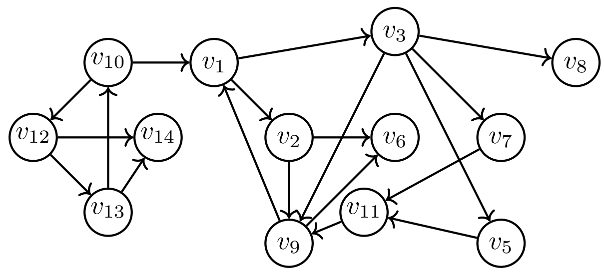
Simon Ibssa
CSC 349

Quiz 2 — Graph Algorithms

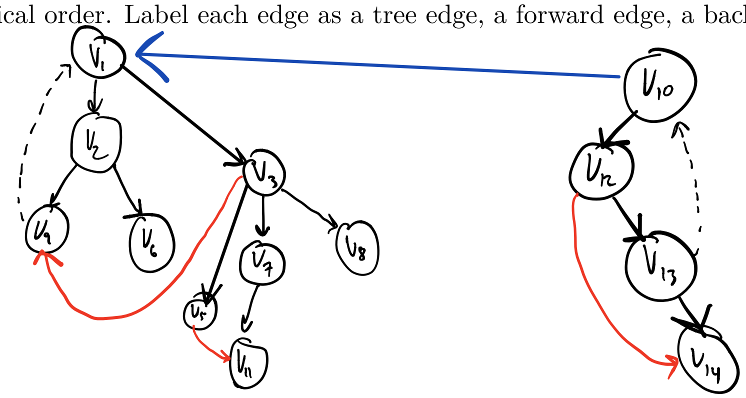
You have **1 day** to complete this quiz. Please ask me if you have any questions. Good luck!

- There are 2 pages in this quiz. Make sure you have all of them.
- This quiz is open-note and open-book, however, all answers must be an individual effort.

1. Consider the following directed graph:



Draw the forest produced by running depth-first search on the above graph, breaking ties in ascending numerical order. Label each edge as a tree edge, a forward edge, a back edge, or a cross edge.



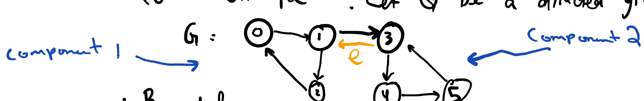
Key
— = tree edge
- - - = back edge
— = forward edge
— = cross edge

2. Consider the following statement:

Proposition: Suppose that G is a directed graph containing exactly two strongly connected components. Then it is *never* possible to make G strongly connected by adding a single edge.

Give a proof of or a counterexample to this statement.

Counter Example : Let G be a directed graph containing exactly two strongly connected components.



- By definition a Graph is strongly connected if there exists a path from u to v and from v to u for all vertecies in the graph.
- There must exist a path in both directions from strongly connected component 1 to strongly connected component 2, in order to start & return to our starting vertex, so we can add an edge e (in orange) to graph G.
- Since there exists a path from u to v and from v to u for all vertecies V, Graph G is strongly connected w/ the addition of only 1 edge.

3. Recall that a directed graph is said to be *strongly connected* if there is a path from u to v and from v to u for all vertices u and v in the graph.

   Suppose that you are given a strongly connected, directed graph $G = (V, E)$ and an edge $e = (u, v)$.

   (a) Give pseudocode for an efficient algorithm that determines whether or not removing e from G would leave G strongly disconnected. You may assume that $e \in E$.

   [*Hint:* Note that you are given that G is strongly connected. You should *not* need to compute its strongly connected components.]

StillSC(G, e)
Input: A directed, strongly connected graph G. An edge $e = (u, v)$, $e \in E$.
Output: A boolean connected that represents if removing e from G will still result in a strongly connected graph. Returns false if G is no longer strongly connected. Returns true if G is still strongly connected.

```
let connected = true
let removable = [] // store all edges that are loops, or forward edges. We can remove these & still maintain strongly connected in G.
for all v ∈ V, do:
    let v be "undiscovered"
for all v ∈ V, do:
    if v is "undiscovered", then:
        let Explore(G, v, e, removable) be a subcomponent of G.
if !(removable.contains(e)), then:
    connected = false
    return connected
else, do:
    return connected
```

Explore(G, v, e, removable)

Input: A directed graph $G = (V, E)$ and a vertex v, and an edge e to remove, where every vertex is either "discovered" or "undiscovered", and a list "removable"
Output: All vertecies reachable from v.

```
let discovered = []
let v be "discovered"
discovered.add(v)
for all neighbors of v, u, do:
    let edge = e(u, v)
    if u == v OR u is "discovered", then:
        // found a loop or a forward edge
        removable.add(edge)
return discovered
```

   (b) Give the time complexity of your algorithm:

isSC() considers every vertex twice. Explore() is called once for every vertex $v \in V$.
Explore() collectively considers each edge once or twice. isSC() has time complexity $O(|V| + |E|)$

$= \boxed{O(n + m)}$