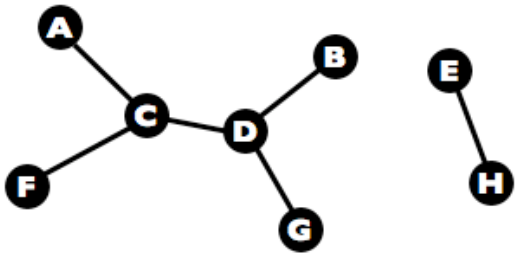
Problem-Based Intro. to Computer Science (20101)
Counting Nodes via Search
Week 9 – Homework

1 Problem

Aside from finding paths in a graph, depth-first (or other) traversals can be used to compute the number of nodes that are connected to a given node. That is, if our graph consists of two separate components, such as the one shown below, we can use a depth-first traversal to detect this condition.



There are several ways of determining the connected-ness of a graph.

One strategy is very similar to the path existence algorithm shown in this week's lecture. After starting at any one node and tracing out all paths, we can simply check the number of visited nodes and see if it is the same as the total number of nodes in the graph. If the graph is a single component, it should not matter where we start the traversal!

We would like also to consider another strategy to check connectedness. We will perform a depth-first traversal, but instead of keeping a list of visited nodes, we will simply keep a count that is incremented for node we visit. In order to avoid revisiting nodes, we will delete each edge as we follow it (this needs to happen just before the recursive call to prevent infinite recursion). The pseudocode for this traversal would look like:

```
DFSnodeWithDeletion( graph, node ):
    count the node itself
    for every neighbor of node:
        remove the edge(s) between the node and neighbor
        add the result of DFSnodeWithDeletion( graph, neighbor ) to count
    return the count
```

The question is, will this second process work correctly for all graphs? If not, why not?

For this assignment, you will need to:

- Implement both strategies for counting nodes in a connected component (you may simply use the first node in your node list as a starting point). Note that the first strategy should be very similar to the lecture code. Also note that the second strategy may simply destroy the graph as long as it is not needed afterward!
- Implement code that asks for the name of an input file (in the same format as in the lecture) and gives the results for each strategy.
- Decide if the second strategy is correct, and, in a comment at the end of the code file submission, do the following:
  - If you think that it is correct, state why you think so.
  - Otherwise, state why you think it is incorrect and prove it by providing a graph that is a counterexample (that is, a graph for which the two strategies produce different node counts). Place the counterexample in your comment.

Put your code and analysis commentary into a file called `nodecount.py` and submit to the MyCourses dropbox.