CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Problem Set 7

Harvard SEAS - Fall 2021 *Due: Wed Nov. 3, 2021 (5pm)*

Your name:
Collaborators:
No. of late days used on previous psets:
No. of late days used after including this pset:

The purpose of this problem set is practice proving optimality and efficiency of a greedy algorithm, practice modelling problems using graphs, and reinforce understanding of the matching algorithm covered in class.

1. (Greedy Coloring for Interval Scheduling) The IntervalScheduling-Optimization problem we studied in class finds the largest group of nonintersecting intervals. In many applications, it is also natural to consider the *coloring* version of the problem, where we want to partition the input intervals into as few groups as possible so that each group is nonintersecting.

   In this problem, you will prove that Greedy Coloring in order of *increasing start time* gives optimal coloring for interval scheduling. (Note the contrast with the *increasing finish time* ordering we used for the version studied in class. It is a common phenomenon that different orderings are better for coloring vs. independent-set problems; for example decreasing vertex degree is a good heuristic for greedy coloring of general graphs, while increasing vertex degree is a good heuristic for independent set.) Let $x = (x_0, \ldots, x_{n-1})$ be an instance of IntervalScheduling, where each x_i is an interval $[a_i, b_i]$ with $a_i, b_i \in \mathbb{Q}$. Let k be the maximum number of input intervals that contain any value $t \in \mathbb{Q}$. That is,

   $$k = \max_{t \in \mathbb{Q}} |\{i \in [n] : t \in x_i\}|.$$

   (a) Prove that every proper coloring for IntervalScheduling uses at least k colors.

   (b) Show that the Greedy Coloring in order of *increasing start time* uses at most k colors. (To develop your intuition, carry out the algorithm on a few examples.)

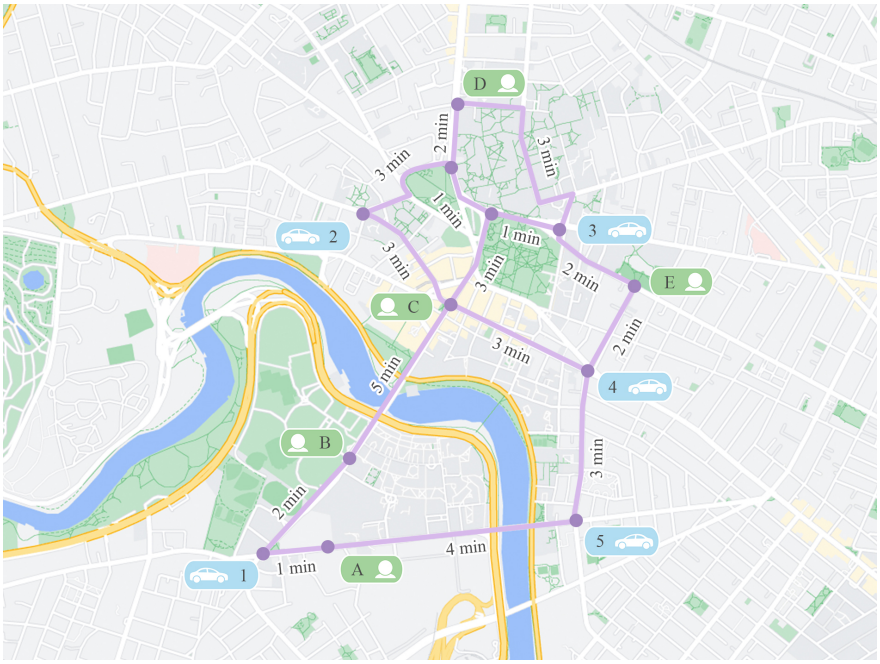
   (c) Show that the Greedy Coloring in order of increasing start time can be implemented in time $O(n \log n)$. Hints:

      i. Keep track of the end times of the most recently scheduled intervals assigned to each color, and use an appropriate data structure to ensure that you spend only $O(\log k)$ rather than $O(k)$ time per iteration, where k is the number of colors used.

      ii. To make life easier for yourselves, you may instead implement a *variant* of Greedy Coloring in which, at every step, you assign a vertex *any color* not assigned to its neighbours that's also less than the largest color (as opposed to standard Greedy Coloring in which you assign the smallest color).

2. (Bipartite Matching) One practical application of matching algorithms is planning logistics, like in the following example from (fictional) ridesharing service Lyber. When a customer

opens the Lyber app and books a ride, these ride requests are sent to some Lyber server and combined with other requests to create a graph like the one drawn in the map below:



Given a graph like this, Lyber's goal is to serve as many customers (labelled A–E in the map) as possible, by assigning each one to a driver (labelled 1–5 in the map). However, the one twist is that they want to make sure that *no customer is waiting for longer than 5 minutes.* They also do not want to assign a driver to more than one customer at once, since serving a single customer can take more than 5 minutes.

(a) To perform the assignment, they reduce to Maximum Matching in bipartite graphs. Draw a bipartite graph corresponding to the drivers and customers in the map above.

(b) Find a maximum matching in the bipartite matching graph you've drawn using the algorithm from class. Draw pictures showing the sequence of matchings and augmenting paths you find. (No need to break down the steps of the algorithm to find the augmenting paths.)