Homework 8: Spanning Trees and Shortest Paths

Data Structures and Algorithms Spring 2020

Sparsh Bansal

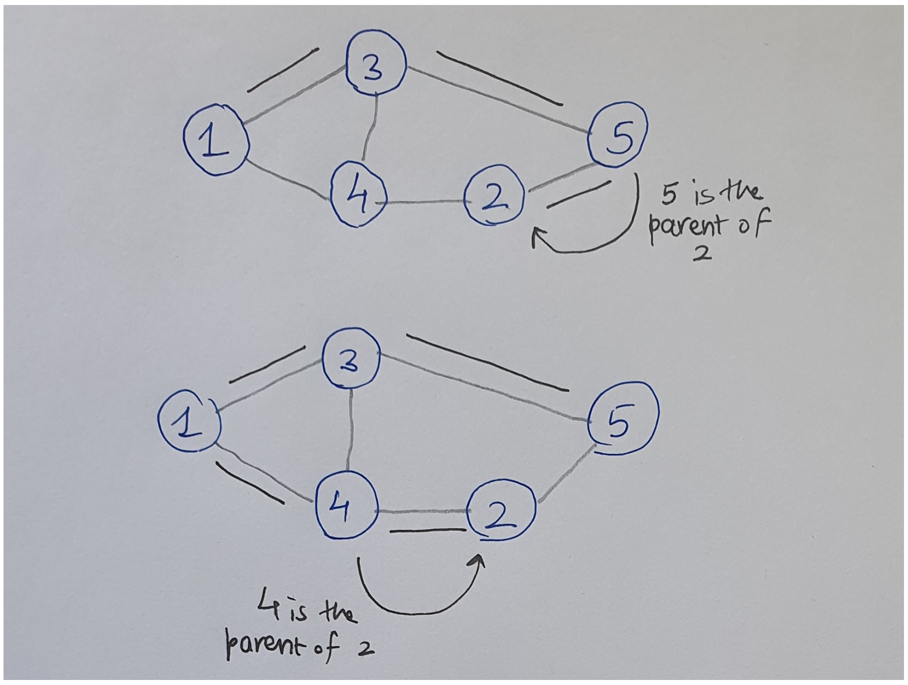
1 Question 1

Suppose that there are two different optimal spanning trees T_1 and T_2. Among all the edges belong to either of the two trees, let the edge of the smallest weight be e_1 in T_1 (which does not belong to T_2). Now, if we add e_1 to the tree T_2, we have added a cycle to it as well. To make T_2 acyclic again, we need to remove any one edge from it. The edge that we remove, call it e_2, must be absent from T_1. Once we do these operations, we have obtained a tree with a smaller span by $e_2 - e_1$ (Since we assumed that e_1 is the smallest of its kind), which a contradiction to the fact that T_2 is a minimum spanning tree. Hence, if a graph $G = (V, E)$ has unique edge weights (i.e. $w_{e1} \neq w_{e2}$ for any two edges $e_1, e \in E$), then there is a single minimum spanning tree.

2 Question 2

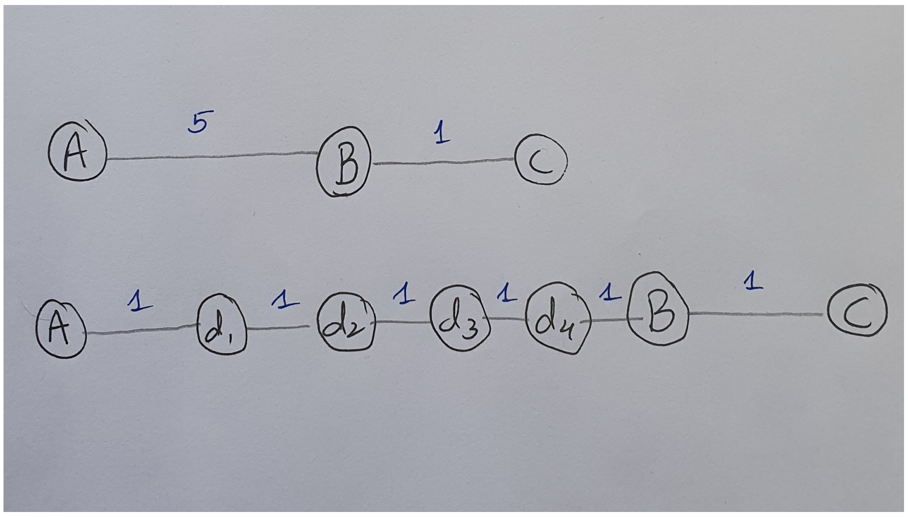
2.1 Part a

To start the argument, let us say that the BFS algorithm does not keep track of the shortest path from i to all the other nodes in the unweighted graph $G = (V, E)$.



Referring to the figure, we can see that in this case, node with value 5 was marked as the parent of node with value 2, and the output path from node with value 1 to the node with value 2 would be 1-3-5-2. However, for every node v added to the queue when processing u in BFS, we mark u as v's parent. This would mean that nodes with values 3 and 4 would be processed prior to nodes with values 2 and 5. As a consequence, node with value 3 would be marked as a parent of node with value 5, and the node with value 4 would be marked as a parent of node with value 2, suggesting that the shortest path to node with value 2 would be 1-4-2. Hence, our initial statement is a contradiction to the BFS algorithm!

2.2 Part b



Given a weighted graph $G = (V, E)$ in which each edge weight $w_e \geq 0$ is an integer, we can modify each weighted edge E to be multiple edges, each with weight W standardized over the graph. Referring to the figure, we have splitted an edge of weight 5 between two nodes to a combination of the two initial nodes plus 4 (5-1) temporary/dummy nodes connected in a tree formation with each edge weighing 1. The processed graph is now an 'unweighted' graph $G' = (V', E')$ such that finding the shortest path from i to j in G' gives us the shortest path in G. This new graph would have $\Sigma w_e - num(E_{original}) + num(V_{original})$ vertices and Σw_e edges. Here, $num(E_{original})$ is the number of edges in the original graph, and $num(V_{original})$ is the number of nodes in the original graph.

2.3 Part c

Using the construction stated above, we can pass this modifed graph into the default BFS algorithm and get the correct result. The only change will be in the output when we will have to delete all the temporary/dummy nodes from the resultant path. Since every vertex and every edge is checked once, the runtime for a BFS algorithm should be $O(num_V + num_E)$. For a weighted graph $G = (V, E)$ the worst case runtime can be $O(num^2_{V_{new}})$, where $num_{V_{new}}$ is the new number of vertices after converting a weighted graph to an unweighted graph. The runtime, will then depend on the weights of edges, on top of the initial number of edges and nodes. This tells us that the runtime could run better than Dijkstra's algorithm (which has a runtime of $O(|E|log(|V|))$) in the case of small edge weights, but worse for the cases of larger edge weights.