Example Graphs

```
e = example graphs
r = random graph
any other input will quit the program.
e
G1 Adjacency Matrix
0 1 1 1 0 0 0
0 0 0 1 0 0 0
0 0 0 1 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 1 0 0

Topological Order
e 4/5
d 3/6
b 2/7
c 8/9
a 1/10
f 11/12
g 13/14


G2 Adjacency Matrix
0 1 1 0 0 0
0 0 1 1 1 0
0 0 0 0 1 0
0 0 0 0 0 1
0 1 0 1 0 0
0 0 0 0 1 0
Cycle detected, topological sort is impossible
```

```
e = example graphs
r = random graph
any other input will quit the program.
r
# vertices: 15

# edges, max 120: 50
0 0 0 0 0 0 1 1 0 0 1 0 0 0 0
0 1 0 0 0 1 0 1 0 0 0 1 0 0 1
0 0 0 0 0 0 0 1 0 0 0 0 1 1 0
1 1 0 1 0 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 0 0 1 1 0 0
1 1 1 0 0 0 0 0 0 0 0 0 1 1 0
0 1 0 0 0 0 0 1 0 0 0 1 0 0 0
1 0 0 0 0 1 0 1 0 0 0 0 1 0 0
0 1 1 1 0 1 0 0 1 1 0 0 1 0 0
0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 1 1 0 0 0 0 1 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
Cycle detected, topological sort is impossible
```

```
e = example graphs
r = random graph
any other input will quit the program.
r
# vertices: 15

# edges, max 120: 10
0 0 0 0 0 1 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0

Topological Order
l 2/3
f 4/5
a 1/6
b 7/8
e 9/10
g 11/12
n 15/16
m 14/17
i 13/18
j 19/20
k 21/22
o 23/24
```

Random Graph

Random Graph - Cycle

```
package Lab8;

import java.util.Scanner;

public class Lab8 {
   public static void main(String args[]) {
      Scanner in = new Scanner(System.in);
      Graph g1 = Graph.generateG1();
      Graph g2 = Graph.generateG2();
      while (true) {
         System.out.println("e = example graphs\nr = random graph\nany other input will quit the program.");
         String input = in.next().toLowerCase();
         if (input.equals("e")) {
            System.out.println("G1 Adjacency Matrix");
            g1.printAdjacencyMatrix();
            g1.DFS(g1.getVertex(0));
            System.out.println("\nG2 Adjacency Matrix");
            g2.printAdjacencyMatrix();
            g2.DFS(g2.getVertex(0));
         } else if (input.equals("r")) {
            System.out.print("# vertices: ");
            int v = in.nextInt();
            System.out.printf("\n# edges, max %d: ", v*(v+1) / 2);
            int e = in.nextInt();
            Graph rand = Graph.generateRandom(v, e);
            rand.printAdjacencyMatrix();
            rand.DFS(rand.getVertex(0));
         } else {
            break;
         }
      }
      in.close();
   }
}


package Lab8;

import java.util.ArrayList;

public class Node {

   private char key;
   private int dst;
   private int start;  //start time
   private int end;    //end time
   private Node parent;
   private ArrayList<Node> adj;

   public Node(char name)
   {
      key = name;
      adj = new ArrayList<>();
```

```
        dst = 0;
        parent = null;
        start = -1;
        end = -1;
    }

    //This method was altered to simulate directed edges
    public void add(Node node)
    {
            adj.add(node);
    }

    public boolean checkAdj(Node node)
    {
        for (Node n : adj) {
            if (n.key == node.key)
                return true;
        }
        return false;
    }

    //mutators
    public void setDst(int d) { dst = d; }
    public void setParent(Node n) { parent = n; }
    public void setStart(int i) { start = i; }
    public void setEnd(int i) { end = i; }

    //accessors
    public char getKey() { return key; };
    public int getDst() { return dst; }
    public Node getParent() { return parent; }
    public ArrayList<Node> getAdj() { return adj; }
    public int getStart() { return start; }
    public int getEnd() { return end; }

    public String toString() { return String.valueOf(key); }
}


package Lab8;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Random;

public class Graph {

    private ArrayList<Node> vertices;

    public Graph()
    {
        vertices = new ArrayList();
    }
```

```
public Graph(ArrayList<Node> copy)
{
   vertices = new ArrayList(copy);
}

public void DFS(Node initial)
{
   boolean cycle = false;
   int time = 0;
   Queue<Node> tpl = new LinkedList();
   for (Node node : vertices) {
      if (node.getStart() == -1)
         cycle = dfsVisit(node, time, tpl);
      if  (node.getEnd() != -1 && node.getEnd() > time) //time tracker
         time = node.getEnd();
   }

   if (!cycle) {
      System.out.println("\nTopological Order");
      for (Node n : tpl) {
         System.out.printf("%c %d/%d\n", n.getKey(), n.getStart(), n.getEnd());
      }
      System.out.println();
   } else{
      System.out.println("Cycle detected, topological sort is impossible\n");
   }

}

private boolean dfsVisit(Node node, int time, Queue<Node> list)
{
   boolean cycle = false;
   time++;
   node.setStart(time);
   for (int i = 0; i < node.getAdj().size() && !cycle; i++) {
      Node v = node.getAdj().get(i);
      if (v.getStart() != -1 && v.getEnd() == -1) {
         return true;
      }
      if (v.getStart() == -1) {
         v.setParent(node);
         cycle = dfsVisit(v, time, list);
      }
      if  (v.getEnd() != -1 && v.getEnd() > time) //time tracker
         time = v.getEnd();
   }
   if (!cycle) {
      time++;
      node.setEnd(time);
      list.add(node);
   }
   return cycle;
}
```

```
public void printAdjacencyMatrix()
{
   int[][] matrix = new int[vertices.size()][vertices.size()];
   for (int row = 0; row < vertices.size(); row++) {
      for (int col = 0; col < vertices.size(); col++) {
         Node one = vertices.get(row);
         Node two = vertices.get(col);
         if (one.getAdj().contains(two)) {
            matrix[row][col]++;
         }
         System.out.print(matrix[row][col] + " ");
      }
      System.out.println();
   }
}

public static Graph generateRandom(int v, int e)
{
   Random rand = new Random();
   ArrayList<Node> list = new ArrayList();
   for (char name = 0; name < v; name++){
      list.add(new Node((char) (name + 97)));
   }
   for (int i = 0; i < e; i++) {
      Node nodeOne = list.get(rand.nextInt(v));
      Node nodeTwo = list.get(rand.nextInt(v));
      if (!nodeOne.checkAdj(nodeTwo))
         nodeOne.add(nodeTwo);
      else
         i--;
   }
   return new Graph(list);
}


public static Graph generateG1()
{
   ArrayList<Node> example = new ArrayList();
   Node a = new Node('a');
   Node b = new Node('b');
   Node c = new Node('c');
   Node d = new Node('d');
   Node e = new Node('e');
   Node f = new Node('f');
   Node g = new Node('g');

   a.add(b);
   a.add(c);
   a.add(d);
   b.add(d);
   c.add(d);
   d.add(e);
   f.add(e);
   g.add(e);
```

```
        example.add(a);
        example.add(b);
        example.add(c);
        example.add(d);
        example.add(e);
        example.add(f);
        example.add(g);
        return new Graph(example);
    }

    public static Graph generateG2()
    {
        ArrayList<Node> example = new ArrayList();
        Node a = new Node('a');
        Node b = new Node('b');
        Node c = new Node('c');
        Node d = new Node('d');
        Node e = new Node('e');
        Node f = new Node('f');

        a.add(b);
        a.add(c);
        b.add(c);
        b.add(d);
        b.add(e);
        c.add(e);
        d.add(f);
        e.add(b);
        e.add(d);
        f.add(e);

        example.add(a);
        example.add(b);
        example.add(c);
        example.add(d);
        example.add(e);
        example.add(f);
        return new Graph(example);
    }

    public void add(Node node) { vertices.add(node); }
    public ArrayList<Node> getGraph() { return vertices; }
    public Node getVertex(int i) { return vertices.get(i); }        //list index
    public Node getVertex(char i) { return vertices.get(i - 97); }  //char key
    public int size() { return vertices.size(); };

}
```