CSM 61B
Fall 2017

Graphs & Searches
Mentoring 12: November 13, 2017

1 Tree Traversal

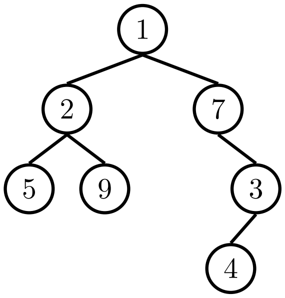
A **queue** is a data structure that orders items in a first-in-first-out (FIFO) manner, meaning that the first element you `add` will be at the front and the last item you `add` will be at the tail. Elements are `removed` from the front.

A **stack** is a data structure that orders items in a last-in-first-out (LIFO) manner, meaning that the first element you `add` will be at the tail and the last item you `add` will be at the front. Elements are `removed` from the front.

```
public class BinaryTree<T> {
    protected Node root;
    protected class Node {
        public T value;
        public Node left;
        public Node right;
    }
}
```

1.1

```
public void treeTraversal(Fringe<Node> fringe) {
    fringe.add(root);
    while (!fringe.isEmpty()) {
        Node node = fringe.remove();
        System.out.print(node.value);
        if (node.left != null) {
            fringe.add(node.left);
        }
        if (node.right != null) {
            fringe.add(node.right);
        }
    }
}
```



What would Java display?

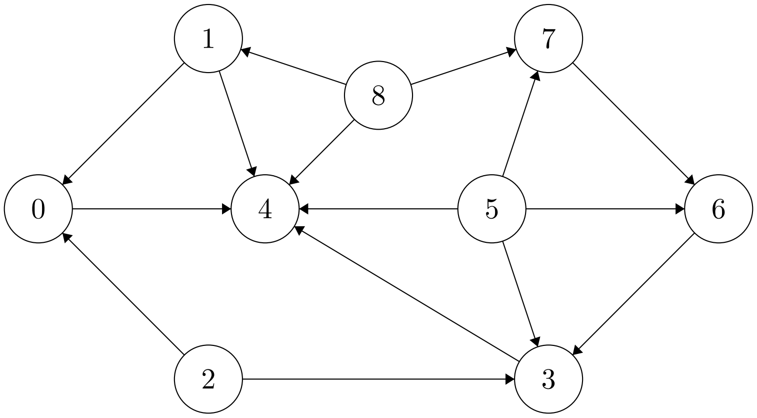
(a) `tree.traversal(new Queue<Node>());`

(b) `tree.traversal(new Stack<Node>());`

2 Graph Traversal

```
procedure GRAPH-TRAVERSAL(start, fringe)
    seen ← an empty set
    ADD(start, fringe)
    while fringe is not empty do
        node ← REMOVE(fringe)
        if node is not in seen then
            ADD(node, seen)
            for child in NEIGHBORS(node) do
                ADD(child, fringe)
```

2.1 Consider the following directed graph. Break ties numerically from least to greatest. For example, when iterating through the edges pointing from vertex 5, consider the edge $(5, 3)$ before the others.



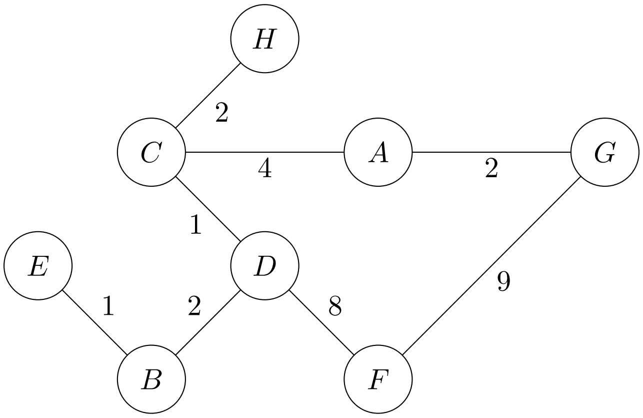
(a) Give the depth-first *pre-order* traversal of the graph.

(b) Give the depth-first *post-order* traversal of the graph.

(c) Give the *reverse* depth-first *post-order* traversal.

3 Searches

3.1 For the graph below, write the order in which vertices are visited using the specified algorithm starting from A. Break ties by alphabetical order. Notice that we have now introduced edge weights to the graph.

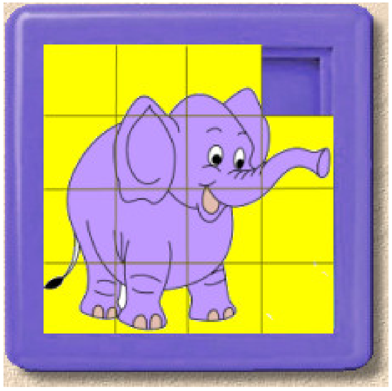


(a) DFS

(b) BFS

(c) Dijkstra's

4 Elephants *Extra Practice*



A `State` of this puzzle is some permutation of the puzzle tiles. There are two things we can do with a `State`:

- Get the set of next possible states from the current `State`.
- Find out if the `State` is a goal: in this case when the puzzle is solved.

Imagine a game graph as a graph of all possible states where each state is a graph node and where a method, `getNextStates`, returns the neighbor nodes. Finding a solution is equivalent to finding a path from some start state to goal state.

4.1 Would BFS or DFS be better for finding the solution that takes the least number of moves to solve the game?

4.2 Define the `solve` method in `PicturePuzzle` which returns the board `State` that is the solution with the least number of moves away from a given starting `State`.

```
public interface State {
    public Set<State> getNextStates();
    public boolean isGoal();
}

public class PicturePuzzle {
    public static State solve(State state) {



    }
}
```