Extra work done:

Node class used to store objects in Graph class

Random graphs

Output

```
A = Part A
B = Part B
All other input will exit.
a
Part A
Example Graph
0 0 1 1 0 0 0 0
0 0 1 0 1 0 0 0
1 1 0 1 0 0 0 0
1 0 1 0 1 1 0 0
0 1 0 1 0 1 0 0
0 0 0 1 1 0 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
Please enter initial vertex from a - h to perform BFS: a
From initial node 'a':
'a', dst 0  Shortest Path: Initial/Unconnected.
'b', dst 2  Shortest Path: b c
'c', dst 1  Shortest Path: c
'd', dst 1  Shortest Path: d
'e', dst 2  Shortest Path: e d
'f', dst 2  Shortest Path: f d
'g', dst 0  Shortest Path: Initial/Unconnected.
'h', dst 3  Shortest Path: h f d
```

```
Random Graph
Please enter number of vertices: 20
Please enter number of edges, max 210: 100
0 0 0 0 1 1 0 0 0 1 1 1 1 0 0 1 1 0 0 0
0 1 1 1 1 1 0 0 1 1 1 0 0 1 1 1 1 1 0 1
0 1 0 0 0 1 0 1 1 1 1 0 1 1 1 0 0 0 0 0
0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0
1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 1 0 0
1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0
0 0 0 1 0 1 1 0 0 1 0 0 1 1 0 1 0 1 1 1
0 0 1 1 0 1 0 1 0 0 1 1 1 0 0 1 1 1 0 0
0 1 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1 0
1 1 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 0 0 1
1 1 1 0 1 1 0 1 0 1 1 1 1 0 0 1 1 1 1 1
1 0 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 0 1 1
1 0 1 0 1 0 1 1 1 0 1 1 1 0 1 0 0 1 0 0 1
0 1 1 0 1 1 1 0 1 1 0 1 1 1 0 0 0 1 0 0 1
0 1 1 0 1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0
1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0
1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 0 1 1 1
0 1 0 0 1 1 1 1 1 0 1 0 0 0 0 1 0 0 1
0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 1 0 0 1
0 1 0 0 0 0 1 0 0 1 1 1 1 1 0 0 1 1 1 1
Please enter initial vertex from a - t to perform BFS: g
From initial node 'g':
'a', dst 2  Shortest Path: a j
'b', dst 2  Shortest Path: b r
'c', dst 2  Shortest Path: c j
'd', dst 1  Shortest Path: d
'e', dst 2  Shortest Path: e r
'f', dst 1  Shortest Path: f
'g', dst 0  Shortest Path: Initial/Unconnected.
'h', dst 2  Shortest Path: h r
'i', dst 2  Shortest Path: i r
'j', dst 1  Shortest Path: j
'k', dst 2  Shortest Path: k r
'l', dst 2  Shortest Path: l t
'm', dst 1  Shortest Path: m
'n', dst 1  Shortest Path: n
'o', dst 3  Shortest Path: o i r
'p', dst 1  Shortest Path: p
'q', dst 2  Shortest Path: q r
'r', dst 1  Shortest Path: r
's', dst 1  Shortest Path: s
't', dst 1  Shortest Path: t
```

```
Part B
Example Graph
0 0 0 1 0 0 0 0 0
0 0 0 1 0 1 0 0 0
0 0 0 1 1 0 0 0 0
1 1 1 0 0 1 0 0 0
0 0 1 0 0 0 0 0 0
0 1 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1
0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0
Graph is bipartite.
'a' blue
'b' blue
'c' blue
'd' red
'e' red
'f' red
'g' blue
'h' red
'h' red
```

```
Random Graph
Please enter number of vertices: 10
Please enter number of edges, max 55: 20
0 1 0 1 1 1 1 1 0 0
1 0 1 0 0 0 1 0 0 0
0 1 0 0 0 0 1 1 0 1
1 0 0 0 0 0 0 0 1 1
1 0 0 0 0 1 0 1 1 0
1 0 0 0 1 0 0 0 0 0
1 1 1 0 0 0 0 1 1 0
1 0 1 0 1 0 1 0 0 0
0 0 0 1 1 0 1 0 0 1
0 0 1 1 0 0 0 0 1 1
Graph is bipartite.
'a' blue
'b' red
'c' blue
'd' red
'e' red
'f' red
'g' red
'h' red
'i' blue
'j' red
```

```
package Lab7;

import java.util.Scanner;

public class Lab7 {
   public static void main(String args[]) {
      //part A
      Scanner in = new Scanner(System.in);
      boolean menu = true;
      while (menu) {
         System.out.println("A = Part A\nB = Part B\nAll other input will exit.");
         String selection = in.next().toUpperCase();
         if (selection.equalsIgnoreCase("A")) {
            //part A
            Graph example = Graph.generateExampleA();
            System.out.print("Part A\nExample Graph\n");
            example.printAdjacencyMatrix();
            System.out.print("Please enter initial vertex from a - h to perform BFS: ");
            char c = in.next().charAt(0);
            example.BFS(example.getVertex(c));

            System.out.print("\nRandom Graph\nPlease enter number of vertices: ");
            int v = in.nextInt();
            System.out.printf("Please enter number of edges, max %d: ", (v * (v + 1)) / 2);
            int e = in.nextInt();
            example = Graph.generateRandom(v, e);
            example.printAdjacencyMatrix();
            System.out.printf("Please enter initial vertex from %s - %s to perform BFS: ",
                  example.getGraph().get(0), example.getGraph().get(example.size() - 1));
            c = in.next().charAt(0);
            example.BFS(example.getVertex(c));
            System.out.println();
         } else if (selection.equalsIgnoreCase("B")){
            //part B
            System.out.println("\nPart B\nExample Graph");
            Graph example = Graph.generateExampleB();
            example.printAdjacencyMatrix();
            example.explore();
            System.out.print("\nRandom Graph\nPlease enter number of vertices: ");
            int v = in.nextInt();
            System.out.printf("Please enter number of edges, max %d: ", (v * (v + 1)) / 2);
            int e = in.nextInt();
            example = Graph.generateRandom(v, e);
            example.printAdjacencyMatrix();
            example.explore();
            System.out.println();
         }
         else { menu = false; }
      }
      in.close();
   }
}
```

```
package Lab7;

import java.util.ArrayList;

public class Node {

    private char key;
    private int dst;
    private Node parent;
    private ArrayList<Node> adj;
    private String color;

    public Node(char name)
    {
        key = name;
        adj = new ArrayList<>();
        dst = 0;
        parent = null;
    }

    public void add(Node node)
    {
            adj.add(node);
            node.adj.add(this);
    }

    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 setColor(String str) { color = str; }

    //accessors
    public int getDst() { return dst; }
    public Node getParent() { return parent; }
    public String getColor() { return color; }
    public ArrayList<Node> getAdj() { return adj; }

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


package Lab7;

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 BFS(Node initial)
   {
      Queue<Node> q = new LinkedList();
      q.add(initial);
      while(q.size() > 0) {
         Node nextNode = q.remove();
         for (Node n : nextNode.getAdj()) {
            if(n.getParent() == null && n != initial) {
               q.add(n);
               n.setParent(nextNode);
               n.setDst(nextNode.getDst() + 1);
            }
         }
      }
      printBFS(initial);
   }

   //helper method to print BFS
   private void printBFS(Node initial)
   {
      System.out.printf("From initial node '%s':", initial);
      for (Node n : vertices) {
         System.out.printf("\n'%s', dst %d\tShortest Path: ", n, n.getDst());
         if (n.getParent() == null)
            System.out.print("Initial/Unconnected.");
         while (n.getParent() != null) {
            System.out.print(n + " ");
            n = n.getParent();
         }
      }
      System.out.println();
   }

   public void explore()
   {
      Node first = vertices.get(0);
      for (Node n : vertices)
```

```
        n.setColor("gray");
    first.setColor("blue");
    boolean result = isBipartite(first);
    for (Node n : vertices) {
        if (n.getColor() == "gray") {
            n.setColor("blue");
            result = isBipartite(n);
        }
    }
    if (result)
        System.out.println("Graph is bipartite.");
    else
        System.out.println("NOT bipartite.");
    for (Node n : vertices)
        System.out.printf("'%s'\t%s\n", n, n.getColor());
}

public boolean isBipartite(Node node)
{
    Queue<Node> q = new LinkedList();
    boolean bipartite = true;
    q.add(node);
    while(q.size() > 0 && bipartite) {
        Node u = q.remove();
        for (Node v : u.getAdj()) {
            if (v.getColor() == "gray") {
                v.setColor("red");
            } else if (v.getColor() == u.getColor()) {
                bipartite = false;
            }
        }
    }
    return bipartite;
}

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 generateExampleA()
{
    ArrayList<Node> list = new ArrayList();
    Node a = new Node('a');
    list.add(a);
    Node b = new Node('b');
    list.add(b);
    Node c = new Node('c');
    list.add(c);
    Node d = new Node('d');
    list.add(d);
    Node e = new Node('e');
    list.add(e);
    Node f = new Node('f');
    list.add(f);
    Node g = new Node('g');
    list.add(g);
    Node h = new Node('h');
    list.add(h);
    a.add(c);
    a.add(d);
    b.add(c);
    b.add(e);
    c.add(d);
    d.add(f);
    d.add(e);
    e.add(f);
    f.add(h);
    return new Graph(list);
}

public static Graph generateExampleB()
{
    ArrayList<Node> list = new ArrayList();
    Node a = new Node('a');
    list.add(a);
    Node b = new Node('b');
    list.add(b);
```

```
        Node c = new Node('c');
        list.add(c);
        Node d = new Node('d');
        list.add(d);
        Node e = new Node('e');
        list.add(e);
        Node f = new Node('f');
        list.add(f);
        Node g = new Node('g');
        list.add(g);
        Node h = new Node('h');
        list.add(h);
        Node i = new Node('i');
        list.add(h);
        a.add(d);
        c.add(d);
        c.add(e);
        b.add(d);
        b.add(f);
        d.add(f);
        g.add(h);
        g.add(i);
        i.add(h);
        return new Graph(list);
    }

    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(); };

}
```