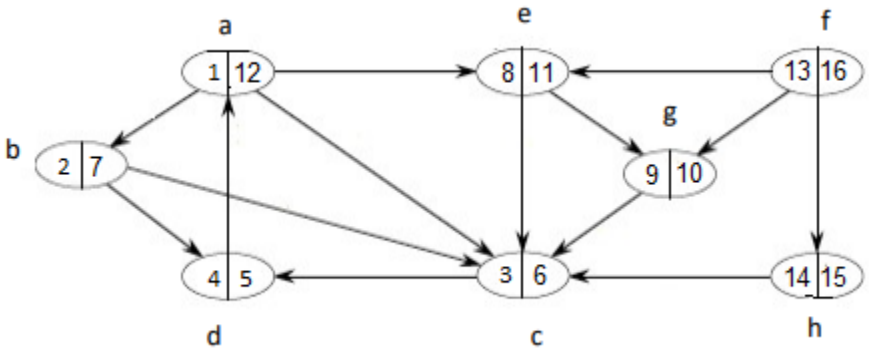
CSE 3500
Homework 3
Alex McLeod

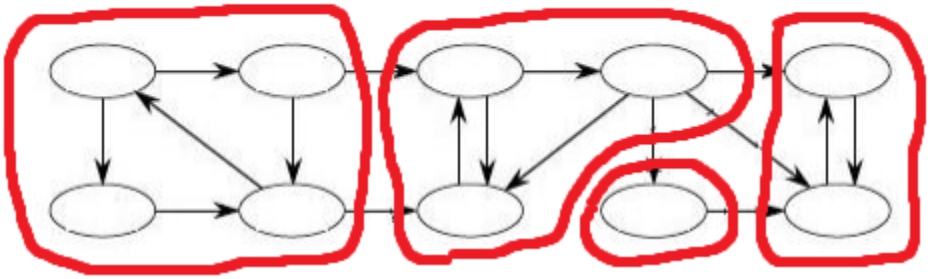
1. **Run DFS and find the discovery time and the finishing time for each vertex in this graph (use the alphabetical order starting from 'a')**



Here is the graph with a DFS run starting at vertex A. Keep in mind that the number on the left side of each vertex is the discovery time, and on the right side is the finishing time.

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

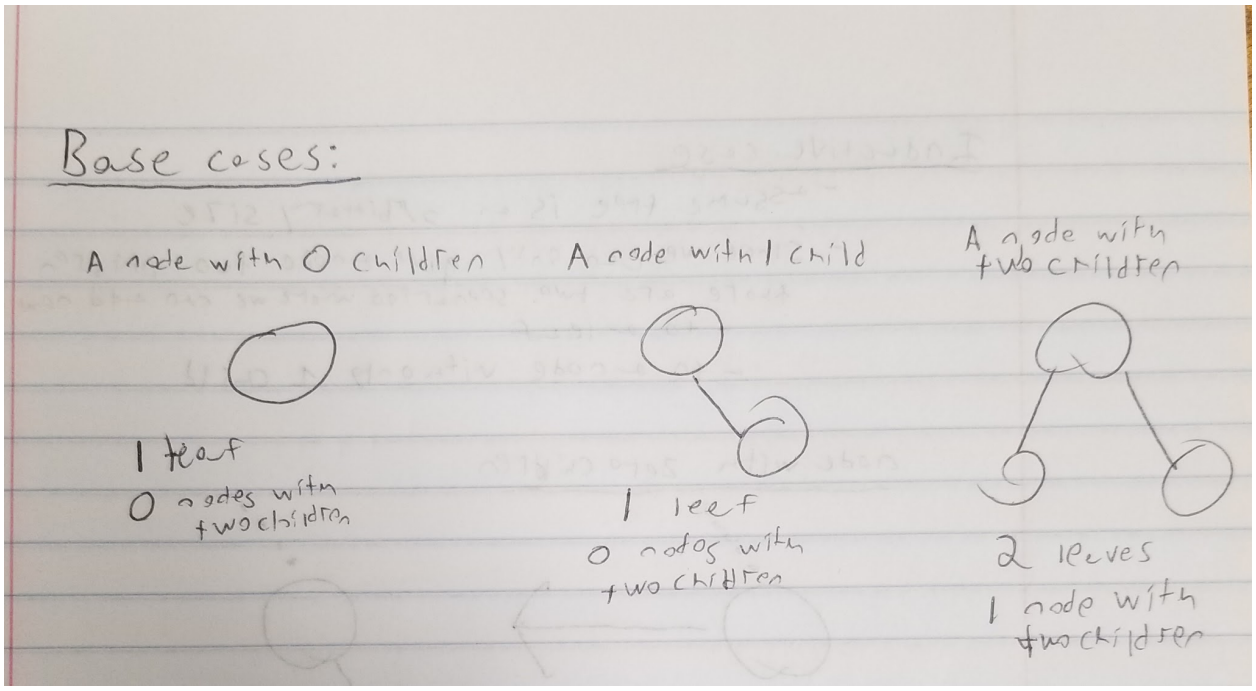
2. **(20 points)** Find the Strongly Connected Components (SCC) (Circle them)



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.**

Note: I'm going to refer to "in any binary tree the number of nodes with two children is exactly one less than the number of leaves" as the "statement" throughout this proof.

To prove this statement with induction, we must show that the statement holds in a base case, in this proof with trees of 1,2,3 nodes, and in an inductive case, where the number of nodes in the tree is arbitrary.

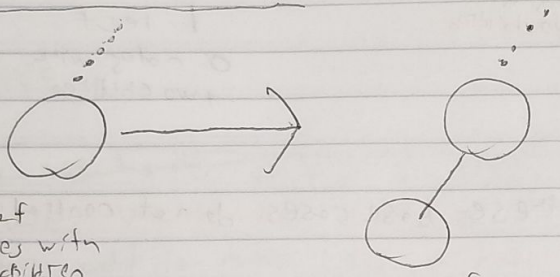


In these base cases, we have the simplest possible trees one can build. In all of them, the number of nodes with two children is exactly one less than the number of leaves, which adheres to our original statement. Since these are the "smallest" trees that are possible to build, when we deal with a larger tree, it is useful to imagine that we are simply adding more nodes to these simple trees.

Inductive case

- assume tree is an arbitrary size
- since we can only give a node two children there are two scenarios where we can add new nodes
  - to a leaf
  - to a node with only 1 child

node with zero children



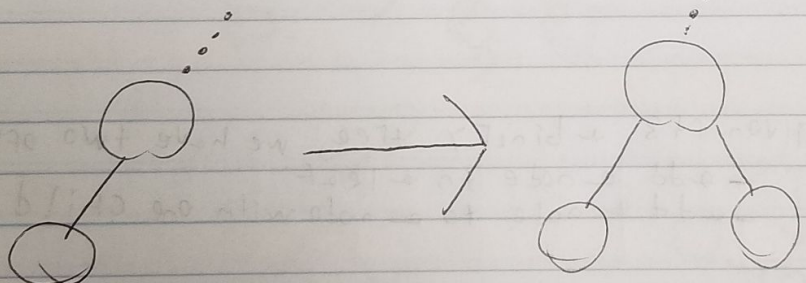
1 leaf
0 nodes with two children

1 − 0 = 1

1 leaf
0 nodes with two children

1 − 0 = 1

node with one child



1 leaf
0 nodes with two children

1 − 0 = 1

2 leaves
1 node with two children

2 − 1 = 1

~~state as we add nodes~~

Since the property remains true even as we add nodes, we have proved it using induction.

In the proof above, we want to add a node to a tree of ANY size. It is important to emphasize the fact that the size of the tree is arbitrary, as we are trying to prove the statement for all binary trees. Since a binary tree can have a maximum of 2 children, we can only add a new node to either a leaf, or a node with only 1 child.

In the case where we add a node to a leaf, starting out there is 1 leaf in this subtree and 0 nodes with two children (as shown above). Once we add the node, there is still only 1 leaf and 0 nodes with two children (as shown above). Because we know that the base cases uphold the statement, we know that ANY time we add a node to a leaf, the statement will hold.

In the case where we add a node to a node with 1 child (which is a leaf), starting out there is 1 leaf in this subtree and 0 nodes with two children (as shown above). Once we add the node, there are 2 leaves and 1 node with two children (as shown above). This upholds the statement as well because there is exactly 1 less node with 1 child in this subtree than leaves. Because we know that the base cases uphold the statement, we know that ANY time we add a node to a node with 1 child, the statement will hold.

Because we have proven that anytime we add a node to a tree, our statement holds, we have proven by induction that for ALL binary trees, the number of nodes with two children is exactly one less than the number of leaves.

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.**

Because of the properties of semi connectedness, we can say two nodes are semi connected if there's a connection between them, no matter the direction.

Here is an algorithm that can determine if a graph is semiconnected:

1. Pick any node u
2. Run a BFS or DFS on u, and track which nodes are reached in set A
3. Reverse the orientation of every edge in G, let's call this graph H
4. Run a BFS or DFS on u in H, and track which nodes are reached in set B
5. Combine sets A and B into set C
6. Return True iff every node in G is in the set C

We know our algorithm is correct from the statement we made before the algorithm.

We also know that the algorithm runs in O(|V | + |E|) time as it runs only two BFS's (both run in O(|V | + |E|)).