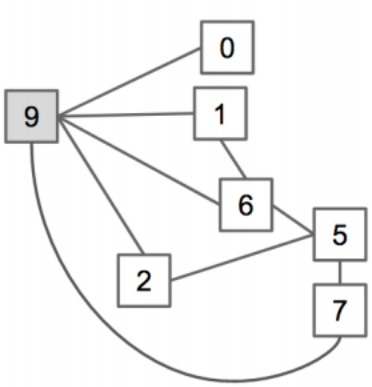
CS 61B Spring 2018

Graphs

Exam Prep 11: April 2, 2018

1 Warmup with DFS and BFS



(a) For the graph above, give the depth first search preorder traversal starting from vertex 9, assuming that we break ties by visiting smaller numbers first.

(b) For the graph above, give the depth first search postorder traversal starting from vertex 9, assuming that we break ties by visiting smaller numbers first.

(c) For the graph above, give the breadth first search traversal starting from vertex 9, assuming that we break ties by visiting smaller numbers first.

2 Cycle Detection

Given an undirected graph, provide an algorithm that returns true if a cycle exists in the graph, and false otherwise. Also provide a Θ bound for the worst case runtime of your algorithm. You may use either an adjacency list or an adjacency matrix to represent your graph.

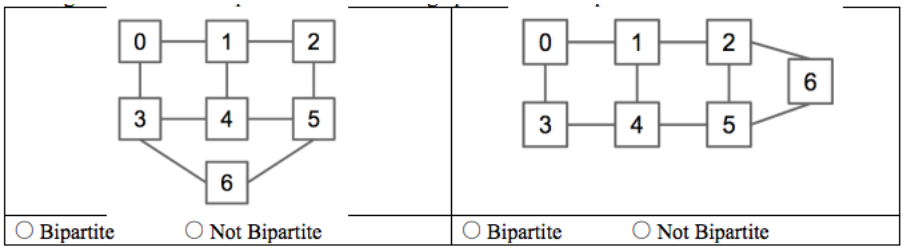
3 Compiler Dependencies

Come up with an efficient algorithm for the following problem: you are given a list of java files, some of which are dependent on one another. For example, `TestRODI.java` from the midterm was dependent on the `ReverseOddDigitIterator` class because it used that object's methods in the test. Therefore, if you want to compile `TestRODI.java`, you have to compile `ReverseOddDigitIterator.java` first. Given the list of java files, your job is to determine the ordering in which to compile the files so that all dependencies are compiled first.

Hint: Think about the different graph traversals that you learned in lecture.

4 Bipartite Graphs

(a) Suppose we want to color every vertex of a graph either blue or green such that no vertex touches anther vertex of the same color. This is possible for some graphs but not others. A graph where a valid coloring exists is called bipartite. Which of the graphs below are bipartite?



(b) Now fill in the method `twocolor` below such that a correct assignment to the blue vertices is printed out when the code runs, or if no such assignment is possible, an exception is thrown. Write only one statement per line. Please use the provided Graph API given to you.

```
public class Graph {
    public Graph(int V):                // Create empty graph with v vertices
    public void addEdge(int v, int w):  // add an edge v-w
    Iterable<Integer> adj(int v):       // vertices adjacent to v
    int V():                            // number of vertices
    int E():                            // number of edges
    ...
}
```

```
HashSet<Integer> blue = new HashSet<Integer>();
HashSet<Integer> green = new HashSet<Integer>();
twocolor(G, 0, blue, green);
System.out.println(Blue vertices are:  + blue.toString());

public static void twocolor(Graph G, int v, Set<Integer> a, Set<Integer> b){
    _________________________________
    for (_____________________________________________) {
        if (_____________________________________) {
            throw new IllegalArgumentException("graph is not bipartite"); }
        if (_____________________________________) {
            _________________________________
        }
    }
}

```