Polytechnic Institute of NYU *CS6033: Design&Analysis of Algorithms I — Fall 2013*
Prof. Boris Aronov (aronov@poly.edu) RH315 Mon 6-8:30pm
Office: 2MTC 10.008 (718) 260-3092 Office hours: Mon 4-6pm

Homework #9: Greedier.

Due December 9, 2013.
Refer to HOMEWORK RULES on PIAZZA.

#1: Recall the activity selection problem, our first example of a greedy algorithm:

> ACTIVITY SELECTION: Given a collection I of intervals $[a_i, b_i]$, $i = 1, \ldots, n$ (for simplicity, assume the $2n$ numbers a_i, b_i are distinct), find a subcollection of pairwise disjoint intervals from I that contains the maximum number of intervals.

We will consider a generalization of this problem:

> WEIGHTED ACTIVITY SELECTION: Given a collection I of intervals $[a_i, b_i]$, with interval i given real weight $w_i > 0$, $i = 1, \ldots, n$ (again numbers a_i, b_i are distinct), find a subcollection of I so that (a) the intervals are pairwise disjoint and (b) their total weight is as large as possible.

Notice that this is indeed a generalization of ACTIVITY SELECTION, as we get the original problem back by setting all $w_i = 1$, so that maximizing the total weight of a collection of intervals and maximizing their number is the same.

The motivating example is as follows: Again, you are renting out an auditorium. The intervals are the different time slots for which different events want to rent it. But now event i will pay you rent w_i and you want to pick a collection of events that (a) do not conflict with each other and (b) bring you the maximum total amount of money.

Now we try a few greedy heuristics for this problem. For each of these heuristics, prove that it *does not work*, which means that there is an input on which its output is *not* the optimal answer. Try to find a simple example—for most heuristics below there is an example that shows that it fails with five, and sometimes even only two, intervals. (Give a bad example for each heuristic, even though sometimes there are other ways to prove the heuristic is bad.)

(i) EARLIESTENDTIMEFIRST: pick the event i with the earliest ending time b_i. Add it to your set. Remove all events conflicting with it. Repeat.

(ii) LONGESTFIRST: pick the event i with longest duration $b_i - a_i$. Add to the output. Remove conflicts. Repeat.

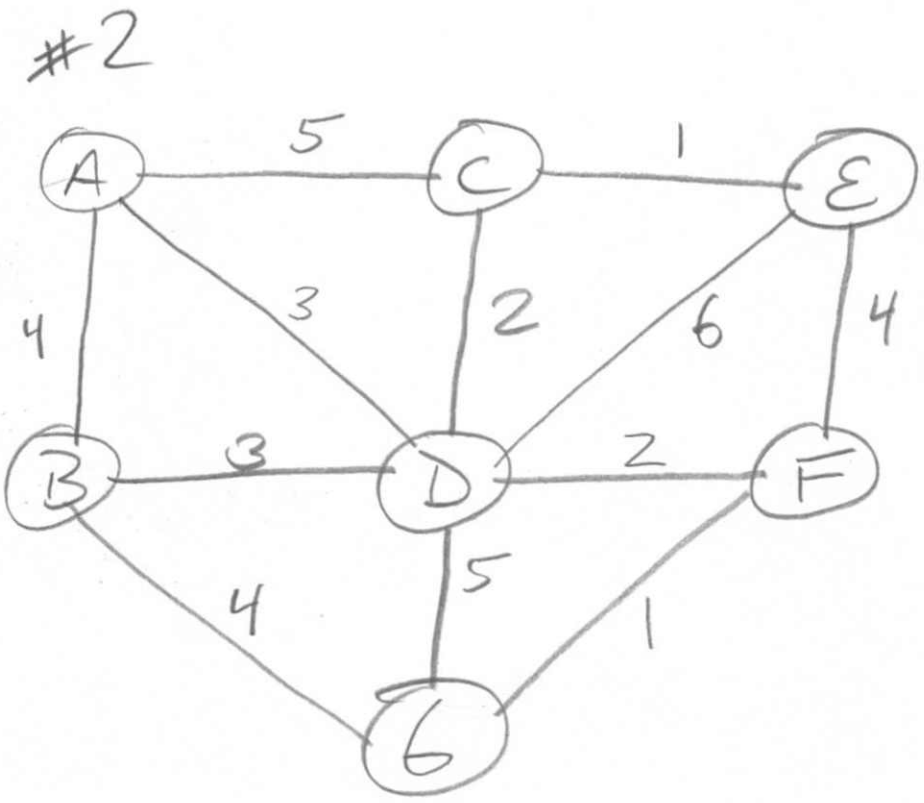
(iii) SHORTESTFIRST: pick the event i with shortest duration $b_i - a_i$. Add to the output. Remove conflicts. Repeat.

(iv) HIGHESTPRICEFIRST: pick the event i with highest price w_i. Add to the output. Remove conflicts. Repeat.

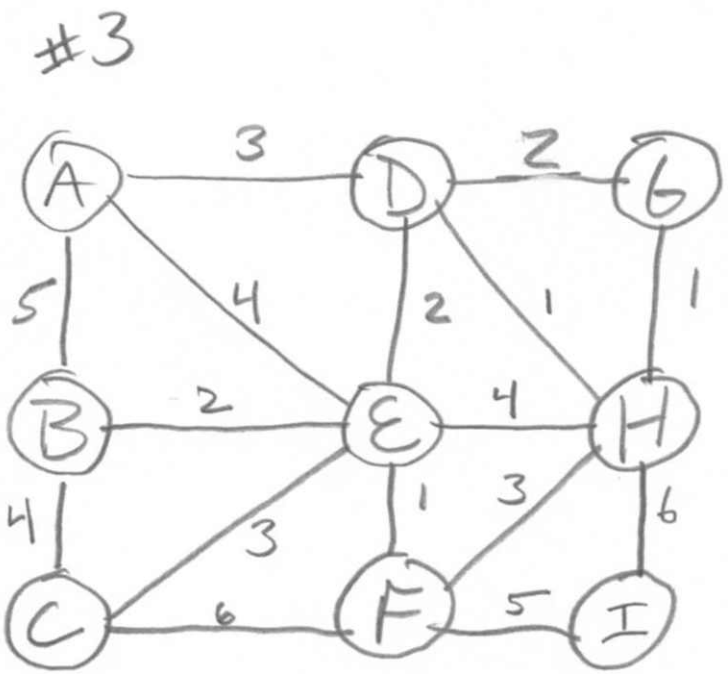
(v) HighestPricePerHourFirst: pick the event i with highest price per hour $w_i/(b_i - a_i)$. **Fixed! Was: "$(b_i - a_i)/w_i$".** Add to the output. Remove conflicts. Repeat.

(vi) Give an example of another, different heuristic for this problem that makes sense. Show by an example that it does not work either.

#2: Run Kruskal's MST algorithm on the following graph. List the edges in the order they are considered by the algorithm, mark the edges included in the MST and draw the resulting MST.



#3: Run Prim's MST algorithm on the following graph, starting at vertex A. Show the minimum weights for each vertex at each step of the algorithm, along with the predecessor values. (Refer to the description of the algorithm in the textbook.)



#4: During the holiday weekend you had to solve the following optimization problem:

WHERE TO RECHARGE: You have to drive along a highway, from your starting point at mile 0 (kilometers are OK too, for metric people :-) to your destination at mile M; we will model the road as an interval $[0, M]$. You are driving a new and amazing all-electric vehicle that has a replaceable set of batteries. The batteries cannot be recharged. Along the way there are service stations, where you can replace your current batteries with new ones. You can carry only one set of batteries with you at any given point. If you run out of charge before reaching a service station, you fail.

The problem is that some service stations give you good batteries that last a long time, while others give you bad ones. You are given the information about the n service stations i, $i = 1, \ldots, n$, as follows. Station i is described by an interval $[s_i, d_i]$, where s_i is the position of the station on the road and d_i is the position on the road where the battery received at station i will die (run out of charge). In other words, if you decide to replace your batteries at that station, the battery will be OK over the interval $[s_i, d_i]$ of the road. If $d_i < M$, you need to stop at another service station at or before d_i, or your car will run out of charge and you fail.

You may assume there is a charge station number 1, with interval $[0, d_1]$, so that you

start with working batteries.

Changing batteries is very slow (or maybe very expensive), so you want to get from 0 to M in as few battery changes as possible. Find where you need to stop to achieve that. (It is also possible that for the given input you cannot reach your destination M. Your algorithm should detect that.)

(i) Re-state the problem in precise mathematical language. What's the input? What's the output? What needs to be optimized?

(ii) Give a simple heuristic that does *not* work, i.e., that does not always produce the optimal answer. Show a simple example where it fails.

(iii) Give a correct heuristic. Prove that it produces the correct answer for all possible inputs.

(iv) Describe an efficient implementation. Analyze its worst-case running time.

Reminder: In the world of greedy and dynamic programming algorithms it is very easy to construct heuristics that seem to do sensible things. It is not always very easy to justify their correctness. And for some problems there are no known greedy algorithms and it is very hard to argue that such a thing is impossible: it is less difficult to argue that a given greedy heuristic is wrong, by providing an example where it does not find the correct optimum answer.