CS 256: Algorithm Design and Analysis

Assignment 3 (due 10/07/20)

Instructor: Sam McCauley

All problems are worth 10 points. Latex typsetting is worth 5 points.

Greedy Stays Ahead and Exchange Argument

Problem 1. Let X be a set of n intervals on the real line. We say that a set P of points stabs X (or is *the stabbing set*) if every interval in X contains at least one point in P. See Figure 1. In this and the next question, we will design and analyze an efficient algorithm to compute minimum set of points that stabs X. Assume that your input consists of pairs (ℓ_i, r_i), representing the left and right endpoints of ith interval in X, for $1 \leq i \leq n$.

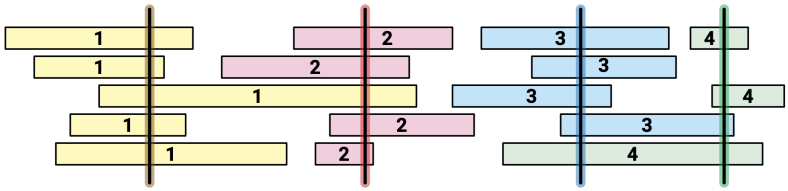

Figure 1: A set of intervals stabbed by four points. From Jeff Erickson's Algorithms book.

(a) *Structure of optimal.* If we were to choose stabbing points anywhere on the intervals, the possibilities would be endless. So first, we prove a structural property about any optimal solution to the problem, as this will guide our algorithm design. Show that for any set of intervals, there exists an optimal solution (a stabbing set of minimum size) where every stabbing point is the right endpoint r_i of some interval. This means that without loss of generality (WLOG), we can restrict ourselves to stabbing sets that satisfy this property.

Consider the following greedy strategy: sort the intervals in increasing order of their right endpoints. Take the first interval from this list, and add its right endpoint to the stabbing set. Remove all intervals that contain this point. Repeat until no intervals remain. Clearly, this produces a valid stabbing set, but is it optimal?

(b) *Greedy is optimal.* Prove that the above greedy strategy is optimal, i.e., it produces a stabbing set of minimum size. (*Hint.* Use (a) and compare to an optimal solution where every stabbing point is a right endpoint of some interval.)

(c) *Running time of greedy.* Analyze the running time of the greedy strategy and show that it can be implemented in $O(n \log n)$ time.

Solution. □

Problem 2. A store that sells glasses is almost out of stock—they only have n pairs of glasses remaining. Yesterday, the store received a large number of orders. By chance, they received exactly n orders.

This seems like great news, as every customer's order can be filled! Unfortunately, now the store must decide who gets which pair of glasses.

Each customer has a desired prescription lens power for their glasses $p_1, p_2, \ldots p_n$ (so p_i is customer i's desired lens power). The shop has glasses with power $g_1, g_2, \ldots g_n$. Each lens power is a floating point number. There's no guarantee that each customer has a perfect match—some customers may be assigned their desired power, whereas others may need to make do with lenses that are slightly off.

Our goal is to match glasses to customers so that the prescription power matches as closely as possible. That is to say, we want to minimize the sum (over all customers) of how close their desired prescription is to the power of the classes they are assigned. That is to say, we want to find an assignment A (so customer i receives glasses with index $A[i]$ and prescription power $g_{A[i]}$) to minimize

$$\sum_{i=1}^{n} \left| p_i - g_{A[i]} \right| .$$

Give an efficient algorithm for this problem, analyze its running time, and prove that it always gives the correct solution.

Solution. □

Greedy Algorithms on Graphs

Problem 3. Are the following statements True of False? You must justify your choice with a brief explanation or a counterexample.

(a) The minimum spanning tree of a graph includes the minimum-weight edge in *every* cycle.
(b) Kruskals algorithm for finding minimum spanning trees works for negative edge weights.
(c) Dijkstra's shortest path algorithm works even on graphs with negative edge weights.

Solution. □

Problem 4. It is often the case that updating the output of an algorithm when the input changes is easier than computing it from scratch. One such example is that of dynamically changing edge weights in a graph. This problem asks you to consider how you would maintain a minimum-cost spanning tree in such an environment. Suppose you are given a graph G with weighted edges and a minimum spanning tree T of G. Describe an algorithm to update the minimum spanning tree when the weight of a single edge e is decreased. The input to your algorithm is the edge e and its new weight; your algorithms should modify T so that it is still a minimum spanning tree. State and justify the running time of your algorithm. You may assume that all edge costs are distinct. *Hint: Consider $e \in T$ and $e \notin T$ separately and use the properties of cycles and cuts.*[1]

Solution. □

[1] Food for thought (no need to answer): how would you update T if weight of a single edge is increased?

Problem 5. A *feedback edge set* in a graph G is a subset F of the edges of G such that every cycle in G contains at least one edge in F. In other words, removing every edge in F makes the graph G acyclic. Notice that the set of all the edges of the graph trivially forms a feedback edge set. Design and analyze a fast algorithm to compute the *minimum-weight* feedback-edge set of a given edge-weighted graph. Assume that G is connected. (*Hint.* Reduce this problem to some kind of spanning tree problem.)

Solution. □