Lab 15

CSE 274

I. The Vertex class

The `Vertex` class is as follows:

```
class Vertex
{
String label;

Vertex(String theLabel)
      {
      label = theLabel;
      }
}
```

II. The Graph class

The `Graph` class is as follows:

```
class Graph
{
private boolean[][] Adjacency;
private Vertex[] VertexArray;
int size;

Graph()
      {
      size = 0;
      }
}
```

The variable `Adjacency` is the adjacency matrix of the graph data structure. The `VertexArray` stores the vertexes of the graph. The `size` is the size of `VertexArray` and shows the number of vertexes in the graph. The `Graph` class has an `addVertex` method to add vertexes to the graph:

```
public void addVertex(String label)
```

The above method increases the `size` by one, creates a new adjacency matrix of `size` × `size` and copies the content of the old adjacency matrix to the top-left corner of the new adjacency matrix. The `Graph` method has an `addEdge` method to add edges to the graph:

```
public void addEdge(String uLabel, String vLabel){
int u=VertexToIndex(LabelToVertex(uLabel));
int v=VertexToIndex(LabelToVertex(vLabel));
Adjacency[u][v] = true;
Adjacency[v][u] = true;}
```

Copy the following lines of code into the body of the `main` method, run the application, and make sure that you see the expected output:

```
Graph myGraph = new Graph();

myGraph.addVertex("A");
myGraph.addVertex("B");
myGraph.addVertex("C");
myGraph.addVertex("D");
myGraph.addVertex("E");
myGraph.addVertex("F");
myGraph.addVertex("G");
myGraph.addVertex("H");
myGraph.addVertex("I");
myGraph.addVertex("J");
myGraph.addVertex("K");

myGraph.addEdge("A", "B");
myGraph.addEdge("B", "A");
myGraph.addEdge("B", "E");
myGraph.addEdge("E", "B");
myGraph.addEdge("A", "C");
myGraph.addEdge("C", "A");
myGraph.addEdge("B", "D");
myGraph.addEdge("D", "B");
myGraph.addEdge("D", "C");
myGraph.addEdge("C", "D");
myGraph.addEdge("C", "E");
myGraph.addEdge("E", "C");
myGraph.addEdge("E", "F");
myGraph.addEdge("F", "E");
myGraph.addEdge("D", "G");
myGraph.addEdge("G", "D");
myGraph.addEdge("E", "G");
myGraph.addEdge("G", "E");
myGraph.addEdge("G", "H");
myGraph.addEdge("H", "G");
myGraph.addEdge("C", "I");
myGraph.addEdge("I", "C");
myGraph.addEdge("A", "J");
myGraph.addEdge("J", "A");

myGraph.display();
```

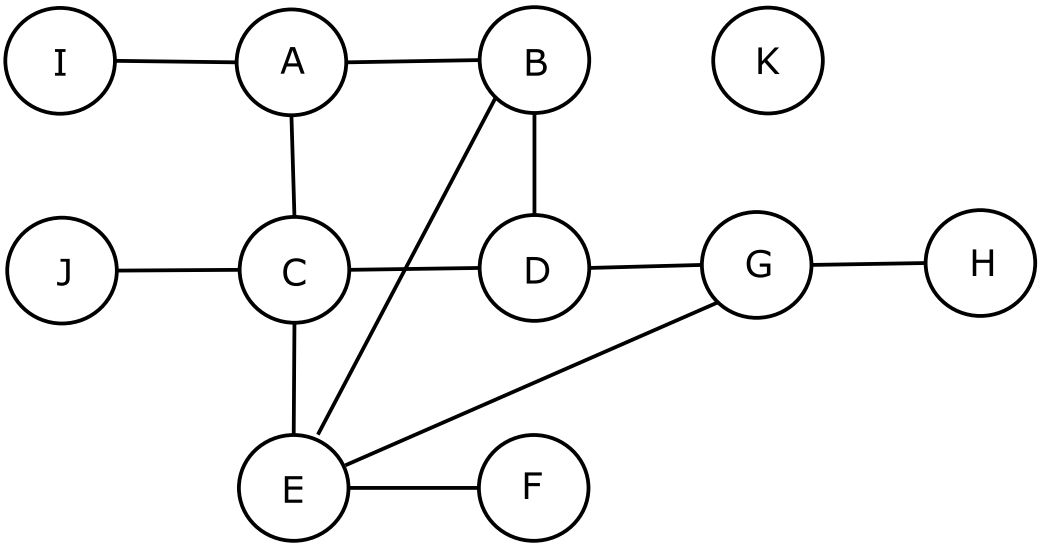
The expected output is printed below:

```
Adjacency Matrix:

      A      B      C      D      E      F      G      H      I      J      K
A     false  true   true   false  false  false  false  false  false  true   false
B     true   false  false  true   true   false  false  false  false  false  false
C     true   false  false  true   true   false  false  false  true   false  false
D     false  true   true   false  false  false  true   false  false  false  false
E     false  true   true   false  false  true   true   false  false  false  false
```

```
F     false  false  false  false  true   false  false  false  false  false  false
G     false  false  false  true   true   false  false  true   false  false  false
H     false  false  false  false  false  false  true   false  false  false  false
I     false  false  true   false  false  false  false  false  false  false  false
J     true   false  false  false  false  false  false  false  false  false  false
K     false  false  false  false  false  false  false  false  false  false  false
```

The above lines of code create the following graph:



III. Breadth First Traversal

In breadth-first traversal, we start from an initial vertex, visit the vertex, then visit the neighbors of the initial vertex, then visit the neighbors of the neighbors of the initial vertex ... . The below method traverses vertexes of a graph using the breadth-first traversal algorithm. The method visits the initial vertex and `enQueue` the initial vertex. The method `deQueue` vertexes from the Queue until the Queue becomes empty. Every time the method `deQueue` a vertex, the method checks all the neighbors of the `deQueue`-ed vertex to see which ones are not visited. The method visits those neighbors that are not visited and `enQueue` them. The method keeps a record of visited vertexes in a `VisitedVertexes` array.

```
public void breadthFirstTraversal(String label)
{
   System.out.println("Breadth First Traversal from Vertex: "+
      label);

   Vertex[] VisititedVertexes = new Vertex[size];
   int VisitedCounter = 0;

   Vertex CurrentVertex = LabelToVertex(label);
```

```
    if (CurrentVertex == null)
       return;

    Visit(CurrentVertex);
    VisititedVertexes[VisitedCounter] = CurrentVertex;
    VisitedCounter++;

    Queue myQueue = new Queue();

    myQueue.enQueue(CurrentVertex);

    while (!myQueue.isEmpty())
    {
       CurrentVertex = myQueue.deQueue();

       for (int j = 0; j < size; j++)
       {
             Vertex AnotherVertex = VertexArray[j];

             if (Adjacency[VertexToIndex(CurrentVertex)][j] == true)
             {
             if (IsVisited(AnotherVertex, VisititedVertexes))
                   continue;

             Visit(AnotherVertex);
             VisititedVertexes[VisitedCounter] = AnotherVertex;
             VisitedCounter++;

             myQueue.enQueue(AnotherVertex);
             }
       }
    }
 }
```

The below methods are used in the body of the above method:

```
void Visit(Vertex myVertex)
Vertex LabelToVertex(String label)
int VertexToIndex(Vertex myVertex)
boolean IsVisited(Vertex myVertex,Vertex[] VisititedVertexes)
```

The `Visit` method takes `myVertex` as an argument and prints the label of `myVertex` in a separate line. The `LabelToVertex` method takes label of a vertex as an argument and `return` the corresponding vertex by searching the `VertexArray`. The `VertexToIndex` takes a vertex as an argument and `return` the index of this vertex in the `VertexArray`. The `IsVisited` method takes `myVertex` and the `VisititedVertexes` array as arguments and `return` true only if `myVertex` is in the `VisititedVertexes` array. The `LabelToVertex` and `VertexToIndex` methods are already developed in the `Application.java` file. Develop the `Visit` and `IsVisited` methods. Use the following lines of code to test the developed methods:

```
Graph myGraph = new Graph();

myGraph.addVertex("A");
myGraph.addVertex("B");
myGraph.addVertex("C");
myGraph.addVertex("D");
myGraph.addVertex("E");
myGraph.addVertex("F");
myGraph.addVertex("G");
myGraph.addVertex("H");
myGraph.addVertex("I");
myGraph.addVertex("J");
myGraph.addVertex("K");

myGraph.addEdge("A", "B");
myGraph.addEdge("B", "A");
myGraph.addEdge("B", "E");
myGraph.addEdge("E", "B");
myGraph.addEdge("A", "C");
myGraph.addEdge("C", "A");
myGraph.addEdge("B", "D");
myGraph.addEdge("D", "B");
myGraph.addEdge("D", "C");
myGraph.addEdge("C", "D");
myGraph.addEdge("C", "E");
myGraph.addEdge("E", "C");
myGraph.addEdge("E", "F");
myGraph.addEdge("F", "E");
myGraph.addEdge("D", "G");
myGraph.addEdge("G", "D");
myGraph.addEdge("E", "G");
myGraph.addEdge("G", "E");
myGraph.addEdge("G", "H");
myGraph.addEdge("H", "G");
myGraph.addEdge("C", "I");
myGraph.addEdge("I", "C");
myGraph.addEdge("A", "J");
myGraph.addEdge("J", "A");

myGraph.breadthFirstTraversal("A");
myGraph.breadthFirstTraversal("C");
myGraph.breadthFirstTraversal("G");
myGraph.breadthFirstTraversal("K");
```

The expected output is printed below:

```
Breadth First Traversal from Vertex: A
A
B
C
J
D
E
I
G
F
H
Breadth First Traversal from Vertex: C
C
A
D
E
I
B
J
G
F
H
Breadth First Traversal from Vertex: G
G
D
E
H
B
C
F
A
I
J
Breadth First Traversal from Vertex: K
K
```

From above, when the breadth-first traversal starts from vertex K, the only vertex that is visited is vertex K itself. That is because vertex K is not connected to any other vertex.

IV. Depth First Traveral

In depth-first Traversal of a tree, we visit an initial vertex, then visit one neighbor of the initial vertex, then visit one neighbor of the neighbor of the initial vertex and keep repeating until reaching to a dead end. We then move back and visit an unvisited vertex that is closest to the initial vertex and repeat the process until all vertexes are visited. Develop the code for depth-first traversal of a tree using the following logic:

```
public void depthFirstTraversal(String label)

      System.out.println("Depth First Traversal from Vertex: "+
         label)

      Vertex[] VisititedVertexes = new Vertex[size]

      int VisitedCounter = 0

      Vertex CurrentVertex = LabelToVertex(label)

      if (CurrentVertex == null)
            return;

      Visit(CurrentVertex)
      VisititedVertexes[VisitedCounter] = CurrentVertex
      VisitedCounter++

      StackV myStack = new StackV(size)

      push CurrentVertex on myStack

      while myStack is not empty

            CurrentVertex = myStack.pop()

            for (int j = 0; j < size; j++)

                  Vertex AnotherVertex = VertexArray[j]

                  if CurrentVertex and AnotherVertex are neighbors

                        if AnotherVertex is already visited
                              continue

                        Visit AnotherVertex
                        VisititedVertexes[VisitedCounter] =
                           AnotherVertex
                        VisitedCounter++

                        push CurrentVertex on myStack
                        push AnotherVertex on myStack

                        break
```

Use the following lines of code to test the developed method:

```
 Graph myGraph = new Graph();

myGraph.addVertex("A");
myGraph.addVertex("B");
myGraph.addVertex("C");
myGraph.addVertex("D");
myGraph.addVertex("E");
myGraph.addVertex("F");
myGraph.addVertex("G");
myGraph.addVertex("H");
myGraph.addVertex("I");
myGraph.addVertex("J");
myGraph.addVertex("K");

myGraph.addEdge("A", "B");
myGraph.addEdge("B", "A");
myGraph.addEdge("B", "E");
myGraph.addEdge("E", "B");
myGraph.addEdge("A", "C");
myGraph.addEdge("C", "A");
myGraph.addEdge("B", "D");
myGraph.addEdge("D", "B");
myGraph.addEdge("D", "C");
myGraph.addEdge("C", "D");
myGraph.addEdge("C", "E");
myGraph.addEdge("E", "C");
myGraph.addEdge("E", "F");
myGraph.addEdge("F", "E");
myGraph.addEdge("D", "G");
myGraph.addEdge("G", "D");
myGraph.addEdge("E", "G");
myGraph.addEdge("G", "E");
myGraph.addEdge("G", "H");
myGraph.addEdge("H", "G");
myGraph.addEdge("C", "I");
myGraph.addEdge("I", "C");
myGraph.addEdge("A", "J");
myGraph.addEdge("J", "A");

myGraph.depthFirstTraversal("A");
myGraph.depthFirstTraversal("C");
myGraph.depthFirstTraversal("G");
myGraph.depthFirstTraversal("K");
```

The expected output is printed below:

```
 Depth First Traversal from Vertex: A
A
B
D
C
E
F
G
H
I
J
Depth First Traversal from Vertex: C
C
A
B
D
G
E
F
H
J
I
Depth First Traversal from Vertex: G
G
D
B
A
C
E
F
I
J
H
Depth First Traversal from Vertex: K
K
```

From above, when the depth-first traversal starts from vertex K, the only vertex that is visited is vertex K itself. That is because vertex K is not connected to any other vertex.

V. Submitting the Assignment

When submitting your response to this assignment, keep the above lines of code in the body of the `main` method.