CS 535 Design and Analysis of Algorithms **Fall Semester, 2021**

Homework 3 Version 1.1

Assigned: September 29 *Due: October 13*

Remote students: please use Blackboard to submit your solutions. Sections 01 and 02: submit both hard copy and on Blackboard.

Notes for pseudocode usage:

1. C/C++/Java instructions are fine. But do not write object-oriented additions. Do not declare or use any class. Declare only procedures (if necessary) and explain in words what each procedure does, and what is the use of each parameter. Feel free to use as procedures algorithms from the textbook; indicate page and edition.
2. One instruction per line
3. Match the brackets with a horizontal line
4. Number your lines
5. Write down if your array is indexed $0 \ldots n-1$ or $1 \ldots n$.

Problem 1 When analyzing splay trees, we have defined the rank function on nodes: $r(v) = \lfloor \lg |V(T_v)| \rfloor$, where lg is the base 2 logarithm, and T_v is the subtree rooted at v (V and all its descendants).

We also called $r'(v)$ as being the rank of v after one operation - a double rotation or a single rotation. We proved that the amortized cost of a double rotation at x does not exceed $3(r'(x) - r(x))$, and that the amortized cost of a single rotation at x does not exceed does not exceed $1 + 3(r'(x) - r(x))$.

This problem asks to show that the $+1$ term for a single rotation is needed. In other words, give an example of a binary search tree and a single rotation whose amortized cost as defined in class (actual cost plus the sum of ranks after the operation minus the sum of ranks before the operation) exceeds $3(r'(x) - r(x))$.

As an aside (no need to turn in anything about this paragraph), with single rotations only, the splay operations will not have amortized cost of $O(\lg n)$ each.

Problem 2 Suppose we implement the UNION-FIND Data structure with union-by-rank but no path-compression. For every n, give an example of a sequence of operation with n MAKE-SET(), p UNION and r FIND operations whose total running time exceeds $c(n + p + r) \log n)$ for some constant c. We also require that each element is a parameter in FIND() at most once in this sequence (any element can appear many times in UNION(,) operations).

Problem 3 We describe below a data structure that maintains the transitive closure of a directed graph while arcs (directed edges) are added to the graph.

Formally, a set of vertices V is given (with $|V| = n$), and arcs $e_1, e_2, \ldots, e_m$ become available one by one (e_i is not known before computing R_{i-1}, defined below). Let $G_i = (V, E_i)$, where $E_0 = \emptyset$ and $E_i = E_{i-1} \cup e_i$. Let R_i, a $n \times n$ matrix, have $R_i[u, v] = 1$ if u has a directed path to v, and $R_i[u, v] = 0$ otherwise. Thus R_i stores the transitive closure of G_i.

Note that R_0 has entries that are 1 only on the main diagonal.

1. Give a series of instances (one for each n) such that there exists an i with the number of entries 1's in R_i being $\Omega(n^2)$ higher than the number of entries 1 in R_{i-1}.

2. Consider however the code

   ```
   ADD($e_i$) where the tail of $e_i$ is $u$ and the head of $e_i$ is $v$:
   for all $x \in V$
     if $R[x, u] = 1$ AND $R[x, v] = 0$
       for all $y \in V$
         $R[x, y] \leftarrow \max(R[x, y], R[v, y])$
   ```

   Prove that if $R = R_{i-1}$ before the code is executed, then $R = R_i$ after the code is executed.

3. Use the first part of this problem to show that ADD(e_i) may have running time $\Omega(n^2)$.

4. Prove that despite this, the running time of m operations ADD(.) is $O(nm + n^3)$.