CS 340 Fall 2014

Assignment 1

Due 11:59 PM Friday September 19

Assignment 1

- The primary goal of this assignment is to implement the topological sorting algorithm for both the adjacency matrix representation of a graph and the adjacency list representation of a graph. The secondary goal is for you to review Java, review some data structures you learned in CS 220 and to practice reading Java documentation.
- The next slide shows the main static method for the program. The method expects one command line argument. The argument is the name of a file that contains a representation of a graph. The file is assumed to be located in the current working directory. **Do not change the contents of main**.
- Note the simple use of polymorphism when topo is called

Main

```
public class TopoTest {

        public static void main(String args[]) {

                DirectedGraph g1 = new AdjMatrixGraph(args[0]);
                DirectedGraph g2 = new AdjListGraph(args[0]);

                System.out.println("A topological order for the "+
                        "graph in the file " +args[0]+":"+g1.topo());

                System.out.println("A topological order for the "+
                        "graph in the file " +args[0]+":"+g2.topo());
        }
}
```

File Format

- The representation of a graph in the file contains the following information. A line containing an integer, n, where n is the number of nodes in the graph. A sequence of n lines each of which contains a name associated with a node. The names will only contains letters and digits. Following the names will be one line for each edge. An edge will be represented by a pair of names separated by a space. On the next slide is the file representation for the first directed acyclic graph shown in the introduction slides.

File Format

8
A
B
C
D
E
F
G
H
A B
A C
B E
B D
C D
C F
F G
E H
E G
D G

Classes

- Use the following classes and interface as a starting point. You must use these classes and interface. For the adjacency list class you cannot create an array of generic lists so create an ArrayList of generic lists. You might not have used ArrayList before but you can look at the documentation to figure out to how to use it.

Directed Graph

```
public abstract class DirectedGraph {

    protected String vertexNames[];
    protected BufferedReader b;
    protected int numNodes;

    public DirectedGraph(String filename) {
    //Do not change this code
        try {
            b = new BufferedReader(new FileReader(filename));
            numNodes = Integer.parseInt(b.readLine());
        }
        catch (Exception e) {
            System.out.println("Error in file contents or file not found");
        }
    }
```

DirectedGraph

```
protected void buildGraph() {
//PRE: b has been assigned and the first line in the file associated with b has been
//        read and its value has been assigned to numNodes
//This method reads the remaining contents of the file associated with b. It stores the
//vertex names in the vertexNames array and builds the graph representation using
//the contents of the file and the addEdge method



}
```

DirectedGraph

```
    protected int getNodeNum(String v) {
    //PRE: the vertexNames array has been created and contains the names of numNode
    //         vertices
    //The function can be used to find the number associated with a vertexName
            for (int i = 0; i < numNodes; i++)
                    if (vertexNames[i].equals(v))
                            return i;
            return -1; //this means the name was not found. This should not happen if
                          //the data is correct. You can assume the data is correct
    }


    public abstract void addEdge(int v1, int v2);

    public abstract String topo();
}
```

AdjMatrixGraph

```
public class AdjMatrixGraph extends DirectedGraph {
        int g[][]; //the adjacency matrix

        public AdjMatrixGraph(String filename) {
        //Do not change this constructor
                super(filename);
                g = new int[numNodes][numNodes];
                buildGraph();}
        }

        public void addEdge(int v1, int v2) {
        //adds an edge to the adjacency matrix

        }
```

AdjMatrixGraph

```
      public String topo() {
      //returns a string of vertex names in a topological order
      //the names are separated by commas
      //if a topological order does not exist return the string "no topological order found"
      //Use an instance of MyQueue for the fifo queue used in the algorithm


      }
}
```

AdjListGraph

```
public class AdjListGraph extends DirectedGraph {
    ArrayList<DoubleLinkedList<Integer>> g; //adjacency list representation

    public AdjListGraph(String filename) {
    //do not change this constructor
        super(filename);
        g = new ArrayList<DoubleLinkedList<Integer>>();
        for (int i = 0; i < numNodes; i++)
            g.add(new DoubleLinkedList<Integer>());
        buildGraph();
    }


    public void addEdge(int v1, int v2) {
    //add v2 to the adjacency list associated with vertex v1

    }
```

AdjListGraph

```
    public String topo() {
    //returns a string of vertex names in a topological order
    //the names are separated by commas
    //if a topological order does not exist return the string "no topological order found"
    //Use an instance of MyQueue for the fifo queue used in the algorithm

    }

}
```

FIFOQueue

```
public interface FIFOQueue<T> {
//The interface implemented by MyQueue
        void enqueue(T t);

        T serve();

        boolean empty();

}
```

MyQueue

```
public class MyQueue<T> implements FIFOQueue<T> {

        DoubleLinkedList<T> q; //Use a insatnce DoubleLinkedList to implement MyQueue

        public MyQueue() {
                q = new DoubleLinkedList<>();
        }
        public boolean empty() {
        //return true if the queue us empty otherwise return false
        }
        public void enqueue(T d) {
        //add an item to the rear of the queue
        }
        public T serve() {
        //PRE: the queue us not empty
        //remove and return the item at the front of the queue
        }
}
```

DoubleLinkedList

```
public class DoubleLinkedList<T> {
//A generic double linked list
//Implement the list with a sentinel node at the  front and end of the list

    class Node {
        Node prev;
        T data;
        Node next;

        Node(Node p, T d, Node n) {
            prev = p;
            data = d;
            next = n;
        }
    }
```

DoubleLinkedList

```
Node head;
Node current; //references the node following the node in the current position
               //unless the list is empty in which case it references the same node as
              //last
Node last;
int position; //this value is only meaningful when the list is not empty
int size;
```

DoubleLinkedList

```
public DoubleLinkedList() {
        head = new Node(null, null, null);
        head.next = new Node(head, null, null);
        size = 0;
        position = 0;
        current = head.next;
        last = current;

}

public void insertLast(T d) {
//add a new value to the end of the list

}
```

DoubleLinkedList

```
public void insertAt(int p, T d) {
//PRE: 0>= p <= size
//add a value at position p
//positions begin with 0
//Use the value of position (unless the list is empty) and current so the method
//does not always have to traverse the list beginning at the head
//Write a separate test driver to test this method because you might not use it in the
//topological sort implementation
}

public T deleteFirst() {
//PRE: size > 0
//remove and return the first element in the list
}
```

DoubleLinkedList

```
    public int getSize() {
        return size;
    }

    public int getPosition() {
    //PRE: size > 0
        return position;
    }

    public T getDataAt(int p) {
    //PRE: size > 0 AND 0 <= p < size
    }

}
```

Assignment Submission

- You must email the following files. Do not email me an Eclipse project or other files.
  - DirectedGraph.java
  - AdjMatrixGraph.java
  - AdjListGraph.java
  - MyQueue.java
  - DoubleLinkedList.java
  - TestDoubleLinkedList.java
    - This should test all methods in the DoubleLinkedList class
  - Two test data files containing graph representations
    - Your program must work for these files
    - One file should contain a graph representation for a graph that has a topological ordering and the other file should contain a graph representation for a graph that has a cycle