Eric Yang

304263623

CS 180

Dis 1B

Homework 3

1. A. We store the nodes V in a priority queue with d’(v) as the key for v ϵ V. For the next v node to be inserted into S, we extract the minimum from the queue. Then we update all the values d’(w) for w not in S that have an edge with v that was just added to S. If d’(w) > d(v) + , then we change the key of node w to the lower number. Doing this means we don’t have to recalculate the minimum distance on every loop and thus the running time for finding a node is O(n).

B. If we implement the priority queue above using a heap structure, we can easily extract the smallest key. Making the heap will take O(m) time and fixing the heap on every cycle causes a run time of O(logn). This means the total time is O(mlogn).

1. To reduce the problem to finding the longest path in a DAG, we can simply make each interval a node in the DAG. Each interval, starting from the earliest one, will point to the next interval and branch off to several nodes if there are time conflicts. The edges connecting the current node to the next nodes will have weight according to the price of each interval. The longest path in this DAG will then give us the best schedule for highest total price.
2. Given source node s and undirected graph with integer edge weights

Use two directed edges to replace each undirected edge with same weight

Let S be the set of explored nodes

For each u in S, we store a distance d(u)

Initially S = {s} and d(s) = 0

Create priority queue with V as nodes and d’(v) as keys for v in V

While S != V

Select minimum node u from queue

Update values d’(w) for w not in S that have an edge with v to minimum of

d’(w) and d(v) +

Add u to S and define d(u)=d’(u)

EndWhile

This algorithm should run the same way as the shortest path algorithm for directed graphs once we replace the undirected edges with two directed edges. The run time should be O(n^2)

1. A. The diameter can be found if we run BFS on any node in the tree. Suppose the last node we find is node x. Then we run BFS again on node x and suppose the last node we find is node y. The diameter of the tree is distance between node x and node y.

B. Recursive algorithm:

F(n) where n is the set of nodes in the tree

Select two leaves on deepest level of tree that don’t share parent node

If they share parent or there is only 1 node

If tree has more than 1 node

Remove all leaf nodes from n

Return 1+F(n)

Else

Return 0

Else

If tree has more than 1 node

Remove all leaf nodes from n

Return 2+F(n)

Else

Return 0

Iterative algorithm:

Diameter = 0

For each level in the tree with n nodes starting from the bottom

Select two nodes on current level that don’t share a parent

If there is only 1 node or both share a parent

If node is root

Return Diameter

Else

Diameter += 1

Else

Diameter += 2

The complexity of the iterative algorithm is O(n) since it loops through the n nodes in the tree.