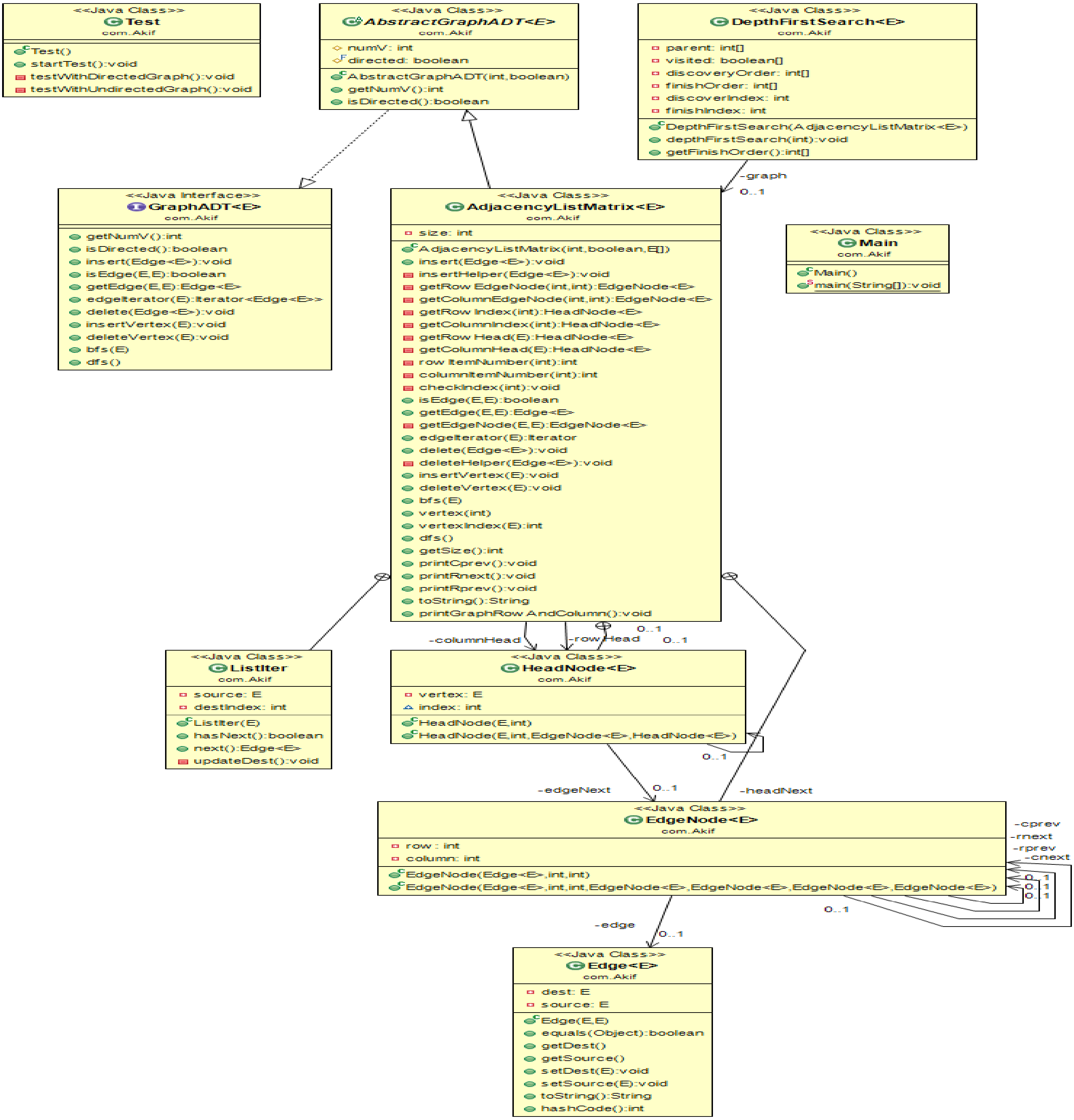
GIT Department of Computer Engineering CSE 222/505 - Spring 2020 Homework 8 Report

Akif KARTAL
171044098

Q2 REPORT

1. CLASS DIAGRAMS



2. PROBLEM SOLUTION APPROACH

My Problem solution steps are;

– Specify the problem requirements
– Analyze the problem
– Design an algorithm and Program
– Implement the algorithm
– Test and verify the program
– Maintain and update the program

1) **Specify the problem requirements :** I understand the problem.
2) **Analyze the problem :** I identify;
   – Input data
   – Output data
   – Additional requirements and constraints

3) **Design an algorithm and Program :** I divide the problem into sub-problems. I listed major steps (sub-problems). I break down each step into a more detailed list. To do these We have to divide this big project into small pieces.

⇨ **Implement the algorithm :** I wrote the algorithm in Java by converting each step into statements of Java (classes ,methods etc.)

Firstly, I wrote the **generic Edge class** from the book.

After I wrote **extended and updated version of Graph Interface.** After I took common methods for a graph in **AbstractGraphADT class.**

Lastly, I wrote **AdjacencyListMatrix** class that implementation of extended Graph Interface. Also, to perform depth-first search, I wrote **DepthFirstSearch** class.

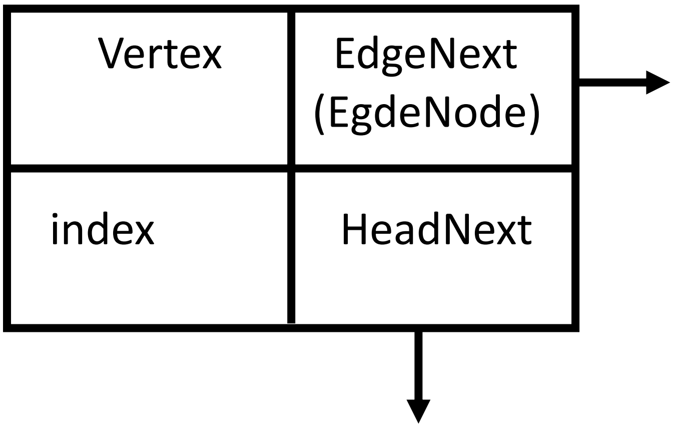
For more detail about implementation **check following explanation.**

4) **Test and verify the program:** To test program I wrote the **Test class**. In this class I wrote two method two test program with different graph structure. After each operation on graph I printed the results.
5) **Maintain and update the program :** I keep the program up-to-date

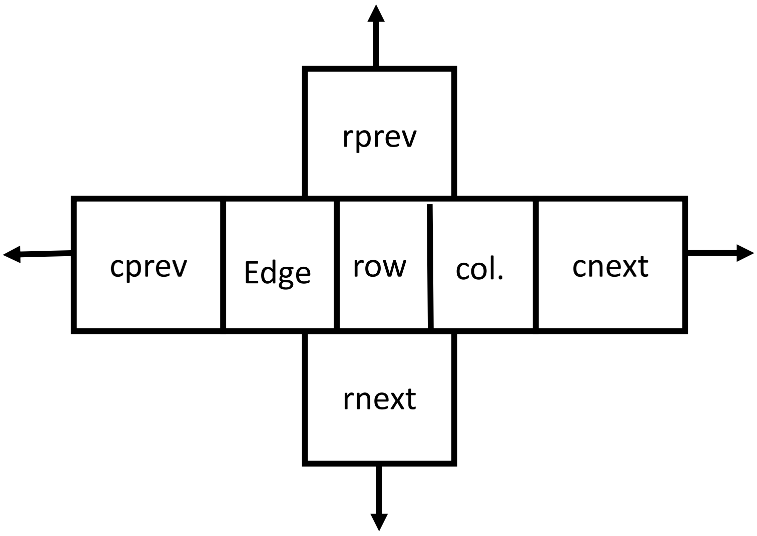
Explanation on my 2D linked-list structure

To implement this structure, I have 2 node class one of for vertices one of them for edges.

HeadNode

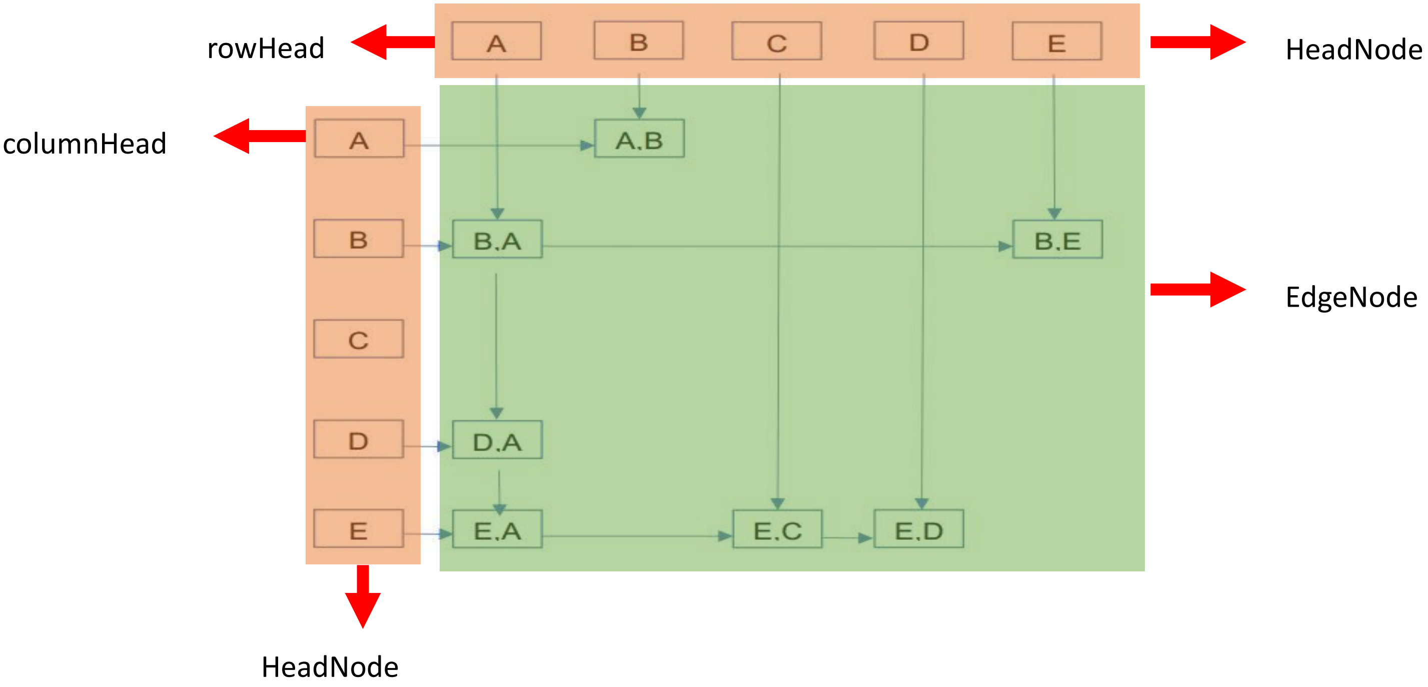


EdgeNode



In AdjacencyListMatrix class I have two variable these are;

```
private HeadNode<E> rowHead;
private HeadNode<E> columnHead;
```



In this structure as you can see vertices are **HeadNode** edges are **EdgeNode**.

Each HeadNode have edgeNext reference that is head edge reference of that vertex.

Also, they have headNext next reference of next vertices.

Note that, since our graph is Generic, we don't need to keep the vertices in **sorted** order.

Insertion an Edge

Note that even if each egdeNode has row and column index, this is a 2D linked-list generic graph. So, when you insert a new edge first it finds correct position in both headNode according to given source and destination vertex. After that by using edgeNext references of headNodes it makes connection of that new Edge correctly.(for all sides)

Deletion an Edge

While deletion an edge first it finds correct position in columnHead headNode according to given source a vertex. After that by using edgeNext references of headNodes it makes new connection of that Edge to delete. In other words it deletes references that shows that edge.(update connections correctly)

Insertion a Vertex

While insertion a new vertex first it goes end of the both headNode references(rowHead, columnHead) then it adds new vertex end of the both list and increments the numV.

As I mentioned, we don't need keep the vertices to be sorted.

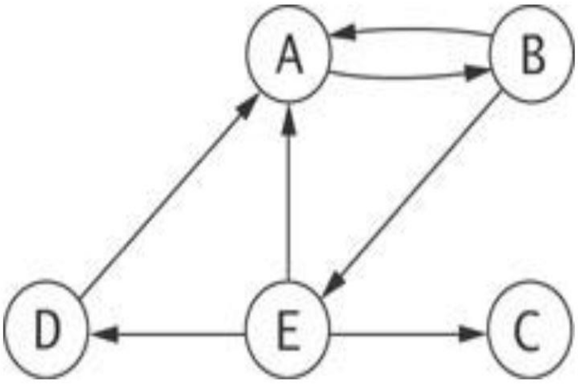
Deletion a Vertex

To delete a vertex is not easy as insertion. First, we need to delete all edges that belongs that vertex. After this operation we need to decrement index of succeeding vertices of the deleted vertex both headNode references(rowHead, columnHead). Then, according to new indexes we need to set row and column index of edges that are positions after the deleted vertex. Note that we don't need to change coordinates of edges that belongs the predecessor vertices of the deleted vertex.

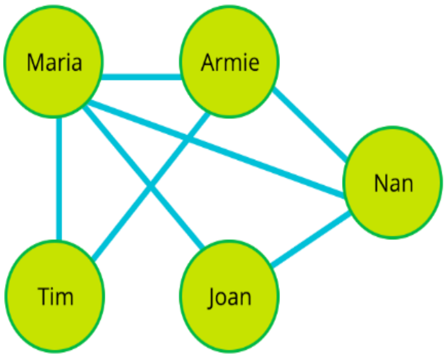
Lastly, After updating of IDs we need to delete that vertex from both list(rowHead, columnHead) by updating headNext references and by decrementing numV number. After, all these operations deletion is done.

Graphs that I used for Testing:

- Directed Graph



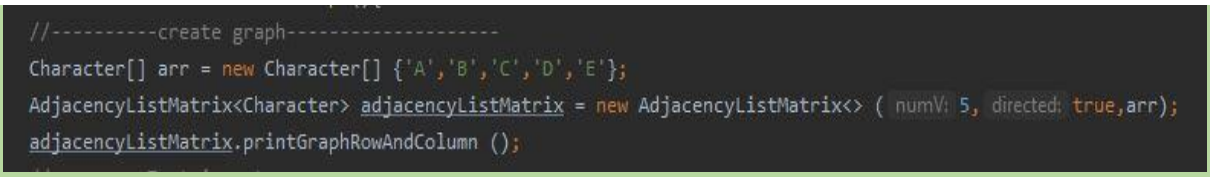
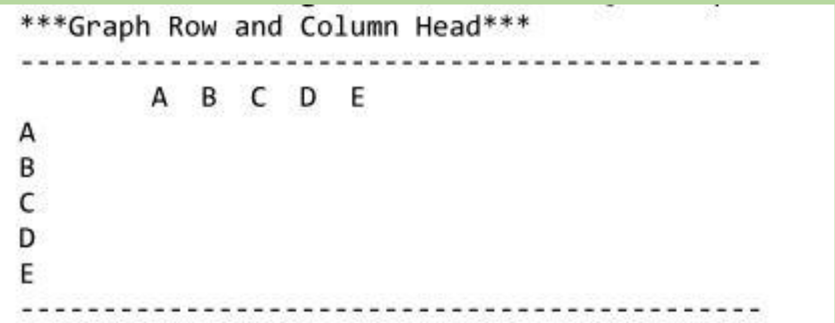
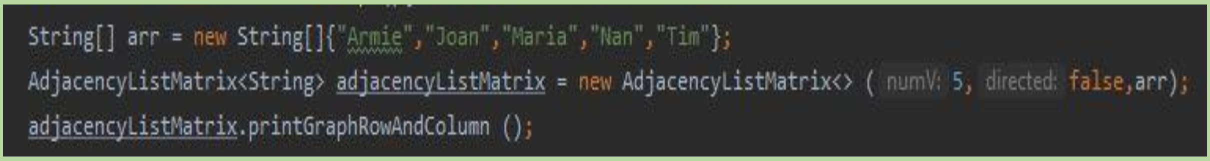
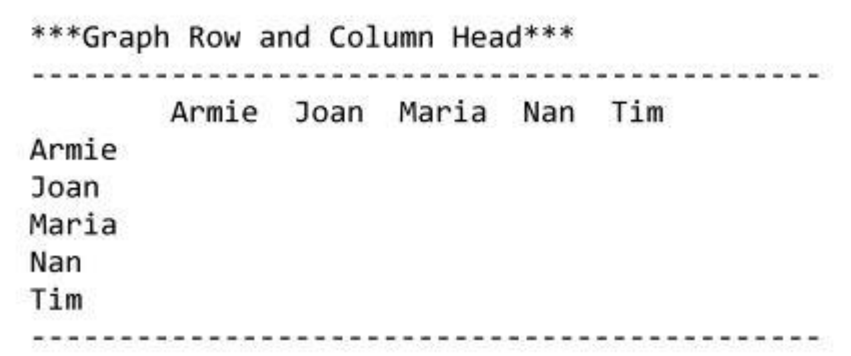
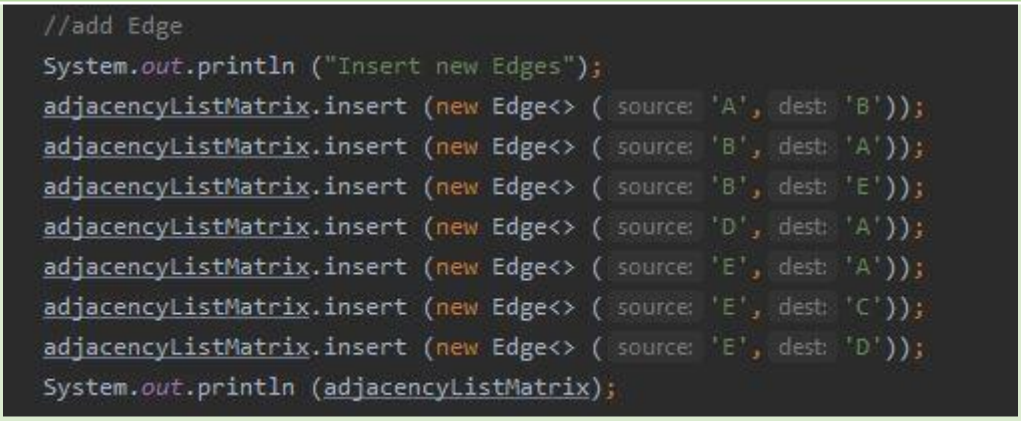
- Undirected Graph



3. TEST CASES

Test ID	Test Case	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T1	Create Graph Representation Correctly.	Create object of classes with proper parameter.	Array of Character Array of String	The Graphs are created successfully.	As Expected	Pass
T2	Insertion of an individual edge.	Call method with proper parameter.	'A','B' "Maria","Armie"	The edge inserted successfully.	As Expected	Pass
T3	Get an egde.	Call method with proper parameter.	'E','A' "Nan","Joan"	The edge reached successfully.	As Expected	Pass
T4	Deletion of an individual edge.	Call method with proper parameter.	'A','B' "Maria","Joan"	The edge deleted successfully.	As Expected	Pass
T5	Insertion of an individual Vertex.	Call method with proper parameter.	'K' "Carol"	The vertex inserted successfully.	As Expected	Pass
T6	Deletion of an individual Vertex.	Call method with proper parameter.	'E' "Armie"	The vertex deleted successfully.	As Expected	Pass
T7	Perform breadth-first search of the graph	Call method with proper parameter.	'A' "Maria"	Breadth-first search performed successfully.	As Expected	Pass
T8	Perform depth-first search of the graph	Call method.	-	Depth-first search performed successfully.	As Expected	Pass
T9	Print Edges Using all references To see connections are true.	Call methods.	-	The graph has printed in a good shape.	As Expected	Pass

4. RUNNING AND RESULTS

Test ID	Test Result
T1	   
T2	

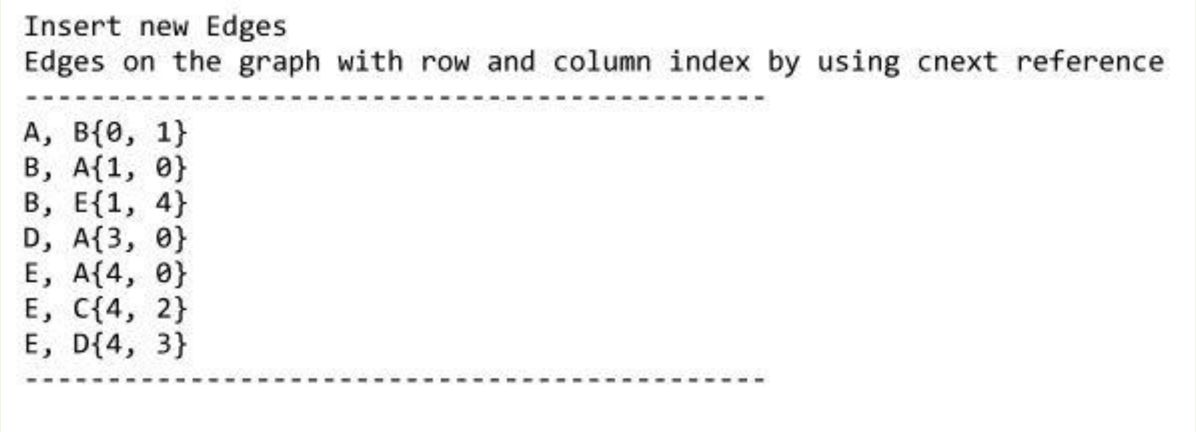
```
//----------create graph-------------------
Character[] arr = new Character[] {'A','B','C','D','E'};
AdjacencyListMatrix<Character> adjacencyListMatrix = new AdjacencyListMatrix<> ( numV: 5, directed: true,arr);
adjacencyListMatrix.printGraphRowAndColumn ();
```

```
***Graph Row and Column Head***
----------------------------------------------
        A  B  C  D  E
A
B
C
D
E
----------------------------------------------
```

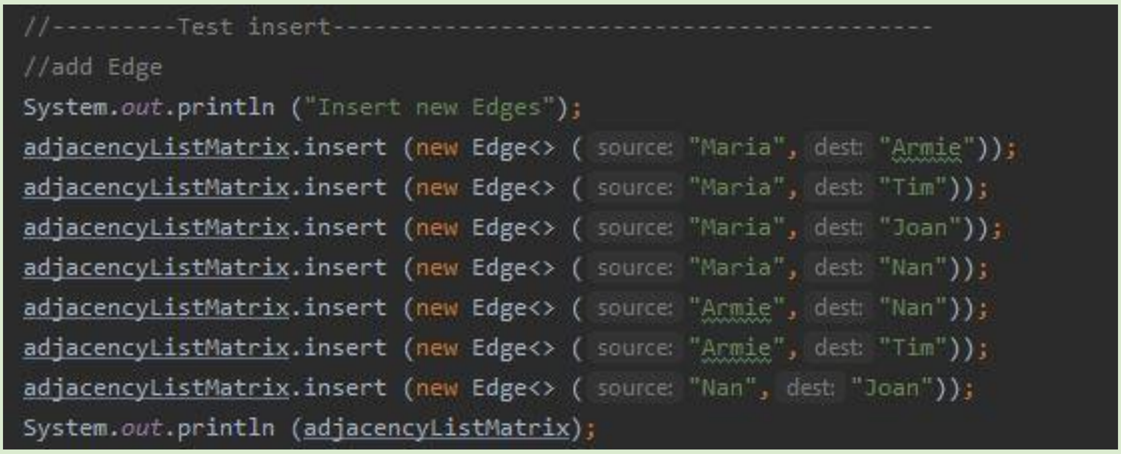
```
String[] arr = new String[]{"Armie","Joan","Maria","Nan","Tim"};
AdjacencyListMatrix<String> adjacencyListMatrix = new AdjacencyListMatrix<> ( numV: 5, directed: false,arr);
adjacencyListMatrix.printGraphRowAndColumn ();
```

```
***Graph Row and Column Head***
---------------------------------------------
        Armie  Joan  Maria  Nan  Tim
Armie
Joan
Maria
Nan
Tim
---------------------------------------------
```

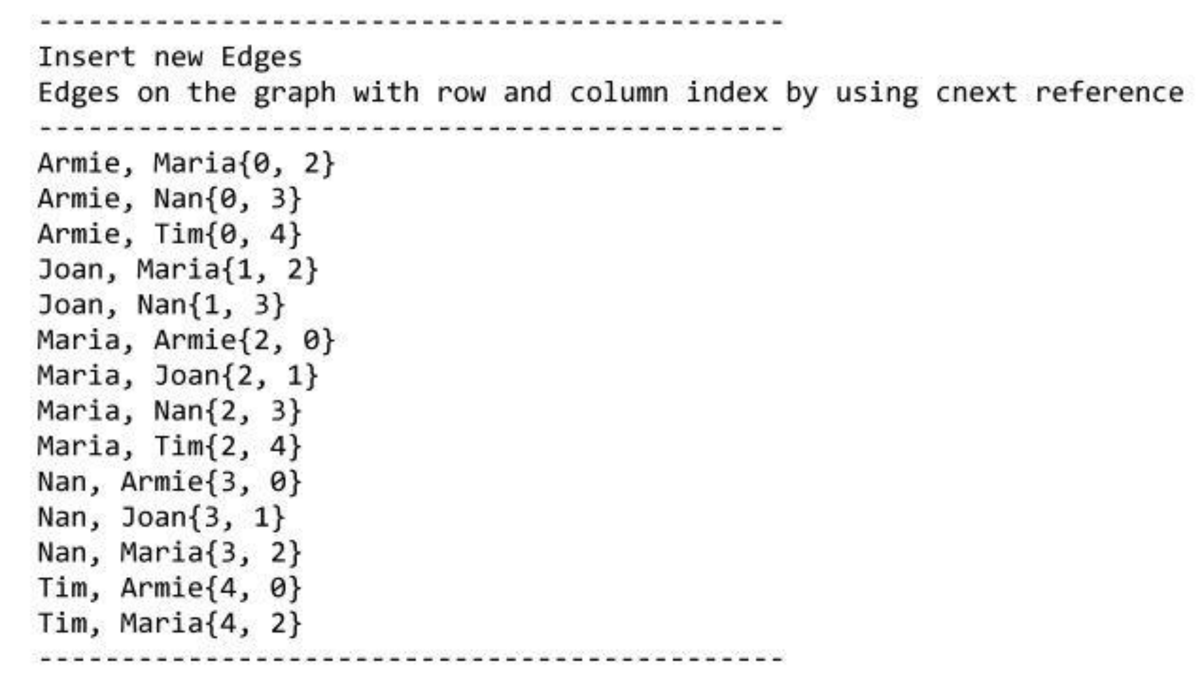
```
//add Edge
System.out.println ("Insert new Edges");
adjacencyListMatrix.insert (new Edge<> ( source: 'A', dest: 'B'));
adjacencyListMatrix.insert (new Edge<> ( source: 'B', dest: 'A'));
adjacencyListMatrix.insert (new Edge<> ( source: 'B', dest: 'E'));
adjacencyListMatrix.insert (new Edge<> ( source: 'D', dest: 'A'));
adjacencyListMatrix.insert (new Edge<> ( source: 'E', dest: 'A'));
adjacencyListMatrix.insert (new Edge<> ( source: 'E', dest: 'C'));
adjacencyListMatrix.insert (new Edge<> ( source: 'E', dest: 'D'));
System.out.println (adjacencyListMatrix);
```



```
Insert new Edges
Edges on the graph with row and column index by using cnext reference
---------------------------------------------
A, B{0, 1}
B, A{1, 0}
B, E{1, 4}
D, A{3, 0}
E, A{4, 0}
E, C{4, 2}
E, D{4, 3}
---------------------------------------------
```

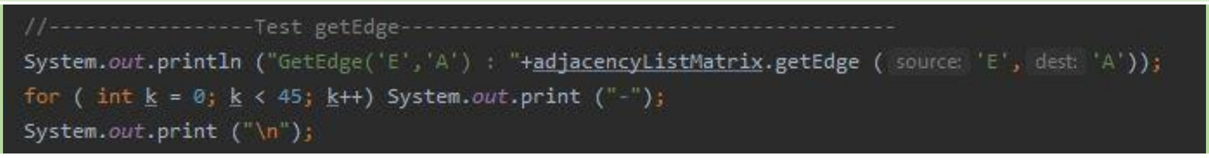


```
//---------Test insert----------------------------------------
//add Edge
System.out.println ("Insert new Edges");
adjacencyListMatrix.insert (new Edge<> ( source: "Maria", dest: "Armie"));
adjacencyListMatrix.insert (new Edge<> ( source: "Maria", dest: "Tim"));
adjacencyListMatrix.insert (new Edge<> ( source: "Maria", dest: "Joan"));
adjacencyListMatrix.insert (new Edge<> ( source: "Maria", dest: "Nan"));
adjacencyListMatrix.insert (new Edge<> ( source: "Armie", dest: "Nan"));
adjacencyListMatrix.insert (new Edge<> ( source: "Armie", dest: "Tim"));
adjacencyListMatrix.insert (new Edge<> ( source: "Nan", dest: "Joan"));
System.out.println (adjacencyListMatrix);
```

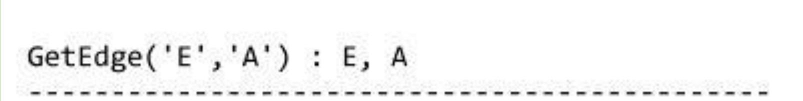


```
---------------------------------------------
Insert new Edges
Edges on the graph with row and column index by using cnext reference
---------------------------------------------
Armie, Maria{0, 2}
Armie, Nan{0, 3}
Armie, Tim{0, 4}
Joan, Maria{1, 2}
Joan, Nan{1, 3}
Maria, Armie{2, 0}
Maria, Joan{2, 1}
Maria, Nan{2, 3}
Maria, Tim{2, 4}
Nan, Armie{3, 0}
Nan, Joan{3, 1}
Nan, Maria{3, 2}
Tim, Armie{4, 0}
Tim, Maria{4, 2}
---------------------------------------------
```

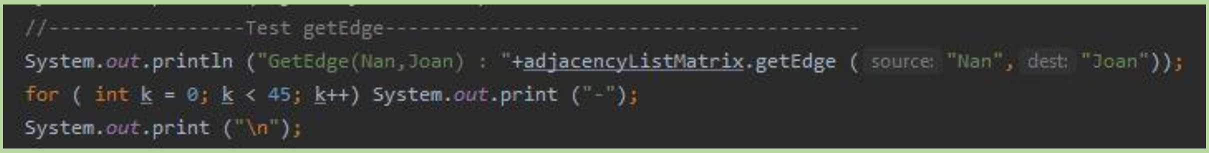
T3



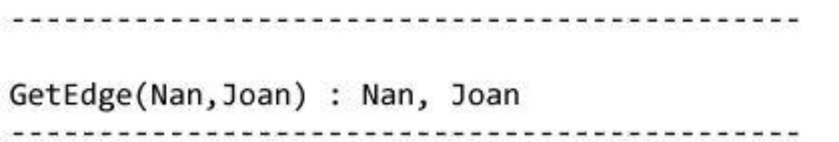
```
//-----------------Test getEdge----------------------------------------
System.out.println ("GetEdge('E','A') : "+adjacencyListMatrix.getEdge ( source: 'E', dest: 'A'));
for ( int k = 0; k < 45; k++) System.out.print ("-");
System.out.print ("\n");
```



```
GetEdge('E','A') : E, A
---------------------------------------------
```

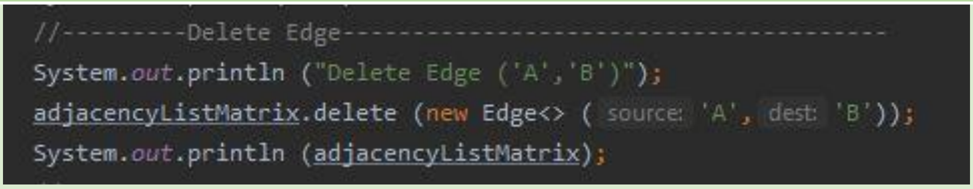


```
//----------------Test getEdge----------------------------------------
System.out.println ("GetEdge(Nan,Joan) : "+adjacencyListMatrix.getEdge ( source: "Nan", dest: "Joan"));
for ( int k = 0; k < 45; k++) System.out.print ("-");
System.out.print ("\n");
```

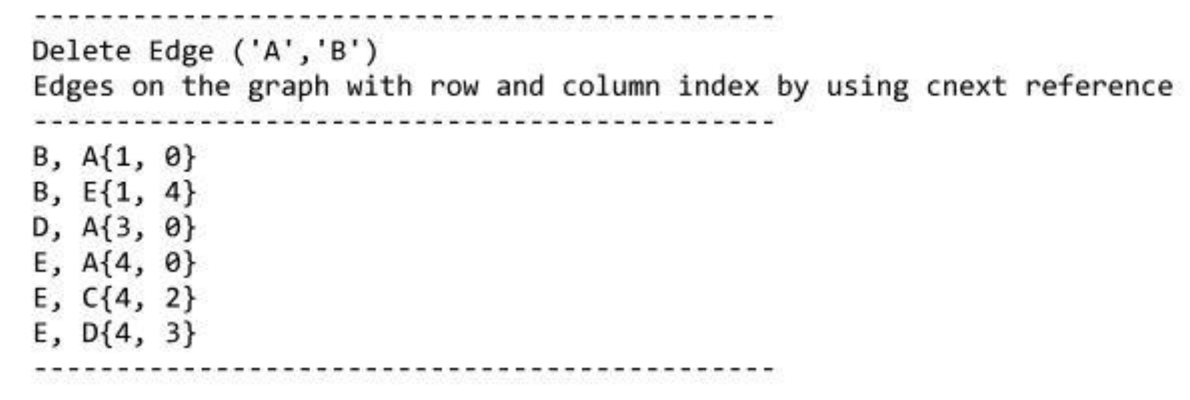


```
---------------------------------------------

GetEdge(Nan,Joan) : Nan, Joan
---------------------------------------------
```

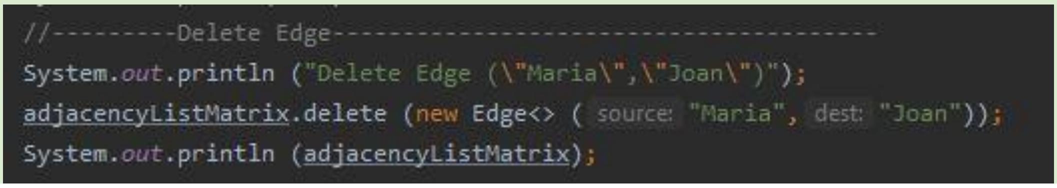
T4



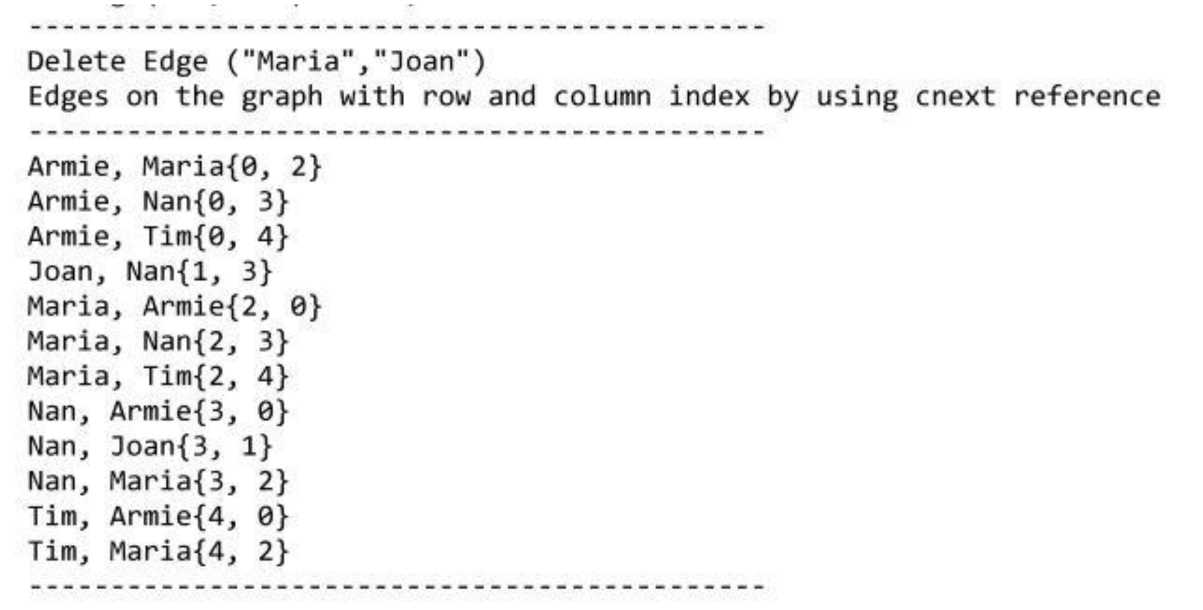
```
//---------Delete Edge---------------------------------------
System.out.println ("Delete Edge ('A','B')");
adjacencyListMatrix.delete (new Edge<> ( source: 'A', dest: 'B'));
System.out.println (adjacencyListMatrix);
```



```
---------------------------------------------
Delete Edge ('A','B')
Edges on the graph with row and column index by using cnext reference
---------------------------------------------
B, A{1, 0}
B, E{1, 4}
D, A{3, 0}
E, A{4, 0}
E, C{4, 2}
E, D{4, 3}
---------------------------------------------
```

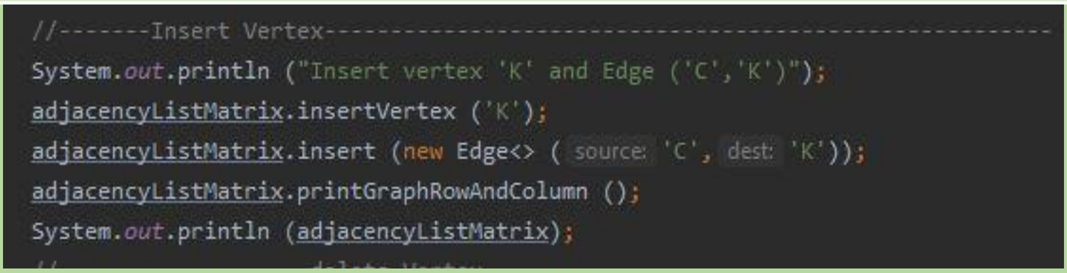


```
//---------Delete Edge---------------------------------------
System.out.println ("Delete Edge (\"Maria\",\"Joan\")");
adjacencyListMatrix.delete (new Edge<> ( source: "Maria", dest: "Joan"));
System.out.println (adjacencyListMatrix);
```

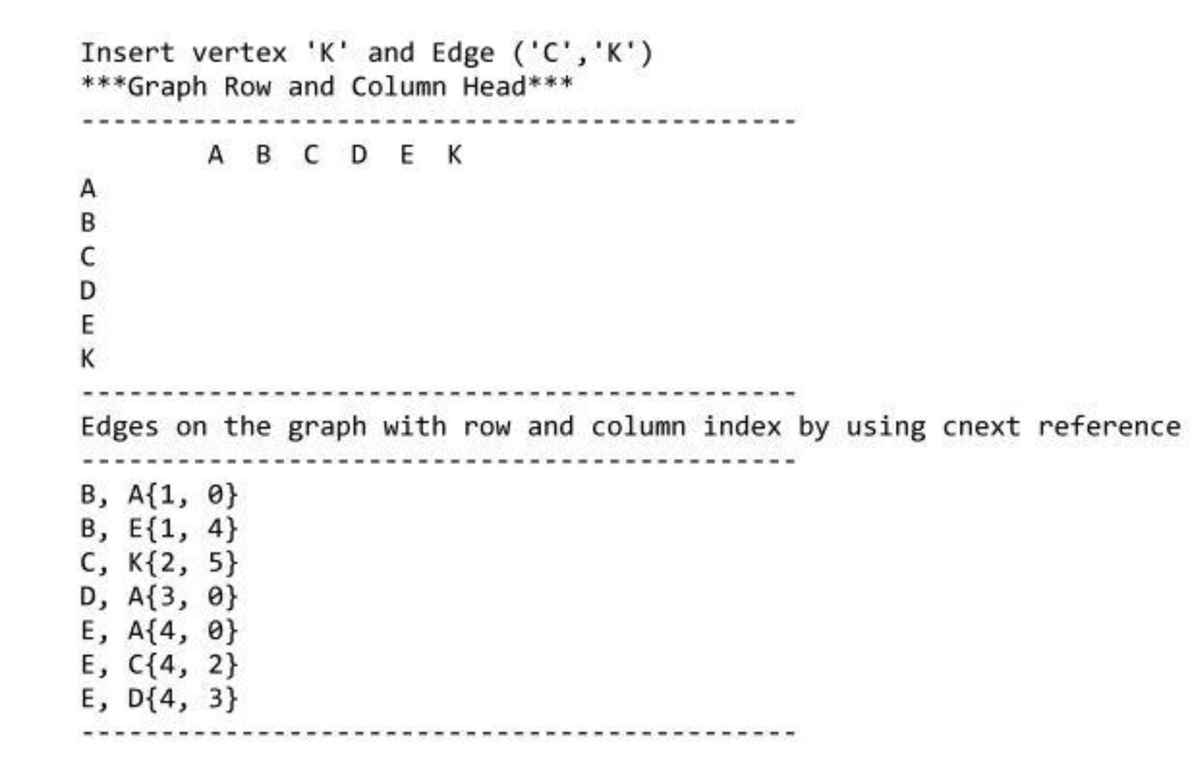


```
--------------------------------------------
Delete Edge ("Maria","Joan")
Edges on the graph with row and column index by using cnext reference
--------------------------------------------
Armie, Maria{0, 2}
Armie, Nan{0, 3}
Armie, Tim{0, 4}
Joan, Nan{1, 3}
Maria, Armie{2, 0}
Maria, Nan{2, 3}
Maria, Tim{2, 4}
Nan, Armie{3, 0}
Nan, Joan{3, 1}
Nan, Maria{3, 2}
Tim, Armie{4, 0}
Tim, Maria{4, 2}
--------------------------------------------
```

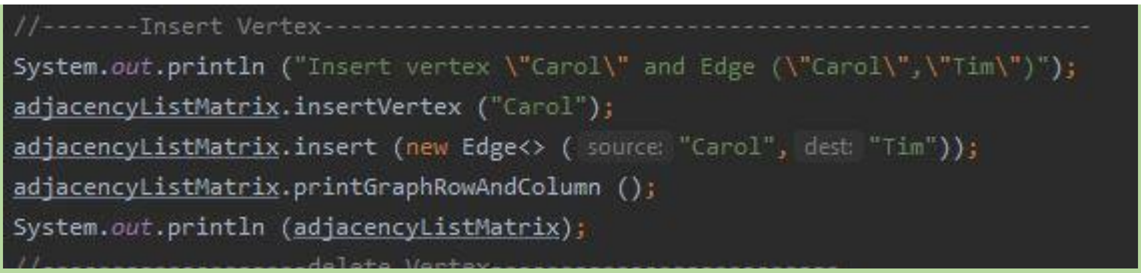
T5



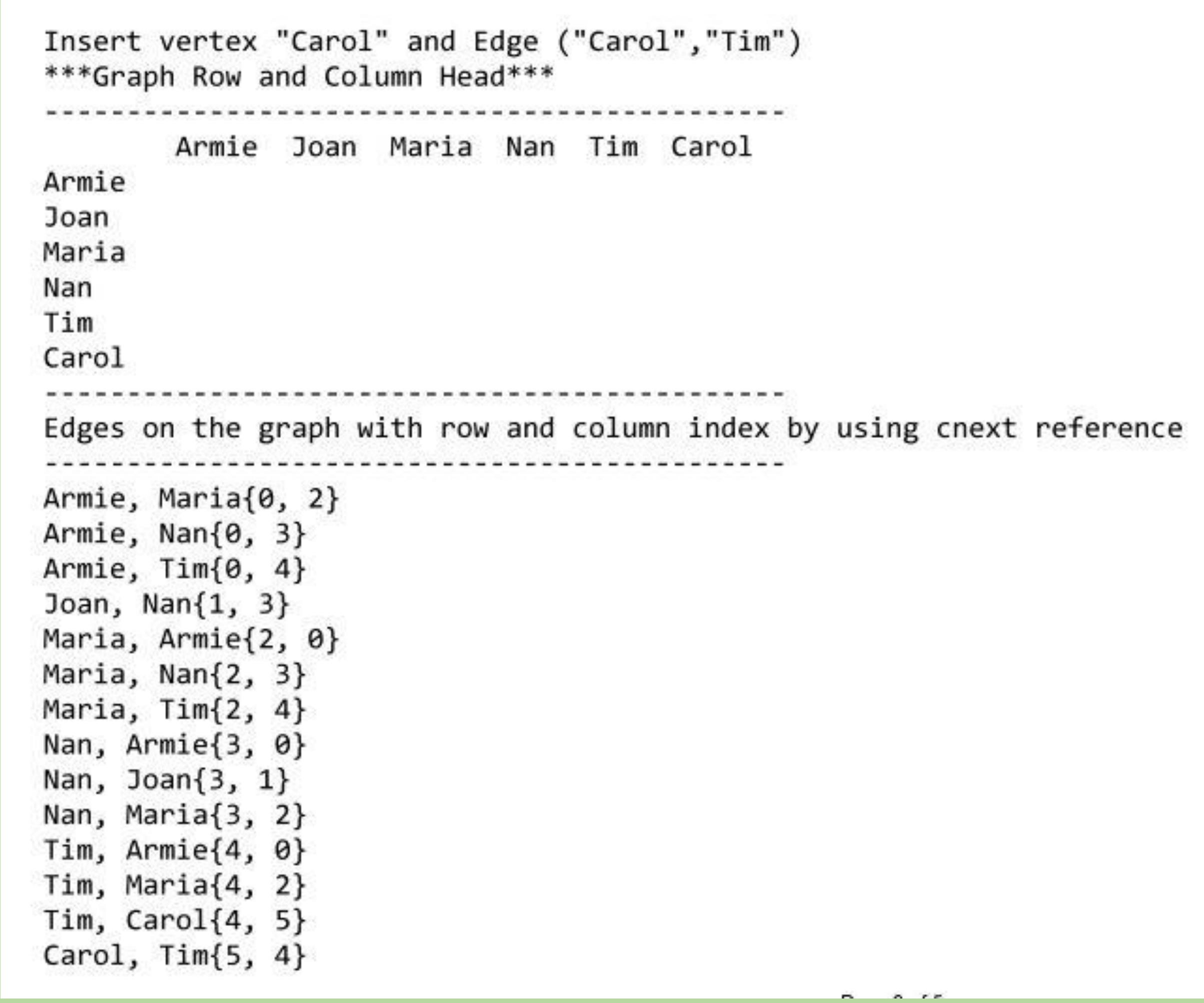
```java
//-------Insert Vertex-------------------------------------------------------
System.out.println ("Insert vertex 'K' and Edge ('C','K')");
adjacencyListMatrix.insertVertex ('K');
adjacencyListMatrix.insert (new Edge<> ( source: 'C', dest: 'K'));
adjacencyListMatrix.printGraphRowAndColumn ();
System.out.println (adjacencyListMatrix);
```



```
Insert vertex 'K' and Edge ('C','K')
***Graph Row and Column Head***
--------------------------------------------
        A  B  C  D  E  K
A
B
C
D
E
K
--------------------------------------------
Edges on the graph with row and column index by using cnext reference
--------------------------------------------
B, A{1, 0}
B, E{1, 4}
C, K{2, 5}
D, A{3, 0}
E, A{4, 0}
E, C{4, 2}
E, D{4, 3}
--------------------------------------------
```

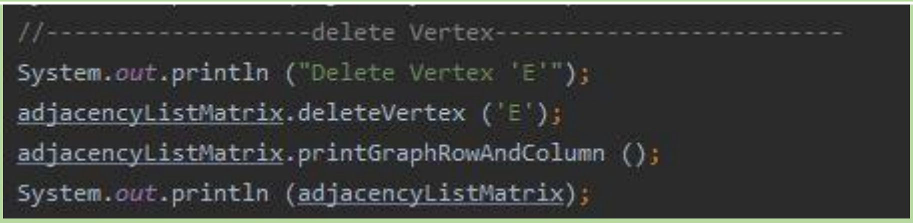


```
//-------Insert Vertex-----------------------------------------------------
System.out.println ("Insert vertex \"Carol\" and Edge (\"Carol\",\"Tim\")");
adjacencyListMatrix.insertVertex ("Carol");
adjacencyListMatrix.insert (new Edge<> ( source: "Carol", dest: "Tim"));
adjacencyListMatrix.printGraphRowAndColumn ();
System.out.println (adjacencyListMatrix);
//--------------------delete Vertex---------------------------
```

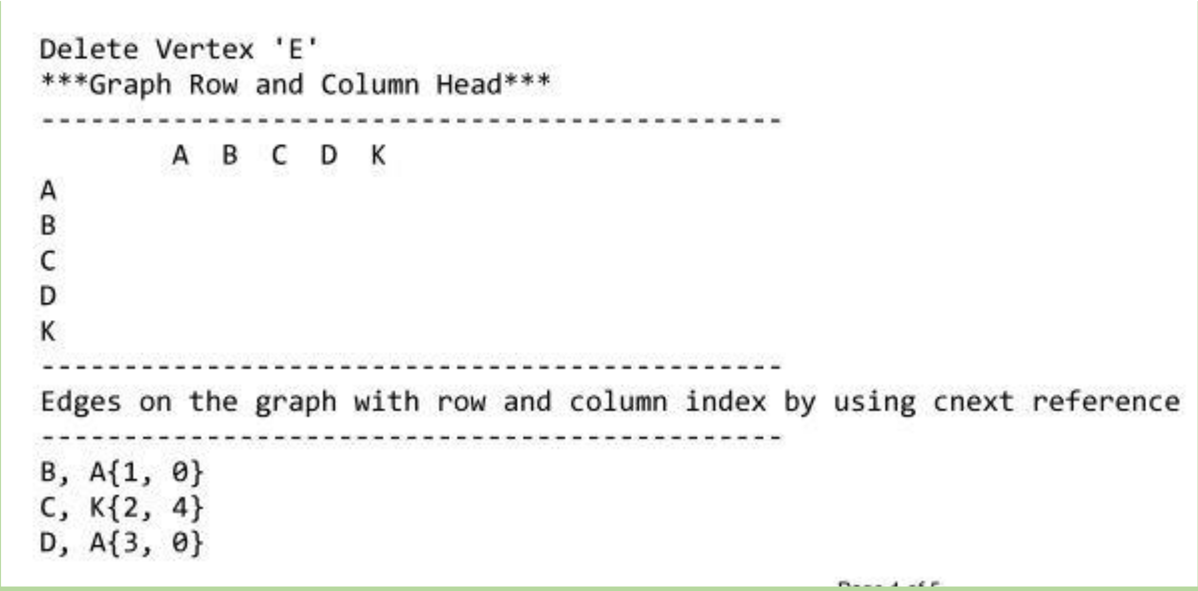


```
Insert vertex "Carol" and Edge ("Carol","Tim")
***Graph Row and Column Head***
---------------------------------------------
        Armie  Joan  Maria  Nan  Tim  Carol
Armie
Joan
Maria
Nan
Tim
Carol
---------------------------------------------
Edges on the graph with row and column index by using cnext reference
---------------------------------------------
Armie, Maria{0, 2}
Armie, Nan{0, 3}
Armie, Tim{0, 4}
Joan, Nan{1, 3}
Maria, Armie{2, 0}
Maria, Nan{2, 3}
Maria, Tim{2, 4}
Nan, Armie{3, 0}
Nan, Joan{3, 1}
Nan, Maria{3, 2}
Tim, Armie{4, 0}
Tim, Maria{4, 2}
Tim, Carol{4, 5}
Carol, Tim{5, 4}
```

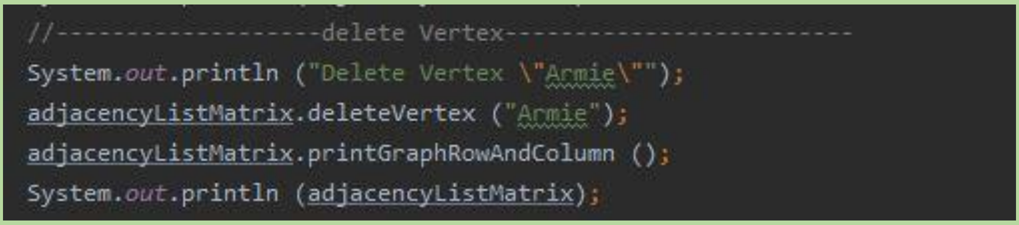
T6



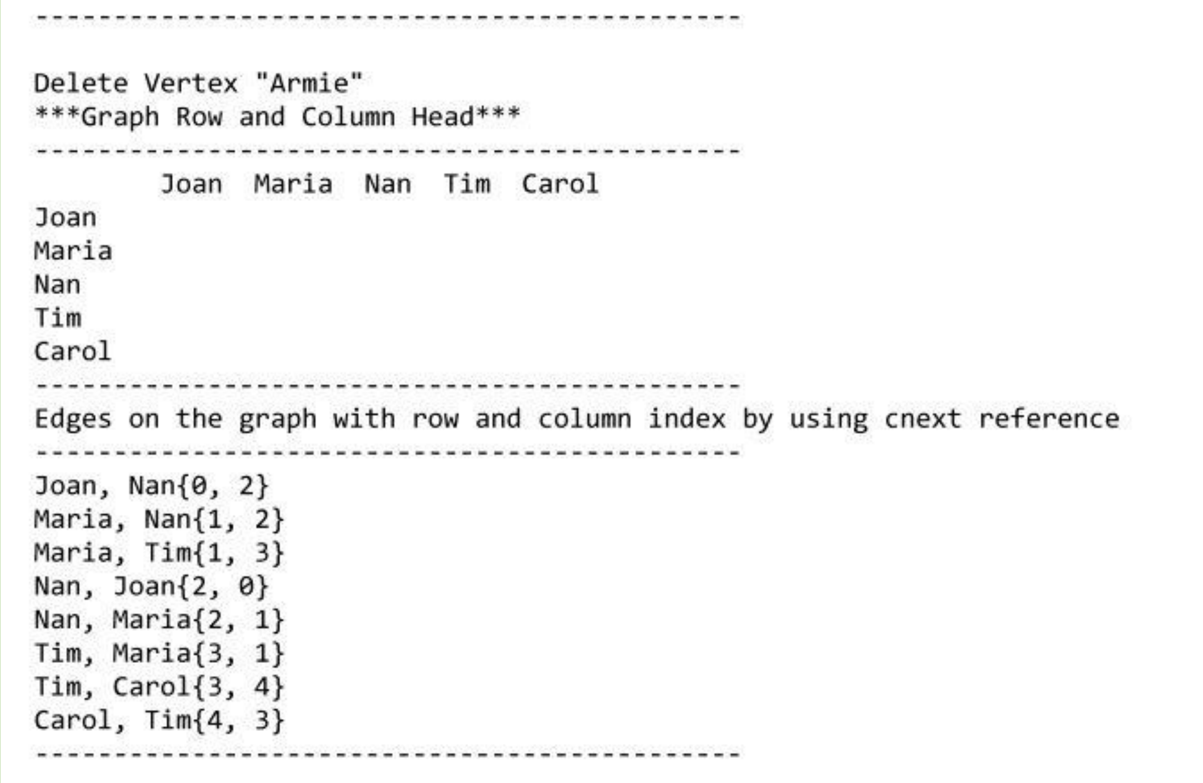
```
//------------------delete Vertex------------------------
System.out.println ("Delete Vertex 'E'");
adjacencyListMatrix.deleteVertex ('E');
adjacencyListMatrix.printGraphRowAndColumn ();
System.out.println (adjacencyListMatrix);
```



```
Delete Vertex 'E'
***Graph Row and Column Head***
---------------------------------------------
        A  B  C  D  K
A
B
C
D
K
---------------------------------------------
Edges on the graph with row and column index by using cnext reference
---------------------------------------------
B, A{1, 0}
C, K{2, 4}
D, A{3, 0}
```

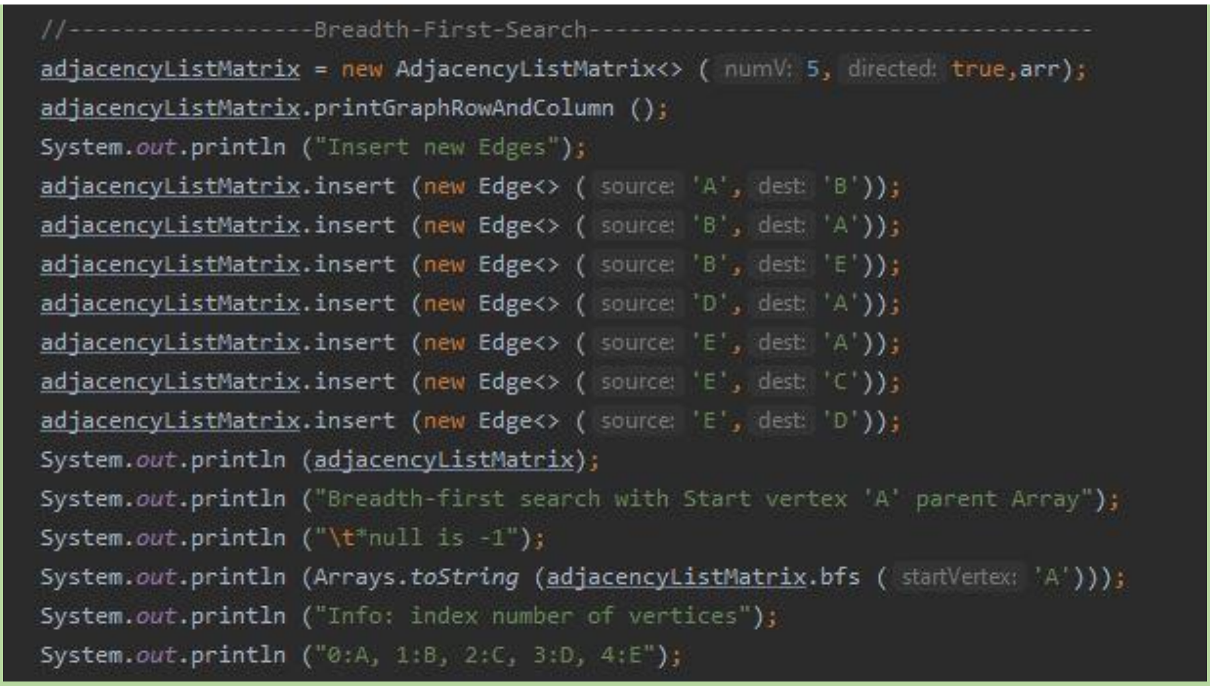


```java
//-------------------delete Vertex-----------------------
System.out.println ("Delete Vertex \"Armie\"");
adjacencyListMatrix.deleteVertex ("Armie");
adjacencyListMatrix.printGraphRowAndColumn ();
System.out.println (adjacencyListMatrix);
```

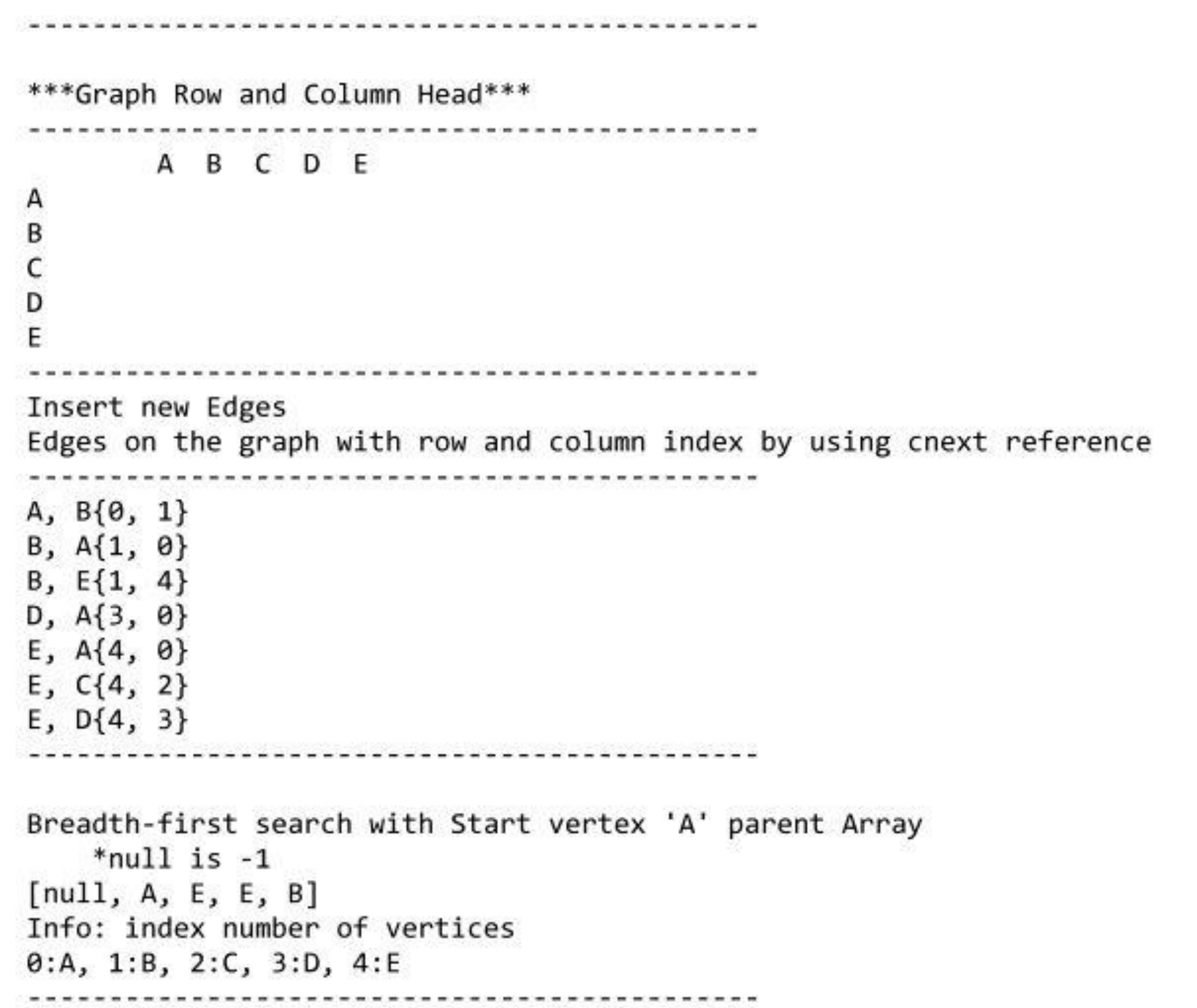


```
---------------------------------------------

Delete Vertex "Armie"
***Graph Row and Column Head***
---------------------------------------------
        Joan  Maria  Nan  Tim  Carol
Joan
Maria
Nan
Tim
Carol
---------------------------------------------
Edges on the graph with row and column index by using cnext reference
---------------------------------------------
Joan, Nan{0, 2}
Maria, Nan{1, 2}
Maria, Tim{1, 3}
Nan, Joan{2, 0}
Nan, Maria{2, 1}
Tim, Maria{3, 1}
Tim, Carol{3, 4}
Carol, Tim{4, 3}
---------------------------------------------
```

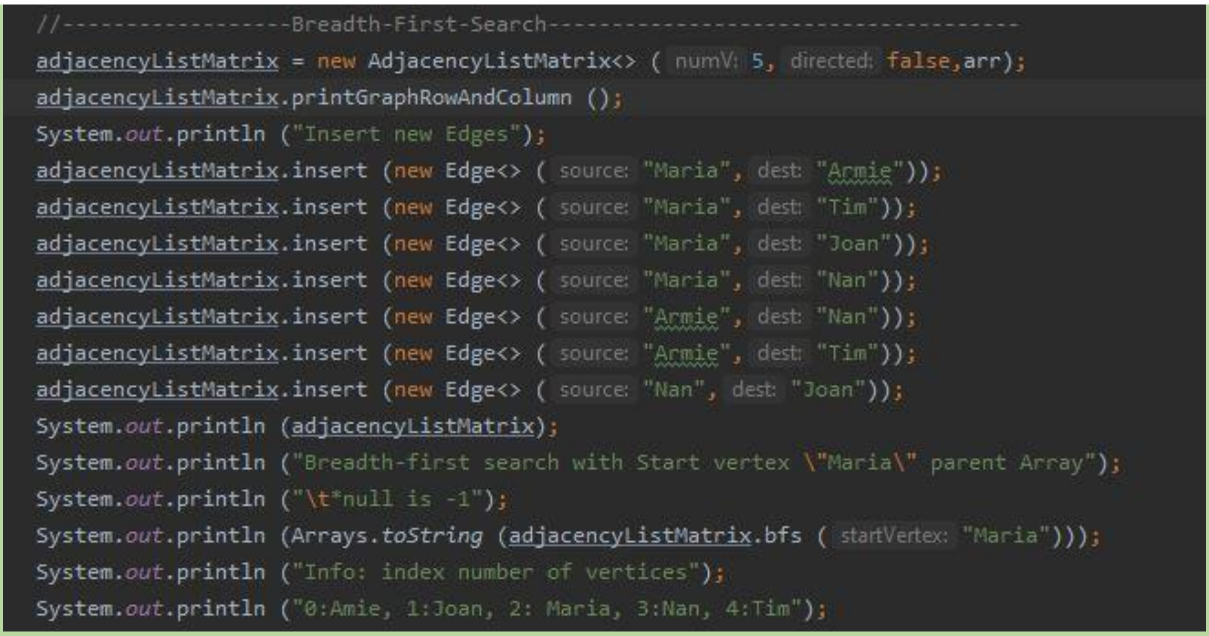
T7



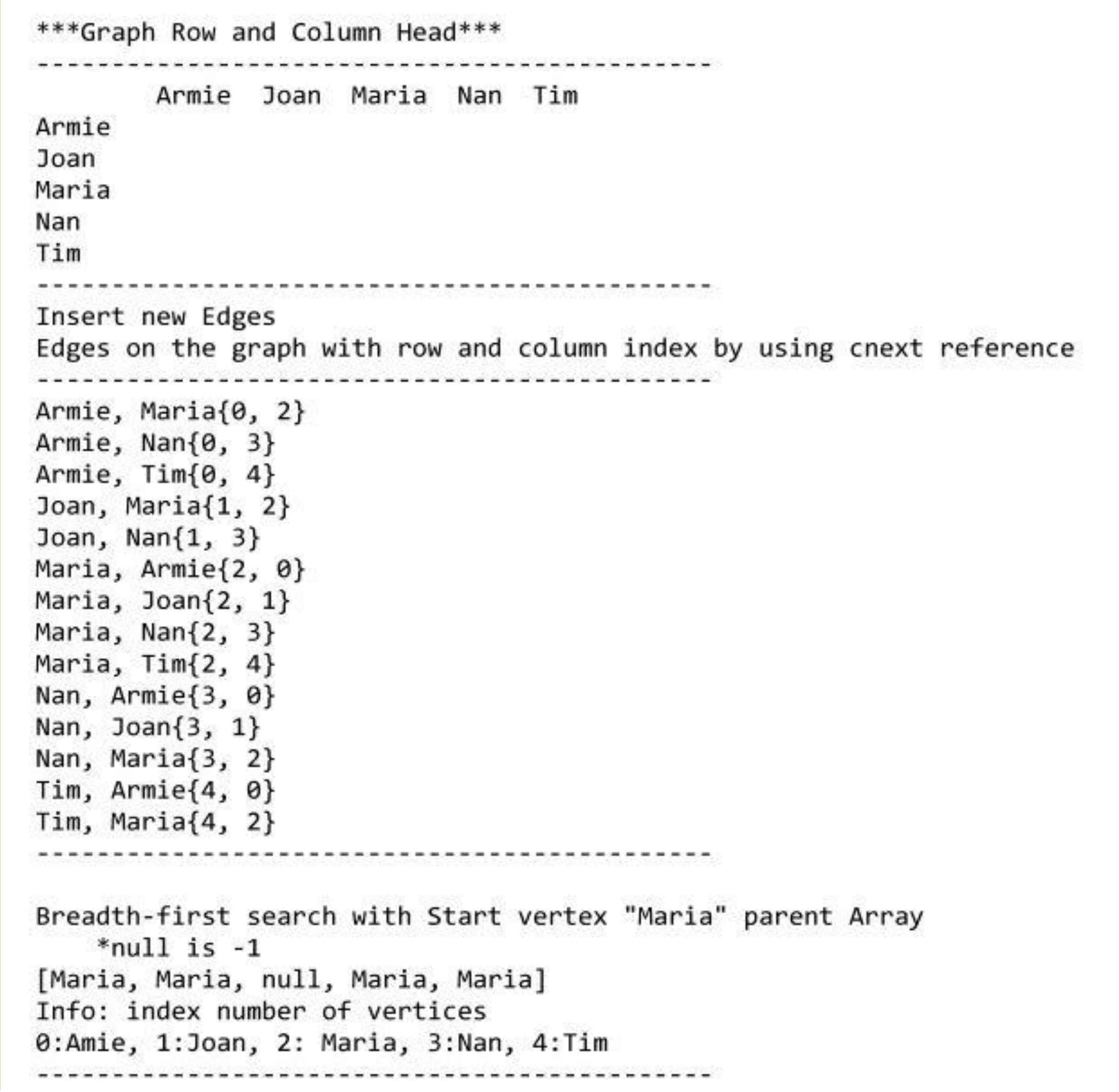
```
//------------------Breadth-First-Search-------------------------------------
adjacencyListMatrix = new AdjacencyListMatrix<> ( numV: 5, directed: true,arr);
adjacencyListMatrix.printGraphRowAndColumn ();
System.out.println ("Insert new Edges");
adjacencyListMatrix.insert (new Edge<> ( source: 'A', dest: 'B'));
adjacencyListMatrix.insert (new Edge<> ( source: 'B', dest: 'A'));
adjacencyListMatrix.insert (new Edge<> ( source: 'B', dest: 'E'));
adjacencyListMatrix.insert (new Edge<> ( source: 'D', dest: 'A'));
adjacencyListMatrix.insert (new Edge<> ( source: 'E', dest: 'A'));
adjacencyListMatrix.insert (new Edge<> ( source: 'E', dest: 'C'));
adjacencyListMatrix.insert (new Edge<> ( source: 'E', dest: 'D'));
System.out.println (adjacencyListMatrix);
System.out.println ("Breadth-first search with Start vertex 'A' parent Array");
System.out.println ("\t*null is -1");
System.out.println (Arrays.toString (adjacencyListMatrix.bfs ( startVertex: 'A')));
System.out.println ("Info: index number of vertices");
System.out.println ("0:A, 1:B, 2:C, 3:D, 4:E");
```



```
---------------------------------------------


***Graph Row and Column Head***
---------------------------------------------
        A  B  C  D  E
A
B
C
D
E
---------------------------------------------
Insert new Edges
Edges on the graph with row and column index by using cnext reference
---------------------------------------------
A, B{0, 1}
B, A{1, 0}
B, E{1, 4}
D, A{3, 0}
E, A{4, 0}
E, C{4, 2}
E, D{4, 3}
---------------------------------------------


Breadth-first search with Start vertex 'A' parent Array
    *null is -1
[null, A, E, E, B]
Info: index number of vertices
0:A, 1:B, 2:C, 3:D, 4:E
---------------------------------------------
```

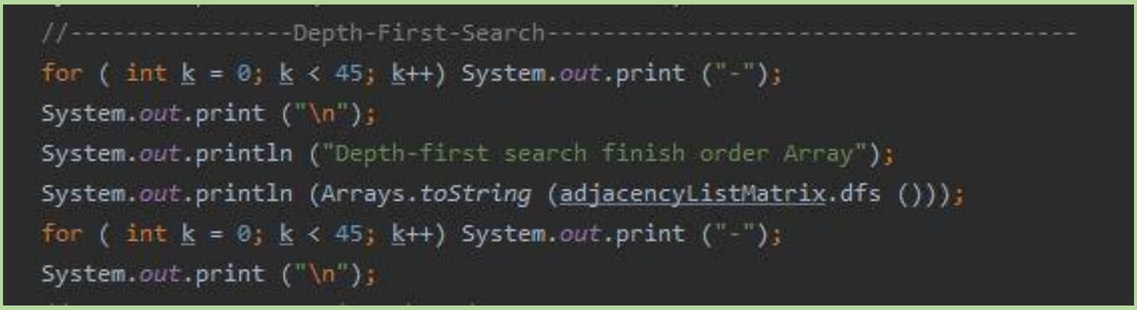


```
//-----------------Breadth-First-Search------------------------------------
adjacencyListMatrix = new AdjacencyListMatrix<> ( numV: 5, directed: false,arr);
adjacencyListMatrix.printGraphRowAndColumn ();
System.out.println ("Insert new Edges");
adjacencyListMatrix.insert (new Edge<> ( source: "Maria", dest: "Armie"));
adjacencyListMatrix.insert (new Edge<> ( source: "Maria", dest: "Tim"));
adjacencyListMatrix.insert (new Edge<> ( source: "Maria", dest: "Joan"));
adjacencyListMatrix.insert (new Edge<> ( source: "Maria", dest: "Nan"));
adjacencyListMatrix.insert (new Edge<> ( source: "Armie", dest: "Nan"));
adjacencyListMatrix.insert (new Edge<> ( source: "Armie", dest: "Tim"));
adjacencyListMatrix.insert (new Edge<> ( source: "Nan", dest: "Joan"));
System.out.println (adjacencyListMatrix);
System.out.println ("Breadth-first search with Start vertex \"Maria\" parent Array");
System.out.println ("\t*null is -1");
System.out.println (Arrays.toString (adjacencyListMatrix.bfs ( startVertex: "Maria")));
System.out.println ("Info: index number of vertices");
System.out.println ("0:Amie, 1:Joan, 2: Maria, 3:Nan, 4:Tim");
```

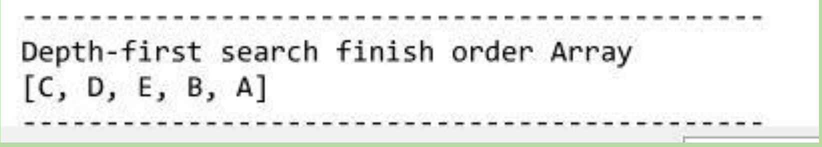


```
***Graph Row and Column Head***
---------------------------------------------
        Armie  Joan  Maria  Nan  Tim
Armie
Joan
Maria
Nan
Tim
---------------------------------------------
Insert new Edges
Edges on the graph with row and column index by using cnext reference
---------------------------------------------
Armie, Maria{0, 2}
Armie, Nan{0, 3}
Armie, Tim{0, 4}
Joan, Maria{1, 2}
Joan, Nan{1, 3}
Maria, Armie{2, 0}
Maria, Joan{2, 1}
Maria, Nan{2, 3}
Maria, Tim{2, 4}
Nan, Armie{3, 0}
Nan, Joan{3, 1}
Nan, Maria{3, 2}
Tim, Armie{4, 0}
Tim, Maria{4, 2}
---------------------------------------------

Breadth-first search with Start vertex "Maria" parent Array
    *null is -1
[Maria, Maria, null, Maria, Maria]
Info: index number of vertices
0:Amie, 1:Joan, 2: Maria, 3:Nan, 4:Tim
---------------------------------------------
```

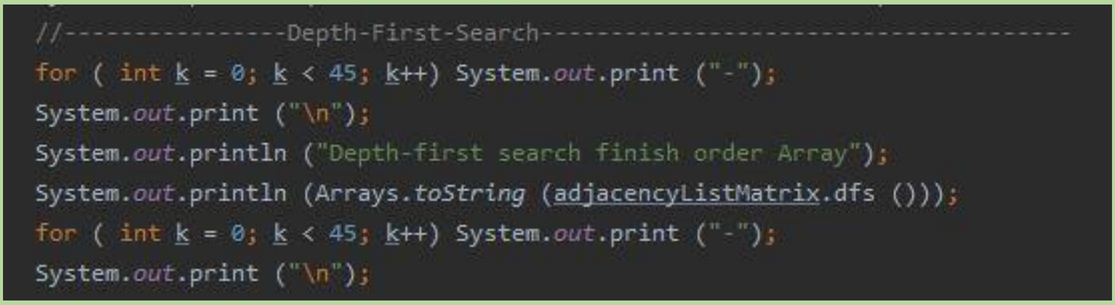
T8



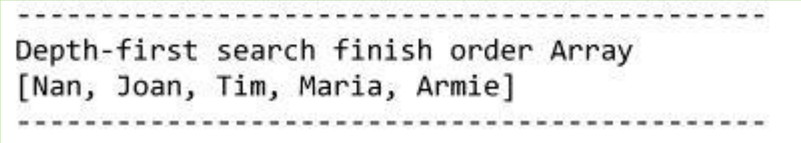
```
//---------------Depth-First-Search-------------------------------------
for ( int k = 0; k < 45; k++) System.out.print ("-");
System.out.print ("\n");
System.out.println ("Depth-first search finish order Array");
System.out.println (Arrays.toString (adjacencyListMatrix.dfs ()));
for ( int k = 0; k < 45; k++) System.out.print ("-");
System.out.print ("\n");
```



```
---------------------------------------------
Depth-first search finish order Array
[C, D, E, B, A]
---------------------------------------------
```

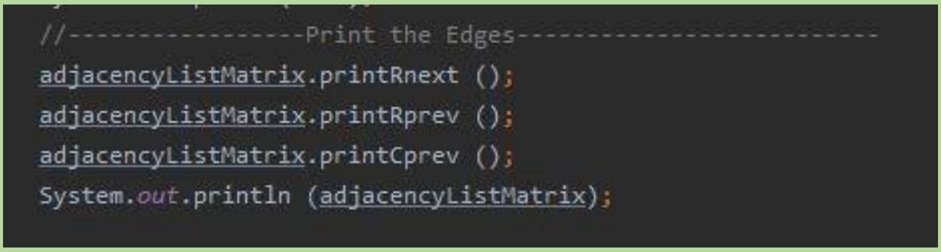


```
//---------------Depth-First-Search-------------------------------------
for ( int k = 0; k < 45; k++) System.out.print ("-");
System.out.print ("\n");
System.out.println ("Depth-first search finish order Array");
System.out.println (Arrays.toString (adjacencyListMatrix.dfs ()));
for ( int k = 0; k < 45; k++) System.out.print ("-");
System.out.print ("\n");
```

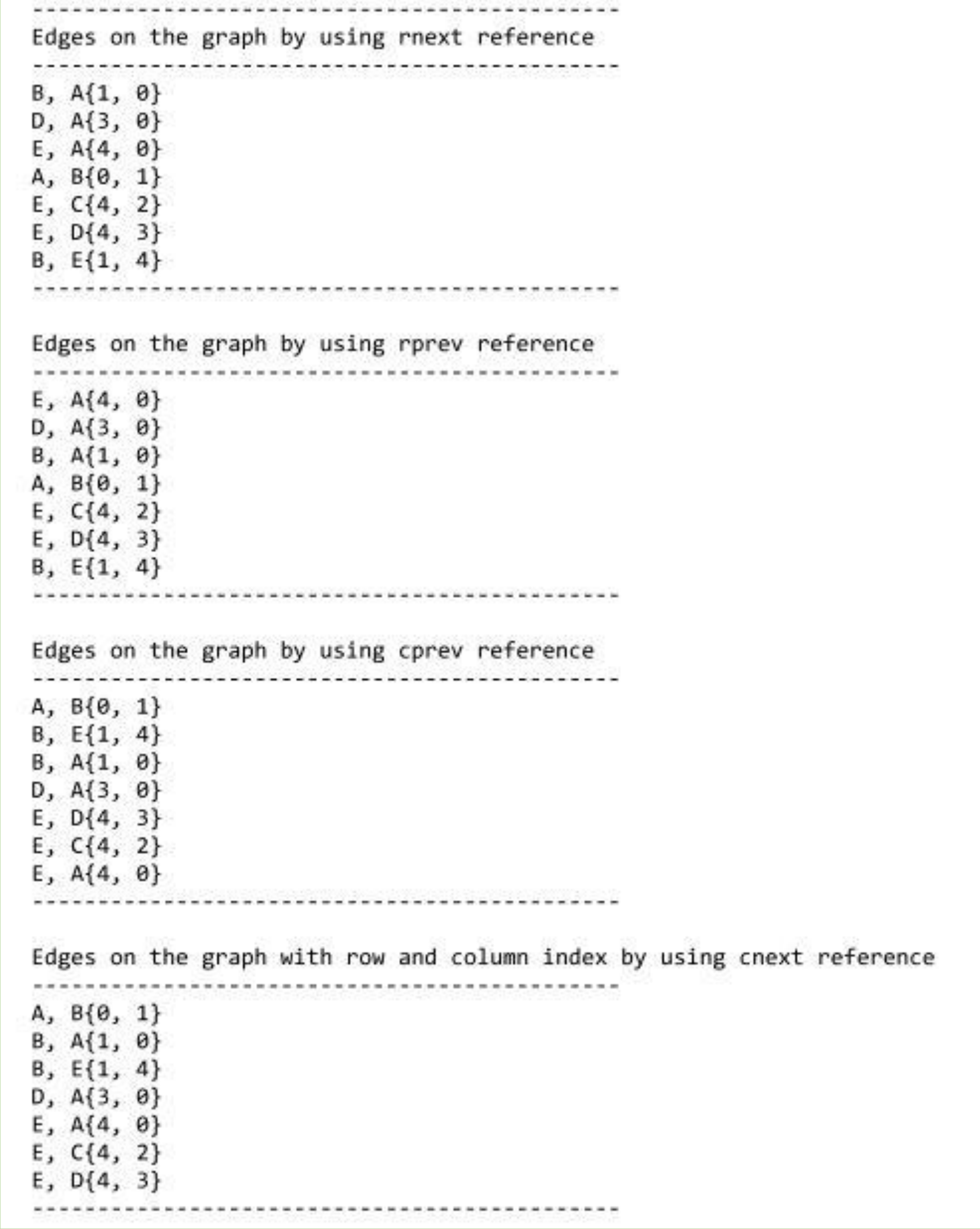


```
---------------------------------------------
Depth-first search finish order Array
[Nan, Joan, Tim, Maria, Armie]
---------------------------------------------
```

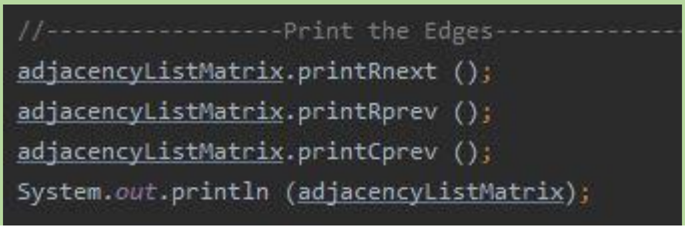
T9



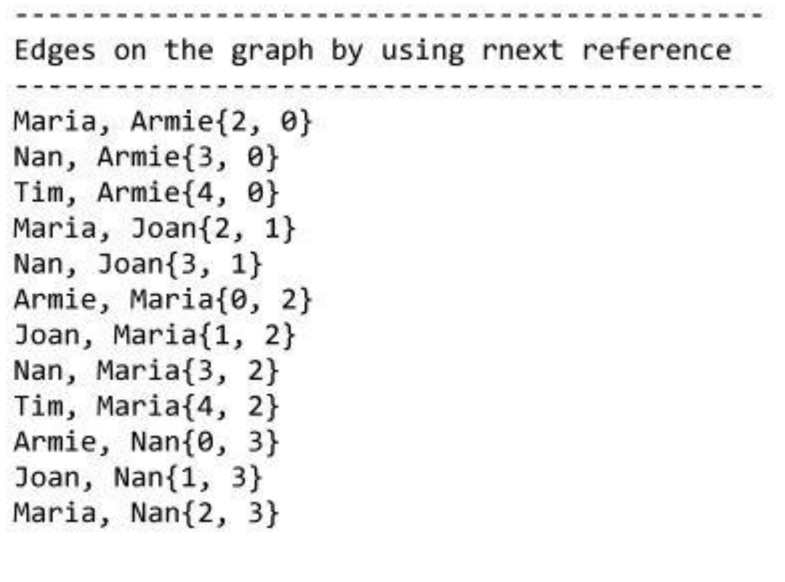
```
//----------------Print the Edges-------------------------
adjacencyListMatrix.printRnext ();
adjacencyListMatrix.printRprev ();
adjacencyListMatrix.printCprev ();
System.out.println (adjacencyListMatrix);
```



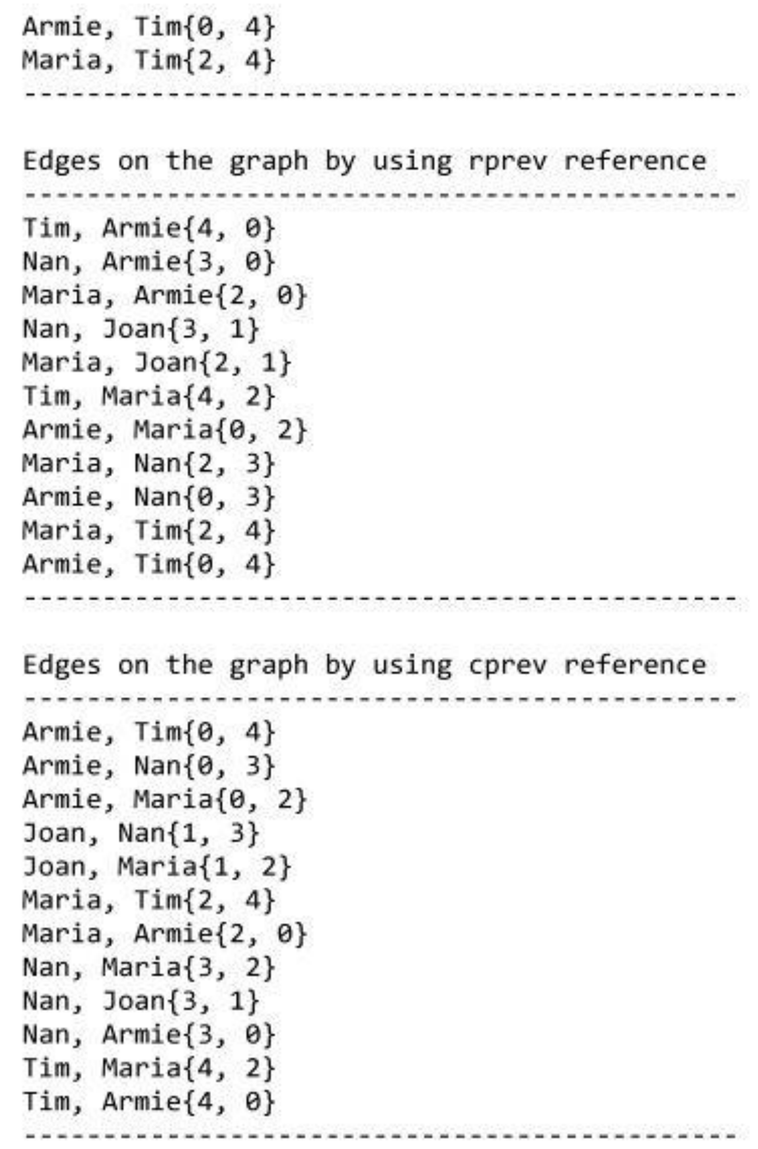
```
---------------------------------------------
Edges on the graph by using rnext reference
---------------------------------------------
B, A{1, 0}
D, A{3, 0}
E, A{4, 0}
A, B{0, 1}
E, C{4, 2}
E, D{4, 3}
B, E{1, 4}
---------------------------------------------

Edges on the graph by using rprev reference
---------------------------------------------
E, A{4, 0}
D, A{3, 0}
B, A{1, 0}
A, B{0, 1}
E, C{4, 2}
E, D{4, 3}
B, E{1, 4}
---------------------------------------------

Edges on the graph by using cprev reference
---------------------------------------------
A, B{0, 1}
B, E{1, 4}
B, A{1, 0}
D, A{3, 0}
E, D{4, 3}
E, C{4, 2}
E, A{4, 0}
---------------------------------------------

Edges on the graph with row and column index by using cnext reference
---------------------------------------------
A, B{0, 1}
B, A{1, 0}
B, E{1, 4}
D, A{3, 0}
E, A{4, 0}
E, C{4, 2}
E, D{4, 3}
---------------------------------------------
```



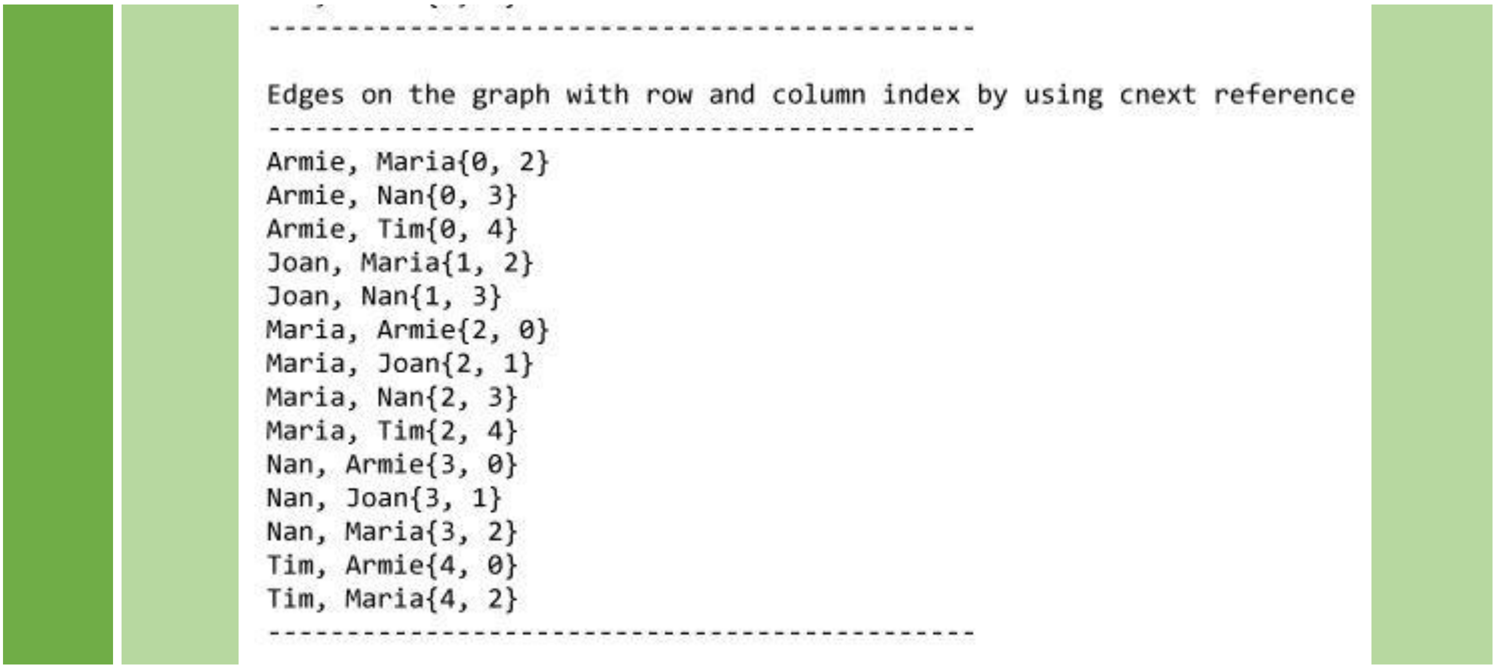
```
//----------------Print the Edges-------------
adjacencyListMatrix.printRnext ();
adjacencyListMatrix.printRprev ();
adjacencyListMatrix.printCprev ();
System.out.println (adjacencyListMatrix);
```



```
---------------------------------------------
Edges on the graph by using rnext reference
---------------------------------------------
Maria, Armie{2, 0}
Nan, Armie{3, 0}
Tim, Armie{4, 0}
Maria, Joan{2, 1}
Nan, Joan{3, 1}
Armie, Maria{0, 2}
Joan, Maria{1, 2}
Nan, Maria{3, 2}
Tim, Maria{4, 2}
Armie, Nan{0, 3}
Joan, Nan{1, 3}
Maria, Nan{2, 3}
```



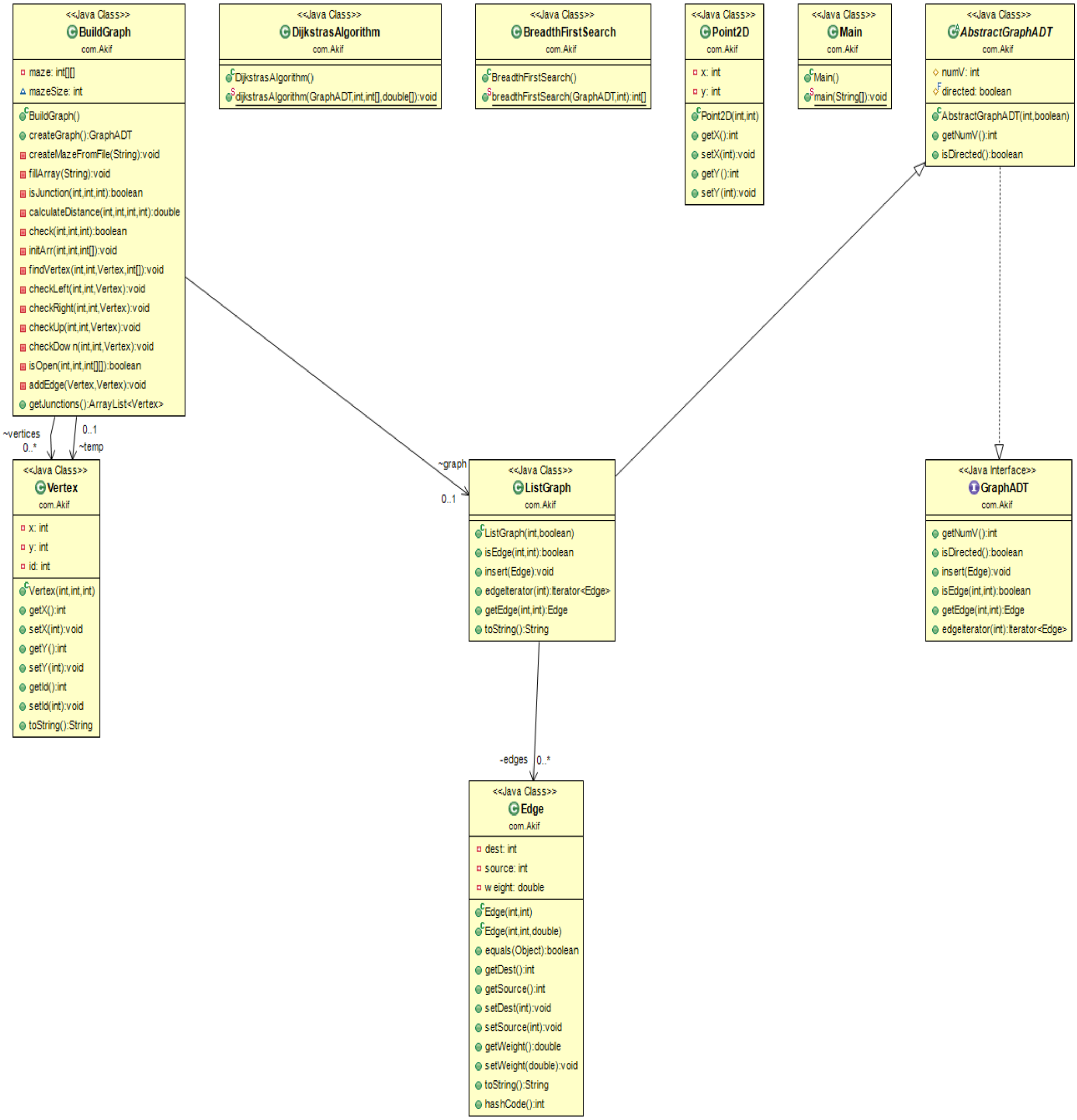
```
Armie, Tim{0, 4}
Maria, Tim{2, 4}
---------------------------------------------

Edges on the graph by using rprev reference
---------------------------------------------
Tim, Armie{4, 0}
Nan, Armie{3, 0}
Maria, Armie{2, 0}
Nan, Joan{3, 1}
Maria, Joan{2, 1}
Tim, Maria{4, 2}
Armie, Maria{0, 2}
Maria, Nan{2, 3}
Armie, Nan{0, 3}
Maria, Tim{2, 4}
Armie, Tim{0, 4}
---------------------------------------------

Edges on the graph by using cprev reference
---------------------------------------------
Armie, Tim{0, 4}
Armie, Nan{0, 3}
Armie, Maria{0, 2}
Joan, Nan{1, 3}
Joan, Maria{1, 2}
Maria, Tim{2, 4}
Maria, Armie{2, 0}
Nan, Maria{3, 2}
Nan, Joan{3, 1}
Nan, Armie{3, 0}
Tim, Maria{4, 2}
Tim, Armie{4, 0}
---------------------------------------------
```



```
---------------------------------------------

Edges on the graph with row and column index by using cnext reference
---------------------------------------------
Armie, Maria{0, 2}
Armie, Nan{0, 3}
Armie, Tim{0, 4}
Joan, Maria{1, 2}
Joan, Nan{1, 3}
Maria, Armie{2, 0}
Maria, Joan{2, 1}
Maria, Nan{2, 3}
Maria, Tim{2, 4}
Nan, Armie{3, 0}
Nan, Joan{3, 1}
Nan, Maria{3, 2}
Tim, Armie{4, 0}
Tim, Maria{4, 2}
---------------------------------------------
```

Q3 REPORT

1. CLASS DIAGRAMS



2. PROBLEM SOLUTION APPROACH

My Problem solution steps are;

– Specify the problem requirements
– Analyze the problem
– Design an algorithm and Program
– Implement the algorithm
– Test and verify the program
– Maintain and update the program

1) **Specify the problem requirements :** I understand the problem.

2) **Analyze the problem :** I identify;

   – Input data

   – Output data

   – Additional requirements and constraints

3) **Design an algorithm and Program :** I divide the problem into sub-problems. I listed major steps (sub-problems). I break down each step into a more detailed list. To do these We have to divide this big project into small pieces.

➡ **Implement the algorithm :** I wrote the algorithm in Java by converting each step into statements of Java (classes ,methods etc.)

Firstly, I wrote the **Edge class** .

After I wrote **GraphADT Interface.** After I took common methods for a graph in **AbstractGraphADT class.**

To keep Junction square information as a vertex I wrote **Vertex** class.

To read maze and create the graph, I wrote **BuildGraph** class.

Lastly, I wrote **Dijkstra's Algorithm** to get shortest distance and I wrote **BreadthFirstSearch** Algorithm to show vertices on the shortest path.

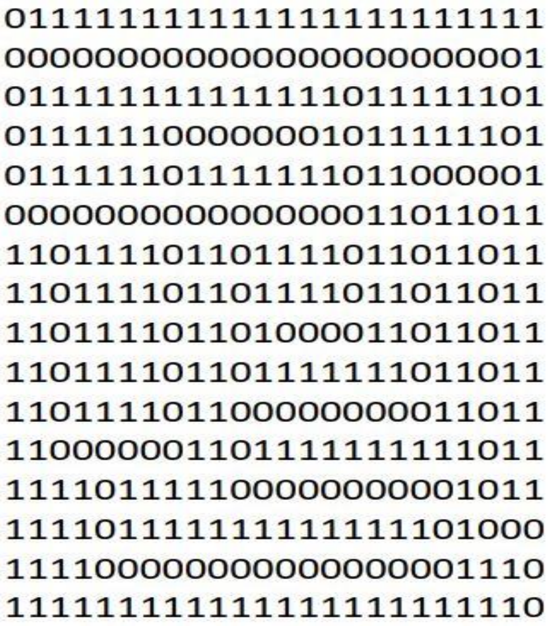
Note: I found the **shortest path** by using both Dijkstra's Algorithm and BreadthFirstSearch Algorithm. Of course since graph is weighted Dijkstra's Algorithm is better but I wanted to try both of them to find path not the distance. To find distance I used Dijkstra's Algorithm.

Note: To represent graph I used **list graph** since it's performance is better.

4) **Test and verify the program:** To test program I wrote the **Main class.** In this class I applied BreadthFirstSearch and Dijkstra's Algorithm I printed the results. Note that to run this program you need to have **maze.txt file** it contains a sequence of lines consisting of Os and 1s.

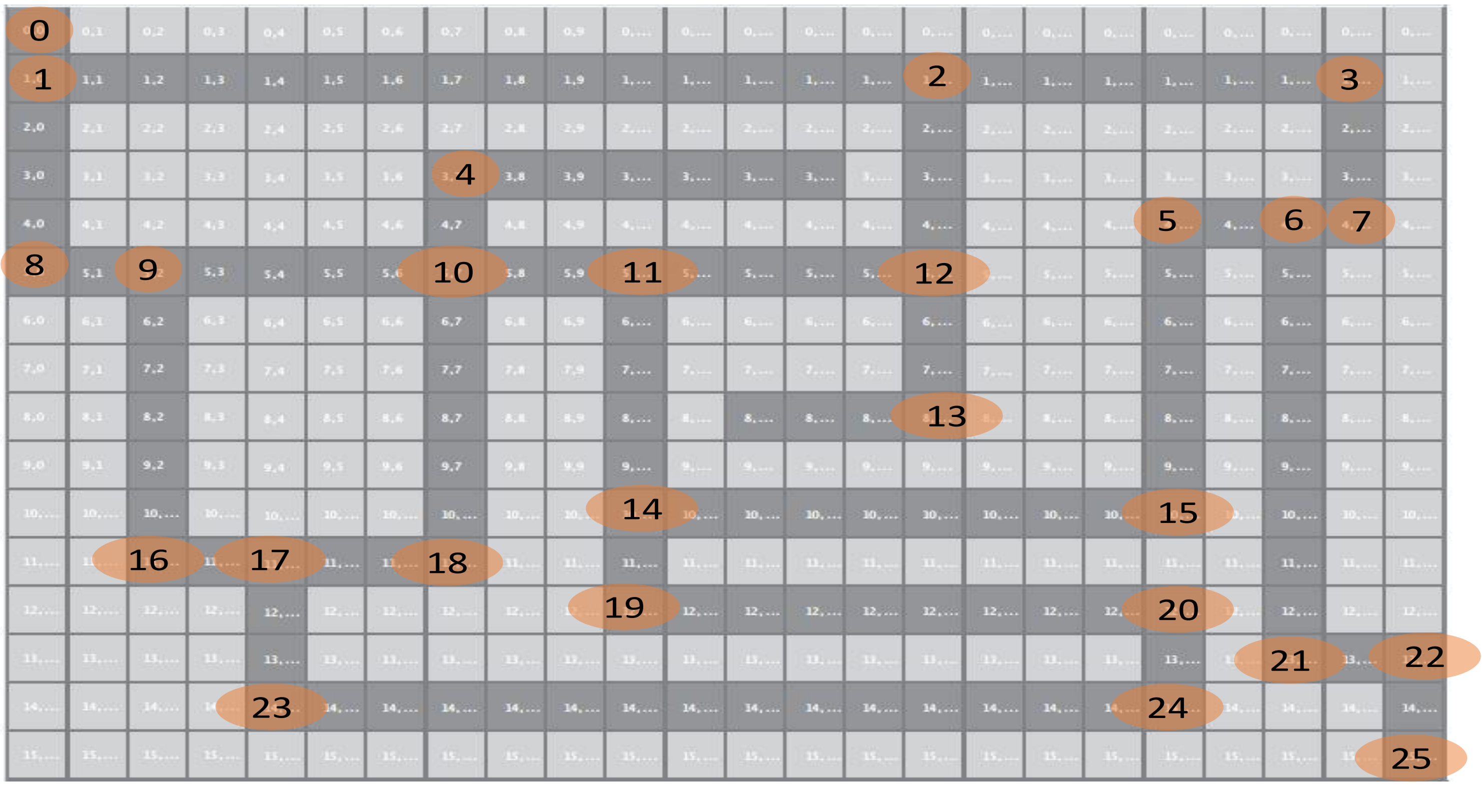
5) **Maintain and update the program :** I keep the program up-to-date

Explanation on my Solution



```
011111111111111111111111
000000000000000000000001
011111111111111011111101
011111100000001011111101
011111101111111011000001
000000000000000011011011
110111101101111011011011
110111101101111011011011
110111101101000011011011
110111101101111111011011
110111101100000000011011
110000001101111111111011
111101111100000000001011
111101111111111111101000
111100000000000000001110
111111111111111111111110
```

To solve this problem I found the vertices I created the graph where the vertices are junction squares and the weight of an edge is the distance defined by the number of squares from the junction square represented by the source vertex to the next junction square represented by the destination vertex as shown in figure. **In following figure my vertices are 0 to 25 which is junction squares.** Firstly, I found the vertices and after I made the connections(edges) between vertices by using a simple algorithm.
Lastly, to find shortest path I used 2 methods, to get shortest distance I used **Dijkstra's Algorithm** and to show vertices on the shortest path, I used **BreadthFirstSearch Algorithm** and I showed vertices on the shortest distance and weights between vertices.



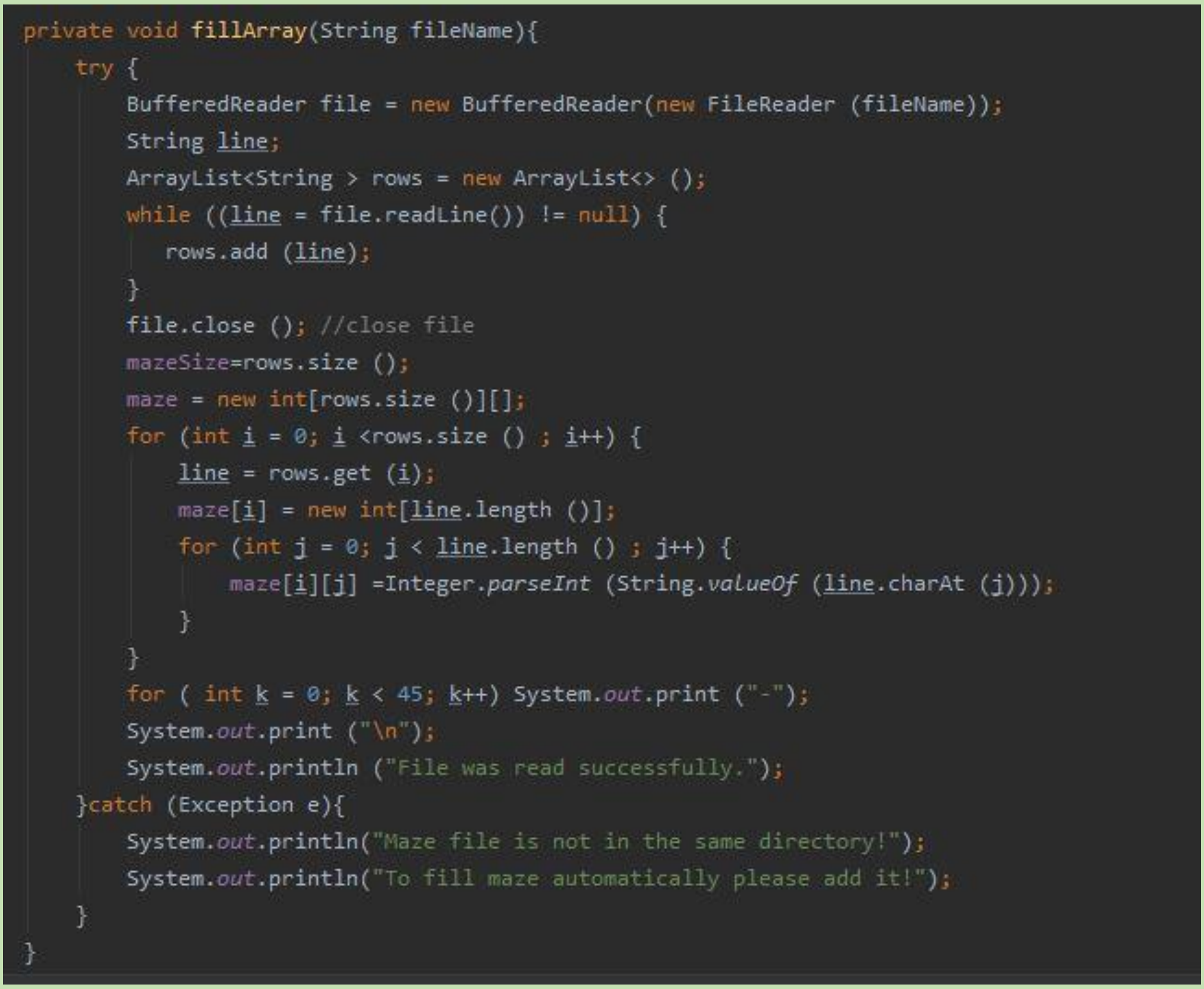
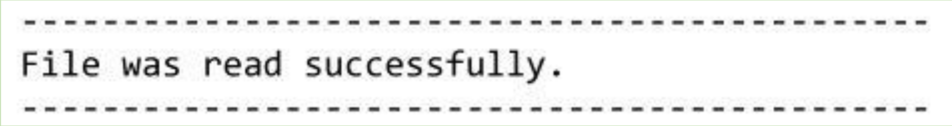
*To test program I used the maze above from homework pdf file.

* You can see these vertices on my test results.

3. TEST CASES

Test ID	Test Case	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T1	Read the Maze.	1) Create Maze file 2) Call method with proper parameter. 3) Run Program	Maze file sequence of lines consisting of Os and 1s	Maze has created successfully.	As Expected	**Pass**
T2	Find Vertices on the Maze.(junction squares)	1) Create Maze file 2) Call method with proper parameter. 3) Run Program	Maze file sequence of lines consisting of Os and 1s	Vertices has found successfully on the maze.	As Expected	**Pass**
T3	Convert the Maze to a weighted graph.	1) Create Maze file 2) Call method with proper parameter. 3) Run Program	Maze file sequence of lines consisting of Os and 1s	Graph has created successfully	As Expected	**Pass**
T4	Find the shortest path from upper-left corner to lower-right Corner.	1) Create Maze file 2) Call method with proper parameter. 3) Run Program	Maze file sequence of lines consisting of Os and 1s	Finding shortest path	As Expected	**Pass**

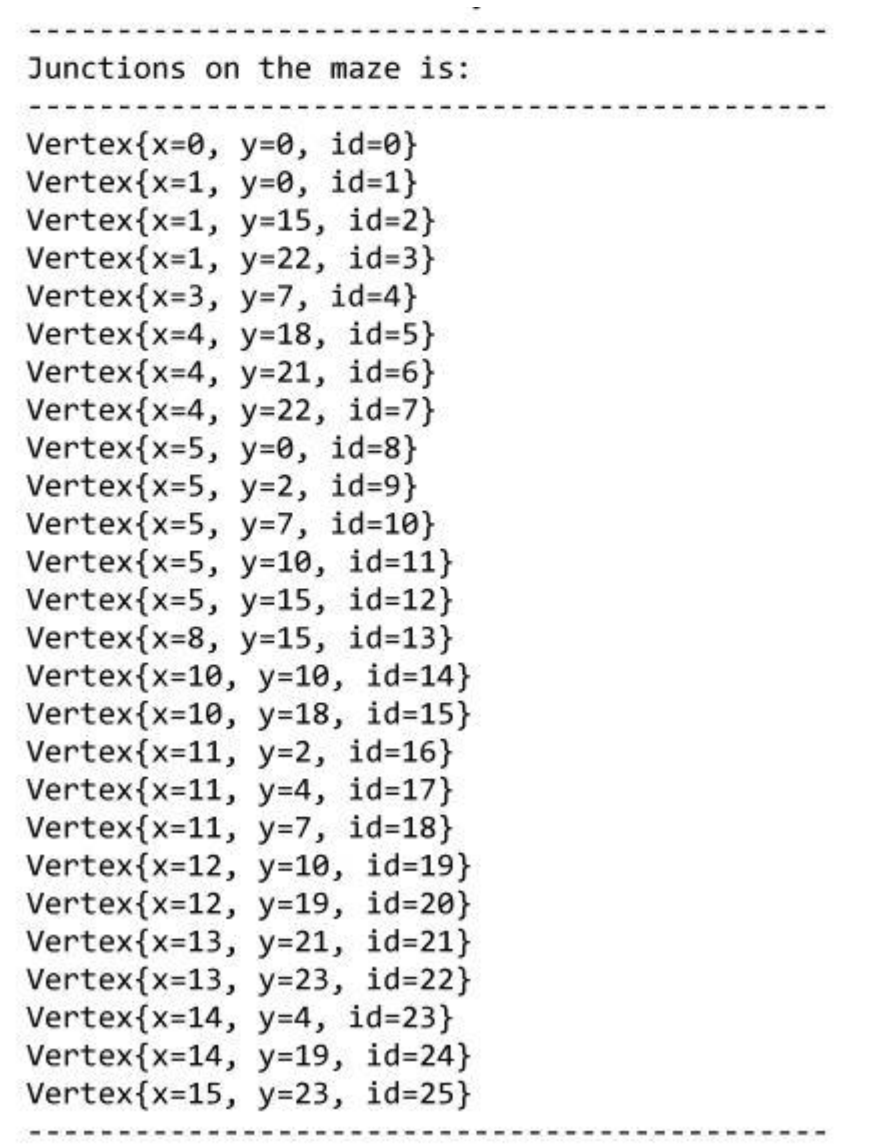
4. RUNNING AND RESULTS

Test ID	Test Result
T1	 

```
private void fillArray(String fileName){
    try {
        BufferedReader file = new BufferedReader(new FileReader (fileName));
        String line;
        ArrayList<String > rows = new ArrayList<> ();
        while ((line = file.readLine()) != null) {
            rows.add (line);
        }
        file.close (); //close file
        mazeSize=rows.size ();
        maze = new int[rows.size ()][];
        for (int i = 0; i <rows.size () ; i++) {
            line = rows.get (i);
            maze[i] = new int[line.length ()];
            for (int j = 0; j < line.length () ; j++) {
                maze[i][j] =Integer.parseInt (String.valueOf (line.charAt (j)));
            }
        }
        for ( int k = 0; k < 45; k++) System.out.print ("-");
        System.out.print ("\n");
        System.out.println ("File was read successfully.");
    }catch (Exception e){
        System.out.println("Maze file is not in the same directory!");
        System.out.println("To fill maze automatically please add it!");
    }
}
```

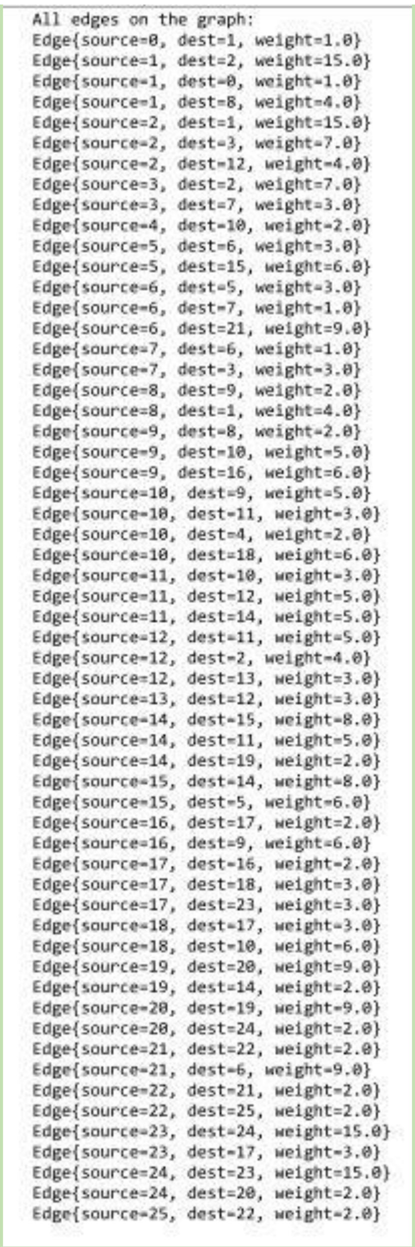
```
---------------------------------------------
File was read successfully.
---------------------------------------------
```

T2



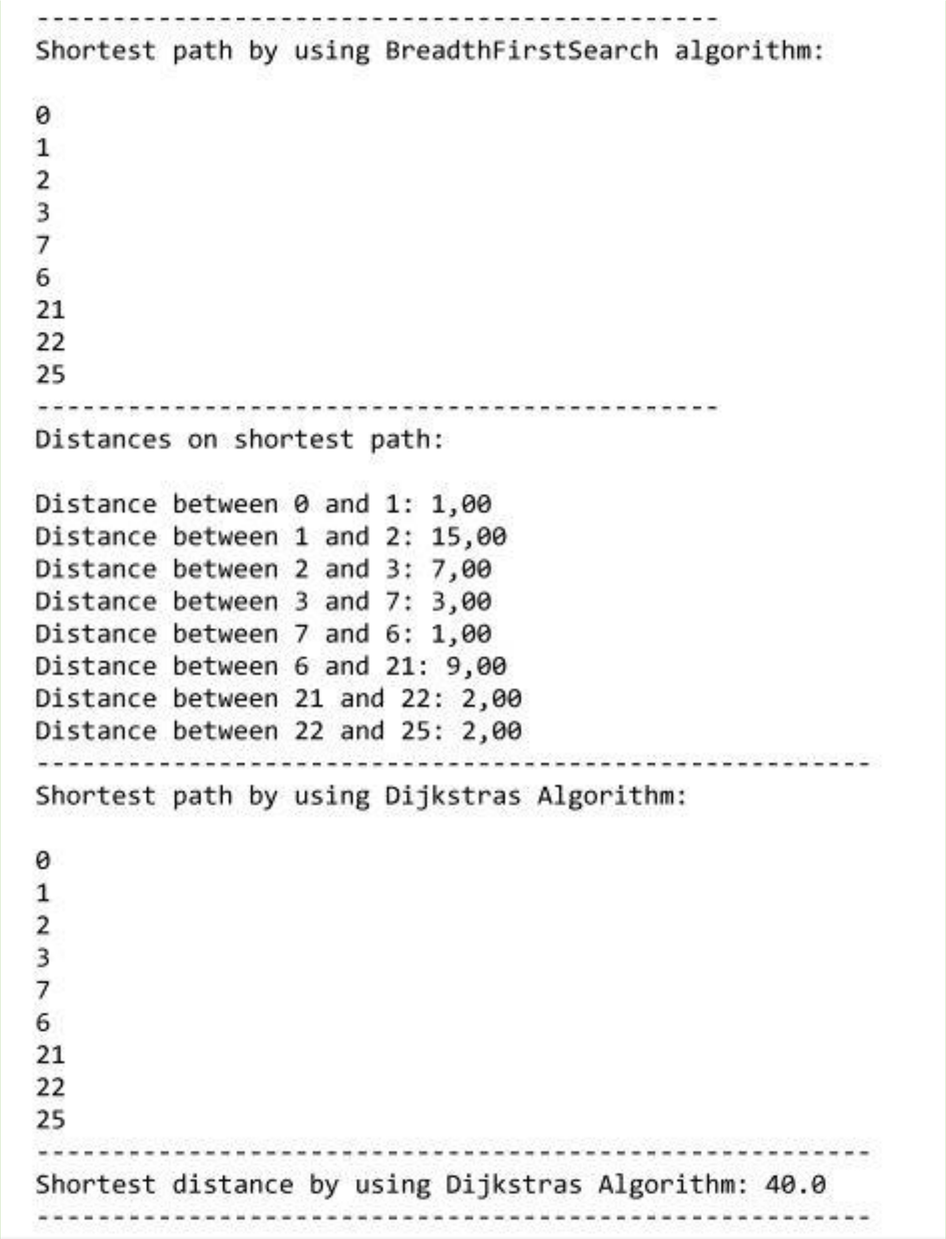
```
---------------------------------------------
Junctions on the maze is:
---------------------------------------------
Vertex{x=0, y=0, id=0}
Vertex{x=1, y=0, id=1}
Vertex{x=1, y=15, id=2}
Vertex{x=1, y=22, id=3}
Vertex{x=3, y=7, id=4}
Vertex{x=4, y=18, id=5}
Vertex{x=4, y=21, id=6}
Vertex{x=4, y=22, id=7}
Vertex{x=5, y=0, id=8}
Vertex{x=5, y=2, id=9}
Vertex{x=5, y=7, id=10}
Vertex{x=5, y=10, id=11}
Vertex{x=5, y=15, id=12}
Vertex{x=8, y=15, id=13}
Vertex{x=10, y=10, id=14}
Vertex{x=10, y=18, id=15}
Vertex{x=11, y=2, id=16}
Vertex{x=11, y=4, id=17}
Vertex{x=11, y=7, id=18}
Vertex{x=12, y=10, id=19}
Vertex{x=12, y=19, id=20}
Vertex{x=13, y=21, id=21}
Vertex{x=13, y=23, id=22}
Vertex{x=14, y=4, id=23}
Vertex{x=14, y=19, id=24}
Vertex{x=15, y=23, id=25}
---------------------------------------------
```

T3



```
All edges on the graph:
Edge{source=0, dest=1, weight=1.0}
Edge{source=1, dest=2, weight=15.0}
Edge{source=1, dest=0, weight=1.0}
Edge{source=1, dest=8, weight=4.0}
Edge{source=2, dest=1, weight=15.0}
Edge{source=2, dest=3, weight=7.0}
Edge{source=2, dest=12, weight=4.0}
Edge{source=3, dest=2, weight=7.0}
Edge{source=3, dest=7, weight=3.0}
Edge{source=4, dest=10, weight=2.0}
Edge{source=5, dest=6, weight=3.0}
Edge{source=5, dest=15, weight=6.0}
Edge{source=6, dest=5, weight=3.0}
Edge{source=6, dest=7, weight=1.0}
Edge{source=6, dest=21, weight=9.0}
Edge{source=7, dest=6, weight=1.0}
Edge{source=7, dest=3, weight=3.0}
Edge{source=8, dest=9, weight=2.0}
Edge{source=8, dest=1, weight=4.0}
Edge{source=9, dest=8, weight=2.0}
Edge{source=9, dest=10, weight=5.0}
Edge{source=9, dest=16, weight=6.0}
Edge{source=10, dest=9, weight=5.0}
Edge{source=10, dest=11, weight=3.0}
Edge{source=10, dest=4, weight=2.0}
Edge{source=10, dest=18, weight=6.0}
Edge{source=11, dest=10, weight=3.0}
Edge{source=11, dest=12, weight=5.0}
Edge{source=11, dest=14, weight=5.0}
Edge{source=12, dest=11, weight=5.0}
Edge{source=12, dest=2, weight=4.0}
Edge{source=12, dest=13, weight=3.0}
Edge{source=13, dest=12, weight=3.0}
Edge{source=14, dest=15, weight=8.0}
Edge{source=14, dest=11, weight=5.0}
Edge{source=14, dest=19, weight=2.0}
Edge{source=15, dest=14, weight=8.0}
Edge{source=15, dest=5, weight=6.0}
Edge{source=16, dest=17, weight=2.0}
Edge{source=16, dest=9, weight=6.0}
Edge{source=17, dest=16, weight=2.0}
Edge{source=17, dest=18, weight=3.0}
Edge{source=17, dest=23, weight=3.0}
Edge{source=18, dest=17, weight=3.0}
Edge{source=18, dest=10, weight=6.0}
Edge{source=19, dest=20, weight=9.0}
Edge{source=19, dest=14, weight=2.0}
Edge{source=20, dest=19, weight=9.0}
Edge{source=20, dest=24, weight=2.0}
Edge{source=21, dest=22, weight=2.0}
Edge{source=21, dest=6, weight=9.0}
Edge{source=22, dest=21, weight=2.0}
Edge{source=22, dest=25, weight=2.0}
Edge{source=23, dest=24, weight=15.0}
Edge{source=23, dest=17, weight=3.0}
Edge{source=24, dest=23, weight=15.0}
Edge{source=24, dest=20, weight=2.0}
Edge{source=25, dest=22, weight=2.0}
```

T4



```
---------------------------------------------
Shortest path by using BreadthFirstSearch algorithm:

0
1
2
3
7
6
21
22
25
---------------------------------------------
Distances on shortest path:

Distance between 0 and 1: 1,00
Distance between 1 and 2: 15,00
Distance between 2 and 3: 7,00
Distance between 3 and 7: 3,00
Distance between 7 and 6: 1,00
Distance between 6 and 21: 9,00
Distance between 21 and 22: 2,00
Distance between 22 and 25: 2,00
-------------------------------------------------------
Shortest path by using Dijkstras Algorithm:

0
1
2
3
7
6
21
22
25
-------------------------------------------------------
Shortest distance by using Dijkstras Algorithm: 40.0
-------------------------------------------------------
```