Homework #5
Introduction to Algorithms
601.433/633
Spring 2020

Due on: Saturday, March 28th, 12pm
Where to submit: On Gradescope, please mark the pages for each question

1 Problem 1 (25 points)

1.1 Problem 1.1 (10 points)

Suppose we wish not only to increment a counter of length $m \in \mathbb{N}$ but also to reset it to zero (i.e., make all bits in it 0). Counting the time to examine or modify a bit as $\Theta(1)$, show how to implement INCREMENT and RESET operations on a counter (represented as an array of bits) so that *any* sequence of n operations takes $O(1)$ amortized time per operation.

You may assume that the counter is initially zero and that you may access and modify any specific bit (say bit $j \in [m]$) in $O(1)$ time. Additionally, you may assume that the total count in the counter never exceeds $2^m - 1$ during the course of the n operations.

Prove correctness and running time of your algorithms.

(Hint: Keep a pointer to the highest-order 1.)

1.2 Problem 1.2 (15 points)

Design a data structure to support the following two operations for a set S of integers, which allows duplicate values:

- INSERT(S, x) inserts x into S.
- DELETE-LARGER-HALF(S) deletes the largest $\lceil |S|/2 \rceil$ elements from S.

Explain how to implement this data structure so that any sequence of m INSERT and DELETE-LARGER-HALF operations runs in amortized $O(1)$ time per operation. Your implementation should also include a way to output the elements of S in $O(|S|)$ time.

Prove the running time of your implementation.

2 Problem 2 (10 points)

Let $G = (V, E)$ be a directed graph. $a \in V$ is a *central* vertex if for all $b \in V$ there exists a path from a to b. Provide an $O(|V| + |E|)$ time algorithm to test whether graph G has a central vertex.

Prove the correctness of your algorithm and analyze the running time.

3 Problem 3 (15 points)

You're helping some analysts monitor a collection of networked computers, tracking the spread of fake information. There are n computers in the system, labeled $C_1, C_2, ..., C_n$, and as input you're given a collection of trace data indicating the times at which pairs of computers communicated. Thus the data is a sequence of ordered triples (C_i, C_j, t_k); such a triple indicates that C_i and C_j exchanged bits at time t_k. There are m triples total.

We'll assume that the triples are presented to you in sorted order of time. For purposes of simplicity, we'll assume that each pair of computers communicates at most once during the interval you're observing. The analysts you're working with would like to be able to answer questions of the following form: If the fake information was generated by computer C_a at time x, could it possibly have been sent to C_b by time y? The mechanics of communicating the information are simple: if a computer containing the fake information C_i communicates with another computer C_j that hasn't received that information yet by time t_k (in other words, if one of the triples (C_i, C_j, t_k) or (C_i, C_j, t_k) appears in the trace data), then computer C_j receives the fake information, starting at time t_k.

The fake information can thus spread from one machine to another across a sequence of communications, provided that no step in this sequence involves a move backward in time. Thus, for example, if C_i has received the fake information by time t_k, and the trace data contains triples (C_i, C_j, t_k) and (C_j, C_q, t_r), where $t_k \leq t_r$, then C_q will receive the fake information via C_j. (Note that it is okay for t_k to be equal to t_r; this would mean that C_j had open connections to both C_i and C_q at the same time, and so the information would have been sent from C_i to C_q.)

Design an algorithm that answers questions of this type: given a collection of trace data, the algorithm should decide whether the fake information generated by computer C_a at time x could have been received by computer C_b by time y. The algorithm should run in time $O(m + n)$.

Prove correctness and running time as usual.