Name: IBRAHIM ALSHAYEA, ID: 202176470

- Task1:
  addEdge(int i, int j): This method connects vertex i to vertex j and vice versa, effectively adding an edge between them in an undirected graph.

- removeEdge(int i, int j): This method disconnects vertex i from vertex j and vice versa, effectively removing the edge between them in an undirected graph.

- isEdge(int i, int j): This method checks if there's a connection between vertex i and vertex j in the graph. It returns true if there's an edge between them, false otherwise, in an undirected graph.

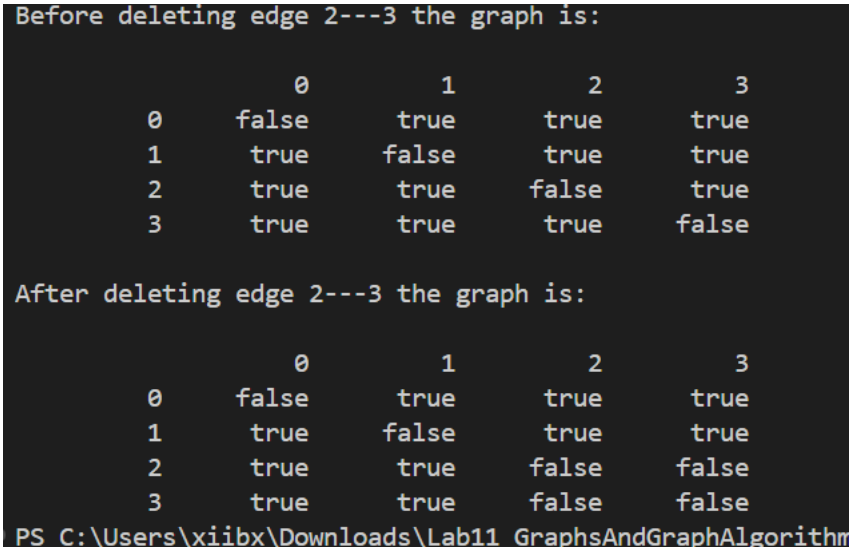
Methods Code:

```
public void addEdge(int i, int j) {
      // Add edge from i to j
      adjacencyMatrix[i][j] = true;
      // Since it's an undirected graph, also add edge from j to i
      adjacencyMatrix[j][i] = true;
  }


public void removeEdge(int i, int j) {
      // Remove edge from i to j
      adjacencyMatrix[i][j] = false;
      // Also remove edge from j to i
      adjacencyMatrix[j][i] = false;
}

public boolean isEdge(int i, int j) {
      // Check if there is an edge from i to j and from j to i
      return adjacencyMatrix[i][j] && adjacencyMatrix[j][i];
}
```

Task1 Output:



Task2:

- The modification to the Graph class made it simpler by using string labels instead of integer identifiers for vertices. The methods addDirectedEdge and addUndirectedEdge now work with string labels, and displayGraph prints these labels for better comprehension

  The code:

```
3   class Graph {
8       Graph(int numVertices, String[] labels) {
9           this.labels = labels;
10          this.numVertices = numVertices;
11          adjacencyList = new LinkedList[numVertices];
12
13          for (int i = 0; i < adjacencyList.length; i++)
14              adjacencyList[i] = new LinkedList<String>();
15      }
16
17      // To add a directed edge to graph
18      void addDirectedEdge(int v, int w) {
19          adjacencyList[v].add(labels[w]); // Add label of w to v's list.
20      }
21
22      // To add undirected edge to graph
23      void addUndirectedEdge(int v, int w) {
24          adjacencyList[v].add(labels[w]);
25          adjacencyList[w].add(labels[v]);
26      }
27
28      void displayGraph() {
29          for (int i = 0; i < adjacencyList.length; i++) {
30              System.out.print(labels[i] + " ----> ");
31              for (String vertex : adjacencyList[i]) {
32                  System.out.print(vertex + " ");
33              }
34              System.out.println();
35          }
36          System.out.println();
37      }
```

The output:

```
The directed graph is:
A ----> C D
B ----> A E
C ----> B
D ---->
E ---->

The undirected graph is:
A ----> B C D
B ----> A C E
C ----> A B
D ----> A
E ----> B

PS C:\Users\xiibx\Downloads\Lab11_Graphs
```

Task3:

- The isReachable method in the Graph class recursively checks if a destination vertex is reachable from a given source vertex by performing Depth-First Search traversal. It marks vertices as visited to prevent revisiting and returns true if the destination vertex is found reachable, otherwise false.

The method Code:

```java
public boolean isReachable(int src, int dest) {
    boolean[] visited = new boolean[numVertices];
    return isReachable(src, dest, visited);
}

private boolean isReachable(int src, int dest, boolean[] visited) {
    if (src == dest) {
        return true;
    }

    visited[src] = true;
    for (int neighbor : adjList.get(src)) {
        if (!visited[neighbor]) {
            if (isReachable(neighbor, dest, visited)) {
                return true;
            }
        }
    }

    return false;
}
```

Task3 Output:

```
\jdk-17.0.7.7-hotspot\bin\java.exe  -XX:+ShowCodeDeta
2b61ad4667\redhat.java\jdt_ws\Task03_19ec46b2\bin' 'Gr
Enter the source vertex [0 - 7]: 2
Enter the destination vertex [0 - 7]: 3
No path exists between 2 and 3
```

```
Enter the source vertex [0 - 7]: 1
Enter the destination vertex [0 - 7]: 7
Path exists from vertex 1 to vertex 7
PS C:\Users\xiibx\Downloads\Lab11_GraphsAndGra
```