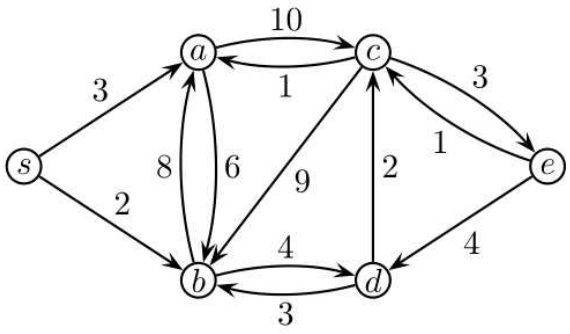
**CSE421: Introduction to Algorithms** January 19, 2022

# Homework 3

*Yin Tat Lee* Due: Jan 26, 2022 (before class)

Unless otherwise mentioned, you always need to prove that your algorithm halts, shows it runtime and outputs the correct answer. See Homework Guideline in Ed for more details.

1. (5 Marks) Run Dijkstra's algorithm in the following instance with source node $s$. Edges $(u, v)$ are labelled with their length $\ell_{uv}$.




   You simply need to write down the list of vertices according to the order Dijkstra's algorithm visited and their distance to $s$.

2. (5 Marks) In class we discussed that we can solve the interval scheduling problem by sorting all jobs in the order of their finishing time. Show that if we sort the jobs in the order of the starting times, the greedy algorithm may not return the optimum. In other words, construct a set of jobs with their starting times and finishing times such that if we sort the jobs based on their starting times the greedy algorithm schedules a smaller number of jobs than the optimum.

3. (10 Marks) A small business faces the following scheduling problem: Each morning they get a set of jobs from customers. They want to do the jobs on their single machine in an order that keeps their customers happiest. Customer i's job will take $t_i$ time to complete. Given a schedule (i.e., an ordering of jobs) let $C_i$ denote the finishing time of job $i$. For example, if job $j$ is the first to be done, we would have $C_j = t_j$; and if job $j$ is done right after job $i$, we would have $C_j = C_i + t_j$. Each customer $i$ also has a given weight $w_i$ that represents his or her importance to the business. So, the company wants to order the jobs to minimize weighted sum of completion times, $\sum_{i=1}^{n} w_i C_i$. Design a polynomial time algorithm that orders the jobs so as to minimize the weighted sum of the completion times.

   (Hint: the proof uses the exchange argument.)

4. (10 Marks) Given a long sequence $S$ of stocks to buy in order, your friends want an efficient way to detect certain patterns in them – for example, they may want to know if the four stocks:

Apple, Amazon, GameStop, Apple

occur in this sequence $S$, *in order* but not necessarily consecutively.

They begin with a collection of possible companies and a sequence $S$ of $n$ of them. A given company's stock may be bought multiple times in $S$. For example, the sequence of four companies above is a subsequence of the sequence

Amazon, Apple, Meta, Amazon, GameStop, Microsoft, Apple.

Their goal is to be able to dream up short sequences and quickly detect whether they are subsequences of $S$. So this is the problem they pose to you:

Given two sequences $S'$ and $S$ of companies of length $m$ and $n$ (each possibly containing a company more than once), where $m \leq n$, describe an $O(n)$-time greedy algorithm that decides whether $S'$ is a subsequence of $S$.

5. (**Extra Credit)** Up to now, all the greedy algorithms discuss in the class involve picking subsets. So, one may ask for under what general conditions, greedy method works for picking an optimal subset. In this question, we explore one of such condition.

   Let $[n] = \{1, 2, 3, \cdots, n\}$ and $\mathcal{I}$ be a collection of subsets of $[n]$. We call any set $I \in \mathcal{I}$ is nice.

   We know that $\mathcal{I}$ satisfy two main axioms:

   (a) If $X \subset Y$ and $Y \in \mathcal{I}$, then $X \in \mathcal{I}$. Namely, any subset of a nice set is nice.

   (b) If $X \in \mathcal{I}$, $Y \in \mathcal{I}$ and $|Y| > |X|$, then there exists $i \in Y \backslash X$ such that $X \cup \{i\} \in \mathcal{I}$. Namely, if $X$ is nice and there exists a larger nice set $Y$, then $X$ can be extended to a larger nice set by adding an element of $Y \backslash X$.

   The collection $\mathcal{I}$ may have exponentially size and is only defined implicitly. However, we assume that we can test if a set $I$ is nice or not in polynomial time.

   Given a cost $c_1, c_2, c_3, \cdots, c_n$, design a greedy polynomial time algorithm to find a nice set $X$ with maximum total cost $c(X) = \sum_{x \in X} c_x$.