CMPSCI 311: Introduction to Algorithms **Fall 2016**

Homework 1

Released 9/9/2016 Due 8:00pm 9/23/2016 in Moodle

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

Instructions. You make work in groups, but you must individually write your solutions yourself. List your collaborators on your submission.

If you are asked to design an algorithm as part of a homework problem, please provide: (a) the pseudocode for the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.

Submission instructions. This assignment is by 8:00pm on 9/23/2016 in Moodle. Please submit a pdf file. You may submit a scanned handwritten document, but a typed submission is preferred.

1. **(10 points) Stable Marriage Running Time.** In class, we saw that the Propose-and-reject algorithm terminates in at most n^2 iterations, when there are n students and n colleges. Construct an instance so that the algorithm requires at least cn^2 iterations for some constant $0 < c \leq 1$ and prove that it requires cn^2 iterations. Your construction should be possible for all values of n.

2. **(20 points) Stable Marriages: K&T Ch 1, Ex 5.** Consider a version of the stable matching problem where there are n students and n colleges as before. Assume each student ranks the colleges (and vice versa), but now we allow ties in the ranking. In other words, we could have a school that is indifferent two students s_1 and s_2, but prefers either of them over some other student s_3 (and vice versa). We say a student s *prefers* college c_1 to c_2 if c_1 is ranked higher on the s's preference list and c_1 and c_2 are not tied.

   (a) **Strong Instability.** A strong instability in a matching is a student-college pair, each of which prefer each other to their current pairing. In other words, neither is indifferent about the switch. Does there always exist a matching with no strong instability? Either give an example instance for which all matchings have a strong instability (and prove it), or give and analyze an algorithm that is guaranteed to find a matching with no strong instabilities.

   (b) **Weak Instability.** A weak instability in a matching is a student-college pair where one party prefers the other, and the other may be indifferent. Formally, a student s and a college c with pairs c' and s' form a weak instability if either

   - s prefers c to c' and c either prefers s to s' or is indifferent between s and s'.
   - c prefers s to s' and s either prefers c to c' or is indifferent between c and c'.

   Does there always exist a perfect matching with no weak instability? Either give an instance with a weak instability or an algorithm that is guaranteed to find one.

3. **(10 points) Big-Oh.** For each function $f(n)$ below, find the smallest *integer* constant c such that $f(n) = O(n^c)$, or indicate that no such constant exists. All logarithms are with base 2.

   (a) $f(n) = \frac{1}{2}n^2$.

   (b) $f(n) = n(\log n)^3$

   (c) $f(n) = \sum_{i=0}^{\lceil \log n \rceil} \frac{n}{2^i}$.

   (d) $f(n) = \sum_{i=1}^{n} i^3$.

   (e) $f(n) = 2^{(\log n)^2}$

4. **(20 points) Asymptotics. K&T Ch 2, Ex 6.** Given an array A consistent of n integers, you'd like to output a two-dimensional $n \times n$ array B in which $B[i,j] = A[i] + A[i+1] + \ldots + A[j]$ for each $i < j$. For $i \geq j$ the value of $B[i,j]$ can be left as is.

   For $i = 1, 2, \ldots, n$
   For $j = i+1, \ldots, n$
   Add up $A[i] + A[i+1] + \ldots + A[j]$.
   Store in $B[i,j]$.

   (a) What is the running time of this algorithm as a function of n? Specify a function f such that the running time of the algorithm is $\Theta(f(n))$.

   (b) Design and analyze a faster algorithm for this problem. You should give an algorithm with running $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.

5. **(10 points) DFS and BFS. K&T Ch 3, Ex 5.** Suppose we have a connected graph $G = (V, E)$ and a vertex $u \in V$. If we run DFS from u, we obtain a tree T. Suppose that if we run BFS from u we obtain exactly the same tree T. Prove that $G = T$.

6. **(30 points) Disrupting Communication Networks.** Suppose we have an undirected connected graph $G = (V, E)$ representing a communication network (e.g., the vertices denote routers and the edges denote links). We'd like to disrupt communication in this network by removing a node so that the resulting in a highly fragmented network. Precisely, we'd like to identify a node $v \in V$ whose removal splits the graph into as many different connected components as possible. In this question, we'll design a linear time $O(|V| + |E|)$ algorithm for this problem.

   Let $f(v)$ denote the number of connected components in the graph once the vertex v is removed.

   (a) How can you use a DFS tree starting at some $r \in V$ to calculate $f(r)$ efficiently?

   (b) Suppose $v \in V$ is a node in the resulting DFS tree, $v \neq r$, and that no descendant of v has any non-tree edge to any ancestor of v. How could you calculate $f(v)$ from the DFS tree in an efficient way?

   (c) For each node in the DFS tree, let $d(v)$ denote the depth of v in the DFS tree (so r has $d(r) = 0$, r's children have depth 1, etc.) How can we compute $d(v)$ for all vertices $v \in V$ in linear time. Your algorithm should output an array `d` with `d[v]` $= d(v)$.

   (d) If w is a node in the DFS tree, let $\text{up}(w)$ denote the depth of the shallowest node y such that $\{x, y\} \in E$ is a graph edge and either $x = w$ or x is a descended of w. Let $\text{up}(w) = \infty$ if no such edge exists.

   If v is a non-root node in the tree, which children $w_1, \ldots, w_k$, how can we compute $f(v)$ as a function of $k, \text{up}(w_1), \ldots, \text{up}(w_k)$ and $d(v)$.

   (e) Describe how to compute $\text{up}(v)$ for each vertex $v \in V$ in linear time.

   (f) Describe how to compute $f(v)$ for each vertex $v \in V$ in linear time.

7. **(0 points).** How long did it take you to complete this assignment?