Shallows Explanation

Samuel Kent 22704037

Given the array lanes[ ] that holds the information for a weighted, non-negative and directed graph of size ports. Find the Longest (deepest total) path from the given source node (origin) to every other node (port). To find the longest path I used Dijkstra's algorithm as a priority first search. Once the deepest path to a node from the origin has been established, find the lane on this path with the smallest weight (shallowest) and add it to the array (min_lanes[ ]) in the same position as the path ends (arrives).

Initially create a new inner class Node to store the ID of the node and a corresponding weight. Where the weight is the current total depth to reach that node for the current deepest total path.

I have used PriorityQueue to store each node in a max heap. Such that when an element is removed from the heap it holds the current highest total depth.

Also initialize an array key[ ] to hold the current deepest total path for each node where Node ID is found at key[ID].

Initialize an array to store the Shallowest lane's depth for each corresponding node in key[ ]. With the Origin node's value set to infinity so that it is never the minimum of any amount of nodes and never altered.

To begin Dijkstra's algorithm we add the Node with ID = Origin and the weight = 0.

Once the PriorityQueue is no longer empty, we remove from the front of the queue giving us the Node with the current Deepest total path to it in the heap. We mark this Node as being discovered so that we don't revisit it and it is never added to the heap again. We then find the departing lanes by traversing the lanes[ ] array, a lane is directed away from and attached to the current node if "depart" is the same as the current node ID, i.e. it is departing from the current node.

If we have not seen the node the lane is arriving at then we compare the total depth of the current path that arrived us at the node we are on now plus the depth of the lane we are using to the total depth of the deepest previous path of the node we are arriving at. If the path to this node will be a deeper path than previous due to using this lane then we update it in our key[ ] array. We also add this newly discovered node to the priorityQueue such that it will eventually be dequed and analysed later.

After a node has been seen and all it's lanes have been analysed we compare all the lanes that were on it's deepest path and store the shallowest lane in min_lanes[ ] at Node ID. Once the Priority Queue is empty all nodes have been visited and analysed.

As such we have executed the main loop once for every port, therefore costing O(P) and each time we extract a Node from the max heap it costs O(logV) to find and extract the Node with the highest weight. So extraction costs, O(P logV).

For every P operation (different Node) we traverse the lanes[ ] array of size L to find departing lanes. For every new lane it may cause us to add a new node to the heap, costing O(log V). These operations cost O(PL LogV)

As such, keep dominant terms: O(P logV + PL LogV) = O(P L)