CSCI E-124 Data Structures and Algorithms — Spring 2015

Problem Set 3

Due: 11:59pm, Wednesday, March 4th

See homework submission instructions at
`http://sites.fas.harvard.edu/~cs124/e124/problem_sets.html`

Problem 5 is worth 40% of this problem set, and problems 1-4 constitute the remaining 60%.

1 Problem 1

In the Minimum Spanning Tree (MST) problem we would like to find a spanning tree whose *sum* of weights is minimum when compared to all other spanning trees. An Internet Service Provider (ISP) might use the solution to such a problem to determine the network of least cost (and therefore most profit) where each edge cost is measured in dollars ($). Suppose instead that the ISP wishes to maximize their profit in another way. The cost of each edge now measures the bandwidth (in Mb/s), and the ISP wishes to find a spanning tree (of its network) whose *maximum* bandwidth is minimum, thereby guaranteeing the smallest possible maximum transfer speed on the network.

In the following three problem parts, you may assume you have access to a "selection algorithm" $T(A, k)$ such that, given a length n array A of numbers and an integer $1 \leq k \leq n$, $T(A, k)$ outputs the kth smallest number in A (breaking ties arbitrarily). For example $T([6, 10, 5, 1], 2)$ would return 5. $T([6, 10, 5, 5], k)$ would return 5 both for $k = 1$ and $k = 2$ and would return 6 for $k = 3$. The running time of T is $\Theta(n)$.

(a) Before hiring you, the ISP wants to make sure you know your stuff. Show that any MST is a solution to the ISPs new problem.

(b) Great! However, the ISP still isn't satisfied. Show that the ISPs new problem can be solved in time $O(n + m)$ when all bandwidths are distinct in an undirected network with n customers and m links.

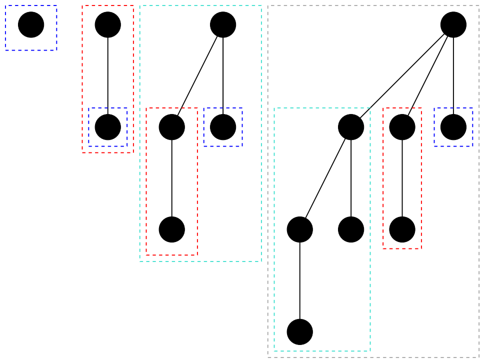
(c) For the final touch (before you get the job!), modify your algorithm from (b) to handle the case when bandwidths might be equal.

2 Problem 2

As you'll recall from lecture, the number of spanning trees of a graph G can grow rather quickly (there are n^{n-2} for the complete graph). A Civil Engineer is currently considering a road network for his hometown and, as is natural, wishes to minimize the cost of building such network. The road network can be thought of as a graph $G = (V, E)$ where each $e \in E$ is a road segment connecting two intersections, $u, v \in V$. Suppose the Civil Engineer has access to the SpanTree software suite (through his job). SpanTree works as follows: when given a weighted connected undirected graph G on n vertices, SpanTree(G) returns the number of spanning trees of G modulo 2015 in time $T(n)$. Note the lack of the word "minimum". Using the SpanTree software as a black box, the Civil Engineer wants to figure out just how many possible road networks fit his criteria, modulo 2015. Show that the number of minimum cost road networks of G modulo 2015 can be computed in time $O(nT(n) + m \log n)$. You can assume $T(\cdot)$ is a monotonically increasing function, and for the purposes of the story, will be small enough for the Civil Engineer to be home in time for dinner.

3 Problem 3

In class we saw that with the disjoint forests data structure, if you use both "union by rank" and "path compression", a sequence of n make-set and $m \geq n$ find and union operations takes time $O(m \log^* n)$. As you probably noticed, the proof for that bound is non-trivial, which might leave you wondering, could I possibly simplify the data structure and still achieve this bound? Along that train of thought, suppose we use only one of "union by rank" or "path compression", but not both. Show that if we only use union by rank, there is an infinite family of operation sequences (with m, n going to ∞) such that the total runtime to serve a sequence in this family is $\Omega(m \log n)$. Similarly, show that if we only use path compression, we can also make the total time $\Omega(m \log n)$. In both of your example sequences, m should be $\Theta(n)$. **Hint**: you may find the following *sequence* of trees useful to think about:



4 Problem 4

You know what can get tough when running an automated factory? Checking that each robot is up and running! Suppose each robot has a set schedule on which it runs. For simplicity, a robot can only run once and must run continuously until some end time. We can then represent such a schedule as n horizontal line segments in the plane. The ith segment has some height h_i (which may be negative and represents the equity on robot i)

and runs from the start time, $x = a_i$, to the end time, $x = b_i$ ($a_i < b_i$). For some bizarre reason, the line segments are half-open: they contain their end time, but not the start time. For simplicity, "checking in" at the factory can be represented by a vertical line (note *line* and not *line segment*), and can occur at non-integer times. The only limitation that exists is that each robot (horizontal line segment) is checked on (intersected by a vertical line) at least once.

(a) Help the manager solve this problem! Give a greedy algorithm which minimizes the total number of "check-ins" and prove its correctness. The running time should be $O(n \log n)$.

(b) However, the robots actually form part of a union, and are rather wary of being constantly checked on. Suppose the objective was instead to minimize the maximum number of times that any robot is checked on. That is, each robot should be checked on at least once, and *at most* z times such that z is minimized. Show that if the optimal solution for a given instance achieves a value of z_{opt}, then there is a greedy solution which is optimal for part (a) while achieving $z \leq z_{opt} + 1$ for this new objective.

(c) Well, the robots really *do* insist that you try your best to absolutely achieve z_{opt}. Assuming that the statement of (b) is true, derive an algorithm which achieves the objective of (b) but with $z = z_{opt}$. Any algorithm running in polynomial time (i.e. $O(n^c)$ for some constant c) will receive full credit because the robots are on the verge of mutiny.

5 Problem 5

Solve "Problem A - Game" on the programming server; see the "Problem Sets" part of the course web page for the link.