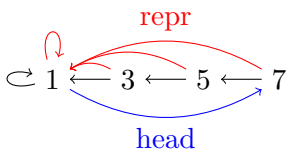
University of Konstanz
Department of Computer & Information Science
Sabine Cornelsen

Design and Analysis of Algorithms
Winter 2018/2019

Assignment 3

Post Date: 05 Nov 2018 **Due Date:** 12 Nov 2018, 11:30 a.m.
You are permitted and encouraged to work in groups of two.

Problem 1: Union-Find with Extended Linked List Representation — 6 Points



Consider the following extended variant of the linked list representation:
Store for each item of the list also the representative of the list. Then an operation Find can be performed in $\Theta(1)$, but for each Union operation, we have to update the pointers to the representative of the items in the list that is appended to the head of the other list.

We also store the length of each list and always append the shorter list to the longer one when performing a Union operation.

(a) Write easily understandable pseudocode for the operations MakeSet, Union, and Find.

Consider a sequence of m operations MakeSet, Union, and Find, of which n are MakeSet operations.

(b) Prove that the pointer repr of a fixed item can be updated at most $\log n$ times.

(c) Conclude that the whole sequence can be performed in $\mathcal{O}(m + n \log n)$ time.

Problem 2: Sequence of Operations — 6 Points

Consider sequences of operations MakeSet, Find with path compression, and weighted Union where all Union operations are performed before the first Find operation.

(a) Show that the amortized cost for m operations is in $\mathcal{O}(m)$.

(b) Does (a) hold if Find still performs path compression but Union is unweighted?

(c) Does (a) hold if Union is still weighted but Find does not perform path compression?

[please turn over]

Problem 3: Application of Union-Find **8 Points**

In this exercise we want to use Union-Find as the basis to solve a different problem.

We maintain a forest $\mathcal{F} = \{T_i\}$ of rooted trees under three operations:

- MAKETREE(v): create a tree whose only node is v.
- FINDDEPTH(v): return the depth (distance to the root) of node v within its tree.
- APPEND(r, v): make 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 APPEND(r, v) by setting $p[r] \leftarrow v$ and FINDDEPTH(v) by following the path from v up to the root in T_i and returning a count of all nodes other than v encountered, show that the worst-case running time of a sequence of m MAKETREE, FINDDEPTH, and APPEND operations is in $\Theta(m^2)$.

By using the weighted union and path compression heuristic, we can reduce the worst-case running time. We use the disjoint-set forest $\mathcal{S} = \{S_i\}$, where each set S_i (which is itself a tree) corresponds to a tree T_i in the forest $\mathcal{F}$. The tree structure within a set S_i, however, does not necessarily correspond to that of T_i. In fact, the implementation of S_i 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 a **pseudodistance** $d[v]$ in each node v, which is defined so that the sum of pseudodistances along the path from v to the root of its set S_i equals the depth of v in T_i. That is, if the path from v to its root in S_i is $v_0, v_1, \ldots, v_k$, where $v_0 = v$ and v_k is S_i's root, then the depth of v in T_i is $\sum_{j=0}^{k} d[v_i]$.

(b) Give an implementation of MAKETREE.

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

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

Let S_r and S_v be the trees containing r and v. Note that the choice to append S_r to the root of S_v or vice-versa depends on whether $|S_r| < |S_v|$ or $|S_r| \geq |S_v|$.