CENG 567 Design and Analysis of Algorithms Fall 2020

# Homework 3 - Greedy Algorithms

Released: 20/11/2020 Due: 30/11/2020

**Instructions.** You may work with other students, but you must individually write your solutions **in your own words**. If you work with other students or consult outside sources (such as Internet/book), cite your sources.

If you are asked to design an algorithm, provide:
(a) a description of the algorithm in English and, if helpful pseudocode,
(b) at least one worked example or diagram to show more precisely how your algorithm works,
(c) a proof (or indication) of the correctness of the algorithm,
(d) an analysis of the running time of the algorithm.

**Submissions.** Submit a pdf file through odtuclass. LaTeX or Word typed submission is required.

1. **Job scheduling**
Consider distributed indexing of documents. You have one supercomputer and an unlimited number of affordable PCs. The overall process is broken into $n$ distint jobs which can be performed independently. Each job has two steps: preprocessing and indexing. The preprocessing step is performed on the supercomputer and the indexing is finished on one of the PCs. Thus, each job needs $p_i$ preprocessing time and $f_i$ finishing the index time. While the supercomputer can work on only a single job at a time, the PCs can work independently, finishing the jobs in parallel. When the first job in order is done on supercomputer, it is passed to a PC for indexing; at that point the next job is fed to the supercomputer and when its preprocessing step is done, it is passed to a (possibly different) PC without waiting for the first job to finish.

You want to minimize the completion time which is the earliest time at which all jobs will have finished processing on the PCs. Give a polynomial-time greedy algorithm that finds an ordering of the jobs for the supercomputer with as small completion time as possible. Describe your algorithm by explicitly stating the greedy choice you make. Prove that your algorithm minimizes the completion time. Also give the running time complexity of your algorithm.

2. **Spanning Tree**
Assume you are given a graph $G(V, E)$ where each edge is labelled *red* or *white*. Give a polynomial-time algorithm that takes $G$, and either (i) returns a spanning tree with exactly $k$ edges labelled *red*, or (ii) reports correctly that no such tree exists. Note that this can be a valid real world problem. Consider a connected graph of $n$ nodes where each node is a

district and the edges represent roads. Two municipalities maintain a subset of roads. Thus each edge is labeled by one of the municipalities. Suppose a spanning tree of the graph will be chosen to upgrade the roads corresponding to the edges of the spanning tree, with the constraint that $k$ edges will belong to one municipality and $n-k$ edges will belong to the other.

3. **Shortest-paths and min spanning trees**
a) Suppose you are given a weighted graph $G(V, E)$ with a distinguished node $s$ and where all edge weights are positive and distinct. Is it possible for a tree of shortest paths from $s$ and a minimum spanning tree in $G$ to not share any edges? If so, give an example. If not, give a reason.
b) In class we discussed the minimum-cost arborescence in directed graphs. Now, consider the case for Directed Acyclic Graphs (DAG). As in general directed graphs, there can be many minimum-cost solutions. Suppose you are given a directed acyclic graph $G(V, E)$ and an arborescence $A \subseteq E$ such that for every $e \in A$, $e$ belongs to *some* minimum-cost arborescence in $G$. Can we conclude that $A$ itself must be a minimum-cost arborescence in $G$? Give a proof or a counter example with explanation.