Homework 8: Spanning Trees and Shortest Paths

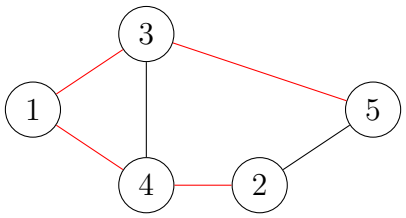
Elias Gabriel

1. (6 points) Prove that if a graph $G = (V, E)$ has unique edge weights (i.e. $w_{e1} \neq w_{e_2}$ for any two edges $e_1, e_2 \in E$), then there is a single minimum spanning tree. In other words, there is only one optimal solution. **Hint:** Use a proof by contradiction and suppose that there are two optimal spanning trees T_1^* and T_2^*.

   Solution: Suppose that there are two or more optimal spanning trees for the uniquely-edge-weighted graph G, say T_1^* and T_2^*. Assuming they are both valid, the total edge weight for both trees would need to be equal else one would be more optimal than the other.

   If both trees start at edge V_0 and we iterate through the edges of T_1^*, there necessarily must be at least one node at which T_1^* selects an edge E_i and T_2^* selects a different edge E_j (else the trees would be identical). To maintain optimality, we know that the weights for each edge w_{e_i} and w_{e_j} must be equal. However, we also know that the tree G has no edges that share a common edge weight. This is a contradiction, so the uniquely-edge-weighted graph G must only have one minimum spanning tree.

2. (15 points) Suppose that you are given an unweighted graph $G = (V, E)$ and a node $i \in V$. One way to find the shortest path from i to all other nodes $j \in V$ is to run breadth-first search from i. In particular, for every node v added to the queue when processing u in BFS, we mark u as v's parent. This induces a tree of shortest paths from i, as shown below in red for $i = 1$. In this case, the shortest path from 1 to 5 is 1-3-5.



   (a) (6 points) Use a proof by contradiction to show that BFS keeps track of the shortest path from i to all other nodes. **Hint:** Consider the closest node v to the source i whose BFS path is not a shortest path.

   Solution: Starting at any node i, we know that we iterate through all its children before iterating through all of its grandchildren. This is because we prioritize our search space using a queue, which by definition maintains the FIFO property. If BFS

produces a non-optimal solution between nodes i and j and we retrace each edge decision, there must be an edge k where BFS selected to scan from k's grandchild before scanning from each of k's children. However, this is impossible since we know we add all of k children to our queue before adding all of k's grandchildren. This is a contradiction as is breaks a queue's FIFO property, and is therefore impossible.

(b) (6 points) Given a weighted graph $G = (V, E)$ in which each edge weight $w_e \geq 0$ is an integer, explain how to convert G into an unweighted graph $G' = (V', E')$ such that finding the shortest path from i to j in G' gives you the shortest path in G. How many vertices and edges does this new graph have?

Solution: To convert the weighted tree G to a unweighted tree, we can simply replace every edge with weight w_{e_i} with a chain of w_{e_i} linked nodes. This yields a new graph where each previous edge is a chain of nodes and whose combined edge weight is $w_{e_i} + 1$. For the case where the previous weight was 0, we simply set the existing edge weight to 1. By doing so every edge weight is effectively incremented by 1, so graph's connectedness remains unchanged. If we didn't increment zero-weighted edges, network information and possible paths would be lost.

(c) (3 points) Using your construction above, explain how to find the shortest path from i to all other nodes j in a weighted graph $G = (V, E)$ using BFS. What is the runtime of your overall algorithm? How does it compare to the runtime of Dijkstra's algorithm?

Solution: To find the shortest path for graph G, we can convert it to an unweighted graph using the process I outlined above. In that step we must iterate over every edge, creating w nodes for each. Assuming that the creation of w nodes takes $O(w)$ time, the total runtime for that pre-processing step would be bounded by the number of edges $|E|$ and the maximum edge weight w_{max}, making it $O(|E| * w_{max}) = O(|E|)$.

To find the shortest path, we can then make use of the fact that BFS will always find the shortest path in an unweighted graph with a worst-case runtime of $O(|V| + |E|)$. In this new graph, $|V|$ and $|E|$ are larger than in the original by an offset equal to the original cumulative edge weight W. In BigO, that is $O((W + |V|) + (W + |E|)) = O(2W + |V| + |E|) = O(|V| + |E|)$.

In totality, this means the algorithm has a runtime of $O(|V| + 2|E|) = O(|V| + |E|)$. In comparison to Dijkstra's algorithm, this runtime is faster as the $|V|$ term does not have a factor of $\log(|V|)$.