

## Homework 2

Simon Zheng  
260744353

October 13<sup>th</sup>, 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

We can represent this situation as a directed graph:

Each person is a vertex;

If person  $u$  knows person  $v$ , then there exists an edge going from  $u$  to  $v$ .

As such, a *celebrity* is a vertex  $c$  with an outdegree of 0 and an indegree of  $|V| - 1$ , where  $|V|$  is the total number of vertices (people).

Now, the question forces us to ask every single person at least once to know who they know. This means visiting every vertex at least once (no matter if they are connected or not as we have to ask to know it), so the algorithm will be at least  $O(n)$ . We know the celebrity must know no one. With this, we can immediately exclude a vertex (person) as a celebrity the moment they know someone. But we must still know who else they know, as to identify the celebrity who is known by everyone. Fortunately, every time we visit a vertex (ask a person who they know), we can exclude anyone who wasn't previously mentioned, as the celebrity must be known by all.

A good way would be to maintain a single collection of people who are known by every single person we have visited. It must *only* include people who do not know anyone and is known by every previous person asked. As such, a good data structure would be a hash table, as we will be doing a lot of searches and deletions.

1. Starting from any person, for every other person, ask if this person knows them. If they do not know anyone, add to set of potential celebrities. Otherwise, add every person they know.

2. Now, for every following person, if they do not know anyone and are not included in the list of potential celebrities, then we are done, as we know there are no celebrities since we have a person who is not a celebrity and cannot know them.

3. If they do not know anyone but are a potential celebrity, then simply go to the next person.

4. If they know anyone and are a potential celebrity, remove them from the set of potential celebrities.

5. Now remove any person in the set that is not mentioned by the current person being asked. If the set becomes empty then we are done, and there are no celebrities.

6. Otherwise, continue until no one is left to ask. Either there will be no celebrity or a single person who is the celebrity in the set, as it is impossible to have more than 1 by virtue of having to be known by *every other person*.

Algorithm: BDFS(Vertex  $u$ )

---

1	Input: A vertex $u$ <span style="color: blue;">in</span> a graph $G$
2	Output:

---

**SECOND WAY:** 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  $p_j$ .

### 3

Run a Depth-First Search from the starting airport. At each airport, sort the possible flights (accessible and that you are in time for) by arrival times to their respective destination. Take the one with the earliest arrival time, keep going until final destination is reached, or a dead-end is reached, in which case backtrack.

### 4

Run DFS on both stones at the same time. But on every step, check if next step will put them adjacent to each other. If it will, make one find another route (including going back on going into a cycle or loop) by checking other nodes.

### 5

Choose any root and run Breadth-First Search to find the two longest paths (the two branches with the longest depths) and sum them. Add 1 to the length to include the root itself.

**6**

Use a BFS