

Homework 2

CS6043 Design and Analysis of Algorithms II

Fall 2009

Due: 10/26/09 in class

Maximum Score: 140 points

Note: This assignment has 2 pages.

1. (10 points)

Assume that the only operations on the keys are comparing two keys (as we do in the operations for binary heaps, binomial heaps and Fibonacci heaps). Show that it is **not** possible to make both INSERT and EXTRACT-MIN operations running in $O(1)$ amortized time for binary heaps, binomial heaps, or Fibonacci heaps.

(Hint: Show that if it were possible, then some well-known lower bound would be violated.) (10 points)

2. (30 points) Consider Fibonacci heaps as given in the textbook.

a. Show that an n -node Fibonacci heap may have height n (rather than $O(\log n)$) by giving, for any positive integer n , a sequence of Fibonacci heap operations that creates a Fibonacci heap consisting of just one tree that is a linear chain of n nodes. **(15 points)**

b. Use an **accounting method** to analyze the amortized time bounds of operations INSERT, UNION, EXTRACT-MIN, and DECREASE-KEY. You can use the fact that the actual cost of the CONSOLIDATE operation (used in EXTRACT-MIN) is linear in the length of the root list when CONSOLIDATE is invoked. You can assume that 1 credit can pay for $O(1)$ work. Your resulting amortized bounds should match the ones given in the textbook.

(Hint: Look at the potential function given in the textbook to see where to put credits.) (15 points)

3. (45 points) In the textbook, the *union* operation for the disjoint-set union-find problem is defined as $\text{UNION}(x, y)$, where x and y are two elements. An alternative way is to define *union* as $\text{UNION}'(A, B)$, where A and B are two disjoint sets being maintained. Actually, UNION' is equivalent to the LINK operation defined in the textbook.

a. Show that using path compression, any sequence of m MAKE-SET, FIND-SET and UNION' operations takes $O(m)$ time if all the UNION' operations occur **before** any FIND-SET operation, even when union by rank is not used.

(Hint: Use an accounting method to show that each operation takes $O(1)$ amortized time.) (15 points)

b. Show why your argument in **part a.** does not work if there are arbitrarily **intermixed** FIND-SET and UNION' operations in the sequence.

(Hint: Give a sequence of operations that creates a long path in which there is no credit left.) (15 points)

c. Textbook Exercise 21.4-2 (page 518), where both union by rank and path compression are used.

(Hint: Define $\text{size}(x)$ to be the number of nodes in the subtree rooted at x . Prove the following claim: If x is a *tree root*, then $\text{size}(x) \geq 2^{\text{rank}(x)}$. To prove the claim, use an induction on the number of LINK operations performed. Finally, use the claim to finish the proof about $\text{rank}(x)$,

for which you need to consider two cases: (i) x is a root, and (ii) x is not a root. For (ii), look at the time when $rank(x)$ was last modified.) **(15 points)**

4. (20 points) In this question, we consider a special case of the *minimum spanning tree* problem. (Consult Chapter 23 or Problem 19-2 on page 474 of the Textbook for the definition of the minimum spanning tree problem.) Given a connected, undirected graph $G = (V, E)$ where V is the set of nodes and E is the set of edges with edge weights in the set of values $\{1, 2, 3, \dots, k\}$ for some **constant** k , design (and analyze) an algorithm that computes a minimum spanning tree of G in time $O(|E| + |V|)$.

(Note: You should **not** consider $\alpha(n)$ as a constant.)

(Hint: Use Prim's Algorithm. Design a simple priority queue using the special property of the edge weights.) **(20 points)**

5. (35 points) 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-ancestor problem**, we are given a rooted tree T and an arbitrary set $P = \{p : p = \{u, v\}\}$ of unordered pairs of nodes in T , and we wish to determine the least common ancestor of each pair in P . We assume that there are n nodes in T and m pairs in P , where $m \geq n$.

To solve the off-line least-common-ancestor 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.

$LCA(u)$

```
1 MAKE-SET( $u$ )
2  $ancestor[FIND-SET(u)] \leftarrow u$ 
3 for each child  $v$  of  $u$  in  $T$ 
4   do  $LCA(v)$ 
5     UNION( $u, v$ )
6      $ancestor[FIND-SET(u)] \leftarrow u$ 
7    $color[u] \leftarrow BLACK$ 
8 for each node  $w$  in  $T$ 
9   do if  $\{u, w\} \in P$ 
10     if  $color[w] = BLACK$ 
11       print "The Least common ancestor of"  $u$  "and"  $w$  "is"  $ancestor[FIND-SET(w)]$ 
```

(Hint: To get a feeling of how the algorithm works, try to work on a small example.)

a. As written, lines 8 and 9 make $LCA(root[T])$ run in time $\Omega(n^2)$. Give an efficient implementation for lines 8 and 9 so that the overall running time for finding all nodes w such that $\{u, w\} \in P$ during the entire execution of $LCA(root[T])$ is $O(m)$. **(6 points)**

b. Argue that at the time of the call $LCA(u)$, the number of sets in the disjoint-set data structure is equal to the depth of u in T . (**Note:** We define the depth of the root to be 0.) **(10 points)**

c. Prove that LCA correctly prints the least common ancestor of u and w for each pair $\{u, w\} \in P$. **(15 points)**

d. Analyze the overall running time of $LCA(root[T])$, assuming that we implement the disjoint-set data structure using union by rank with path compression. **(4 points)**