CS330 – Graph Traversal Algorithms

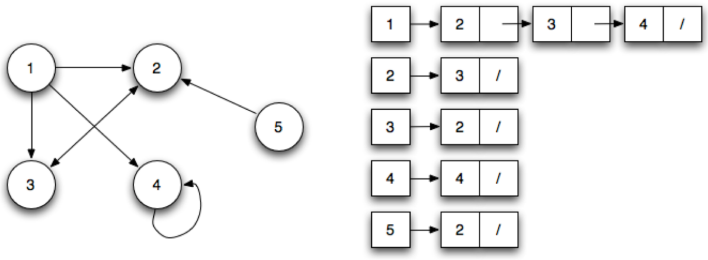
Purpose: A graph is a common and useful data structure in computer science which represents general relationships between items. We will examine how to represent graphs and how to traverse through all the items in a graph. Graph traversals serve as the backbone of a number of algorithms.

Knowledge: This activity will help you become familiar with the following content knowledge:

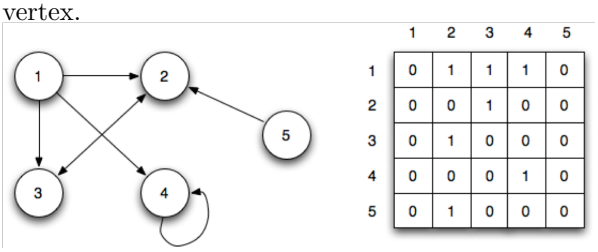
- Graph representations
- Graph traversal algorithms
- Graph algorithms based upon traversals

Activity: With your group perform the following tasks and answer the questions. You will be reporting your answers back to the class in 60 minutes.

1. A graph is comprised of a collection of vertices (or nodes) connected by edges and can be represented with an adjacency list which indicates the edges starting from each vertex.



An alternative representation is to use an adjacency matrix in which one dimension of the matrix indicates the start vertex and the other dimension indicates the end vertex. Values in the matrix indicate where or not there is an edge from the start to the end vertex.



   (a) How much space in Big-Oh notation is required to represent a graph with V number of vertices and E number of edges using an adjacency list?

(b) How much space is required for the adjacency matrix representation?

(c) How much time does it take in Big-Oh notation to add an edge (u,v) to a graph with V number of vertices and E number of edges using an adjacency list?

(d) How much time does it take to add an edge with an adjacency matrix?

(e) The degree of a vertex is the number of edges starting at that vertex. How much time does it take to determine if there is an edge (u,v) in a graph with an adjacency list, assuming that the degree of u is k?

(f) How much time does it take to determine if there is an edge (u,v) in a graph with an adjacency matrix?

2. A graph traversal is an algorithm that visits all the vertices in the graph in some systematic way. A general way to view graph traversals is with the following Tricolor algorithm. In this algorithm **white** nodes are undiscovered nodes that have not been seen yet in the current traversal and may even be unreachable. Whereas **black** nodes are nodes that are reachable and that the algorithm is done with. Finally, **gray** nodes are nodes that have been discovered but that the algorithm is not done with yet. These nodes are on a frontier between white and black.

   Tricolor algorithm:

   Color all nodes white, except for a starting node which is colored gray.

   While some gray node n exists:

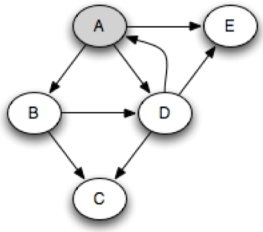
   color some white successor of n gray.

   if n has no white successor then color n black

   We have different ways that we can choose the gray node n. For example, we could put the nodes in a queue and select the nodes in the order in which they become gray. Alternatively we could put the nodes in a stack and select the last node that became gray. We could also select the gray node based on some priority scheme.

Depth First Search is the TriColor algorithm with a stack (implicit in the recursion):

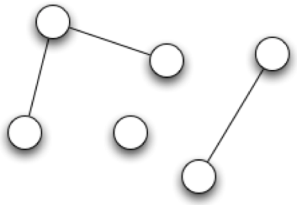
```
DFS(Vertex v){
      set color of v to gray
      for each successor v' of v{
           if v' not yet visited, so white {
                 DFS(v')}
     }
     set color of v to black
}
```



Trace through the algorithm with this graph marking vertices gray and black. What order do they become gray? What order do they become black?

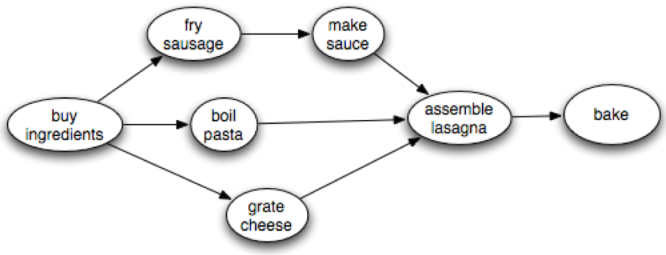
How many calls to DFS are made by the algorithm if we have V vertices and E edges?
What is the total number of times the `for` loop will iterate in the algorithm if we have V vertices and E edges?
What is the order of growth for the time in terms of V and E?

3. We can use graph traversals to solve various problems. One such problem is determining the connected components in a graph. For example the following graph would have three connected components



How can we use DFS to determine the number of connected components in a graph?

4. Another graph problem is a topological sort in which we come up with some ordering of performing tasks, given that some tasks must be performed before others. The ordering is often not unique. This graph specifies which steps must be performed before other steps in making lasagna.

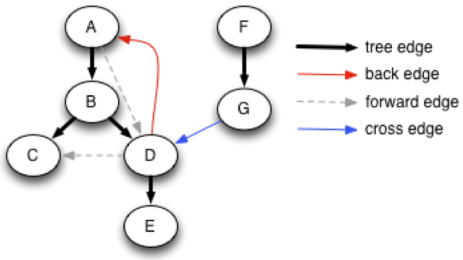


Give some linear ordering of these 7 steps that makes sense.

Now perform a DFS on this graph and note the order in which nodes get marked black. What is your order?

How can we use DFS to give a topological ordering?

5. As we examine edges in the process of performing DFS we can have four different types of edges.

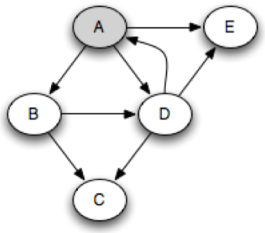


- A tree edge is one in which the destination node is white
- A back edge is one in which the destination node is gray
- A forward is one where the destination node is black and in the same tree
- A cross edge is one where the destination node is black and in another tree

How can we use DFS and back edges to determine if there is a cycle in a graph?

6. Breadth First Search is the TriColor algorithm with a queue:

```
BFS(Vertex root){
      frontier = new Queue()
      set color of root to gray
      while frontier not empty{
           Vertex v = frontier.dequeue()
            for each successor v' of v{
                 if v' not yet visited, so white {
                       frontier.enqueue(v')
                       mark v' gray
                 }
           }
           mark v as black
     }
}
```



Trace through the algorithm with this graph, marking vertices gray and black. What order do they become gray? What order do they become black?

How can we use BFS to find the shortest path (least number of edges) from a given vertex to a second vertex?

Complete the following assignments to be submitted for grading. Each should be done individually but you can consult with a classmate to discuss your strategies.

Download an import the Graph project into Eclipse.

Assignment 1:
Draw the graph in the input file tinyG.txt and determine the order that vertices will be visited using DFS starting at vertex 0. Verify your result be running the code. In Eclipse you can use the run configuration to set the run argument to tinyG.txt

Criteria for Success: You have drawn the graph and listed the order that DFS visits the vertices.

Assignment 2:
Complete the method `connectedComponent` which prints the vertices that are contained in each of the connected components of the graph. You will want to use DFS to do so. Test your code on tinyG.txt

Criteria for Success: Your code lists the vertices in each of the connected components.

Assignment 3:
Make a small modification to the code in the `addEdge` method so that the graphs now have directed rather than undirected edges. Then complete the code for determining if a graph contains a cycle using a modified DFS. Your modified DFS should return true if it finds a back edge. Test your code on graph1.txt, graph2.txt, and graph3.txt.

Criteria for Success: Your code determines if a directed graph contains a cycle or not.