CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2022**

Homework #4

Deadline: Tuesday, April 05, 11:59 PM

Homework Policy

- If you leave a question completely blank, you will receive 25% of the grade for that question. This however does not apply to the extra credit questions.
- 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 (video recordings, notes, textbook, 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.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "use Prim's or Kruskal's algorithm to find an MST of the input graph in $O(m \log m)$ 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** (or any other efficiency measure asked in the question).
- The "Challenge yourself" and "Fun with algorithms" are both extra credit. These problems are significantly more challenging than the standard problems you see in this course (including lectures, homeworks, and exams). As a general rule, only attempt to solve these problems if you enjoy them.

Problem 1. A *walk* in a directed graph $G = (V, E)$ from a vertex s to a vertex t, is a sequence of vertices $v_1, v_2, \ldots, v_k$ where $v_1 = s$ and $v_k = t$ such that for any $i < k$, (v_i, v_{i+1}) is an edge in G. The *length* of a walk is defined as the number of vertices inside it minus one, i.e., the number of edges (so the walk $v_1, v_2, \ldots, v_k$ has length $k - 1$).

Note that the only difference of a walk with a *path* we defined in the course is that a walk can contain the same vertex (or edge) more than once, while a path consists of only distinct vertices and edges.

Design and analyze an $O(n + m)$ time algorithm that given a directed graph $G = (V, E)$ and two vertices s and t, outputs *Yes* if there is a walk from s to t in G whose length is divisible by five, and *No* otherwise.

(25 points)

Problem 2. We say that an undirected graph $G = (V, E)$ is **2-edge-connected** if we need to remove *at least two* edges from G to make it disconnected. Prove that a graph $G = (V, E)$ is 2-edge-connected if and only if for every cut $(S, V - S)$ in G, there are *at least two cut edges*, i.e., $|\delta(S)| \geq 2$. **(25 points)**

Problem 3. Given a connected undirected graph $G(V, E)$ with distinct weights w_e on each edge $e \in E$, a *maximum* spanning tree of G is a spanning tree of G with maximum total weight of edges[1]. Design an $O(m \log m)$ time algorithm for finding a maximum spanning tree of any given graph. **(25 points)**

Hint: You may want to use graph reductions, but remember that you are not allowed to have *negative* weights on edges of graphs (for now at least).

Problem 4. Let $G = (V, E)$ be an undirected connected graph with maximum edge weight $W_{\max}$. Prove that if an edge with weight $W_{\max}$ appears in some MST of G, then all MSTs of G contain an edge with weight $W_{\max}$. **(25 points)**

Challenge Yourself. A **bottleneck spanning tree (BST)** of a undirected connected graph $G = (V, E)$ with positive weights w_e over each edge e, is a spanning tree T of G such that the weight of maximum-weight edge in T is minimized across all spanning trees of G. In other words, if we define the cost of T as $\max_{e \in T} w_e$, a BST has minimum cost across all spanning trees of G. Design and analyze an $O(n + m)$ time algorithm for finding a BST of a given graph. **(+10 points)**

Fun with Algorithms. Let us go back to the bottleneck spanning tree (BST) problem defined above.

(a) Prove that any MST of any graph G is also a BST. Use this to obtain an $O(m \log m)$ time algorithm for the BST problem (notice that this is slower than the algorithm from the previous question). **(+8 points)**

(b) Give an example of a BST of some graph G which is *not* an MST in G. **(+2 points)**

[1] This is exactly the opposite of the minimum spanning tree (MST) problem we studied in the lectures.