Homework 3. Due February 15

CS180: Algorithms and Complexity
Winter 2017

GUIDELINES:

- Upload your assignments to Gradescope by 6:59 PM.
- Follow the instructions mentioned on the course webpage for uploading to Gradescope very carefully (including starting each problem on a new page and matching the pages with the assignments); this makes it easy and smooth for everyone. As the guidelines are simple enough, bad uploads will not be graded.
- You may use results proved in class without proofs as long as you state them clearly.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly. Homework is a stepping stone for exams; keep in mind that reasonable partial credit will be awarded and trying the problems will help you a lot for the exams.

1. Consider the interval scheduling problem we studied in class: Given a sequence of requests with start and finish times $(s(i), t(i))$, $i = 1, \ldots, n$, find a set of non-conflicting jobs of maximum possible size. Show that the following algorithm solves the problem correctly (i.e., returns a set of non-conflicting jobs of maximum size).

   LATEST START TIME (LST):

   (a) Set $R \leftarrow \{1, \ldots, n\}$, and $A \leftarrow \emptyset$.

   (b) While $R \neq \emptyset$:

      i. Pick request $i \in R$ with the latest start time.

      ii. Add i to A.

      iii. Remove all requests that conflict with i (including i) from R.

   (c) RETURN A.

   For full-credit, your answer must be comparable in detail to our analysis in class. [.75 points]

   (Hint: You can follow the same approach we used for analyzing EARLIEST FINISH TIME in class.)

   (Thanks to Andrew Xue for asking a question during office hours that led to this problem.)

2. Problem 5.1 from [DPV]. You should break ties between edges of same weight (if needed) in lexicographic order. [.75 points]

   (In second part of part(c), "For each edge in this sequence, give a cut that justifies its addition." means to find a cut for which the edge being added is an edge of minimum weight crossing the cut.)

3. You are given a connected graph G with n vertices and m edges, and a minimum spanning tree T of the graph. Suppose one of the edge weights $c(e)$ for an edge $e \in T$ is updated. Give an algorithm that runs in time $O(m)$ to test if T still remains the minimum spanning tree of the graph. You may assume that all edge weights are distinct both before and after the update. Explain why your algorithm runs in $O(m)$ time and is correct. [.75 points]

   (Hint: Consider the cut obtained by deleting e from T. Note that as you are only allowed $O(m)$ time, you cannot recompute a MST in the new graph from scratch. Also, make sure you specify your algorithm fully.)

4* Given an undirected graph $G = (V, E)$, a subset of vertices $I \subseteq V$ is an independent set in G if no two vertices in I are adjacent to each other. Let $\alpha(G) = \max\{|I| : I \text{ an independent set in } G\}$. The goal of the following questions is to give an efficient algorithm for computing an independent set of maximum size in a tree. Recall that a *leaf* in a graph is a vertex of degree at most 1 and that every acyclic graph (graph without any cycles) has at least one leaf.

   Let $T = (V, E)$ be an acyclic graph on n vertices.

   (a) Prove that if u is a leaf in T, then there is a maximum-size independent set in T which contains u. That is, for every leaf u, there is an independent set I such that $u \in I$ and $|I| = \alpha(T)$. [.3 points]

   (b) Give the graph T as input (in adjacency edge representation), give an algorithm to compute an independent-set of maximum size, $\alpha(T)$, in T. To get full credit your algorithm should run in time $O(|V| \cdot |E|)$ (or better) and you must prove correctness of your algorithm. You don't need to analyze the time-complexity of your algorithm and it is sufficient to solve this problem assuming part (1) (if you want) even if you don't solve it. [.45 points]

   (Hint: You can try a greedy approach where you add vertices one after the other based on property (1).)

ADDITIONAL PROBLEMS. DO NOT turn in answers for the following problems - they are meant for your curiosity and understanding.

1. Problems 4.1, 4.2, 4.3, 4.7, 4.13, 4.14 from textbook [KT].

2. Problem 5.5, 5.6, 5.7 from Chapter 5 of [DPV].