Midterm Examination

CS 4104 (Spring 2024)

Assigned: March 18, 2024.
PDF solutions due on Canvas by 11:59pm on March 25, 2024.

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 or any other online sources including ChatGPT or generative AI software of that ilk.
- 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 (25 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 (30 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 by looking at it.[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 of a kindred spirit. Otherwise, the phones will be silent. Suppose you are told that 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: It is possible to think of this problem in terms of graphs: each node is a visitor and two nodes are connected by an edge if their phones ping when the visitors are introduced to each other. You can think about how to complete the algorithm. However, even constructing this graph will take $\Omega(n^2)$ time. So let us discard this approach.

I am looking for an $O(n \log n)$ time divide and conquer 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 instead of solving the sub-problems below, 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.*

To develop a divide-and-conquer algorithm, note that there is no feature by which we can sort the phones. All we can do is divide the phones (arbitrarily) into two halves A and B. Here is a template for the algorithm, where V is the set of n visitors (V does not indicate nodes in a graph here):

MajorityOS(V)

1. If V has two elements, then `do ??  and return ??`
2. Partition V into two subsets A and B of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$, respectively.
3. R_A = MajorityOS(A)
4. R_B = MajorityOS(B)
5. `Linear time operation on` R_A `and` R_B
6. `Return` R_V

In your solution, you will have to describe in words

(i) (2 points) the constant time operation and return value in step 1,

(ii) (3 points) the return values R_A and R_V in steps 3 and 6, respectively (no need to say what R_B in step 4 is),

(iii) (10 points) the linear time operation in step 5, and

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

(iv) (15 points) state and prove a statement based on the following discussion. Let x be the name of the OS that runs on more than half the phones and let S be the set of phones running x. We do not know what x is, so treat it as an unknown variable for the purpose of developing the algorithm. But we do know that $|S| > |V|/2$. Since we do not know S yet, we do not know how the phones in S are split between A and B; they may be equally split or virtually all the phones in S will go to one of the halves. Make a statement about $S \cap A$ and $S \cap B$ and prove this statement. *This statement will be the basis for the proof of correctness of the algorithm and the answer to the previous part.* Saying something like $(S \cap A) \cup (S \cap B) = S$ is true but not useful, since it does not prove the correctness of the algorithm.

You do not need to analyse the running time of the algorithm or provide any additional proof of correctness.

Problem 4 (45 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.

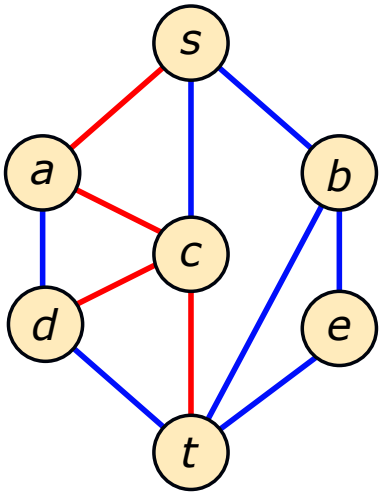

Figure 1: Illustration of monochromatic and colourful paths. The paths s, a, c, t and s, b, e, t are monochromatic. The paths s, c, t and s, c, d, t are colourful. The paths s, a, d, t and s, a, c, d, t are neither monochromatic nor colourful. If the edge (d, t) were red, then the path s, a, d, t would 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.

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) (15 points) Compute all the nodes in V and all the edges in E that lie on a shortest path from s to t? In the figure, the nodes are $\{s, b, c, t\}$ and the edges are $\{(s, c), (c, t), (s, b), (b, t)\}$.

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

Hints and notes: Don't panic that this problem amounts to nearly half of the points. Each of the parts is virtually independent of the others. Draw some simple examples of graphs to help you develop your algorithms.

part (a): Do not develop a new algorithm from scratch. Modify G to create one or more new graphs in which there are no edge colours and on which you can apply an existing algorithm (call it A). You must prove that you can solve the problem (Is there a monochromatic s–t path in G?) by solving an analogous and simpler problem on the new "colourless" graph(s) that you have constructed; algorithm A is applicable to this simpler problem.

part (b): Create a new graph without edge colours and apply an existing algorithm here as well. Is there a way to convert G into a new graph G' where edges in G' have no colours and every colourful path in G "maps" to a path in G' and every path in G' "maps" to a colourful path in G? Describe the construction of G' clearly, for example: "For every node v in G, ..." and "For every edge (u, v) in G, ..." You will find it very useful to introduce systematic names for the nodes in G'.

part (c): *Edge colours do not matter here, so you may ignore them.* You should keep in mind that there could be more than one shortest s–t path. We have not described any algorithm in class 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 can be exponential in the number of nodes in the graph. Therefore, you cannot afford to explicitly compute all shortest s–t paths. *Any solution that attempts to explicitly enumerate all the paths will be severely penalised!*

part (d): *Edge colours do matter here.* Assume you have solved part (c) correctly. Can you use the solution to part (c) to identify a subgraph $G_{s,t}$ of G that will (i) contain every shortest s–t path in G and (ii) every path in $G_{s,t}$ is a shortest s–t path in G? If you can construct such a graph, then you are almost done, although you should watch out for some pitfalls.