CSM 61B Spring 2017

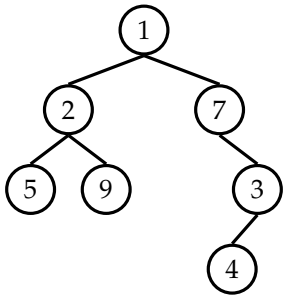
Graphs & Searches

Week 11: April 4, 2017

1 Traversals

1.1
```
public interface Deque<E> {
    void addFirst(E e);
    void addLast(E e);
    E removeFirst(E e);
    E removeLast(E e);
    boolean isEmpty();
}
public class BinaryTree<T> {
    protected Node root;
    protected class Node {
        public T value;
        public Node left;
        public Node right;
    }
    public void queueTraversal() {
        Deque<Node> fringe = new LinkedList<>();
        fringe.addLast(root);
        while (!fringe.isEmpty()) {
            Node node = fringe.removeFirst(); // oldest node
            if (node.left != null) {
                fringe.addLast(node.left);
            }
            if (node.right != null) {
                fringe.addLast(node.right);
            }
            System.out.println(node.value);
        }
    }
    public void stackTraversal() {
        Deque<Node> fringe = new LinkedList<>();
        fringe.addLast(root);
        while (!fringe.isEmpty()) {
            Node node = fringe.removeLast(); // latest node
            if (node.left != null) {
                fringe.addLast(node.left);
            }
            if (node.right != null) {
                fringe.addLast(node.right);
            }
            System.out.println(node.value);
        }
    }
}
```

What will Java display?



(a) `tree.queueTraversal()`

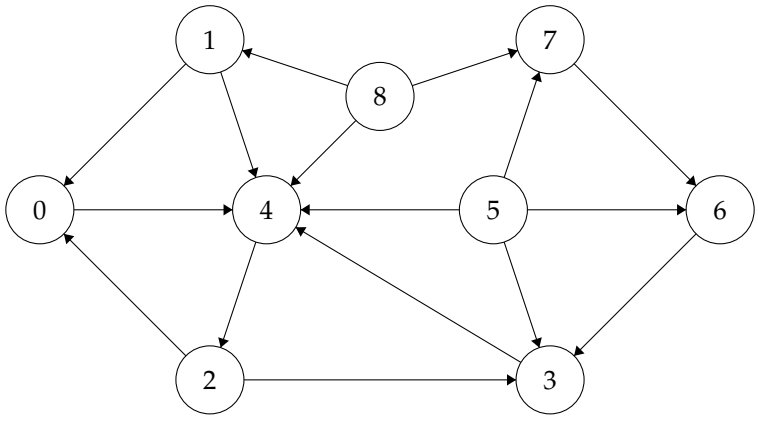
```
1
2
7
5
9
3
4
```

(b) `tree.stackTraversal()`

```
1
7
3
4
2
9
5
```

2 Directed Graphs

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) Starting from 0, give the DFS *pre-order* traversal of the graph.

```
0, 4, 2, 3, 1, 5, 6, 7, 8
```

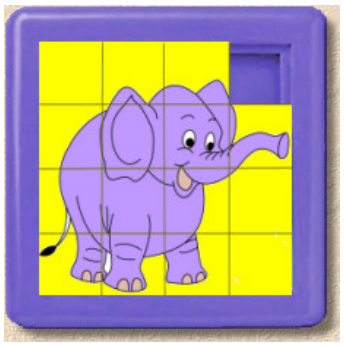
(b) Starting from 0, give the DFS *post-order* traversal of the graph.

```
3, 2, 4, 0, 1, 6, 7, 5, 8
```

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

```
8, 5, 7, 6, 1, 0, 4, 2, 3
```

3 Elephants



Design an algorithm to solve the sliding puzzle.

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.

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

BFS

3.2 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 startState) {
        Set<State> seen = new HashSet<>();
        Queue<State> nextStates = new LinkedList<>();
        seen.add(startState);
        nextStates.add(startState);
        while (!nextStates.isEmpty()) {
            State currentState = nextStates.remove();
            if (currentState.isGoal()) {
                return currentState;
            }
            for (State nextState : currentState.getNextStates()) {
                if (!seen.contains(nextState)) {
                    seen.add(nextState);
                    nextStates.add(nextState);
                }
            }
        }
        return null;
    }
}
```