Directed Graphs

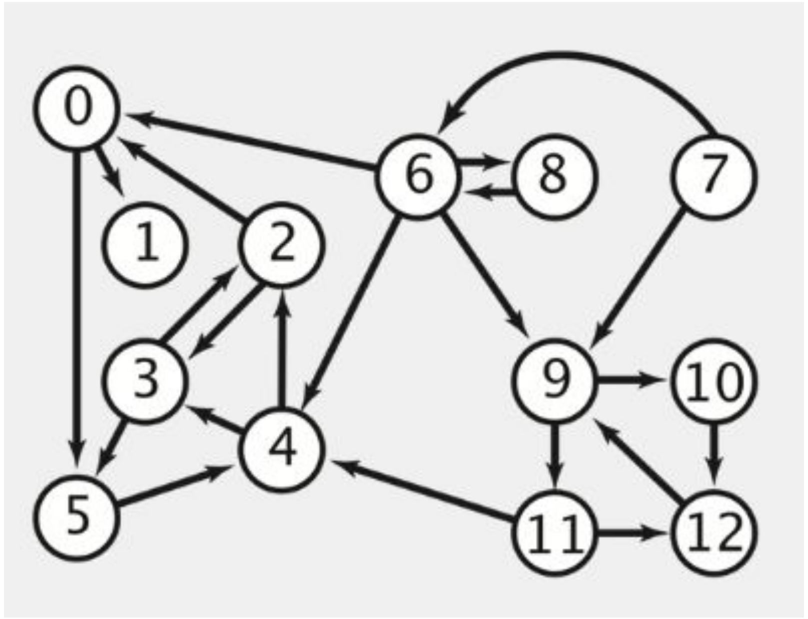
CS112 Recitation

Coding Files:
https://drive.google.com/drive/folders/1WnJX2L6QostvHijbvuOv4yX7dmOhRbMD?usp=sharing

Hoai-An Nguyen
hnn14@scarletmail.rutgers.edu

Let's Review

- Digraphs are almost identical to regular graphs, but differ in that their edges have direction
- Like a regular graph, it can be represented by adjacency lists or an adjacency matrix
- Depth-first searches and breadth-first searches can be performed on either type of graph
- Topological sorts are unique to digraphs



Warm-Up

1. What is the space complexity of a graph…
   a. represented by adjacency lists?
   b. represented by an adjacency matrix?
2. For each representation, identify one advantage.
3. What is the difference between a DFS and a BFS?

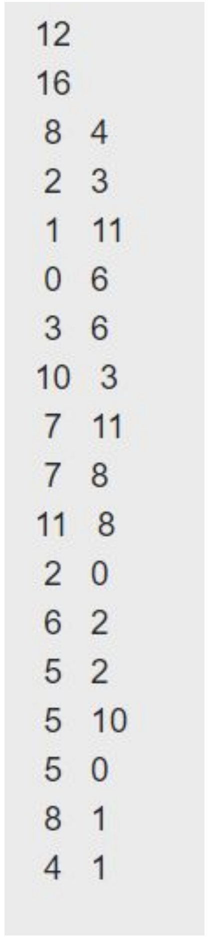
Warm-Up

1. What is the space complexity of a graph…
   a. represented by adjacency lists? **O(V+E)**
   b. represented by an adjacency matrix? **O(V^2)**
2. For each representation, identify one advantage.
   a. **AL: Less memory consumed, you can iterate through all edges of the graph quickly**
   b. **AM: Quick edge lookup time**
3. What is the difference between a DFS and a BFS?
   a. **In a DFS, we start from the root node and visit nodes as far as possible from the root node**
   b. **In a BFS, we proceed level by level, visiting a node's neighbors before moving on to *their* neighbors**

Problem 1: Adjacency Lists

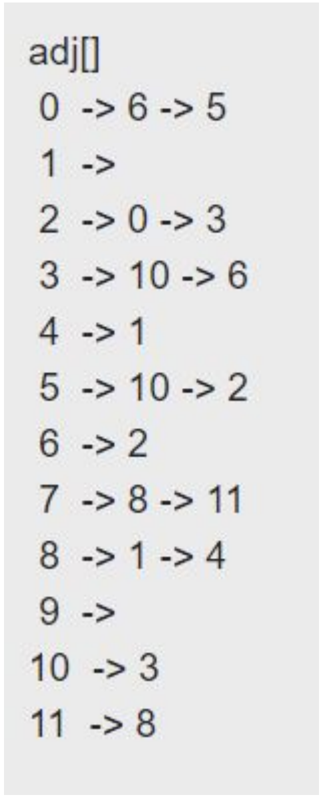
Draw the adjacency list built by Digraph's input stream constructor for the following text file.

The first line corresponds to the number of vertices, the second line corresponds to the number of edges. The remaining lines represent edges between two vertices. (text file on next slide)



```
12
16
 8  4
 2  3
 1  11
 0  6
 3  6
10  3
 7  11
 7  8
11  8
 2  0
 6  2
 5  2
 5  10
 5  0
 8  1
 4  1
```

Problem 1: Solution



```
adj[]
 0  -> 6 -> 5
 1  ->
 2  -> 0 -> 3
 3  -> 10 -> 6
 4  -> 1
 5  -> 10 -> 2
 6  -> 2
 7  -> 8 -> 11
 8  -> 1 -> 4
 9  ->
10  -> 3
11  -> 8
```

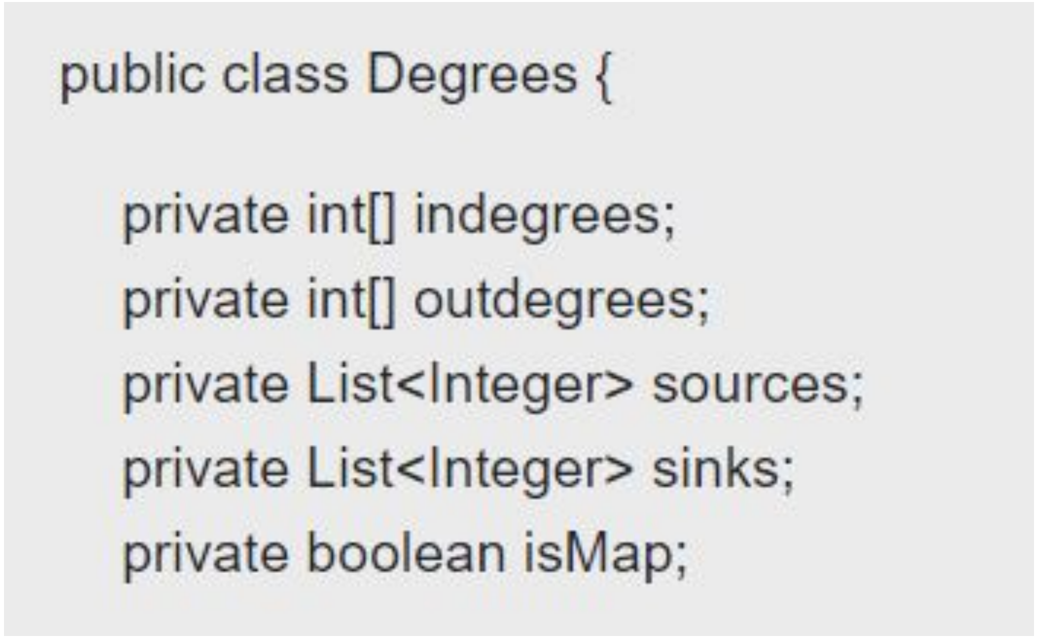
Problem 2: Directed Graphs

The indegree of a vertex in a digraph is the number of directed edges that point to that vertex. The outdegree of a vertex in a digraph is the number of directed edges that emanate from that vertex. No vertex is reachable from a vertex of outdegree 0, which is called a sink; a vertex of indegree 0, which is called a source, is not reachable from any other vertex.

A digraph where self-loops are allowed and every vertex has outdegree 1 is called a map (a function from the set of integers from 0 to V-1 onto itself). Write the program Degrees.java that implements the following API (look at coding file to get started):

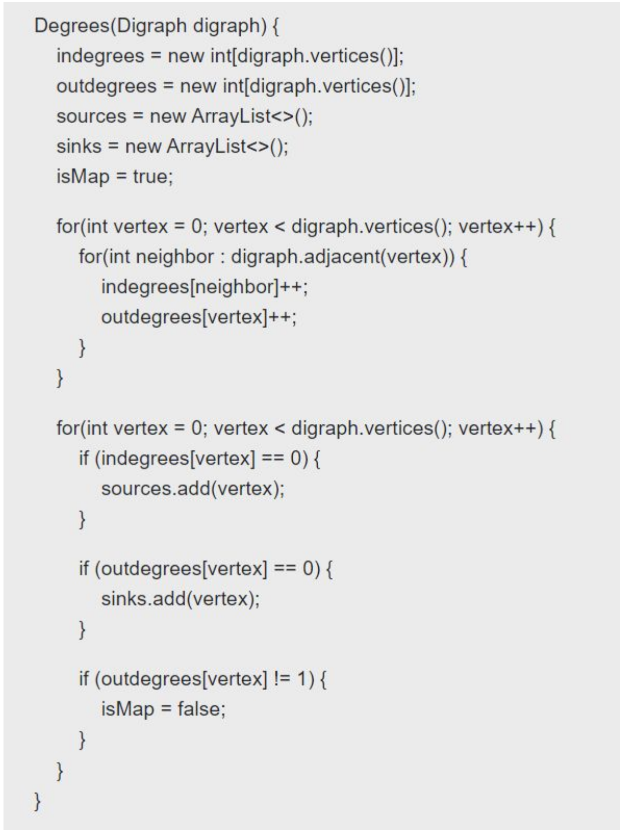
```
public class Degrees {
    Degrees(Digraph G)   // constructor
    int indegree(int v)  // indegree of v
    int outdegree(int v) // outdegree of v
    Iterable<Integer> sources() // sources
    Iterable<Integer> sinks()   // sinks
    boolean isMap()           // is G a map?
}
```

Problem 2: Solution Pt 1



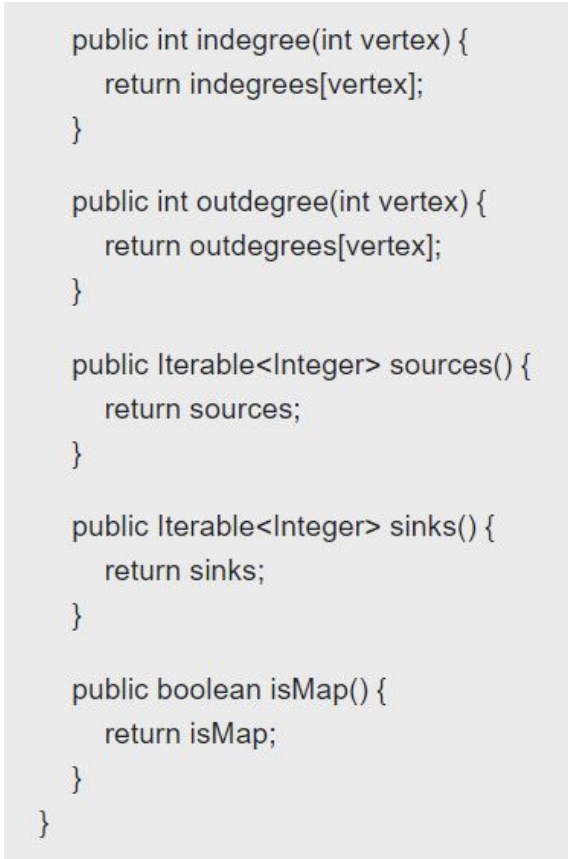
```
public class Degrees {

    private int[] indegrees;
    private int[] outdegrees;
    private List<Integer> sources;
    private List<Integer> sinks;
    private boolean isMap;
```

Problem 2: Solution Pt 2



```
Degrees(Digraph digraph) {
    indegrees = new int[digraph.vertices()];
    outdegrees = new int[digraph.vertices()];
    sources = new ArrayList<>();
    sinks = new ArrayList<>();
    isMap = true;

    for(int vertex = 0; vertex < digraph.vertices(); vertex++) {
        for(int neighbor : digraph.adjacent(vertex)) {
            indegrees[neighbor]++;
            outdegrees[vertex]++;
        }
    }

    for(int vertex = 0; vertex < digraph.vertices(); vertex++) {
        if (indegrees[vertex] == 0) {
            sources.add(vertex);
        }

        if (outdegrees[vertex] == 0) {
            sinks.add(vertex);
        }

        if (outdegrees[vertex] != 1) {
            isMap = false;
        }
    }
}
```

Problem 2: Solution Pt 3



```
    public int indegree(int vertex) {
        return indegrees[vertex];
    }

    public int outdegree(int vertex) {
        return outdegrees[vertex];
    }

    public Iterable<Integer> sources() {
        return sources;
    }

    public Iterable<Integer> sinks() {
        return sinks;
    }

    public boolean isMap() {
        return isMap;
    }
}
```

Good Work!

Go to https://dynrec.cs.rutgers.edu/live/

Enter the Quiz Code: