Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 7 REPORT

CENGİZ TOPRAK
161044087

Course Assistant: Fatma Nur Esirci

1 Q1

This part about Question1 in HW7

1.1 Problem Solution Approach

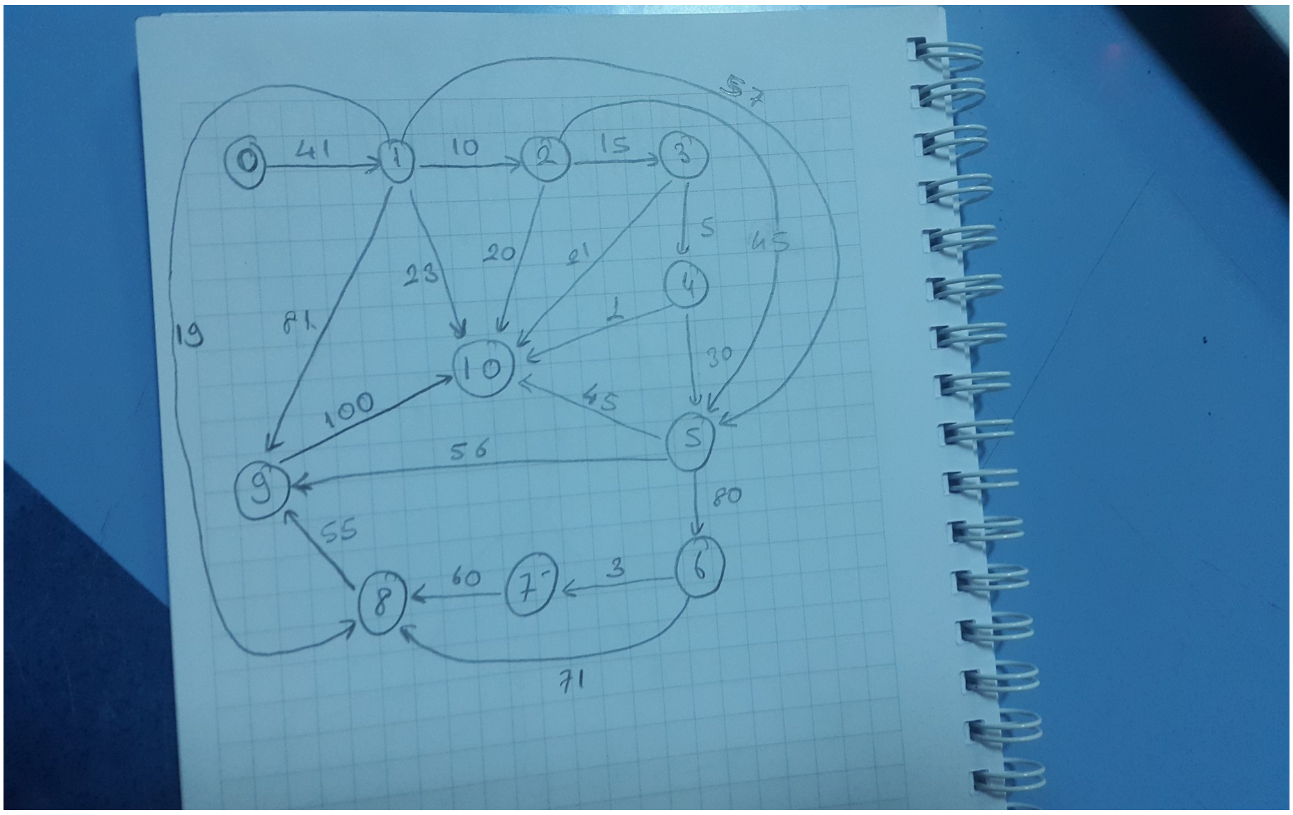
Graphs are created with random vertexes and,it is used Dijkstram Algorithm in order to find shortest path between two vertexes. It is just used ListGraph graph.

1.2 Test Cases

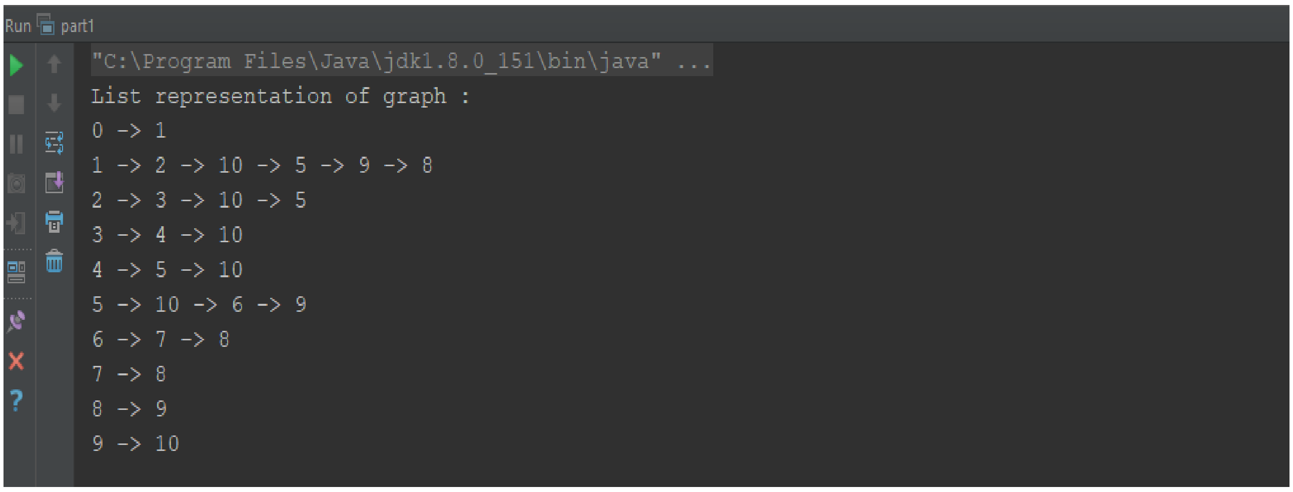
Order of vertex adding to directed graph:

```
AbstractGraph graph = new ListGraph(11,true);
graph.insert(new Edge(0,1,41));
graph.insert(new Edge(1,2,10));
graph.insert(new Edge(2,3,15));
graph.insert(new Edge(2,10,20));
graph.insert(new Edge(3,4,5));
graph.insert(new Edge(4,5,30));
graph.insert(new Edge(5,10,45));
graph.insert(new Edge(5,6,80));
graph.insert(new Edge(5,9,56));
graph.insert(new Edge(6,7,3));
graph.insert(new Edge(7,8,60));
graph.insert(new Edge(8,9,55));
graph.insert(new Edge(1,10,23));
graph.insert(new Edge(3,10,21));
graph.insert(new Edge(4,10,1));
graph.insert(new Edge(1,5,57));
graph.insert(new Edge(9,10,100));
graph.insert(new Edge(1,9,81));
graph.insert(new Edge(2,5,43));
graph.insert(new Edge(6,8,71));
graph.insert(new Edge(1,8,19));
```

Graph representation :

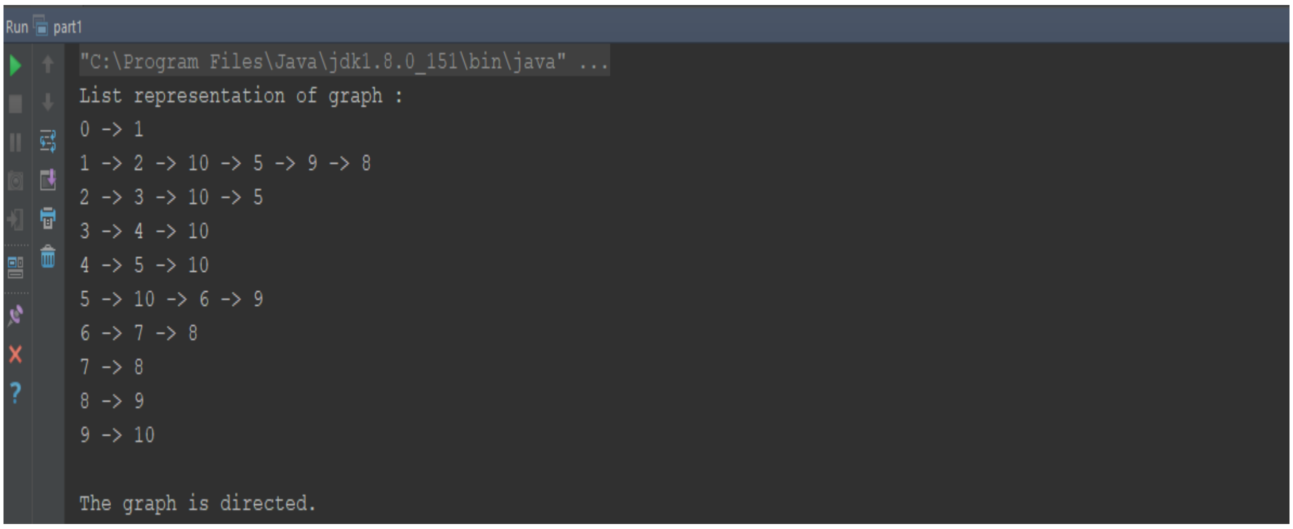


Result of plot_graph :



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
List representation of graph :
0 -> 1
1 -> 2 -> 10 -> 5 -> 9 -> 8
2 -> 3 -> 10 -> 5
3 -> 4 -> 10
4 -> 5 -> 10
5 -> 10 -> 6 -> 9
6 -> 7 -> 8
7 -> 8
8 -> 9
9 -> 10
```

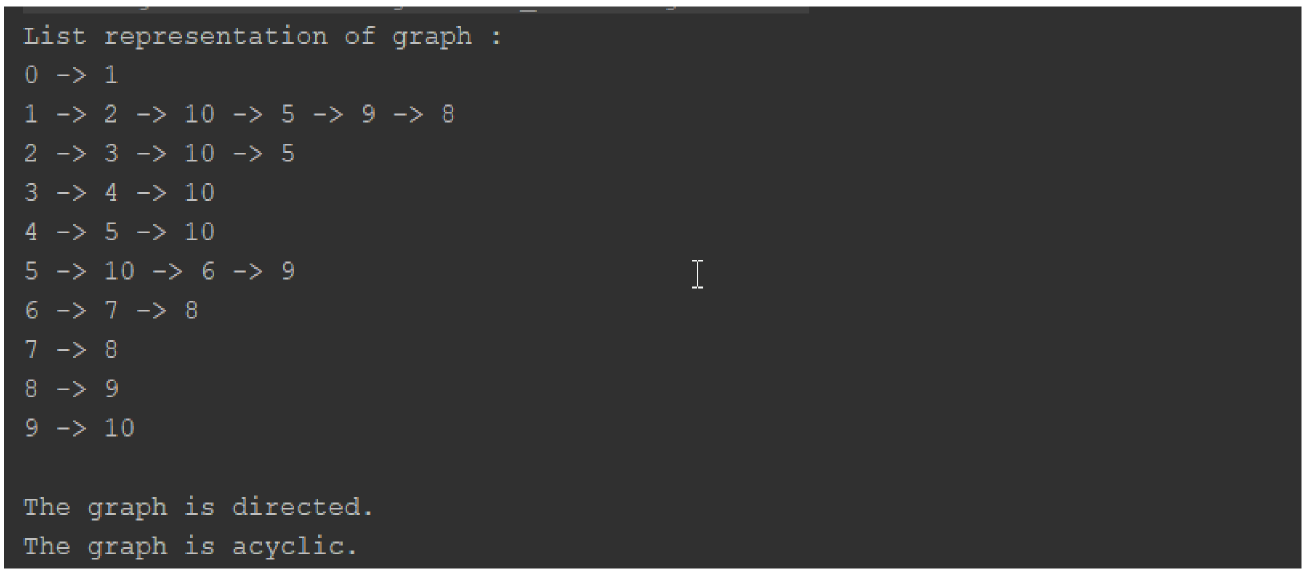
Result of is_undirected:



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
List representation of graph :
0 -> 1
1 -> 2 -> 10 -> 5 -> 9 -> 8
2 -> 3 -> 10 -> 5
3 -> 4 -> 10
4 -> 5 -> 10
5 -> 10 -> 6 -> 9
6 -> 7 -> 8
7 -> 8
8 -> 9
9 -> 10

The graph is directed.
```

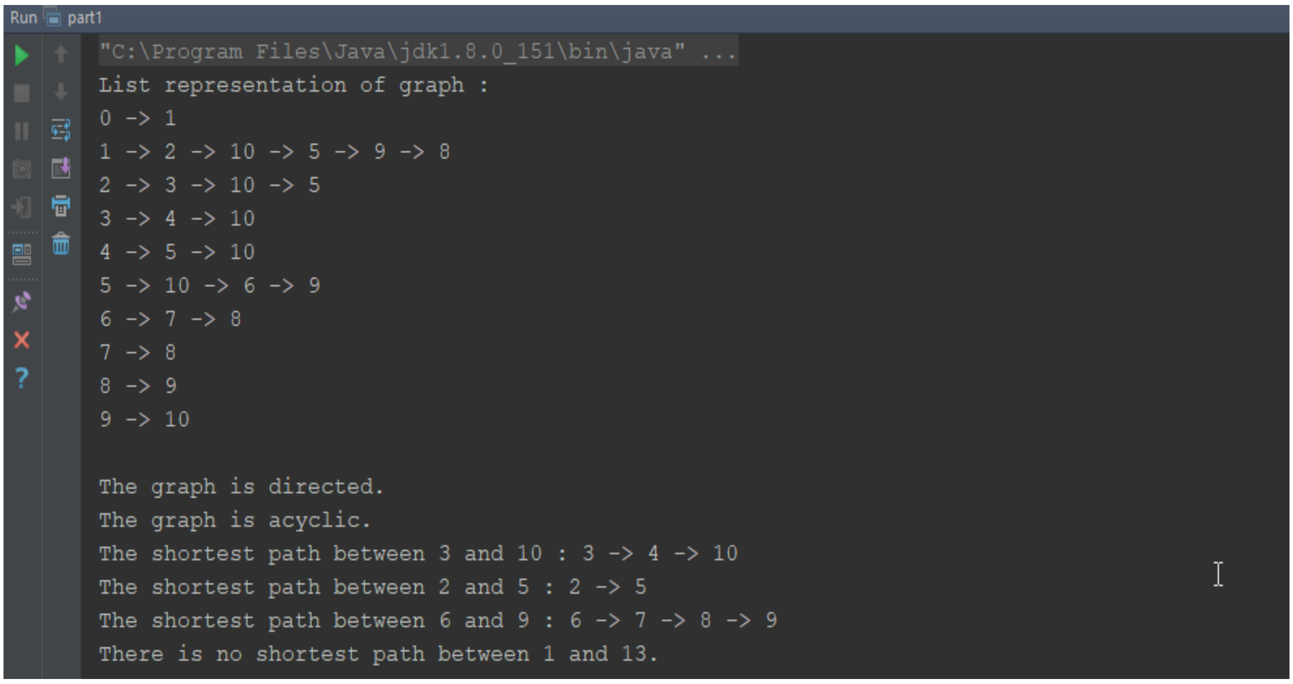
Result of is_acyclic_graph:



```
List representation of graph :
0 -> 1
1 -> 2 -> 10 -> 5 -> 9 -> 8
2 -> 3 -> 10 -> 5
3 -> 4 -> 10
4 -> 5 -> 10
5 -> 10 -> 6 -> 9
6 -> 7 -> 8
7 -> 8
8 -> 9
9 -> 10

The graph is directed.
The graph is acyclic.
```

Result Of shortest_path:


```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
List representation of graph :
0 -> 1
1 -> 2 -> 10 -> 5 -> 9 -> 8
2 -> 3 -> 10 -> 5
3 -> 4 -> 10
4 -> 5 -> 10
5 -> 10 -> 6 -> 9
6 -> 7 -> 8
7 -> 8
8 -> 9
9 -> 10

The graph is directed.
The graph is acyclic.
The shortest path between 3 and 10 : 3 -> 4 -> 10
The shortest path between 2 and 5 : 2 -> 5
The shortest path between 6 and 9 : 6 -> 7 -> 8 -> 9
There is no shortest path between 1 and 13.
```

2 Q2

This part about Question2 in HW7

2.1 Problem Solution Approach

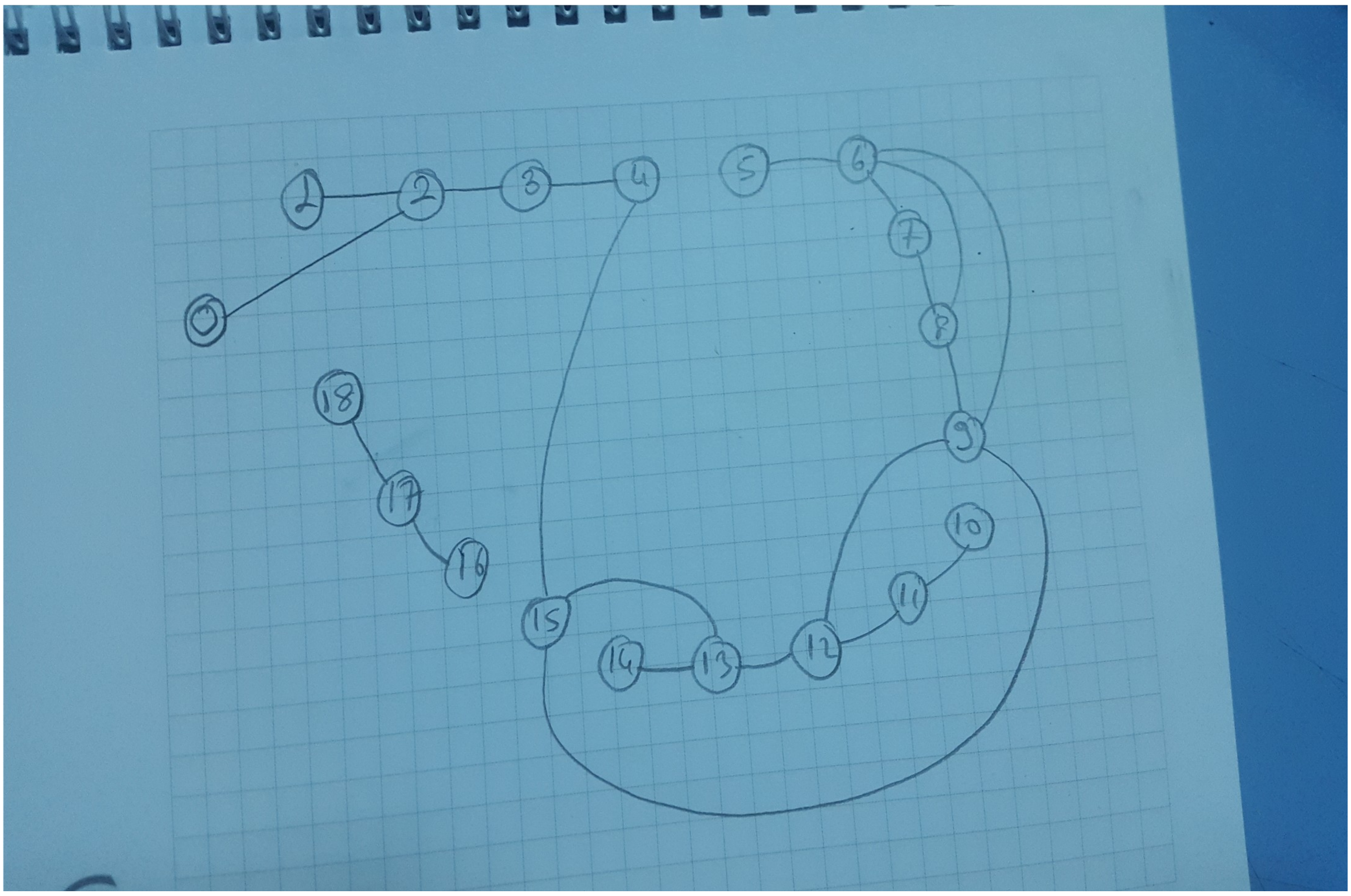
Graphs are created with random vertexes and,it is used Dijkstram Algorithm in order to find shortest path between two vertexes. It is just used ListGraph graph.

2.2 Test Cases

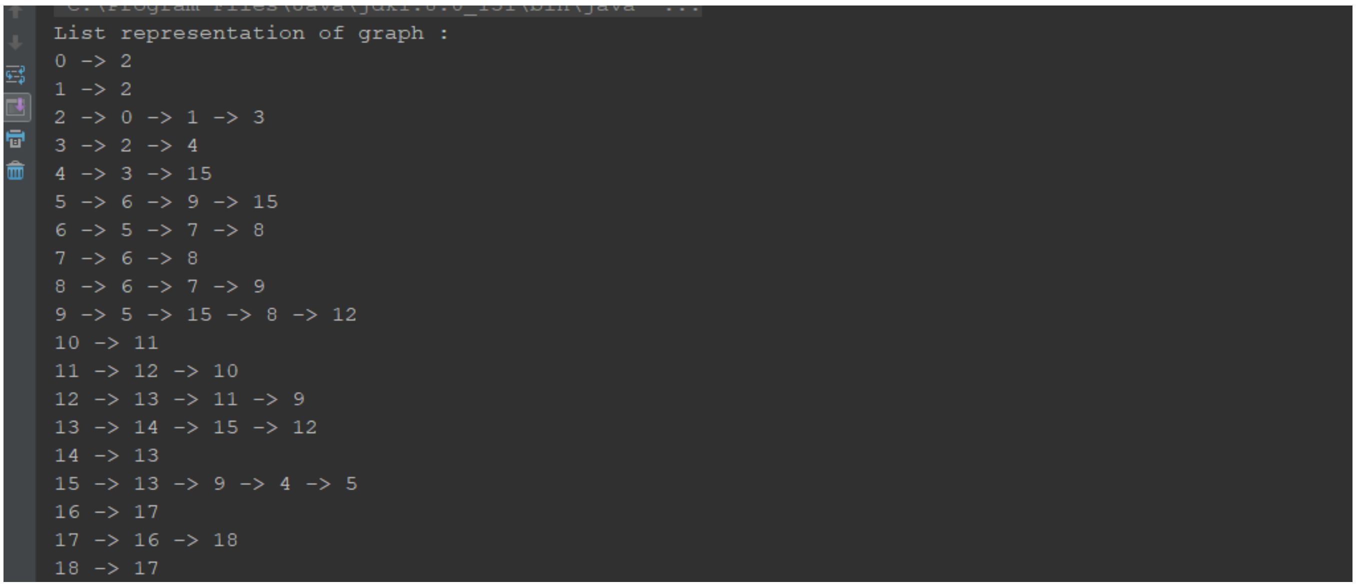
Order of vertex adding to undirected graph:

```
AbstractGraph graph = new ListGraph(19,false);
graph.insert(new Edge(0,2));
graph.insert(new Edge(1,2));
graph.insert(new Edge(2,3));
graph.insert(new Edge(3,4));
graph.insert(new Edge(14,13));
graph.insert(new Edge(15,13));
graph.insert(new Edge(13,12));
graph.insert(new Edge(12,11));
graph.insert(new Edge(11,10));
graph.insert(new Edge(5,6));
graph.insert(new Edge(5,9));
graph.insert(new Edge(15,9));
graph.insert(new Edge(6,7));
graph.insert(new Edge(6,8));
graph.insert(new Edge(7,8));
graph.insert(new Edge(8,9));
graph.insert(new Edge(9,12));
graph.insert(new Edge(4,15));
graph.insert(new Edge(15,5));
graph.insert(new Edge(16,17));
graph.insert(new Edge(17,18));
```

Graph representation :

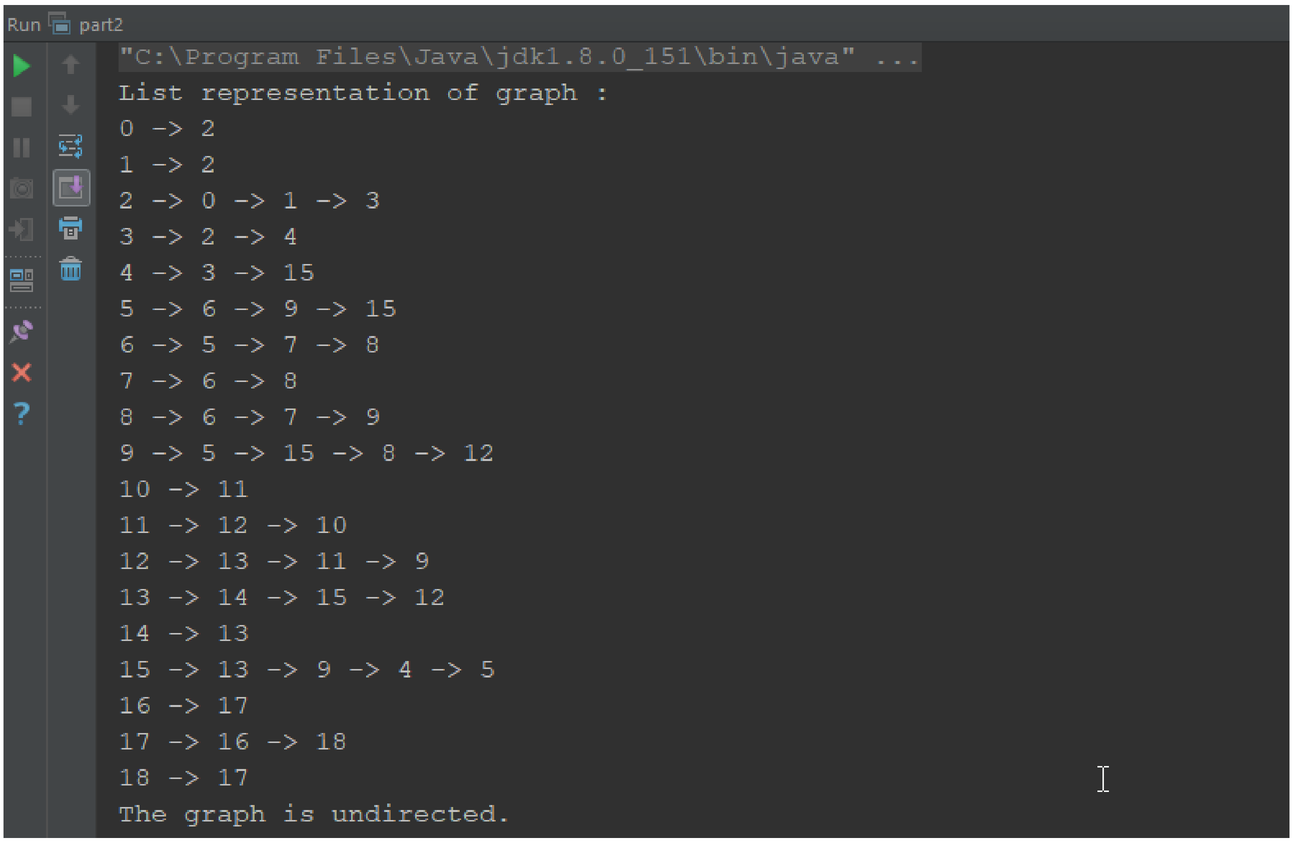


Result of plot_graph :



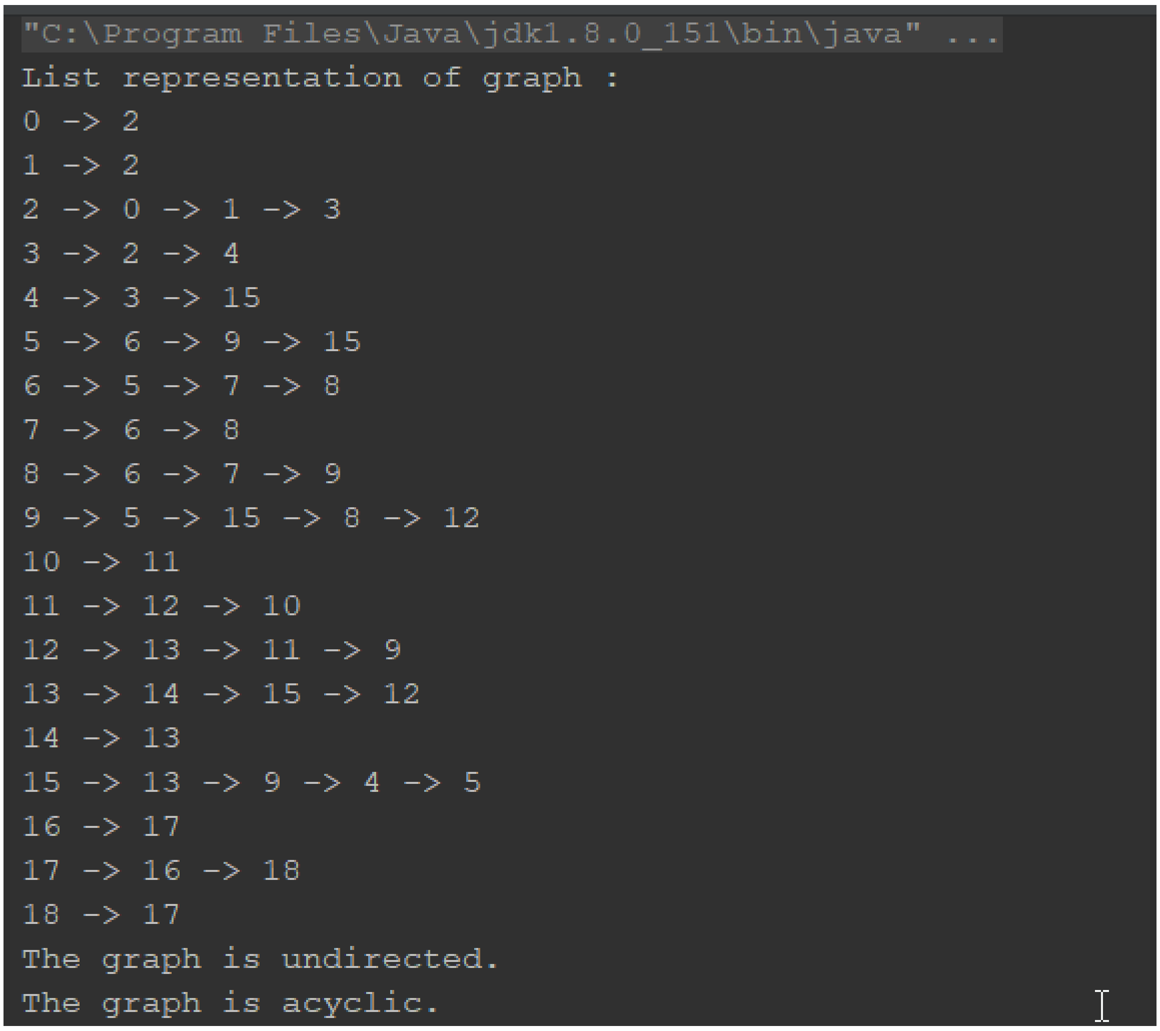
```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
List representation of graph :
0 -> 2
1 -> 2
2 -> 0 -> 1 -> 3
3 -> 2 -> 4
4 -> 3 -> 15
5 -> 6 -> 9 -> 15
6 -> 5 -> 7 -> 8
7 -> 6 -> 8
8 -> 6 -> 7 -> 9
9 -> 5 -> 15 -> 8 -> 12
10 -> 11
11 -> 12 -> 10
12 -> 13 -> 11 -> 9
13 -> 14 -> 15 -> 12
14 -> 13
15 -> 13 -> 9 -> 4 -> 5
16 -> 17
17 -> 16 -> 18
18 -> 17
```

Result of is_undirected:



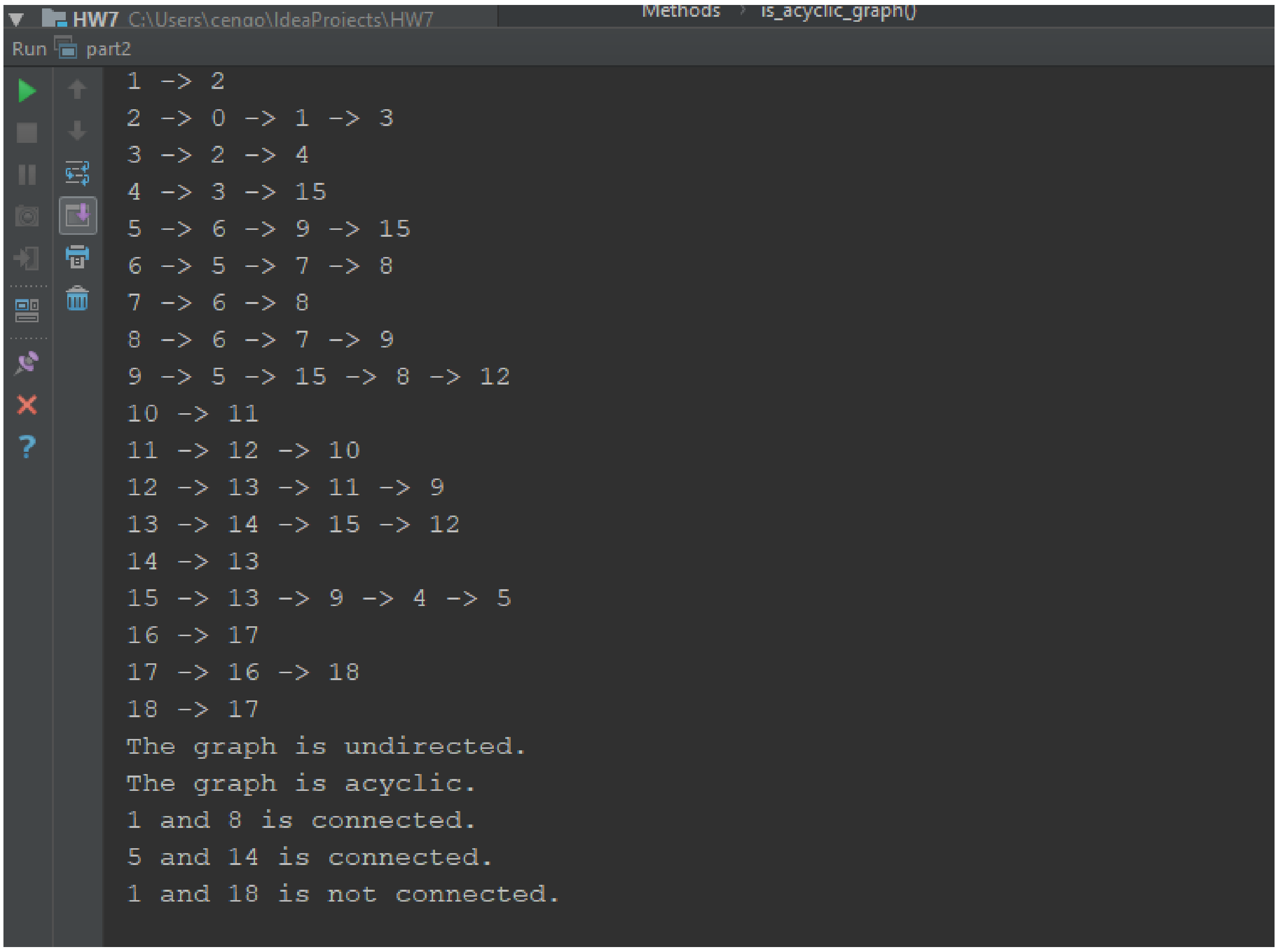
```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
List representation of graph :
0 -> 2
1 -> 2
2 -> 0 -> 1 -> 3
3 -> 2 -> 4
4 -> 3 -> 15
5 -> 6 -> 9 -> 15
6 -> 5 -> 7 -> 8
7 -> 6 -> 8
8 -> 6 -> 7 -> 9
9 -> 5 -> 15 -> 8 -> 12
10 -> 11
11 -> 12 -> 10
12 -> 13 -> 11 -> 9
13 -> 14 -> 15 -> 12
14 -> 13
15 -> 13 -> 9 -> 4 -> 5
16 -> 17
17 -> 16 -> 18
18 -> 17
The graph is undirected.
```

Result of is_acyclic_graph:



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
List representation of graph :
0 -> 2
1 -> 2
2 -> 0 -> 1 -> 3
3 -> 2 -> 4
4 -> 3 -> 15
5 -> 6 -> 9 -> 15
6 -> 5 -> 7 -> 8
7 -> 6 -> 8
8 -> 6 -> 7 -> 9
9 -> 5 -> 15 -> 8 -> 12
10 -> 11
11 -> 12 -> 10
12 -> 13 -> 11 -> 9
13 -> 14 -> 15 -> 12
14 -> 13
15 -> 13 -> 9 -> 4 -> 5
16 -> 17
17 -> 16 -> 18
18 -> 17
The graph is undirected.
The graph is acyclic.
```

Result of is_connected:


HW7 C:\Users\cengo\IdeaProjects\HW7
Methods › is_acyclic_graph()
Run part2

```
1 -> 2
2 -> 0 -> 1 -> 3
3 -> 2 -> 4
4 -> 3 -> 15
5 -> 6 -> 9 -> 15
6 -> 5 -> 7 -> 8
7 -> 6 -> 8
8 -> 6 -> 7 -> 9
9 -> 5 -> 15 -> 8 -> 12
10 -> 11
11 -> 12 -> 10
12 -> 13 -> 11 -> 9
13 -> 14 -> 15 -> 12
14 -> 13
15 -> 13 -> 9 -> 4 -> 5
16 -> 17
17 -> 16 -> 18
18 -> 17
The graph is undirected.
The graph is acyclic.
1 and 8 is connected.
5 and 14 is connected.
1 and 18 is not connected.
```

3 Q3

3.1 Problem Solution Approach

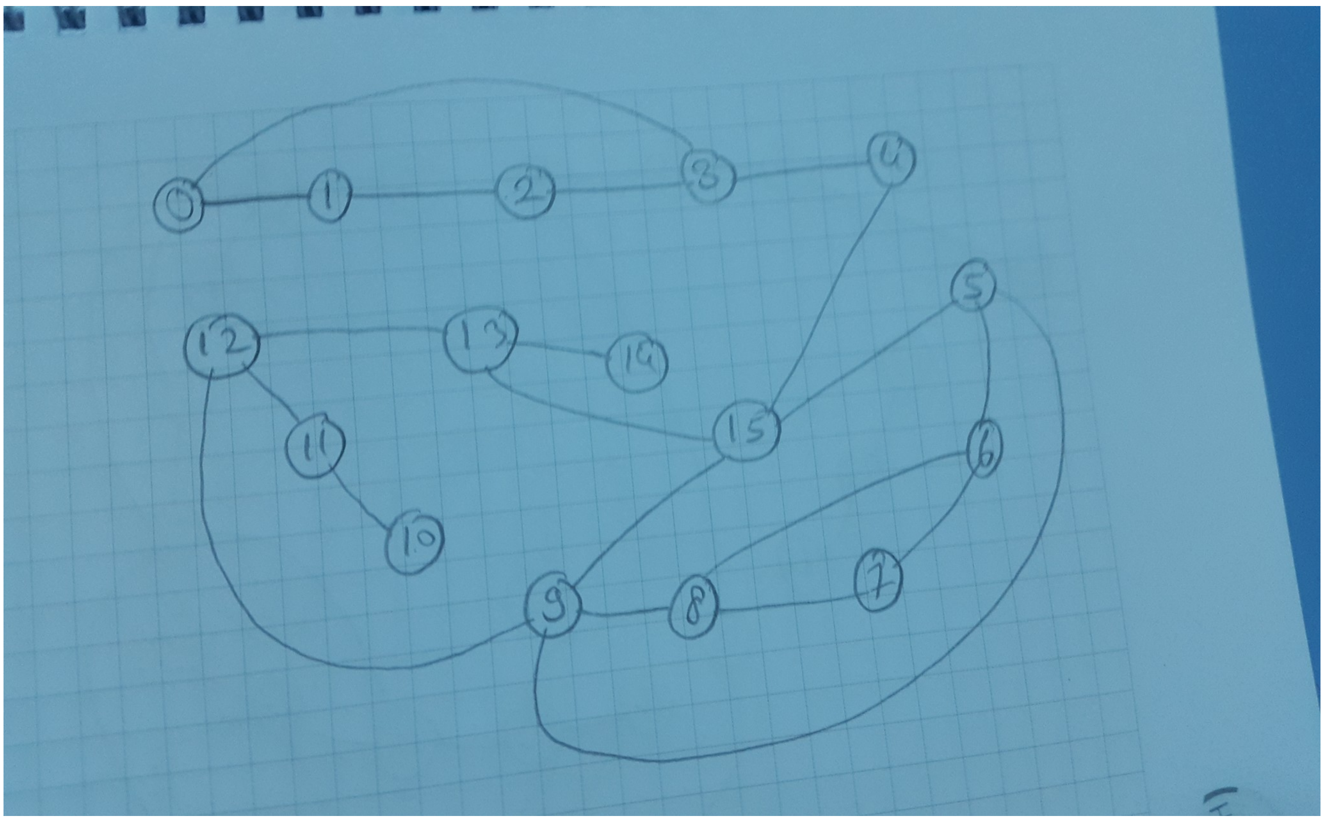
Graphs are created with random vertexes and,it is used Dijkstram Algorithm in order to find shortest path between two vertexes. It is just used ListGraph graph.

3.2 Test Cases

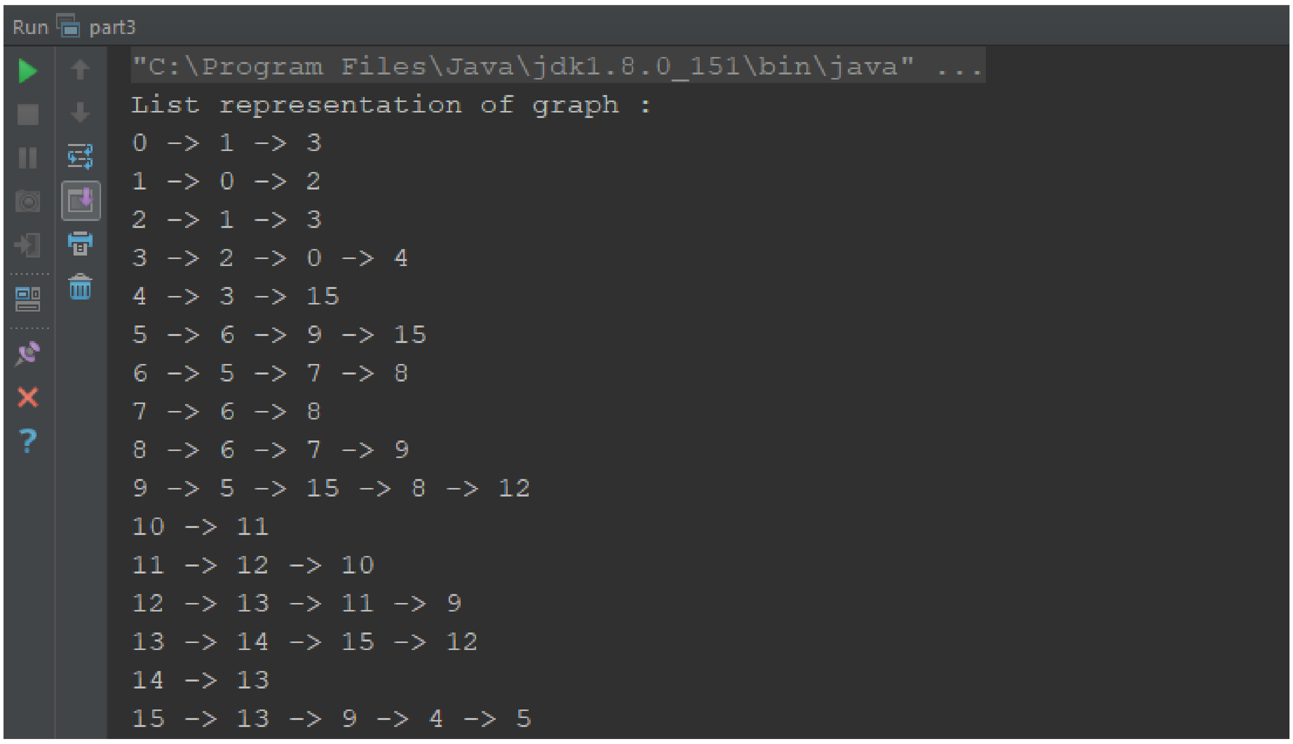
Order of vertex adding to undirected graph:

```
AbstractGraph graph = new ListGraph(16,false);
graph.insert(new Edge(0,1));
graph.insert(new Edge(1,2));
graph.insert(new Edge(2,3));
graph.insert(new Edge(3,0));
graph.insert(new Edge(3,4));
graph.insert(new Edge(14,13));
graph.insert(new Edge(15,13));
graph.insert(new Edge(13,12));
graph.insert(new Edge(12,11));
graph.insert(new Edge(11,10));
graph.insert(new Edge(5,6));
graph.insert(new Edge(5,9));
graph.insert(new Edge(15,9));
graph.insert(new Edge(6,7));
graph.insert(new Edge(6,8));
graph.insert(new Edge(7,8));
graph.insert(new Edge(8,9));
graph.insert(new Edge(9,12));
graph.insert(new Edge(4,15));
graph.insert(new Edge(15,5));
```

Graph representation :



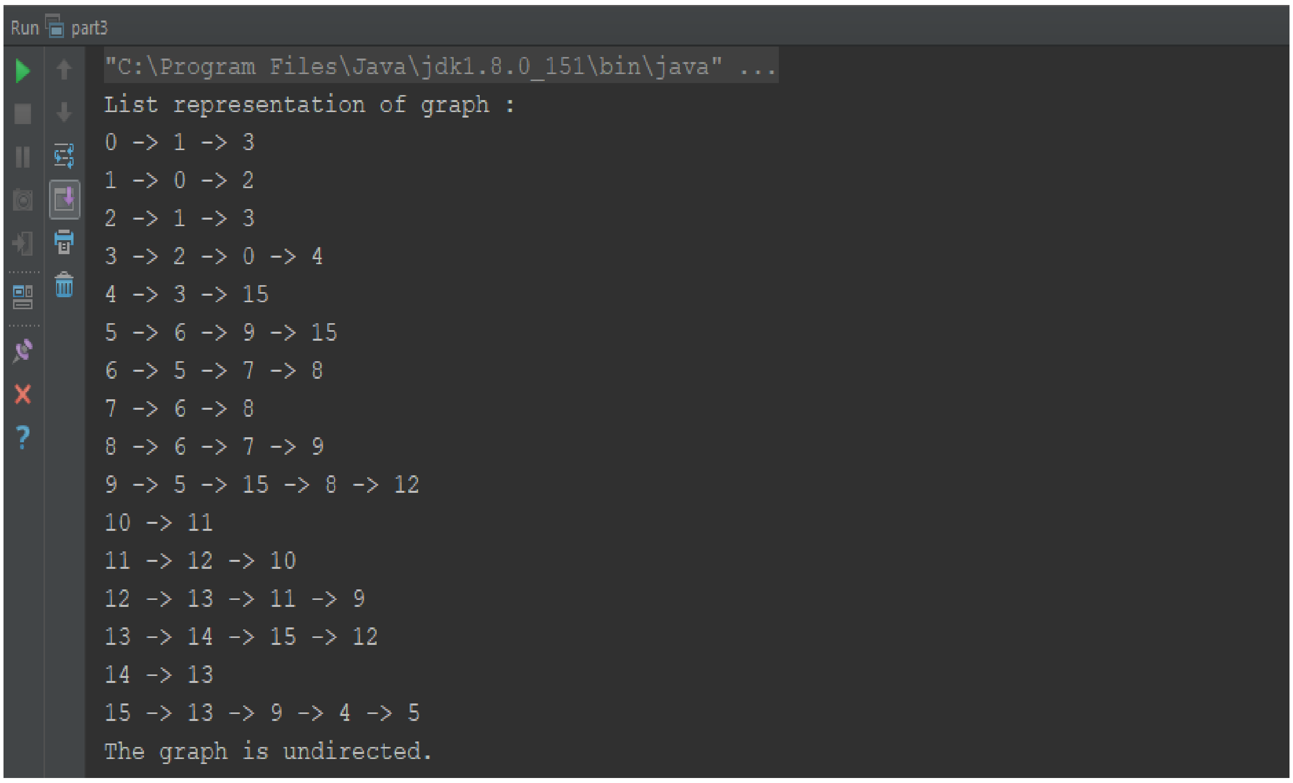
Result of plot_graph :



Run part3

```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
List representation of graph :
0 -> 1 -> 3
1 -> 0 -> 2
2 -> 1 -> 3
3 -> 2 -> 0 -> 4
4 -> 3 -> 15
5 -> 6 -> 9 -> 15
6 -> 5 -> 7 -> 8
7 -> 6 -> 8
8 -> 6 -> 7 -> 9
9 -> 5 -> 15 -> 8 -> 12
10 -> 11
11 -> 12 -> 10
12 -> 13 -> 11 -> 9
13 -> 14 -> 15 -> 12
14 -> 13
15 -> 13 -> 9 -> 4 -> 5
```

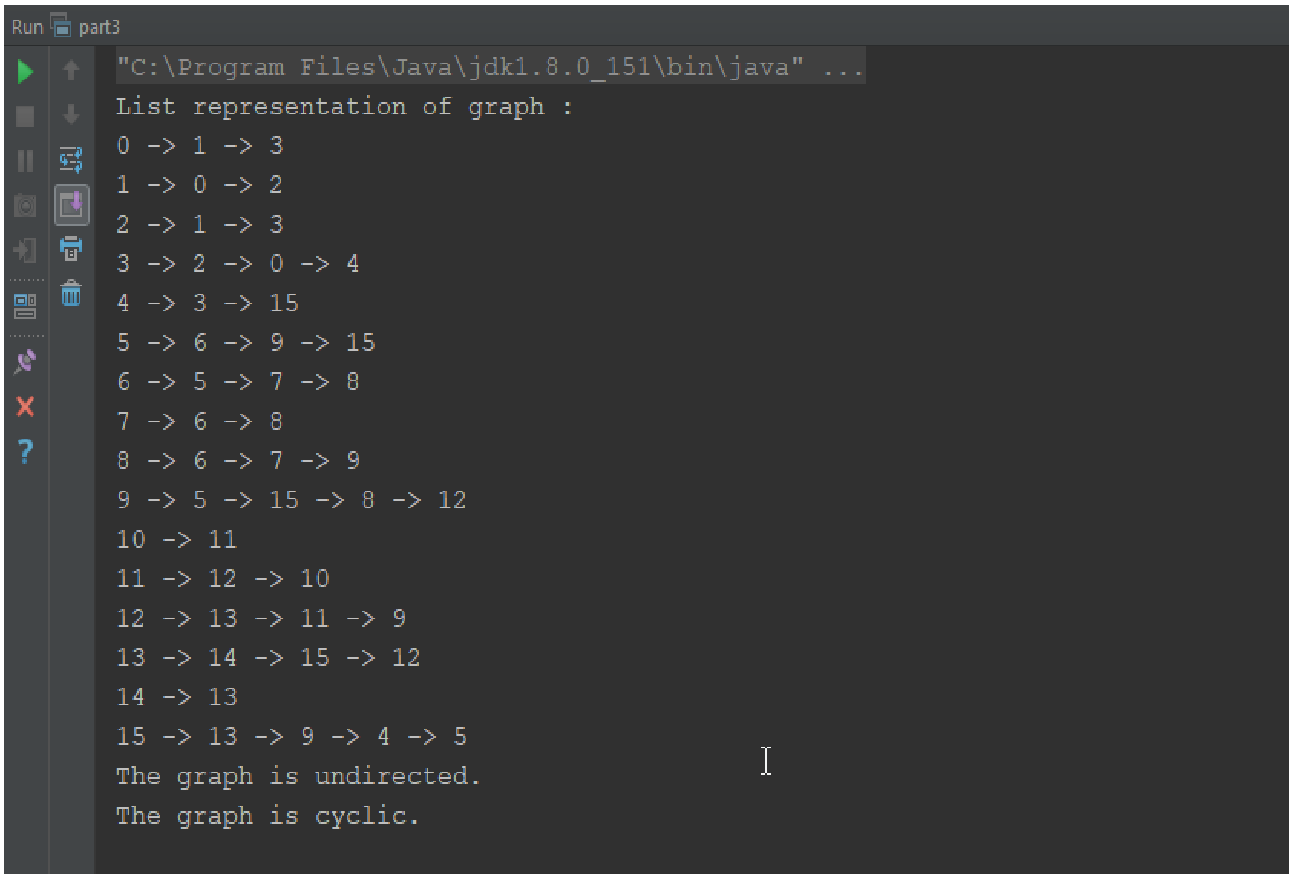
Result of is_undirected:



Run part3

```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
List representation of graph :
0 -> 1 -> 3
1 -> 0 -> 2
2 -> 1 -> 3
3 -> 2 -> 0 -> 4
4 -> 3 -> 15
5 -> 6 -> 9 -> 15
6 -> 5 -> 7 -> 8
7 -> 6 -> 8
8 -> 6 -> 7 -> 9
9 -> 5 -> 15 -> 8 -> 12
10 -> 11
11 -> 12 -> 10
12 -> 13 -> 11 -> 9
13 -> 14 -> 15 -> 12
14 -> 13
15 -> 13 -> 9 -> 4 -> 5
The graph is undirected.
```

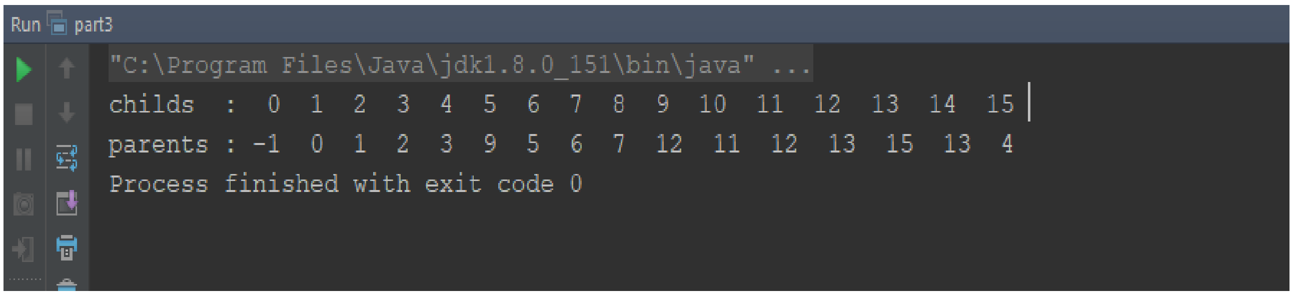
Result of is_acyclic_graph:



```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
List representation of graph :
0 -> 1 -> 3
1 -> 0 -> 2
2 -> 1 -> 3
3 -> 2 -> 0 -> 4
4 -> 3 -> 15
5 -> 6 -> 9 -> 15
6 -> 5 -> 7 -> 8
7 -> 6 -> 8
8 -> 6 -> 7 -> 9
9 -> 5 -> 15 -> 8 -> 12
10 -> 11
11 -> 12 -> 10
12 -> 13 -> 11 -> 9
13 -> 14 -> 15 -> 12
14 -> 13
15 -> 13 -> 9 -> 4 -> 5
The graph is undirected.
The graph is cyclic.
```

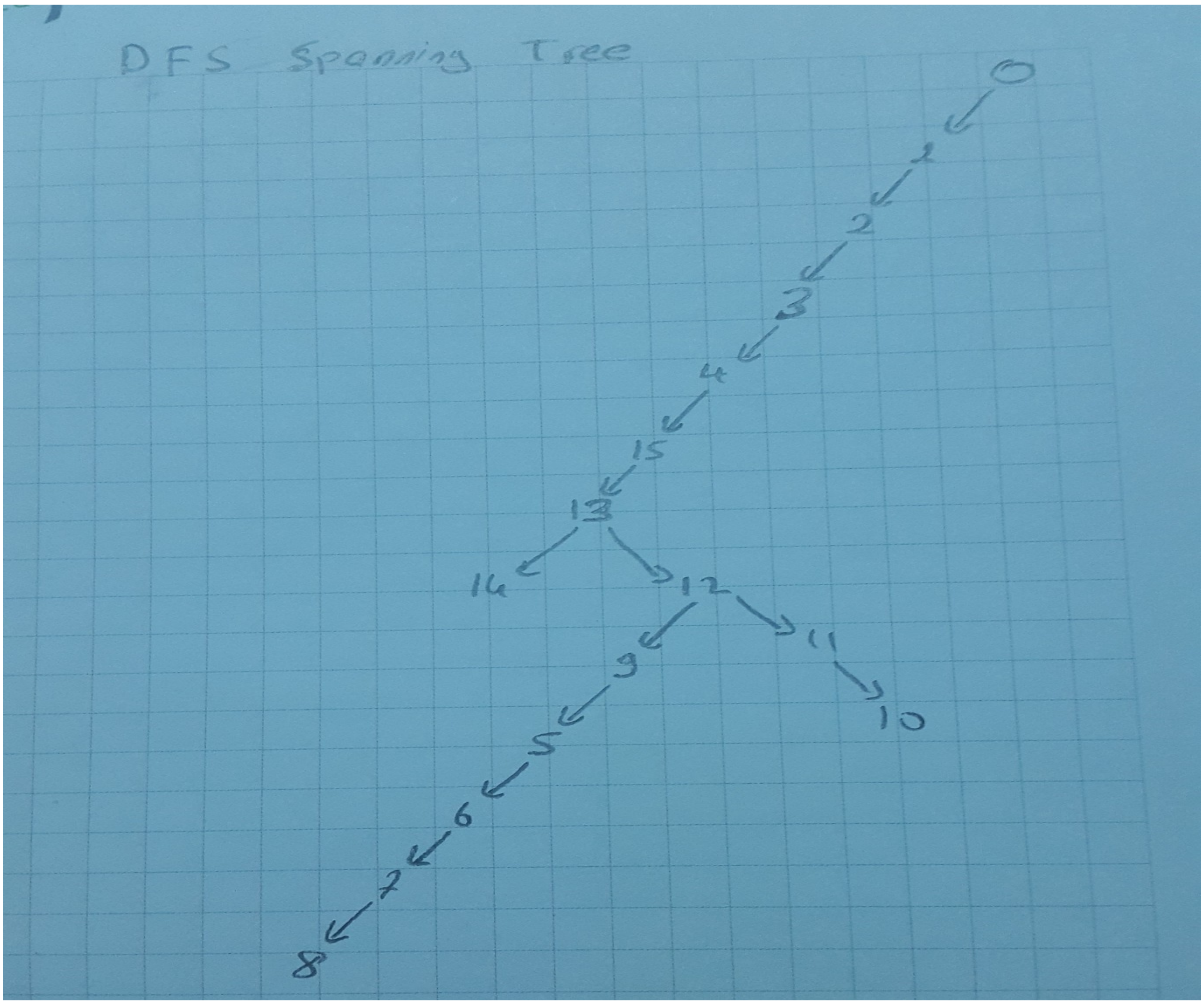
Result of DepthFirstSearch (Show that spanning tree):

parents and childs:



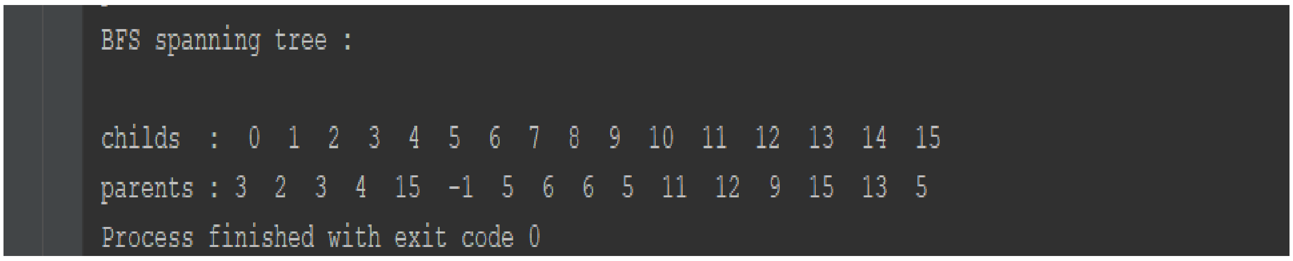
```
"C:\Program Files\Java\jdk1.8.0_151\bin\java" ...
childs  :  0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15
parents : -1  0  1  2  3  9  5  6  7  12  11  12  13  15  13  4
Process finished with exit code 0
```

Spaninig tree:



Result of BreathFirstSearch (Show that spanning tree):

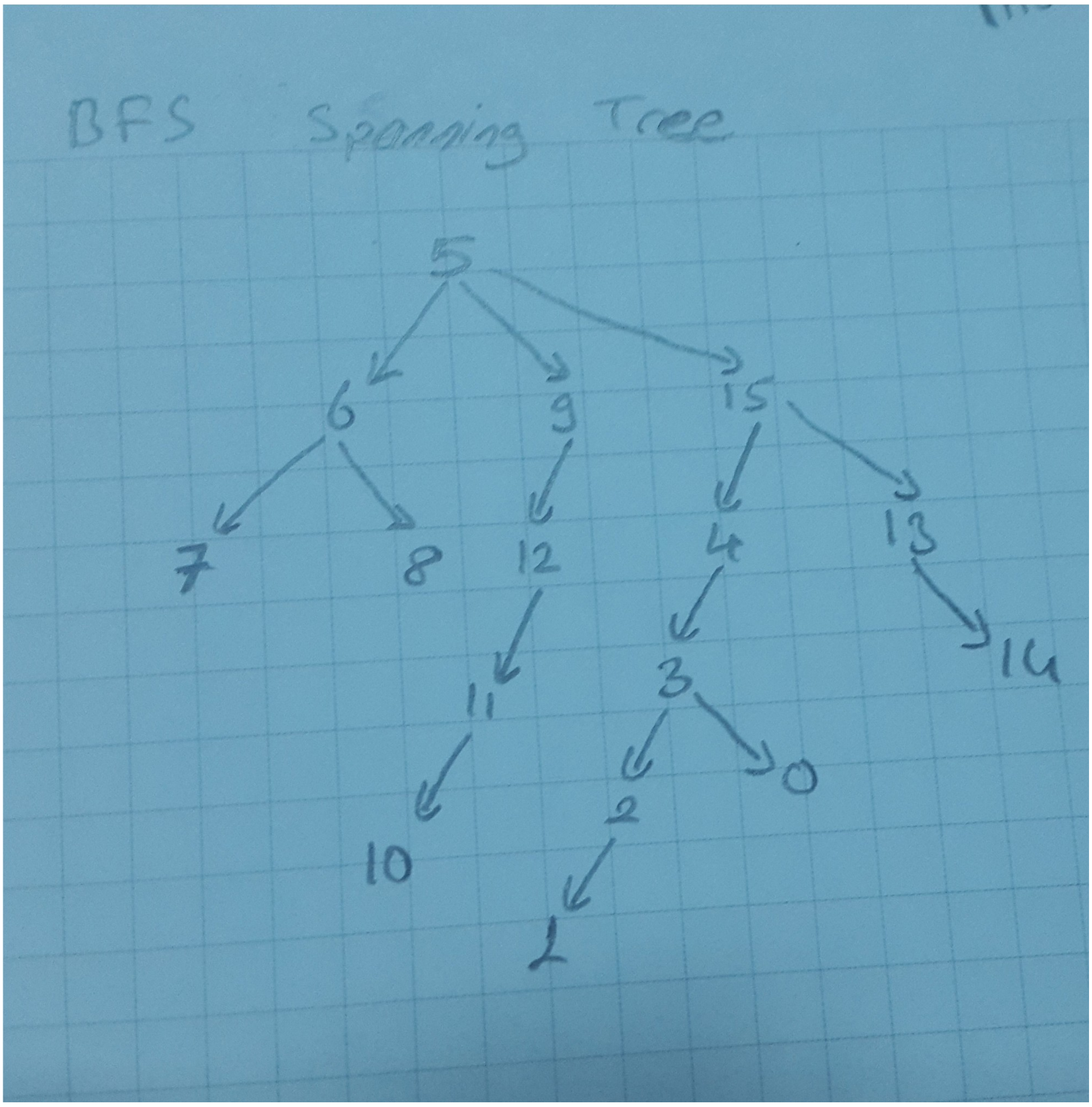
parents and childs:



```
BFS spanning tree :

childs  :  0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15
parents : 3  2  3  4  15  -1  5  6  6  5  11  12  9  15  13  5
Process finished with exit code 0
```

Spaninig tree:



$$\begin{tikzcd}
 & & & 5 \arrow[dll] \arrow[d] \arrow[drr, bend left] & & & & \\
 & 6 \arrow[dl] \arrow[dr] & & 9 \arrow[d] & & 15 \arrow[dl] \arrow[dr] & & \\
7 & & 8 & 12 \arrow[d] & 4 \arrow[d] & & 13 \arrow[dr] & \\
 & & & 11 \arrow[dl] & 3 \arrow[d] \arrow[dr] & & & 14 \\
 & & 10 & & 2 \arrow[dl] & 0 & & \\
 & & & 1 & & & &
\end{tikzcd}$$

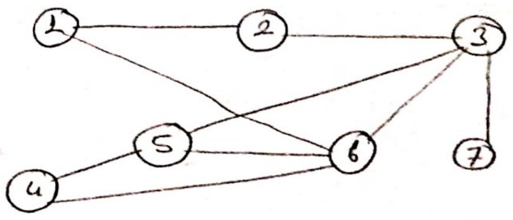
4 Q4

Answer 4: Differences Between DFS And BFS

BFS	DFS:	According to:
- use queue.	- use stack.	- Used data structure.
- is similar to level-order traverse	- is similar to pre-order traverse	- Similarity
- BFS is smaller than DFS	- DFS is more faster than BFS	- Speed
- BFS always finds a shortest path the start Vertex to any other for unweighted graphs	- Depth first search may not finds a shortest path	- Advantage
• Finding the shortest path. • Testing for bipartiteness. • In spanning tree	• Topolosical sorting. • Finding bridges of a graph • Maze generation • Finding strongly connected components	- Applications

	1	2	3	4	5	6	7
1	1	1	0	0	0	1	0
2	1	1	1	0	0	0	0
3	0	1	1	0	1	1	1
4	0	0	0	1	1	1	0
5	0	0	1	1	1	1	0
6	1	0	1	1	1	1	0
7	0	0	1	0	0	0	1

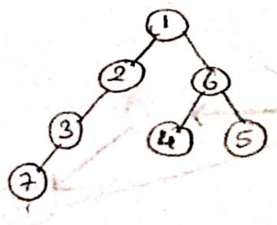
$\xrightarrow[\text{representation}]{\text{graph}}$



b) BFS parent array

Vertex:	0	1	2	3	4	5	6	7
parent:	-1	-1	1	2	6	6	1	3

DFS Tree:



a) DFS parent array

Vertex:	0	1	2	3	4	5	6	7
Parent:	-1	-1	1	2	5	3	4	3

DFS Tree: