**CS 344: Design and Analysis of Computer Algorithms** **Rutgers: Fall 2019**

# Homework #5

Deadline: Thursday, December 5th, 11:59 PM

## Homework Policy

- If you leave a question completely blank, you will receive 25% of the grade for that question.
- You are allowed to discuss the homework problems with other students in the class. **But you must write your solutions independently.** You may also consult all the materials used in this course (notes, textbook, further reading, etc.) while writing your solution, but no other resources are allowed.
- Do not forget to write down your name and whether or not you are using one of your two extensions. Submit your homework on Canvas. Note that if you have already used both your extensions and still submit this homework late, **your homework will not be graded**.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "use topological sort on a DAG in $O(n+m)$ time". You are **strongly encouraged to use graph reductions** instead of designing an algorithm from scratch whenever possible (even when the question does not ask you to do so explicitly).
- Remember to **always prove the correctness** of your algorithms and **analyze their running time**.

---

**Problem 1.** Describe and analyze an algorithm that finds the *second smallest spanning tree* of a given undirected connected (weighted) graph $G$, that is, the spanning tree of $G$ with smallest total weight except for the minimum spanning tree. You may assume that all edge weights are *distinct*s in the graph (recall that in this case the minimum spanning tree would be unique). **(20 points)**

*Hint:* First prove that the second smallest spanning tree of $G$ is different from the MST of $G$ in only one edge. Then use this to design your algorithm by starting from an MST of $G$ and checking which edge should be changed to obtain the second smallest spanning tree.

**Problem 2.** You are given a weighted graph $G(V, E)$ (directed or undirected) with positive weight $w_e > 0$ on each edge $e \in E$ and two designated vertices $s, t \in V$. The goal in this problem is to find a $s$-$t$ path $P$ where the maximum weight edge of $P$ is minimized. In other words, we define the weight of a $s$-$t$ path $P$ to be $w_{MAX}(P) = \max_{e \in P} w_e$ and our goal is to find a $s$-$t$ path $P$ with minimum $w_{MAX}(P)$ (recall that in the shortest path problem, we define weight of a path to be sum (instead of max) of the weights of edges).

Design and analyze an $O(n + m \log m)$ time algorithm for finding such a path in a given graph. **(20 points)**

*Hint:* In this rare instance, it may be easier to modify Dijkstra's algorithm directly, instead of doing a reduction (although the latter option is also completely possible) – just remember to prove the correctness of your modified algorithm from scratch. Another option would be to instead use the DFS algorithm combined with a clever binary search approach.

**Problem 3.** You are given a directed graph $G(V, E)$ with two designated vertices $s, t \in V$, and some of the vertices in $G$ are colored *blue*. Design and analyze an $O(n + m \log m)$ time algorithm that finds a path from $s$ to $t$ that uses the *minimum* number of blue vertices (the path can use any number of non-blue vertices).

**(15 points)**

*Hint:* Unlike the previous problem, the easiest solution here is a simple graph reduction.

**Problem 4.** In the class, we studied the single-source shortest path problem and saw Bellman-Ford and Dijkstra's algorithm for solving this problem. A related problem is the *all-pairs* shortest problem where the goal is to, given a directed weighted graph, compute the shortest path distances from every vertex to every other vertex, i.e., output an $n \times n$ matrix $D$ where $D_{ij} = dist(v_i, v_j)$, namely, the weight of the shortest path from $v_i$ to $v_j$. Our goal in this question is to design an $O(n^3)$ time algorithm for this problem.

(a) Design an $O(n^2)$ time algorithm that takes as input a weighted directed graph $G$ and any arbitrary vertex $v$ in $G$, and constructs a new directed graph $G'(V', E')$ with weighted edges such that $V' = V \setminus \{v\}$, and the shortest path distances between any two vertices in $G'$ is equal to the shortest path distances between them in $G$. **(10 points)**

(b) Now *assume* we have already computed all-pairs shortest path distances in $G'$. Describe an $O(n^2)$ time algorithm to compute the shortest-path distances in the original graph $G$ from $v$ (the vertex chosen in part (a)) to every other vertex in $V$, and from every other vertex in $V$ to $v$. **(10 points)**

(c) Combine your solutions for parts (a) and (b) to design a *recursive* all-pairs shortest path algorithm that runs in $O(n^3)$ time. **(5 points)**

**Problem 5.** You are given an *undirected* graph $G(V, E)$, with three vertices $u$, $v$, and $w$. Design an $O(m+n)$ time algorithm to determine whether or not $G$ contains a (simple) *path* from $u$ to $w$ that passes through $v$. You do *not* need to return the path. Note that by definition of a path, no vertex can be visited more than once in the path so it is *not* enough to check whether $u$ has a path to $v$ and $v$ has a path to $w$. **(20 points)**

*Hint:* While it may not look like it, this is a maximum flow problem. Create a flow network from $G$ by making all edges in $G$ bidirected and adding a new source vertex $s$ that is connected to $u$ and $w$. Assign an appropriate capacity to the *vertices* in this graph, and identify an appropriate target vertex $t$. Finally, find a maximum flow using the Ford-Fulkerson algorithm and argue that this only takes $O(m + n)$ time.

---

**Challenge Yourself.** Consider Problem 2 again and suppose now that the graph is *undirected*. Design and analyze an algorithm for the same problem with the improved running time of $O(n + m)$. You may use the fact that there is an algorithm that given an array of size $k$, can find the *median* of the array in $O(k)$ time.