CSOR W4231 (sec. 002): Analysis of Algorithms Nov 11, 2024

Homework 5: Due on Nov 25, by 2:30pm

Instructor: *Alex Andoni*

Instructions. Please follow the homework policy and submission instructions in the Course Information handout. A few highlights to remember:

- submission is via GradeScope: `https://www.gradescope.com/` using code `KZBNBB`, where the **"student ID" must be your uni** (like `aa3815`, otherwise we can't match your homework to you);
- you are encouraged (but not required) to use LaTeX, using the provided template;
- follow the stated collaboration and academic honesty policies;
- write your name, uni, and collaborators on each problem;
- make sure to identify the start of each problem in GradeScope, including the ones that you didn't solve;
- if you don't know how to solve a particular part of a problem, just write "*empty*".

"Design an algorithm" entails proving both correctness and runtime bound; also the default expectation is for a deterministic algorithms (unless otherwise stated). Note that, for each bullet item of a problem, you can use the previous bullets as "black box", even if you didn't manage to solve them. Similarly, you can use anything from the lectures as black-box.

1 Problem 1 (20pts)

We are given a directed graph G by its adjacency list representation. Suppose each node v has a height $h(v)$. Give an algorithm that computes, for each node v, what is the highest height one can reach starting from the node v. The algorithm should run in linear time, $O(n+e)$, where n is the number of nodes and e the number of edges of the graph.

For example, if the the graph is strongly connected, then for each node v the answer is the same $\max_u h(u)$. Or, if the graph is a line graph going from left to right (a DAG), then the answer for node $i \in [n]$ is $\max_{j \geq i} h(j)$.

2 Problem 2 (25pts)

(a) Consider a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a connected graph $G = (V, E)$, partition the set V of vertices into two equal-sized sets V_1 and V_2 (you can assume that n is a power of 2). Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2. Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2), and

use this edge to unite the resulting two minimum spanning trees into a single spanning tree. Note that the runtime of the algorithm is $O(e \log n)$.

Either 1) argue that the algorithm correctly computes a minimum spanning tree of G, or, instead 2) provide an example for which the algorithm fails.

(b) Consider yet another greedy-like algorithm as follows. Assume all edge weights are distinct. We build the tree T iteratively, starting with $T = \emptyset$. Given the current set of edges T (subset of the eventual MST), let $C_1, \ldots C_k$ be the current connected components (with respect to the edges from the set T); in particular $k = n - |T|$. For each connected component C_i we choose an edge (u, v) where $u \in C_i$ and $v \notin C_i$ of *minimal weight*. Note that we choose k such edges. Then add these edges to the set T. Note that an edge between components C_i, C_j may be added twice (once from each side), in which case we add just one copy of the edge. We repeat until one component remains.

First, prove that the number of iterations is $O(\log n)$. Then, either 1) argue that the algorithm correctly computes a minimum spanning tree of G, or, instead 2) provide an example for which the algorithm fails.

3 Problem 3 (30pts)

We are given a weighted undirected graph $G = (N, E)$ (by its adjacency list representation), with a specified source node s. (You can think of it as representing a road network, with s a place of interest, like warehouse of a company.) You may assume that all weights are positive integers (≥ 1).

a) We call a road (u, v) to be *important* if it lies on some shortest path from s to $t \in N$. Design an algorithm to report all the roads that are important. Expected runtime: $O((n + e) \log n)$.

b) Now fix some target node t. We call a road (u, v) *critical* if every shortest path (from s to t) passes through this road (in other words, if we delete (u, v) the shortest path length to t increases). Design an algorithm to report all the roads that are critical. (Note that none may exist.) Expected runtime: $O(n + e)$ (in addition to (a)). *Hint*: first extract the important edges specifically for the shortest paths towards t and then see how to determine critical edges in the resulting graph.

Here's an example. Consider graph with edges/weights $(u, v, w_{u,v})$ being:
$(1, 2, 5), (1, 3, 2), (3, 2, 3), (2, 4, 7), (1, 4, 20)$. Then the important edges are $(1, 2), (1, 3), (3, 2), (2, 4)$ and the critical edges, for $t = 4$, are $(2, 4)$ (but for $t = 2$, there are no critical edges).

4 Problem 4 (25pts)

Arbitrage is the use of discrepancies in currency exchange rates to transform one unit of a currency into more than one unit of the same currency. For example, suppose that 1 U.S. dollar buys 64 Indian rupees, 1 Indian rupee buys 1.8 Japanese yen, and 1 Japanese yen buys 0.009 U.S. dollars. Then, by converting currencies, a trader can start with 1 U.S. dollar and buy $64 \times 1.8 \times 0.009 = 1.0368$ U.S. dollars, thus turning a profit of 3.68 percent.

Suppose that you are given n currencies $c_1, c_2, \ldots, c_n$ and an $n \times n$ table T of exchange rates, such that one unit of currency c_i buys $T[i, j]$ units of currency c_j. Give an efficient algorithm to determine whether or not there exists a sequence of currencies $\langle c_{i_1}, c_{i_2}, \ldots, c_{i_k} \rangle$ such that $T[i_1, i_2] \cdot T[i_2, i_3] \cdots T[i_{k-1}, i_k] \cdot T[i_k, i_1] > 1$.

Your algorithm should just return yes or no. (Bonus: find the actual sequence.)