CS577 HOMEWORK3

HAOYI LU, WILLIAM SHU

PROBLEM 2:

In this problem, we need to find the number of shortest path so that we need to traverse layer by layer. So, we choose BFS to solve this problem.

Our Algorithm:

// every node has its own depth and count (counts the number of path)

// for the start node, depth = 0 and count = 1

findNumShortestPath(G):

enqueue(v); // add the first node to the end of the queue 🡪 O(1)

while(queue is not empty): 🡪 O(E)

a = dequeue(); // take the first node out from the queue, set as a 🡪 O(1)

if(a is not visited): 🡪 O(1)

mark a as visited; // ensure the node will not be visited multiple times

if(a is w): // if we reach the end of the path🡪 O(1)

return a.count; // return the number of path at node a

for (all the edges of a: b): // traverse all the edges of a, set as b 🡪 O(V)

if(b.depth == 0): // if the depth is 0, which means it’s never reached

b.depth = a.depth + 1; // set depth of b = 1 layer deeper than a

b.count = a.count; // set the count as a, since unreached before

else: // depth is not 0, which means it is reached before🡪 O(1)

if b.depth = a.depth + 1:

// if still 1 layer deeper, then it’s still on the shortest path

b.count = b.count + a.count

enqueuer(b); // add b to the queue🡪 O(1)

return -1; // if the algorithm cannot get to the node w, then no path can reach the end.

Correctness:

This algorithm is a breath-first-search, which can traverse all the nodes until it finds the target one. In this algorithm, it will search layer by layer, which means it will search in depth 1, 2, 3 and so on. Then it will add the all the nodes in the first layer into the queue, dequeue the first node, then find the edges of first nodes and add to the end of the queue. In this process, our algorithm automatically makes the nodes sorted from low depth to high depth. And it can correctly visit all the nodes from start to target node.

For count, we want to prove by induction hypothesis:

Base Case: there is only one node:

Our count is 1 and return a.count = 1; // correct

General Case:

We assume our algorithm holds for depth = n. For depth = n + 1, all the shortest path should connect to the n + 1 node. There are two cases, if the node connect to it is reached before or not reached before. If the node is not reached before, we just need to consider it as a continuous path (count will not change). If the node is reached before, we have to see if the path is still the shortest. We compare the depth of two nodes, if the depth of this node is one layer deeper than the previous node, then we consider it as a “valid node to connect”, then if we connect this node to our path, previously, there was a node connect to it, then we can easily see there is one more possible path. Then we add the count of previous node to this node, since currently it is not simply adding one more path but adding all the possible paths of previous nodes. In result, we make b.count = b.count + a.count. In this process, our count can correctly find the number of paths from depth n to depth n + 1.

Since count can correctly count the number of paths of each node, and depth can correctly visit all the nodes from start to end. Then our algorithm findNumShortestPath(G) can correctly find the number of shortest path of the undirected graph.

Running time:

Since in our algorithm, we are only searching all the nodes and edges one time, then even though we looped twice, the overall running time is just O(E+V), which is linear.