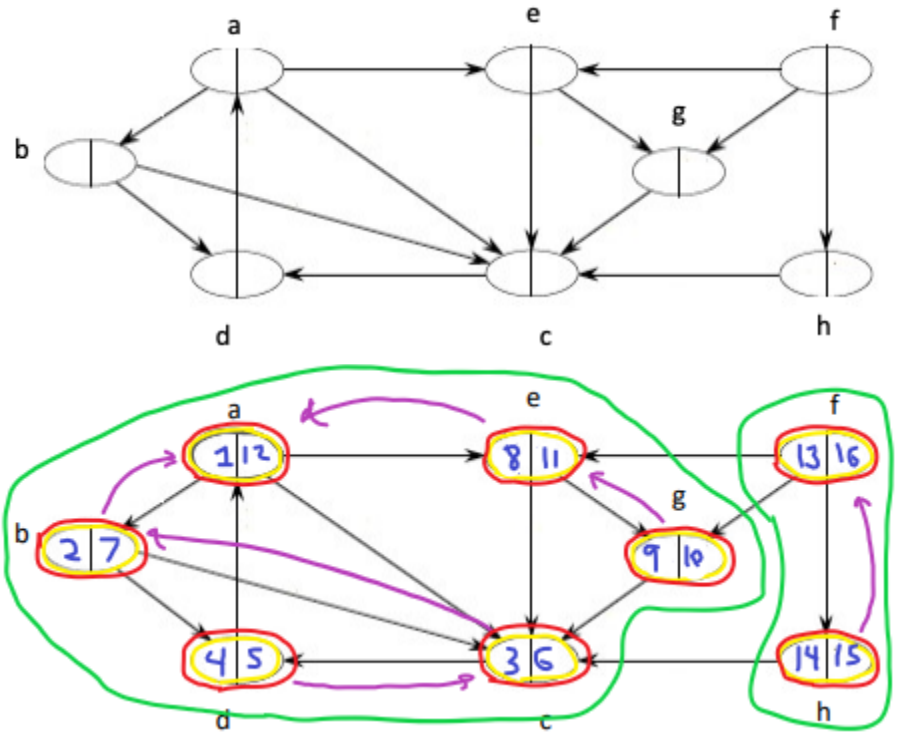
Archie Ross
2/13/23

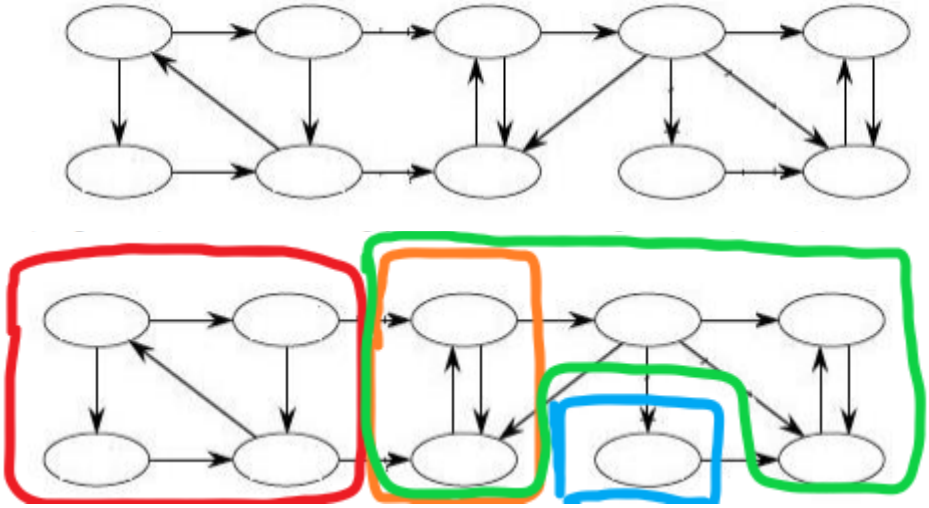
CSE3500 Homework 3

Question 1: **Run DFS and find the discovery time and the finishing time for each vertex in this graph (use the alphabetical order starting from 'a')**



At first, for this question I thought that the algorithm bottomed out three times (once at A, once at g, once at h). However I realized that the function doesn't bottom out at A because it has the adjacent e. Thus, the above diagram demonstrates the order (following the alphabetical bias) the DFS algorithm would take for the discovery time and the finish time of each node. The green circles denote the separation of the trees due to the missing pointers towards f and h.

Question 2: **Find the Strongly Connected Components (SCC) (Circle them)**



Strongly connected components are those that can traverse to and from their nodes. For example, the nodes in the red square are strongly connected because no matter which path you take, you will always have a route back to the starting node. However the Red square does not include the orange square because when you go from one of the nodes in the red square to the orange square, you cannot go back to the red square. This means that these nodes are not strongly connected to those in the red square. I followed this idea for the rest of the graph and found that only one node (the blue square) does not belong to a strongly connected group because it does meet the criteria we discussed above. The other groups have been circled and shown and meet the criteria.

Question 3: **A binary tree is a rooted tree in which each node has at most two children. Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.**

Base Case:

A tree with only one node that is either a leaf or a node with two children (thus it holds true for the base case).

Inductive step:

Consider a binary tree with n+1 nodes, let the root have two subtrees T1 and T2 where the number of nodes with two children are labeled N1 and N2 and the number of leaves L1 and L2.

N1 = L1 - 1 and N2 = L2 - 1
Is a conclusion we can draw by inductive hypothesis. Since the root has two children, it is not a leaf. Therefore the number of leaves in the entire tree is L1 + L2

Now we can count the number of nodes with two children in the entire tree to which there are three cases:

The root has two children (N1 + N2 +1)
The root only has one child, for example T1 (N1)
The root is a leaf. There are no nodes with two children in the entire tree.

Now we have to show that the total number is exactly one less than the number of leaves in the entire tree.

For each case:

N1 + N2 +1 = L1 -1 + L2 -1 + 1 = L1 + L2 - 1
N1 = L1 - 1 = (L1 + L2) - 2 + 1 = L1 + L2 - 1
0 = (L1 + L2) - 2 = L1 + L2 - 1

Therefore in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

Question 4: **directed graph G = (V,E) is semiconnected if, for all pairs of vertices u, v ϵ V , there is a path from u to v or from v to u. That is, different from the strongly connected components where you can go from u to v and then back, here you only need to either go from u to v or from v to u but don't have to get back. Give an efficient algorithm to determine whether or not G is semiconnected. Prove that your algorithm is correct and then analyze its running time. Note: to get full credit, your algorithm needs to run in O(|V | + |E|) time.**

Algorithm:

We want to sort the graph such that for every vertex i, there is an edge from vertex i to vertex(i+1).

We can do so by first finding the Strongly Connected Components, and building a new SCC graph G'=(U,E') where U is a set of SCCs, and E' is the set of edges between SCCs SCC1 and V2 where there is a vertex v1 in SCC1 and a vertex v2 in SCC2 such that (v1,v2) is in E.

Next we can perform a sort on G' and check if every vertex i in the sort has an edge from vertex i to vertex (i+1).

This determines whether or not our graph is acyclical which is a property of being semi-connected.

Pseudo algorithm:

```
SCC = find_max_SCC(G)
G_prime = SCC_graph(SCC, G)
Topological_sort = top_sort(G_prime)
For i = 1 to |Topological_sort| - 1 do
        If there is no edge from topological_sort[i] to topological_sort[i+1] in G_prime
then
                Return False
Return True
```

Proof:

This can be proven to work by supposing there is no edge from vertex i to vertex(i+1) in the sort of the graph. This means that there is not path from one vertex to the other violating that acyclical property as stated in the algorithm. This would mean the graph is not semi connected.

Equally, we can suppose that there is an edge from vertex i to vertex(i+1) for all vertices i in a topological sort of the graph. Then for any two vertices v1 and v2, we can find the SCCs containing them. Finally, since there is an edge from every vertex in SSC1 to every vertex in SCC2, there is a path from v1 to v2. Thus, we can finalize that the graph is semi connected.

Basically, If the algorithm returns true, If SCC1 and SCC2 have a path connected to them, then we can also determine that there is a path from v1 to v2.

Running Time:

The run time of this algorithm is O(|V| + |E|) because of the first and third line in the code which both take O(|V| + |E|) running time which dominates over the rest of the code. Line two only takes O(|E|) and Line 4 only takes O(|V|). This can just be simplified down to O(|V| + |E|)