Homework 4 Solutions
CSI 503 Spring 2021

Rahul Bhenjalia

April 22 2021

1 Problems

1. Assume that for every cut of a graph, there is a unique light edge crossing that cut. Prove that such a graph has a unique minimum spanning tree

   ANSWER:
   Let's start by assuming the contradiction that the graph has two distinct minimum spanning trees, T and T', instead of a unique one
   Let (u, v) be an edge in T, which is not in T'. So we remove edge (u, v) to cut tree in two components.
   Now we have T_u and T_v, which are the vertices in the component containing u and v.
   Considering the cut, (T_u, T_v), we introduce (x, y) to be the unique light edge crossing the cut.
   We first consider that (x, y) != (u, v), then weights will be compared as $w(x, y) < w(u, v)$, which proves that spanning tree $T - (u, v)U(x, y)$ has a better cost than T, which is contradiction to our assumption
   On the other hand let's consider $(x, y) = (u, v)$ that is the unique light edge (u, v) crossing the cut (T_u, T_v), doesn't belong to T'.
   COnsidering a path p from u to v in T' , which starts in T_u and ends in T_v, hence considering this fact, there must be at least one edge (e) on it crossing the cut (T_u, T_v).
   As (u, v) is the unique light edge crossing this cut, the weights $w(u, v) < w(e)$.
   If we add (u, v) to T', we get a cycle composed of (u, v) and p. By removing any edge from the cycle we get again a spanning tree.
   Hence $T(u, v)e$ is a spanning tree and by above it has a better cost than T, again a contradiction.
   Hence, it is proved that for every cut of a graph, if there is a unique light edge crossing the cut, the graph has a unique minimum spanning tree.

2. Write the pseudocode of an algorithm that finds a maximum spanning tree of a graph. That is a spanning tree with maximum total weight of

edges. You can use a modified version of the algorithms for finding a minimum spanning tree.
ANSWER:
In the given Kruskal's algorithm from textbook, if we just sort the edges in decreasing order instead, we have a solution for maximum spanning tree of a graph.
.
.
.
max-ST-KRUSKAL (G,w)
1. $A = \emptyset$
2. for each vertex $v \in G.V$
3. MAKE-SET(v)
4. sort the edges of G.E into **decreasing** order by weight w
5. for each edge $(u, v) \in G.E$, taken in **decreasing** order by weight
6. if FIND-SET(u) ≠ FIND-SET(v)
7. A = D ∪ (u,v)
8. UNION(u,v)
9. return A

3. Consider a directed acyclic graph (dag) $G = \langle V, E \rangle$, and a vertex weight function $w : V \to N$ (N is the set of natural numbers). Our goal is to design an algorithm that calculates function $t : V \to N$ where $t(v_i) = \sum_{v_i \rightsquigarrow v_j} w(v_j)$. In other words, $t(v_i)$ must be equal to the sum of weights of all reachable vertices from v_i. By definition, we consider each vertex being reachable from itself (path of length 0). Your designed algorithm should have time complexity of $O(V + E)$ or better. Correct but less efficient algorithms will receive partial credits.

   (a) Write the pseudocode of your algorithm and briefly explain your overall approach
   ALGORITHM:
   1. sum = 0 //sets initial weight to 0
   2. DFS-Sum-Weight(node):
   3. if visited[node] == 1: //checks if vertex was visited
   4. return
   5. visited[node] = 1 //sets the node as visited
   6. for all u connected to node: //check all nodes connected to u
   7. sum + = weight[node][u] //add weight to sum counter
   8. DFS-Sum-Weight(u) //calling function recursively

   (b) Prove the correctness of your algorithm

   The given algorithm is a modified version of DFS algorithm applied on an adjacency list representation of a graph. Given the reachability of DFS through the graph and provided the weighted edges, the given algorithm provides correct solution for our problem.

(c) Analyze the complexity of your algorithm in detail
Since, the algorithm follows the Depth-First Search method, we can confirm that the time complexity of the code would be same as the DFS algorithm.
We have an adjacency list and the graph is directed, the sum of the sizes of the adjacency lists of all the nodes is E. So, the time complexity in this case is O(V) + O(E) = O(V + E).

4. A graph may have several minimum spanning trees. Can you propose a design of an algorithm based on the Kurskal's algorithm (CLRS, Page 631) to generate all minimum spanning trees corresponding to an input graph? A rough explanation of the idea and why it should work correctly is sufficient. There is no need to provide a pseudocode.

   Considering a weighted graph, we may have, in some cases, multiple edges with same weight which might lead to the graph having multiple minimum spanning trees.
   The process of Kruskal's algorithm is to keep searching for the smallest weighted edge and keep adding that to minimum spanning tree, as long as it does not lead to a cycle.
   Now, in that case if we have multiple edges with same weight, we have an option to create multiple minimum spanning tree, if we cover all the cases.
   Hence, if we create a code where all options of same weighted edges are considered as cases, we can generate all possible minimum spanning trees corresponding to the graph.