Problem 1 [DPV] Problem 3.15 (Computopia)

Part (a):

Create a graph G where each intersection in the city is represent by vertex u . Now there is edge for vertex u to v if there is a path(road) connecting two intersection v to u. We can now run treat each vertex as SSC. Run DFS and check if there is a path from v to u which contains either cycle or has a path to itself then mayor claim is false. So basically, we can run DFS algorithm and check if graph is strongly connected. If graph is strongly connected and there is only one SSC then mayor claim is True , otherwise its false.

Correctness: -

By creating edge between each intersection and then running on DFS check if graph is Strongly connected, we make sure each path goes from intersection u to v using edge e exactly once. But if graph is not strongly connected and there one more than one SSC then path is two ways because it has cycle which goes through edge from u to v twice. So, Mayor claim is false in that case.

Runtime: -
We can Creating graph with vertices and edges for intersection of roads which is linear operation. Running DFS algorithm and checking if there is graph is Strongly connected is linear operation. So above problem can be solved in linear time i.e. O(|V|+|E|) time.

Part (b):

We can now keep the same graph as we have for problem a above where each intersection is represented by vertex and vertex v is connected to u by edge E if there is patch from v to u . We will run DFS algorithm and find all the Strongly connected component of graph. Once we have all SSC components of graph, we will say let townhall be as source s . From this source s , we will find path from s to v such that v using DFS. If there is path from s to v and v is outside the SSC component, then mayor this claim is false. If there is no vertex v which has path from s to v outside the SSC component, then mayor claim is True.

Correctness: -
By first checking getting all components of Graph g and then from townhall as source s , we check if there is a path outside the SSC component (Where townhall is). Using DFS, we check if there is a Path to v and that vertex v is outside the SSC component it means one can came from another street outside the SSC and then there will be one and more path wrong which do not directly go to town hall and mayor claim is false.

Runtime: -
First, we get all the SSC components of graph. Using DFS , from source townhall s we check all vertex v if there is path from s to v and if any v is outside the current SSC component. As we have check path from each street from s to t once for all streets(v) which is linear operation and can be completed in O(|V|+|E|) time.

Problem 2 (Planning a wedding)

Part (a):

We will great a graph G based upon inputs V and T such that for each guest in (v IN V) , we will defined vertex . Next based upon loop up table T , for vertex v we will create edge between v to u if u knows v. Now as our graph G is created with vertex v and edges between them.

We can run DFS on graph G starting from starting vertex s and all the paths reachable from .All vertex/path reachable from s can sit on one table . Continuing search for other vertex's which are not reachable from s and add table for new vertex along with adding the no. of tables increased by one. Keep on doing this until all the vertices are reached. From above as there is no limit how many guests can sit on one table, so no. of tables returns from above is total minimum no. of tables needed to achieve a valid seating. We can run DFS and total no. of SSC components is the minimum no. of tables we need for this.

Correctness:-

Out goal is here is to find minimum no. of tables where all people which knows each other are on same tables. We can this by making sure all edges from a vertex v are on same group and they cannot sit with other people which they have bad terms. So if we run DFS and get all Strongly connected components of graph we make sure there is no cycle between graph and total no. of Strongly connected components is our total no. of tables needed for this.

Runtime :-

Creating graph for guest is O(1) operation for each guest and total time for creating graph is O(n) operation . Running and find SSC using DFS is linear operation . Which is $O(|V|+|E|)$. So total run time for this is $O(|V|+|E|)$.

Part (b):

We will great a graph G based upon inputs V and T such that for each guest in (v IN V) , we will defined vertex . Next based upon loop up table T , for vertex v we will create edge between v to u if u is bad term with v. Now as our graph G is created with vertex v and edges between them.

Now we can formulate this problem if graph is bipartite, and it can be colored with only two colors like dpv exercise. If we start with traversing from vertex v using DFS mark it black. Next mark its neighbor u white if there is direct path from v to u and v is parent of u. Keep on coloring it for all vertexes . If graph can be coloring this way return True and seating arrangement is possible. But if u is visited and its parent color is already white (different than we expected) then return False.

Correctness: -

Our goal is such that no people with bad terms sit with each other which is only possible if people which are on good terms sit with each other. We have created a graph G where edge represent people which are on off terms with u and v vertex. So idea is to check odd vertexes of graph and see if we can all color black and even vertex should be white . Using DFS we traverse each vertex and color it so that all odd vertex have same color and all even vertex has same color. If such an arrangement is possible then u and v which are on bad terms already on different table and this is possible. But if there is edge between u and v and both have same color which means they are on bad terms and such an arrangement not possible.

Runtime:-

Creating a Graph is linear operation with vertex and edges which is O(|V|). Then coloring the odd edges black and even edges white using DFS is also linear operation which is O(|V|+|E|). So total runtime for this is O(|V|+|E|)