CS2800: Design and Analysis of Algorithms

Mini Quiz #2

March 10, 2023

Max Marks: 10
Duration: 1 hour

Instructions

- Write your name and roll number clearly in the answer sheet before you start
- This quiz contains 3 questions totalling 10 marks. All questions are compulsory.
- Please be as precise as possible in your answers. Write only what is necessary to substantiate your answer.
- Anything used in class can be used as is. If you are using a result that was given in the problem set directly, you can mention it and use it. Anything else must be fully justified.
- Write only one answer for a question. If you make multiple attempts, please strike off the attempts that you think are incorrect.
- Whenever you describe an algorithm, you must include a proof of its correctness and an analysis of its running time. **Algorithms/pseudocode without any explanation will receive partial/zero marks.**

---

1. **(2 marks)** Let G be a weighted connected undirected graph (possibly containing negative-weight cycles) and s an arbitrary vertex in G. Assume that you are given an array dist which is claimed to be an array such that dist$[v]$ contains the shortest path distance from s to v. Give an efficient algorithm to check if the array is actually holding the shortest path distances. Note that you are not given any shortest-path tree, but merely the array dist of claimed shortest path distances.

   Solution:

   Slightly inefficient solution: Run Bellman-Ford algorithm to find the shortest distances, and confirm that there are no negative weight cycles. If G has a negative weight cycle, answer that dist-values are incorrect. Otherwise, compare the answers you got with the dist-values. This gives an $O(|V||E|)$-time algorithm.

   A more efficient solution: Verify that for each vertex v, there is a u such that $(u, v) \in E$ and $\text{dist}(v) = \text{dist}(u) + w(u, v)$, and that $\text{dist}(s) = 0$, and these edges form a spanning tree of the graph.

   Now verify that for every edge (u, v), we have $\text{dist}(v) \leq \text{dist}(u) + w(u, v)$.

   If both conditions are satisfied, then dist corresponds to shortest distances. This gives an $O(|E|)$-time algorithm.

2. **(4 marks)** Lyra and Pantalaimon are stuck in two different worlds among the n many. The worlds are interconnected by pathways that both are aware of. For each world W, both Lyra and Pantalaimon knows the worlds W_i that are directly connected to it and to which they can teleport in one step. The teleportation pathways work in both directions. They are being chased by Gobblers and would like to get together; i.e. reach a world at the same time. At every time-step, they *must* move from their current

world to one of the worlds that it is directly connected to. Both Lyra and Pantalaimon are allowed to revisit worlds while trying to find each other. Design an algorithm to check if Lyra and Pantalaimon can ever come together, and if so, the shortest time in which they can meet with each other.

Solution:

Slightly inefficient solution: Construct a new graph $G'(V', E')$ where $V' = V \times V$ and $E' = \{((u_1, v_1), (u_2, v_2)) \mid (u_1, u_2), (v_1, v_2) \in E\}$. Suppose that Lyra is in vertex $\ell \in G$ and Pantalaimon is in $p \in G$. Now, there is a way for Lyra and Pantalaimon to reach a common world iff there exists a path from (ℓ, p) to (u, u) (for some $u \in V$) in the graph G'. Perform a BFS on G' starting from (ℓ, p) and find the first vertex (u, u) reachable.

The graph G' has n^2 vertices and m^2 edges, and the algorithm has a running time $O(n^2 + m^2)$.

A more efficient solution: There is a common world that both Lyra and Pantalaimon can reach iff there is a walk of even length $2k$ from ℓ to p in G (you need to prove this for full credit). Thus if Lyra walks for k steps, and Pantalaimon walks for k steps, both meet at the vertex v.

Now, find the smallest even length walk from ℓ to p if it exists in G using the method described in Problem set 5. This gives an $O(n + m)$-time algorithm.

3. **(4 marks)** Consider the following two greedy strategies for the interval scheduling problem we saw in class.

   (a) If there is an interval I that contains another interval I' completely, then remove I and recurse. If there are no such intervals, choose the interval I'' that *starts first*, and remove all intervals that overlap with I'', and recurse.

   (b) Let I be the interval which starts last, and let I' be the interval that starts second-last.
   - If I' contains I completely, then remove I' and recurse.
   - If I and I' are non-overlapping, then add I and recurse on the remaining intervals.
   - Otherwise, remove I and recurse.

Explain if both, any or none of the two will actually output a maximum set of non-overlapping intervals. Justify your answer.

Solution:

(a) **This gives an optimal set of non-overlapping intervals.**

First, observe that after removing intervals that contain others fully, we only have intervals that are either (i) disjoint, or (ii) overlap without being subsets. Consequently, any interval I that starts first must end first as well. This is because an interval I' that starts after I and ends before I must be completely contained inside I, and we know that we have removed all such cases.

Now, we can use the same argument that was used in class to prove the optimality of this greedy choice.

(b) **This does not give an optimal set of non-overlapping intervals.**

Consider a set of intervals $(0, 5)$, $(4, 8)$, $(7, 10)$. Now, in the first choice $I = (7, 10)$ and $I' = (4, 8)$, and by the third step, $(7, 10)$ is removed. In the next choice, $(4, 8)$ is similarly removed giving $(0, 5)$ as the answer. For this instance OPT $= \{(0, 5), (7, 10)\}$.