Jia Feng Yu
20908183
CS341 A2
Question 2

In order to find the number of different shortest paths in a graph from a starting vertex s and destination vertex t, we can perform a modified BFS algorithm, keeping track of the length of the path for each node we visit. We then store this value into a map, mapping the node to the length of its path from s. Since we do not care about the exact path, we do not need to implemente a parent array. To keep track of the distance from s, we simply apply the same distance array that we have seen in the notes, ie, set distance$[v]$, where v is a neighbor of u to be the distance$[u] + 1$. We also need a counter for the number of different, shortest paths from s to t. Before starting the BFS, simply initialize this counter to 0.

We proceed with BFS as standard: enqueue s into the queue, mark visited$[s] =$ true, and dequeue s and enqueue all of its non visited neighbors. Then proceed with BFS normally, updated visited, the queue, distance as needed. When we have reached t for the first time, save this distance as the minimal distance and increment the counter of numbers of different, shortest paths to 1. After, for every time we meet t, we compare the current distance+1, because we still have to add 1 edge to get to t, with the minimal distance: if it is the same, increment the counter of paths and else do not.

For runtime, since we are only adding the distance array and storing integer counters, all of which are $O(1)$ operations, the overall runtime is the same as BFS, ie, $O(|V| + |E|)$.

To show correctness, by nature of the BFS tree, the first time we have reached t from s, this will be the shortest path. Because BFS will only stop after having visited all the vertices in the graph and it will not revisit the same path twice, this means that every path will eventually be explored. Since all possible paths are explored, therefore all the possible unique paths are explored, and thus, implying that all the different, shortest paths from s to t will be reported.