**CS 38 Introduction to Algorithms** **Spring 2014**

# Problem Set 1

*Out: April 8* *Due:* **April 15**

Reminder: you are encouraged to work in groups of two or three; however you must turn in your own write-up and note with whom you worked. You may consult the course notes and the optional text (CLRS). The full honor code guidelines can be found in the course syllabus.

Please attempt all problems. **To facilitate grading, please turn in each problem on a separate sheet of paper and put your name on each sheet. Do not staple the separate sheets.**

1. (a) Consider the following recurrence: $T(n) = 2 \cdot T(n-1)$ and $T(1) = 1$. Here is a faulty proof that $T(n) \leq O(n)$, by induction on $n$: the base case when $n = 1$ is easy, and

$$T(n) = 2 \cdot T(n-1) = O(T(n-1)) = O(n-1) = O(n)$$

where the third equality uses the induction hypothesis. Explain what is wrong with this proof and give a correct solution to the recurrence together with a correct proof.

(b) Prove that any recurrence of the form: $T(n) = O(T(n/2)) + O(n)$ and $T(1) = O(1)$, satisfies $T(n) \leq O(n^c)$ for some constant $c > 0$.

2. (a) For a directed graph $G$, and a strongly-connected component of $G$, denote by $f(C)$ the *latest* finishing time for a given DFS traversal of $G$. Prove that if there is an edge from a vertex in strongly-connected component $C_1$ to a vertex in strongly-connected component $C_2$ (with $C_1 \neq C_2$), then $f(C_1) > f(C_2)$.

(b) Let $G_{\text{SCC}}$ denote the DAG whose nodes are the strongly-connected components of $G$, with an edge from component $C_1$ to component $C_2$ iff there is an edge in $G$ from a vertex of $C_1$ to a vertex of $C_2$. Prove that a DFS traversal (Lecture 2, slide 12) of $G^T$ that selects vertices in line 3 by decreasing order of finishing times visits the strongly-connected components of $G^T_{\text{SCC}}$ in reverse topological order (i.e, the first component visited is a sink, and the next is a sink after the removal of the first component, etc...). This is the basis for the correctness of the algorithm from class for finding an SCC decomposition.

(c) With all the "reversals" in the previous part, it is tempting to try to identify a SCC decomposition by having the second DFS operate on $G$ (rather than $G^T$), and select vertices in line 3 by *increasing* (rather than decreasing) order of finishing times. Prove that this does not work: specify a small directed graph with two strongly-connected components, label them with discovery and finishing times for an initial DFS traversal, and show that the proposed second DFS produces a DFS forest with only one tree (instead of two, one covering each strongly connected component).

(d) Give an $O(m+n)$ time algorithm to determine if a directed graph $G$ contains an odd length (simple, directed) cycle. Hint: start by considering a strongly connected graph. Consider a DFS traversal that labels vertices with "odd" or "even" discovery times.

3. Consider a *comparison-based* implementation of an $n$-element max-heap: just as with sorting, this means that the implementation only ever makes choices based on *comparisons* between key values. Let $f(n)$ be the worst-case time for a single INSERT and let $g(n)$ be the worst-case time for a single EXTRACT-MAX operation. Prove that $f(n) + g(n) \geq \Omega(\log n)$.

4. Using the min-heaps from class with INSERT, DECREASE-KEY, and EXTRACT-MIN operations all taking $O(\log n)$ time, Dijkstra's algorithm can be implemented to run in $O(m \log n)$ time. Give an alternative implementation of Dijkstra's algorithm that runs in time

$$O\left(\frac{m \log n}{\log(m/n)}\right).$$

Hint: Construct heaps from $d$-ary trees for $d > 2$.

5. In a directed graph with non-negative weights, the *weight* of a path from $u$ to $v$ is the sum of the edges weights along the path. The *bottleneck weight* of a path from $u$ to $v$ is the *maximum* of the edge weights along the path. Give an algorithm to compute the smallest bottleneck-weighted paths from a source vertex $s$ to all other vertices, using a min-heap, with running time $O(n+m)$ plus the time for $n$ INSERT operations, $m$ DECREASE-KEY operations, and $n$ EXTRACT-MIN operations.