Problem Set 5

Name: Asem Ashraf

Collaborators: None

Problem 5-1.

(a)

(b)

(c)

(d) yes, there can be

Problem 5-2.

Problem 5-3. this is is a really good problem i love it. basically in this problem we need to split the droids into to groups where there isn't two droids in the same group that can short-circuit each other. so if we construct a graph where the droids are represented by nodes while the edges in the graph are made between two droids that short-circuits each other. To be successful in separating those droids into two groups we will have to check if the graph is bipartite or not. Basically, a graph is bipartite if it does not contain any cycles with an odd number of edges. All we need to do to solve this problem is to detect if there is any odd-number-edged cycles. To do so we need to use something called spanning trees and spanning forests. Spanning trees are acyclic graphs that are made from a normal undirected graphs where there are no cycles in the resulting tree. If the graph has multiple connected components then the graph can be made of multiple spanning trees called spanning forests. To construct a spanning forest of a graph we could run full-BFS on the graph. It is better to run BFS because it classifies the nodes into levels and that can help us in stating the solution where every level with an odd number can be colored a color while even numbers can be colored another color separating the graph into two groups. After doing so we will have to check the graph for any edge that is connected to two nodes of the same color, if there exist such an edge then the graph cannot be bipartite and a two safe parties would be impossible. Building the spanning forest takes linear time since we will pass every node and every edge once. Checking weather the graph is bipartite or not also is done in linear time.

Problem 5-4. in this problem all i know is a precise but very inefficient solution which basically loops over all the verticies of one river and runs bfs on it and does that for all the verticies of the river, which could potentially take a lot of time like O(n^4). which is more than that of the problem requirement. but the actual solution in the pset answer is rather strange and I don't even know if it is precise or not. Put a new node that has an undirected edge between all the nodes of one river and any path from that river to the other will have to path by that node. based on what. he didn't even state anything more than that which is quit absurd to me to be honest why would this work? So I have seen the problem session and it had a problem like this where they used a super node.

Observation 1: this is an unweighted undirected graph.

we have a n x n map of land. A farmer can have multiple squares adjacent to each other on the map. Our graph would represent the the intersection between four squares as a node and a side of the square as an edge except when one farmer owns multiple squares adjacent to each other then the edge would not be in our graph. i.e. we only take edges into account if it is between two squares owned by different farmers. Now, our problem basically want to find the minimum distance from one river to the other while passing through edges from any node considered to be an Euphris river. We can restate the problem into: there are three sets of nodes nodes considered to be Euphris river, others as Tigrates river and other that are not of the two rivers where the third set of nodes connects the two rivers and we dont know about the connectivity of the sets of nodes of the two rivers. Find the shortest path from the sets of nodes that form a river to the other set of nodes, that is we want to find one path from one node from the first river set to one node in the second river set. find an algorithm that runs in O(n^2) time.

We could run a very inefficient solution here which is run DFS on each node of one set and save the smallest distance to any node in the other set then take the smallest distance from all nodes. We basically took the minimum of the minimums. but that runs in O(n^4) time.

Observation 2: when DFS is run on one node form one set it will explore the whole graph including the distance to nodes in the same set which we don't really need. So if there is a way to not do that would be optimal.

Idea: add a super node that is connected to all the nodes of one set then run BFS not DFS on that node. This way BFS would mark all the nodes in the set as first level nodes and would not explore them again. This will garantee that every level set BFS constructs will be closer to the other set of nodes. Everytime we encounter a node we check for two things: 1- is it any previous level set? 2- is it one of the nodes in the other set of nodes that we want to go to? At the end we will have visited every node once which makes this algorithm O(n^2) time.

Problem 5-5. the solution to this problem uses graph duplication which is rather an overkill because it basically duplicates them exponentially in the number of cards. nibba what? i refuse to solve this problem with that solution. but i have figures out another solution that that will make her get pizza from all the pizza locations in O($|V| + |E|$) time but that will make here go through each location at most eight times because she might need to go through it back and forth and the question was what is the solution that will make here go through each door the least number of time. am sure that by exploiting that there is not a space limitation in the question will solve the problem but i cant be asked atm.

Problem 5-6. i aint solving this. no time.

(a)

(b)

(c)

(d) Submit your implementation to `alg.mit.edu`.