Homework 2

Simon Zheng
260744353

October 15th, 2017

1

A cycle between two vertices in a directed graph is when they each have an edge going to the other.

If adding an edge between two vertices already in the same strongly connected component, then nothing changes.

If adding an edge between a vertex in a strongly connected component and a vertex not in a strongly component: if this creates a cycle between these two vertices, then the strongly connected component the first vertex is in gains a vertex, but the number of strongly connected components still doesn't change. If the edge does not introduce a cycle between these two vertices, then nothing changes.

If adding an edge between two vertices, each in a separate strongly connected component, and it introduces a cycle between these two vertices, then the two separate strongly connected components become a single, bigger one, thus reducing the number of strongly connected components in the graph by one. If the new edge does not introduce a cycle between these two vertices, then nothing changes.

2

Start from any person p_i.

If p_i does not know anyone, mark this person as the potential celebrity, and go to another person p_k.

If p_k does not know p_i, then there are no celebrities.

If p_k does know p_i, then go to another person and also ask them if they know p_i. Do this for everyone left.

If everyone knows p_i, then p_i is a celebrity. Otherwise there are no celebrities.

If p_i does know at least one person, say, p_j, then go to p_j (don't ask for the rest). Mark p_i as being asked (and not a celebrity).

Go back to beginning and do the same thing for p_j, where the new p_i is the current p_j.

In the worst case, you follow a chain of person u_i knowing person u_{i+1} right up to the last person (who is the celebrity) which takes $n-1$ steps. Once you arrive at the celebrity (which

we do not know is one), we must ask that person if he knows any of the other people, which takes $n-1$ questions (steps). Then, finally, you must do the reverse which is asking every other person if *they* know the celebrity, which is also $n-1$ steps (questions) by re-visiting each of them and asking.

Thus, it is at most $O(3(n-1)) = O(n)$.

3

We can imagine the airports as vertices and flights as directed edges but with an extra departure and arrival time.

Take all flights that go to b and only consider those that arrive on time (before time t). As such we will go in reverse.

From those flights, construct a tree of possible paths. Keep track of the current time of the path that's being tried. At every airport, only consider valid flights with arrival times preceding the departure time of the flight you came from (as we are going in reverse).

Do not revisit airports. If you visit an airport that's already been visited from the same flight but in another path, pick the one with latest departure time, as it covers the cases of the earlier one and possibly more, and ignore the other.

Simply give the first path that reaches the source (since we started in reverse from destination).

4

First, run BFS or DFS simulating both stone movements at the same time.

Construct a tree where every vertex stores the position of both stones in the current step.

Edges of this tree connect pairs of positions which the stones can move to.

Do this for every possible path. When you can move the stones to a previously seen pair of positions, then it simply connects back to that position pair in the tree, so ignore it.

Take into account back and forth movements. If both move back to the previous positions, see previous point. Simply ignore it.

If you are forced to move back to a previous position, then it is not a possible path.

If you end with the reverse pair of the beginning, you're done. When you're done, run a simple BFS on the constructed tree to get the shortest path which reached the reversed pair. Otherwise all possible configurations will run out (since same ones are ignored) such that if you run through all the same positions twice, then you know you are done.

Construct the tree with a hash table of pre-existing pairs to know if it already exists in constant time.

5

Choose any root and run Breadth-First Search to find the longest path (the furthest vertex).

You now have its eccentricity, but you do not know if it is the maximum one of the tree (diameter).

You know this vertex will be one of the ends of the diameter, because the longest path from any root in a tree will always find be either vertex of the diameter.

The vertex at one end of the diameter is the furthest from half of the vertices, and the vertex at the other end is the furthest form the other half, otherwise the path between the two would not be the diameter of the tree.

Simply run BFS again from the furthest vertex found and you will find the vertex furthest from this one, which will give you the diameter.

Since a BFS is $O(n + n - 1)$ on a tree (a tree has $n - 1$ edges) and we only run it twice, we get $O(2(n + n - 1)) = O(n)$.

6

Using BFS or DFS, start from vertex with label 1. Run the search backwards (reverse edge direction) such that logically you know every vertex u_i you visit (including itself) are going to have $min_i(u_i) = 1$. Thus, every time you visit a vertex, mark its min as the starting vertex (using an array or if each vertex is an object with an attribute min or etc.).

Then keep doing this by starting on the next unvisited vertex with the smallest label until every vertex has a min.

This will be $O(n)$, because you only run it once per directionally connected component, as every vertex in a directionally connected component should be visited and have their min found once. Let's say worst case every vertex is disconnected, you have to run it n times, but each search will be $O(1)$, so it is proportional and cancels out to $O(n)$.