CS 577: Introduction to Algorithms — Homework 5

Out: 04/05/2022 — **Due: THURSDAY 04/14/2022**

- Read the lecture notes.
- Read Chapters on Graph Algorithms, BFS, DFS, Minimum Spanning Trees.
- You should do ***all*** problems, but hand in only the graded problems.
- Write your name and student ID clearly on your submission.
- Clearly mark the beginning and end of your solution to each problem.

1. One can use DFS on a directed acyclic graph (DAG) to find a topological order.

   Here is another algorithm: Suppose we are given a directed graph G by a list of linked lists, one for each vertex v, to denote the set of vertices u that have a directed edge (v, u) (from v to u). First we try to find a count, one for each w, that counts the in-degree (the number of edges pointing to w.) Then those with in-degree 0 are source nodes, and we can remove them. While removing a source node w we also decrement the count of certain vertices...

   Write a pseudocode for this algorithmic idea to make it a fully specified algorithm, including the data structures you need. Prove your algorithm is correct in finding a topological order.

   What if your input graph G is not a DAG? Will your algorithm find this fact? How and why? Prove it.

   Analyze the worst-case running time of your algorithm. It should be linear time, i.e., $O(n + m)$, where n is the number of vertices and m is the number of edges; prove your answer.

2. **Graded Problem (Page limit: 1 sheet; 2 sides)** This problem is on finding a Minimum *width* spanning tree.

   Let $G = (V, E, W)$ be a weighted connected (undirected) graph, where V is the set of vertices, E is the set of edges, $w_e \in W$ for each $e \in E$ is a nonnegative weight of the edge e. A spanning tree is defined as usual, namely $T = (V, T_E)$ where $T_E \subseteq E$ and T is a tree. The *width* of a spanning tree T is $\max\{w_e \mid e \in T_E\}$. Thus it is the most "expensive" edge in the tree. T is a Minimum *width* spanning tree of G if it is a spanning tree and achieves the minimum width among all spanning trees of G.

   - Show that even if the weights of $G = (V, E, W)$ are all distinct, there could be more than one Minimum *width* spanning trees.
   - Suppose $E_{\leq t} = \{e \in E \mid w_e \leq t\}$ is the set of edges in G with weights at most t. Suppose $G_{\leq t}$ has connected components $C_1, C_2, \ldots, C_s$, for some $s \geq 1$. Let T_i be a Minimum *width* spanning tree of C_i (for $1 \leq i \leq s$), prove that there is a Minimum *width* spanning tree T of G that contains $\cup_{i=1}^{s} T_i$, i.e., T is obtained by adding some number of (zero or more) edges from E to the edge sets of $T_1, T_2, \ldots, T_s$. Can you write down what is the number of edges to be added (in terms of the data already given)?
   - Suppose $w_1 < w_2 < \ldots < w_m$ are all the edges of G. Find an algorithm for Minimum *width* spanning tree that implements the following idea: Imagine a threshold t rises from 0, to w_1, then to w_2, ... (You can use a min-heap for this.) At each threshold level t, you can consider the graph $G_{\leq t} = (V, E_{\leq t})$. Initially you have n isolated vertices. Think of them as n singleton sets. As the threshold t rises, you will perform Union-Find operations, with the idea that each set should be the vertex set of a Minimum *width* spanning tree in $G_{\leq t}$, for the current t. (What's the relationship between these Minimum *width* spanning trees and those components C_i?)

     Write a pseudocode for this algorithm, specify all details. Prove it's correct, and analyze its running time. You can analyze for each segment of your algorithm. What is the best upper bound you can achieve if $m = O(n)$?

3. A *bipartite* graph is an (undirected) graph $G = (V, E)$ whose vertices can be partitioned into two disjoint sets $V = V_1 \cup V_2$ such that all edges in E are between V_1 and V_2.

   - Give a linear time algorithm to determine whether an undirected graph is bipartite.
   - Prove that a graph is bipartite iff there are no cycles of odd length.

4. **Graded Problem (Page limit: 1 sheet; 2 sides)**

   A (valid) graph coloring is to assign colors to vertices of an (undirected) graph such that for every edge $\{u, v\}$ the two vertices are assigned different colors. (You may assume the colors are just some initial segment of integers $\{1, 2, \ldots, t\}$ for t distinct colors.)

   Prove that a graph is bipartite iff it can be colored with two colors.

   - Find an algorithm, that upon an input graph, either produces a (valid) graph coloring with two colors, or outputs "No" when it cannot be colored with two colors.
   - At most how many colors are needed to color in an undirected graph with exactly one cycle of odd length?
   - Find an efficient algorithm, that uses two colors to color all but one vertex for an undirected graph with exactly one cycle of odd length.

5. Design a linear time algorithm which, given an undirected graph G and a particular edge e in it, determine whether G has a cycle containing e.

6. Suppose we are given a map of n cities, call them $1, 2, \ldots, n$. Between any two cities i and j there is a positive distance $d_{i,j}$. Our goal is to start from city 1, visit every city exactly once, and end up in city n. We want to do that so that the total distance traveled is minimized.

   Notice that if we compare all possible solutions by brute force, there are a total of $(n-2)!$ many possibilities to compare.

   Here is a slightly better idea: Suppose we consider a partial trip that starts from city 1 and ends in city $c \in \{2, \ldots, n\}$, and moreover we want to visit (after city 1) exactly the cities that belong to a subset $S \subseteq \{2, \ldots, n\}$. Of course $c \in S$. Can you express the minimal distance traveled $D(S, c)$ by a recursive expression that depends on the last city $i \in S$ traveled before reaching c?

   Develop a dynamic programming algorithm that solves this problem in $O(n^2 2^n)$.