Midterm Examination

CS 4104 (Fall 2021)

Assigned: October 14, 2021.
PDF solutions due on Canvas by 11:59pm on October 21, 2021. **Extended to 11:59pm on October 25, 2021.**

Instructions

- The Undergraduate Honor Code applies to this examination. **Unlike in the case of homework, you must work on the examination individually.**
- You are not allowed to consult sources other than your textbook, the slides on the course web page, your own class notes, the TAs, and the instructor. In particular, do not use a search engine.
- Do not forget to typeset your solutions. *Every mathematical expression must be typeset as a mathematical expression, e.g., the square of n must appear as n^2 and not as "n^2".* You can use the LaTeX version of the homework problems to start entering your solutions.
- Do not make any assumptions not stated in the problem. If you do make any assumptions, state them clearly, and explain why the assumption does not decrease the generality of your solution.
- You must also provide a clear proof that your solution is correct (or a counter-example, where applicable). Type out all the statements you need to complete your proof. *You must convince us that you can write out the complete proof. You will lose points if you work out some details of the proof in your head but do not type them out in your solution.*
- If you are proposing an algorithm as the solution to a problem, keep the following in mind (the strategies are based on mistakes made by students over the years):
  - Describe your algorithms as clearly as possible. The style used in the book is fine, as long as your description is not ambiguous. Explain your algorithm in words. A step-wise description is fine. *However, if you submit detailed code or pseudo-code without an explanation, we will not grade your solutions.*
  - Do not describe your algorithms only for a specific example you may have worked out.
  - Make sure to state and prove the running time of your algorithm. You will only get partial credit if your analysis is not tight, i.e., if the bound you prove for your algorithm is not the best upper bound possible.
  - You will get partial credit if your algorithm is not the most efficient one that is possible to develop for the problem.
- In general for a graph problem, you may assume that the graph is stored in an adjacency list. If n is the number of nodes and m is the number of edges in the graph, then the input size is $m + n$. Therefore, a linear time graph algorithm will run in $O(m + n)$ time.

Good luck!

Problem 1 (15 points) Let us start with some quickies. For each statement below, say whether it is true or false. *You do not have to provide a proof or counter-example for your answer.*

1. Every tree is a bipartite graph and every bipartite graph is a tree.
2. $\sum_{i=1}^{n} \log i = \Omega(n \log n)$.
3. The costliest edge in an undirected graph can never be in any MST for that graph.
4. In a directed graph, $G = (V, E)$, each edge has a unique positive cost. If we increase the cost of each edge by a number c, then the MST of the graph does not change.
5. In a directed graph, $G = (V, E)$, each edge has a unique positive cost. If we increase the cost of each edge by a number c, then the sequence of nodes in the shortest path between two nodes s and t does not change. (The length of the shortest path will change, of course.)

Problem 2 (20 points) After you graduate, you join a catering company as a logistics manager. Every day, the company must create a set of n dishes. Dish i ($1 \leq i \leq n$) requires p_i units of prep time and c_i units of cooking and plating time. The company employs a set of n sous chefs (who do the prep work) and one finicky chef. For every dish, the chef permits one of the sous chefs to do the prep work but insists on cooking and plating the dish themselves. Thus, the sous chefs can work in parallel but the chef is a bottleneck since they can work on only one dish at a time. The chef does not do any prep work; only the sous chefs engage in this task. The chef can start cooking a dish as soon as its prep work is completed. As the logistics manager, your task is to determine an order for scheduling the creation of these n dishes so that the completion time of all the dishes is as small as possible. Here, we define the *completion time* to be the moment when the final dish is cooked and plated by the chef. Develop an algorithm to solve this problem.

Problem 3 (25 points) In a parallel universe, you are a journalist covering a consumer electronics convention. Each visitor to the convention is carrying a cell phone. It is impossible to tell which OS is running on a specific phone.[1] Moreover, you will be told to leave if you ask this question of a visitor. What you can do is to introduce two visitors to each other. If they are both carrying phones running the same OS, the phones will ping in recognition. Otherwise, the phones will be silent. Suppose more than half of the visitors carry phones with the same OS. Describe an efficient algorithm that identifies all members of this majority group. Your analysis should bound the number of introductions made as a function of n, the number of visitors.

Hint: I am looking for an $O(n \log n)$ time algorithm. It is possible that you have seen this problem before and are aware of a solution with an $O(n)$ running time. If you present this algorithm, it is even more essential than ever that you prove its correctness. This proof is quite complex. I will not award any points for an $O(n)$ algorithm without a complete proof of correctness.

Problem 4 (40 points) You are given a connected, undirected, unweighted graph $G = (V, E)$ in which each edge has the colour red or the colour blue . Given two vertices s and t in V, we say that an s–t path is

- *monochromatic* if all edges in the path have the same colour
- *colourful* if the colours of the edges alternate between red and blue (or blue and red). A colourful path must have at least two edges.

See Figure 1 for examples.

Given $G = (V, E)$, the colour of each edge in E, s, and t, develop efficient algorithms to answer the following questions:

(a) (5 points) Is there a monochromatic s–t path?

(b) (15 points) Is there a colourful s–t path?

(c) (10 points) What are all the nodes in V and all the edges in E that lie on a shortest path from s to t?

[1] Think back to the days when there were indeed other OSes than just iOS and Android.

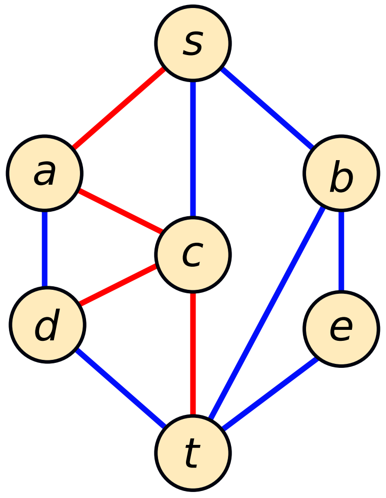

Figure 1: Illustration of monochromatic and colourful paths. The paths s, a, c, t and s, b, e, t are monochromatic. The paths s, c, t and s, c, d, t are colourful. The paths s, a, d, t and s, a, c, d, t are neither monochromatic nor colourful. If the edge (d, t) were red, then the path s, a, d, t would be colourful but the path s, c, d, t would no longer be colourful. With respect to problem (d), there are two shortest s–t paths. Of these s, b, t is monochromatic but s, c, t is not.

(d) (10 points) Are all shortest s–t paths monochromatic?

Hints and notes: For parts (a) and (b), do not develop a new algorithm from scratch. Modify G or create a new graph and apply an existing algorithm. For part (c), you should keep in mind that there could be more than one shortest s–t path. We have not described any algorithm to compute *all* shortest paths between a pair of nodes. To solve this problem, you must first extend one of the algorithms from class or the textbook to identify all the nodes and edges lie on at least one shortest s–t path. It is important to note that the number of shortest paths between a pair of nodes in a graph can be exponential in the number of nodes. Therefore, you cannot afford to explicitly compute all shortest s–t paths.