6.006 Homework	**Problem Set 4, Part A**	# 1 – Cycles
INSERT NAME HERE		April 5, 2011
Collaborators: INSERT COLLABORATORS HERE		

(a) Take the following graph with 3 nodes and 3 edges:

(s,t), (s,v), (v,t). A BFS will discover t twice, because there are two paths to t, but there is no cycle, because the paths only go in one direction (towards t).

(b) If you see a node u twice during a BFS of an undirected graph, then there are two paths from s to u. Simply paste those paths together to make the cycle. It's important that the graph is undirected, because pasting the two paths together to make a cycle requires reversing one of the paths.

(c) Run a DFS on G. If you discover a backedge, declare that there is a cycle. Otherwise, there is no cycle.

If there is a backedge, then it is part of a cycle: Simply take the path currently on the stack for the DFS, and then follow the backedge, and we have a cycle.

If there is a cycle, then there is a backedge. At some point during the DFS, we will visit for the first time a node in the cycle, v. Before we finish with v, we will visit every other node in the cycle, because those nodes are all reachable from v, and none of them have been visited before. In particular, we will visit the node u which precedes v in the cycle, and u will have a backedge to v.

We can check for backedges without affecting the asymptotic running time of DFS, simply by keeping a set of vertices currently on the stack.

(d) An unidirected graph is acyclic (i.e., a forest) if and only if a DFS yields no back edges

(a) if there's a back edge, there is a cycle.

(b) if there are no back edge, we see there are only tree edges, so the graph is acyclic

Thus, we run DFS, and if we find a back edge, there is a cycle. We see by Theorem B.2 on page 1085 of CLRS, in an acyclic (undirected) forest, $|E| \leq |V| - 1$, so the running time is $O(V)$

We shall model the problem using a graph. Let the guys in the club be vertices of the graph, and two vertices share an edge if they are enemies. Our goal is to determine if its nodes can each be assigned a color, either red or blue, such that no red node is adjacent to another red node, and no blue node is adjacent to another blue node. After such coloring is determined, we can let the blue nodes become important members, and red nodes become not important members. In other word, we are trying to decide if the constructed graph is bipartite or not.

Use breadth-first search repeatedly (in case the graph is not connected). Assign the starting point to be red, those at odd depth from the starting point to be blue, and those at even depth to be red. When visiting a node and a neighbor has already been visited, check that they do not have the same color. If they do, return "No assignment possible". There can be no other configuration of the graph coloring which allows for a proper assignment because given only two colors, the colors on opposite ends of an edge must be different so coloring starting from a single source will always yield the same coloring. Starting at a different node will yield the same coloring as well because the graph is undirected. If all nodes are successfully colored, return "Assignment is possible'. This runs in $O(V + E)$, the running time of BFS.

Argument using odd length cycles: In an odd length cycle, coloring with two alternating colors from some start node will end with the last node having the same color as the start node. Thus, any graph containing an odd cycle cannot be bipartite. The above algorithm uses BFS, so any node colored blue is an odd distance away from the starting point while any node colored red is an even distance away from the starting point. If an attempt is made to color a node both colors, this means it is both an odd and even distance from the source node, for an odd total number of nodes in the cycle passing through it, thus an assignment is impossible.