Ian Chadwick
U90596416
EC330 Spring 2021

Homework 7 Written Answers:

1) The general idea is to keep a count of all the alternate predecessors that have an equal distance from the source node by modifying the relax procedure to add one to *alternate* for nodes that are equal in distance to the predecessor starting with 1 for the predecessor itself. Each time relax updates the predecessor with a new shorter path, *alternate* will be reset to 1 because the new shortest path has changed. Instead of running the relax procedure V-1 times you would run it V times to ensure that all equidistant paths are counted. If relax finds a new path with the same distance, *alternate* will be increased by 1. Returning *alternate* for a node will report the number of shortest paths to any given node from a source node.

Bellman-Ford (G, n)

- InitializeSingleSource(G,n) //initialize values for nodes in graph
  - For each vertex v in V[G]
    - **do** d[v] <- inf.
      π[v] <- NULL
      alternate[v] <-0
- d[n] <-0
- Repeat V (number of vertices) times
  //modified Relax procedure
  - **for** each edge (*u,v*) with weight *w* in edgelist **do**
    if d[v] > d[u] + w(u,v)
    then d[v] <- d[u] +w(u,v)
    π[v] <- u
    alternate[v] <-1
    else if d[v] = d[u] + w(u,v) //if there is an equidistant path
    then alternate[v] += 1 //add one to alternate
  - **for** each edge(u,v) in edgelist //check for negative cycles
    do if d[v] > d[u] + w(u,v)
    then return FALSE //return false if there is

  return TRUE //else return true

2) The general approach is to use a modified BFS algorithm that keeps a predecessor list of all the nodes visited until destination node *t* is reached when you start at source node *s*. If node *t* is not reached, you would discard the list. If node *t* is reached the predecessor list will be added to a vector. Then you would have a vector of all the paths from node *s* to node *t*. The running time should be **O(V+E)** because in the worst case you would have to traverse and visit every node and edge once. Since the given info was that it was an acyclic graph, we don't have to worry about node coloring because even if there are repeat visits, there are no cycles. This is important because there may be multiple valid paths that use the same node in the path to *t*.

Node has data field for:
char identifier
predecessor list

Modified_BFS(*G, source, target*)
vector<list<char>> *paths*;
Queue<**Node**> *Q*;
Q.push(*source*)
while (!*Q.empty*) **do** *//will run until all nodes reachable from s have been explored*
Node v = Q.front;
Q.pop;
if (v = target) *//if this node is our target node all nodes in the pred. list are a path*
then paths <- add predecessor list to paths vector;
else do *//otherwise this node is a potential predecessor, so add it to the list*
v.predecessor list <- add v.identifier;
for each node adjacent to v *//then explore all adjacent nodes*
Q <- push that node to Q
return *paths*;

3) The general approach would be to first scan the whole NxN grid for each cell with a 1 and record the coordinates in a list of nodes. **$O(n^2)$**

This would make V nodes, where V is some number less than or equal to n^2. Then use the *Manhattan Distance* formula given to compare each person's coordinates to the other coordinates. **$O(V^2)$** since V could technically be equal to n^2 if every cell was occupied by a person, that could increase overall runtime to $O(n^4)$. However, the average case will likely be that V is much less than n^2 and so the overall average runtime will likely still be closer to **$O(n^2)$**. However, you might be able to reduce the runtime further by sorting all the vertexes by their distance from a fixed point (i.e. 0,0) which could be done in close to linear time with insertion sort, since they would be mostly sorted already. Or you could use merge sort to guarantee O(VlogV). Then after each comparison if the distance between the nodes was already greater

than *sd*, you could stop processing that node, remove it and process the next one. This would reduce the amount of comparisons between nodes to much less than $O(V^2)$. This again would probably bring the runtime closer to **$O(n^2)$**.

If the distance between two nodes is smaller than the allowed distance *sd,* add an undirected edge between those two nodes to each node's adjacency list unless it's already there. Then any node that has a non-empty adjacency list will be in violation based on each other node in their list. You could then report the coordinates of each vertex that is in the list by using either BFS or DFS which both run in **O(V+E)**.

The overall runtime would be the greater of either $O(n^2)$ or O(V+E), but much more likely in **$O(n^2)$**.