Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

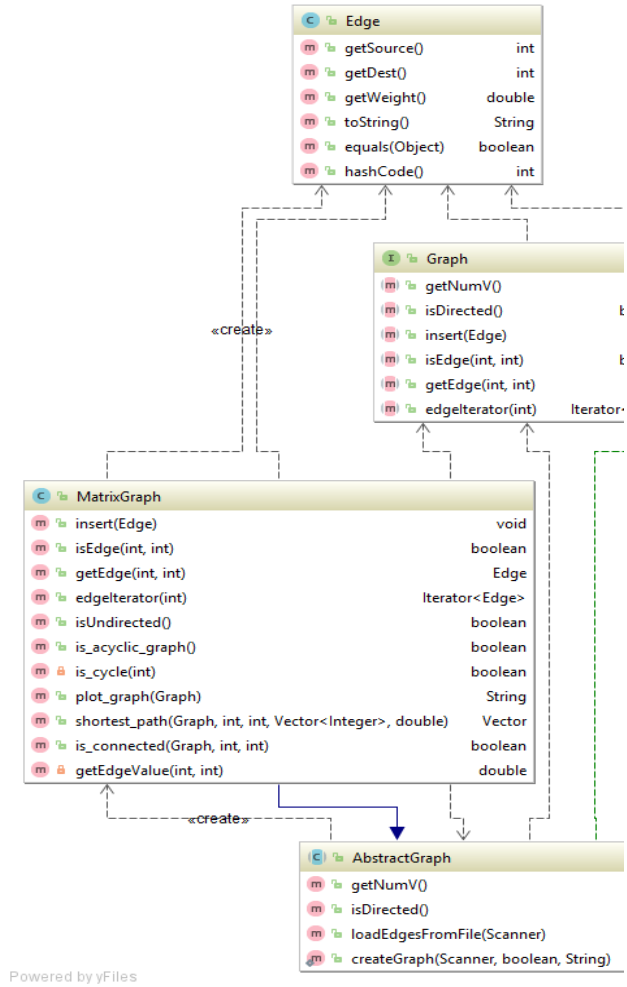
HOMEWORK 7 REPORT

SEVGİ BAYANSALDUZ
151044076

Course Assistant: Fatma Nur Esirci

1 Q1

1.1 Problem Solution Approach



The class AbstractGraph represents a graph in general.

The class MatrixGraph provides representations of graphs using adjacency matrisx.This class contains an inner class,Iter class,which implements the Iterator <Edge> interfaces.

- Shortest Path

The method Shortest_path uses the DijktrasAlgorithm to find the Shortest path from a vertex to another vertex.
Algorithm for find Shortest path:

1. Create DijktrasAlgorithm object
2. Create double array .dist(to stores shortest distances from start vertex to all other vertices)
3. Create int array (use to determine the corresponding path)
4. Call dijktrasAlgoritm with start vertex
5. Set to distance dist[target_vertex]
6. While p[target_index] is not equal the start index,
   →add p[target_index] a collection
   →set the target_index=p[target_index]
7. Reverse the collection and return it.

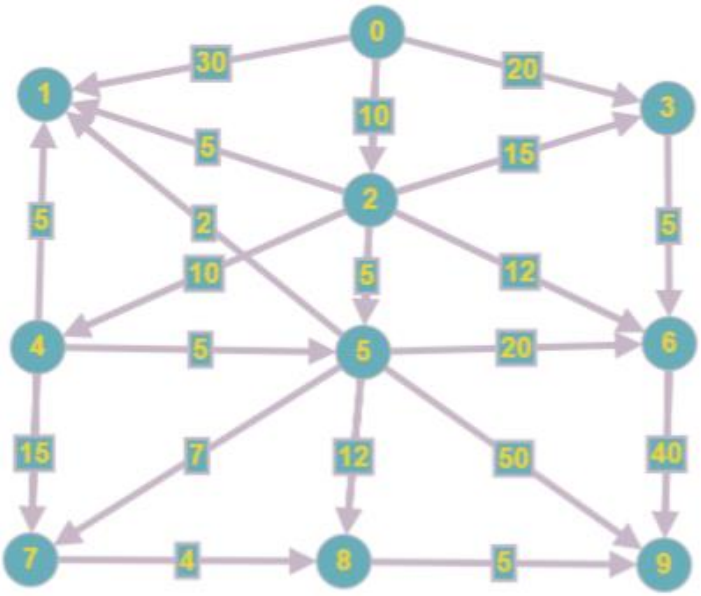
```
/**
 * Find the shortest path from vertex  v1 to vertex  v2 using Dijkst
 */
double distance=0.0;
public Vector shortest_path(Graph graph,int v1,int v2,Vector<Integer>
{
    DijkstrasAlgorithm dijk=new DijkstrasAlgorithm();
    int []pred= new int[graph.getNumV()];
    double []dist=new double[graph.getNumV()];
    dijk.dijkstrasAlgorithm(graph,v1,pred,dist);
    this.distance=dist[v2];
    path.add(v2);
    for(int i=v2;i!=v1;i=pred[i])
        path.add(pred[i]);
    for (int i=path.size()-1;i>=0;--i)
    {
        path.add(path.elementAt(i));
        path.remove(i);
    }
    return path;
}
```

- Graph Creation
  →Create a graph object with the specified number of vertices.
  → Create edges with random weight.Edges includes two vertices and a weight. (Dont make a cycle path when creating the edges)
  →Insert edges into the graph.

1.2 Test Cases

- Test Graph For Q1

→Directed acyclic graph have random weight (v=10,e=20).




```
public static void main(String[] args) {
    MatrixGraph test=new MatrixGraph( numV: 10, directed: true);
    /*Edges*/
    Edge e1=new Edge( source: 0, dest: 1, w: 30);
    Edge e2=new Edge( source: 0, dest: 2, w: 10);
    Edge e3=new Edge( source: 0, dest: 3, w: 20);
    Edge e4=new Edge( source: 2, dest: 1, w: 5);
    Edge e5=new Edge( source: 2, dest: 3, w: 15);
    Edge e6=new Edge( source: 2, dest: 4, w: 10);
    Edge e7=new Edge( source: 2, dest: 6, w: 12);
    Edge e8=new Edge( source: 3, dest: 6, w: 5);
    Edge e9=new Edge( source: 4, dest: 1, w: 5);
    Edge e10=new Edge( source: 4, dest: 5, w: 5);
    Edge e11=new Edge( source: 4, dest: 7, w: 15);
    Edge e12=new Edge( source: 2, dest: 5, w: 5);
    Edge e13=new Edge( source: 5, dest: 6, w: 20);
    Edge e14=new Edge( source: 5, dest: 1, w: 2);
    Edge e15=new Edge( source: 5, dest: 8, w: 12);
    Edge e16=new Edge( source: 5, dest: 7, w: 7);
    Edge e17=new Edge( source: 5, dest: 9, w: 50);
    Edge e18=new Edge( source: 6, dest: 9, w: 40);
    Edge e19=new Edge( source: 7, dest: 8, w: 4);
    Edge e20=new Edge( source: 8, dest: 9, w: 5);
```

- **plot_graph**

MainTest

```
**********************************
Plot of graph:
0-->(w:30.0)1-->(w:10.0)2-->(w:20.0)3
1
2-->(w:5.0)1-->(w:15.0)3-->(w:10.0)4-->(w:5.0)5-->(w:12.0)6
3-->(w:5.0)6
4-->(w:5.0)1-->(w:5.0)5-->(w:15.0)7
5-->(w:2.0)1-->(w:20.0)6-->(w:7.0)7-->(w:12.0)8-->(w:50.0)9
6-->(w:40.0)9
7-->(w:4.0)8
8-->(w:5.0)9
9
```

(This is the output of the MainTest. MainTest is in the Q1 folder).

- **is_undirected**

```
**************************************
Is an undirected graph? : false
**************************************
```

(This is the output of the MainTest, MainTest is in the Q1 folder).

- **is_acyclic_graph**

```
**************************************
Is an acyclic graph? : true
**************************************
```

(This is the output of the MainTest. MainTest is in the Q1 folder).

- **shortest_path (use least 3 different label pair)**

```
********Shortest path  for v1:0, v2:9 *********
Shortest path: [0, 2, 5, 7, 8, 9]
Distance of shortest path: 31.0

********Shortest path  for v1:2, v2:9 *********
Shortest path: [2, 5, 7, 8, 9]
Distance of shortest path: 21.0

********Shortest path  for v1:4, v2:8 *********
Shortest path: [4, 5, 7, 8]
Distance of shortest path: 16.0
**************************************

Process finished with exit code 0
```

(This is the output of the MainTest. MainTest is in the Q1 folder).

2 Q2

2.1 Problem Solution Approach

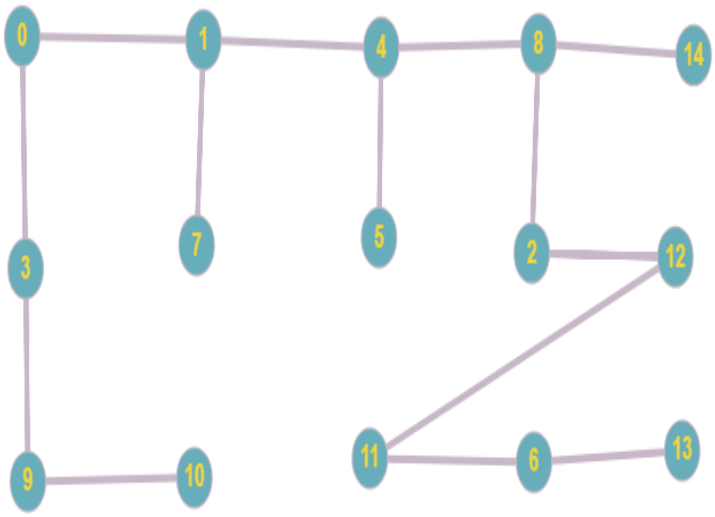
- Graph Creation
  →Create a graph object with the specified number of vertices.
  → Create edges.Edges includes only two vertices. (Dont make a cycle path when creating the edges)
  →Insert edges into the graph.

2.2 Test Cases

- Test Graph For Q2

→Undirected acyclic graph (v=15).

```
lic class MainTest2 {
 public static void main(String[] args) {
    MatrixGraph test=new MatrixGraph( numV: 15, directed: false);
    /*Edges*/
    Edge e1=new Edge( source: 0, dest: 1);
    Edge e2=new Edge( source: 0, dest: 3);
    Edge e3=new Edge( source: 1, dest: 4);
    Edge e4=new Edge( source: 1, dest: 7);
    Edge e5=new Edge( source: 2, dest: 8);
    Edge e6=new Edge( source: 2, dest: 12);
    Edge e7=new Edge( source: 3, dest: 9);
    Edge e8=new Edge( source: 11, dest: 12);
    Edge e9=new Edge( source: 4, dest: 8);
    Edge e10=new Edge( source: 4, dest: 5);
    Edge e12=new Edge( source: 6, dest: 11);
    Edge e13=new Edge( source: 6, dest: 13);
    Edge e14=new Edge( source: 8, dest: 14);
    Edge e15=new Edge( source: 9, dest: 10);
```



- **plot_graph**

```
MainTest2
C:\Users\sevgi\Documents\jdk1.8.0_151\bin\java ...
**************************************
Plot of graph:
0--1--3
1--0--4--7
2--8--12
3--0--9
4--1--5--8
5--4
6--11--13
7--1
8--2--4--14
9--3--10
10--9
11--6--12
12--2--11
13--6
14--8

**************************************
```

(This is the output of the MainTest2. MainTest2 is in the Q2 folder).

- **is_undirected**

```
**************************************
Is an undirected graph? : true
**************************************
```

(This is the output of the MainTest2 .MainTest2 is in the Q2 folder).

- **is_acyclic_graph**

```
**************************************
Is an acyclic graph? : true
**************************************
```

(This is the output of the MainTest2 .MainTest2 is in the Q2 folder).

- **is_connected function (use least 3 different label pair)**

```
**************************************
Is there a path from v1:4 to v2:11  ?  == false
Is there a path from v1:0 to v2:14  ?  == false
Is there a path from v1:6 to v2:13  ?  == true

Process finished with exit code 0
```

(This is the output of the MainTest2 .MainTest2 is in the Q2 folder).

3 Q3

3.1 Problem Solution Approach

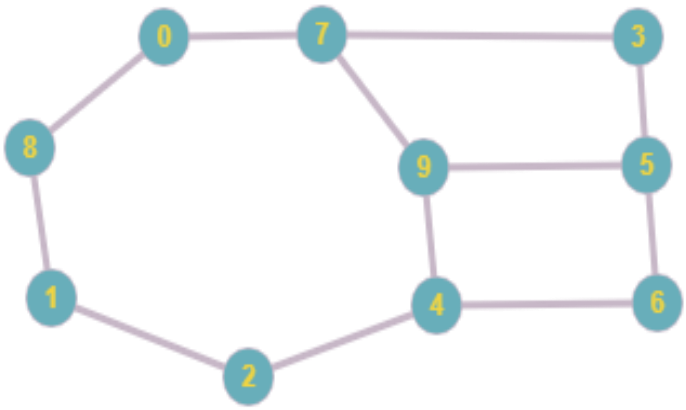
- Graph Creation
  →Create a graph object with the specified number of vertices.
  → Create edges.Edges includes only two vertices. (make a cycle path when creating the edges)
  →Insert edges into the graph.

3.2 Test Cases

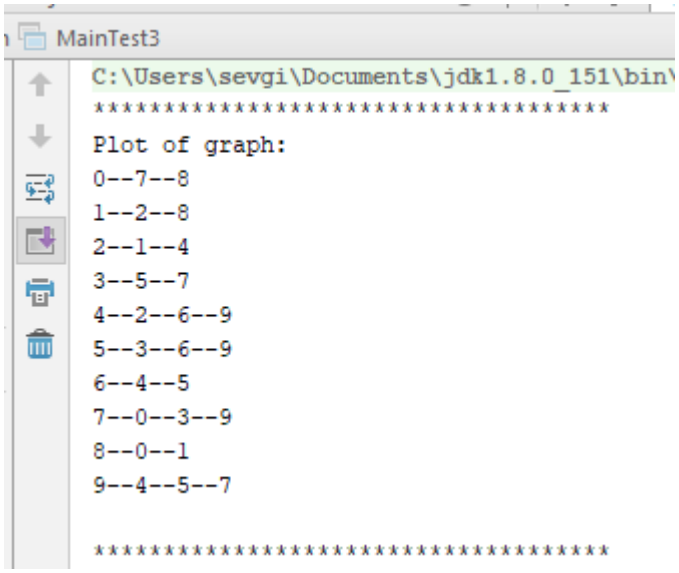
- Test Graph For Q3

→Undirected cyclic graph (v=10).

```
lic class MainTest3 {
 public static void main(String[] args) {
    MatrixGraph test=new MatrixGraph( numV: 10, directed: false);
    Edge e1=new Edge( source: 0, dest: 7);
    Edge e2=new Edge( source: 0, dest: 8);
    Edge e3=new Edge( source: 5, dest: 6);
    Edge e4=new Edge( source: 1, dest: 2);
    Edge e5=new Edge( source: 7, dest: 9);
    Edge e6=new Edge( source: 8, dest: 1);
    Edge e7=new Edge( source: 7, dest: 3);
    Edge e8=new Edge( source: 2, dest: 4);
    Edge e9=new Edge( source: 4, dest: 6);
    Edge e10=new Edge( source: 4, dest: 9);
    Edge e11=new Edge( source: 5, dest: 9);
    Edge e12=new Edge( source: 3, dest: 5);
```

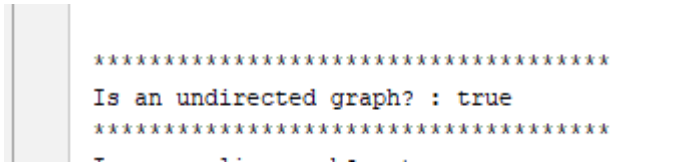


- **plot_graph**



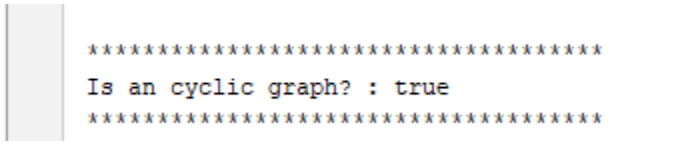
(This is the output of the MainTest3 .MainTest3 is in the Q3 folder).

- **is_undirected**



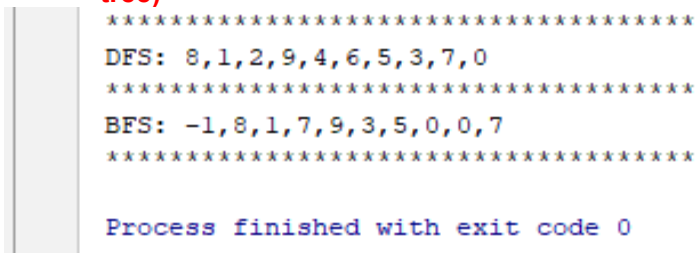
(This is the output of the MainTest3 .MainTest2 is in the Q3 folder).

- **is_acyclic_graph**



(This is the output of the MainTest3 .MainTest3 is in the Q3 folder).

- **DepthFirstSearch (Show that spanning tree) and BreathFirstSearch (Show that spanning tree)**



(This is the output of the MainTest3 .MainTest3 is in the Q3 folder).

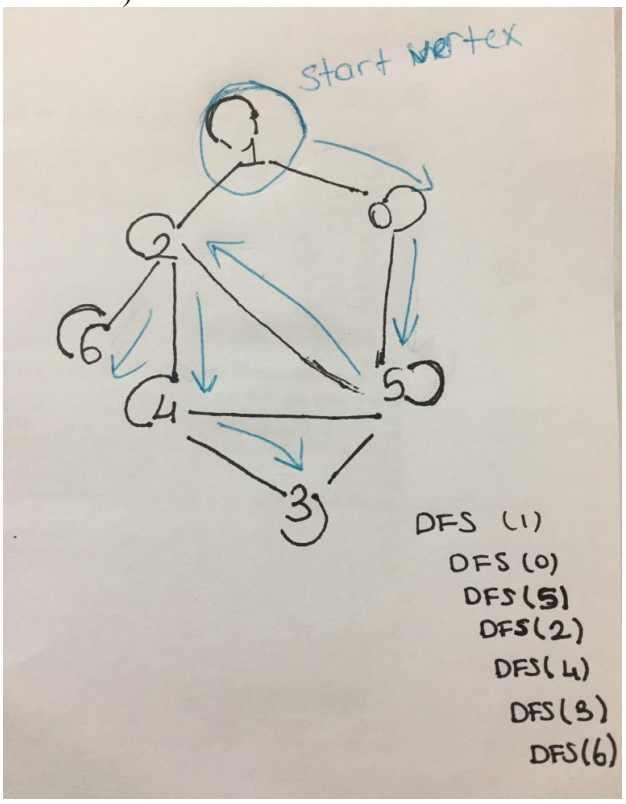
4 Q4

- BFS visit nodes **level by level** in Graph. On the other hand DFS visit nodes of graph **depth wise**. It visits nodes until reach a leaf or a node which doesn't have non-visited nodes.
- BFS is slower and require more memory whereas DFS is faster and require less memory.
- BFS applications:
  - Finding all connected components in a graph.
  - Finding the shortest path between two nodes.
  - Finding all nodes within one connected component.
- DFS aplications:
  - Topological Sorting.
  - Finding connected components.
  - Finding strongly connected components.
  - Finding articulation points (cut vertices) of the graph.
  - Solving puzzles such as maze.

a. Run the DFS algorithm starting from vertex 1, and draw the DFS tree.
b. Run the BFS algorithm starting from vertex 1, and draw the BFS tree.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- A)



- B)