistances correctly.

   (d) Show how to modify the UNION and LINK procedures to implement GRAFT(r, v), which combines the sets containing r and v. Make sure that your implementation updates pseudodistances correctly. Note that the root of a set S_i is not necessarily the root of the corresponding tree T_i.

   (e) Give the best bound you can on the worst-case running time of a sequence of m MAKE-TREE, FIND-DEPTH, and GRAFT operations, n of which are MAKE-TREE operations.