Objective(s):

a. To practice various graph's representation
b. To practice depth-first-search algorithm

Task 1: GraphList_XXYYYY.java, GraphMap_XXYYYY.java and GraphN_XXYYYY.java

- GraphList_XXYYYY keeps adjacency lists of each node. A list contains neighbors id.
- GraphMap_XXYYYY keep track of list of neighbors id for each node.
- GraphN_XXYYYY keep a list of nodes. Each node keeps track of its neighbors id.

Complete the code for each class.

Note that, common graph traversal is depth-first-search because breadth-first-search could store up to n nodes during its process.

```
static void q1_1() {
    System.out.println("Graph List");
    GraphList_XXYYY graphL = new GraphList_XXYYY(5);
    graphL.addEdge(0, 1);
    graphL.addEdge(0, 4);
    graphL.addEdge(1, 2);
    graphL.addEdge(1, 3);
    graphL.printGraph();
        // 0 -> 1 4
        // 1 -> 0 2 3
        // 2 -> 1
        // 3 -> 1
        // 4 -> 0
    System.out.println(graphL.hasEdge(1, 3));   // true
    System.out.println(graphL.hasPath(4, 2) );  // true
    System.out.println(graphL.neighborsOf(2) );     // [1]

    System.out.println("DFS:");
    graphL.dfs(0); // 0 1 2 3 4
}
```

```
static void q1_2() {
    System.out.println("Graph Map");
    GraphMap_XXYYY graphM = new GraphMap_XXYYY();
    graphM.addEdge(1, 2);
    graphM.addEdge(1, 3);
    graphM.addEdge(2, 4);
    graphM.printGraph();
        // 1 -> [2, 3]
        // 2 -> [1, 4]
        // 3 -> [1]
        // 4 -> [2]
    System.out.println(graphM.hasEdge(1, 4));   // false
    System.out.println(graphM.hasPath(4, 2) );  // true
    graphM.addVertex(5);    // add isolated node 5 to the graph
    System.out.println(graphM.hasPath(5, 2) );  // false
    System.out.println("***"+ graphM.neighborsOf(2) ); // [1, 4]

    System.out.println("DFS:");
    graphM.dfsAll();
    // 1 2 4 3
    // 5
}
```

```
static void q1_3() {
    System.out.println("Graph Node");
    GraphNode_XXYYY graphN = new GraphNode_XXYYY();
    Node a = graphN.addNode(1);
    Node b = graphN.addNode(2);
    Node c = graphN.addNode(3);

    graphN.addEdge(a, b);
    graphN.addEdge(b, c);
    graphN.addEdge(a, c);

    graphN.printGraph();
        // 1 -> 2 3
        // 2 -> 1 3
        // 3 -> 2 1
    System.out.println(graphN.hasEdge(1, 3));   // true
    System.out.println(graphN.hasPath(3, 2) );  // true
    System.out.println("***"+ graphN.neighborsOf(2) ); // [1, 3]

    System.out.println("DFS:");
    graphN.dfs(a);  // 1 2 3
}
```

Task 2: DFS on adjacency matrix

```
static void q2() {

  int[][] thisGraph = {{0,3,inf,inf,inf},
                       {inf,0,1,inf,inf},
                       {inf,inf,0,4,inf},
                       {inf,inf,inf,0,5},
                       {inf,inf,inf,inf,0}};
  System.out.println("computing dfs");
  q2_dfs(thisGraph);
            // Edge 0, 1
            // Edge 1, 2
            // Edge 2, 3
            // Edge 3, 4

}

private static void q2_dfs(int[][] thisGraph) {
  ArrayList<Integer> stack = new ArrayList<>();
  ArrayList<Integer> visited = new ArrayList<>();

  stack.add(0);
  while (!stack.isEmpty()) {
    int parent = 0; /* your code 9 */
    visited.add(parent);
    for (int x = 0; x < thisGraph.length; x++) {
      if (0 < thisGraph[parent][x] && thisGraph[parent][x] < inf /* your code 10 */) {
            stack.add(x);
            System.out.println("Edge " + parent + ", " + x);
      }
    }

  } // while
}
```

Task 3: detect cycle in graph

complete a method boolean hasCycle() that returns true if the graph contains at least one cycle.

```
static void q3_2() {
    GraphMap graphM = new GraphMap();
    graphM.addEdge(1, 2);
    graphM.addEdge(1, 4);
    graphM.addEdge(2, 3);
    graphM.addEdge(4,3);
        // 1 - 2
        // |   |
        // 4 - 3
    System.out.println(graphM.hasCycle());  // true
}

// GraphMap_XXYYYY
public boolean hasCycle() {
    Set<Integer> visited = new HashSet<>();

    for (int v : graph.keySet()) {
        if (!visited.contains(v)) {
            if (hasCycleDFS(v, -1, visited)) {
                return true;
            }
        }
    }
    return false;
}

private boolean hasCycleDFS(int current, int parent, Set<Integer>
visited) {
    visited.add(current);

    for (int neighbor : graph.get(current)) {
        if (!visited.contains(neighbor)) {
            // explore deeper
            if (true /* your code 10 */) {
                return true;
            }
        } else if (neighbor != parent) {
            // found a back edge -> cycle
            return false /* your code 11 */;
        }
    }
    return false;
}
```

Submission: Your **GraphList_XXYYYY.java, GraphMap_XXYYYY.java, GraphNode_XXYYY.java** and **Lab12a_Graph1_XXYYYY.java** where. XX are the first two digit of your student id and YYYY are the last four.

Due date: TBA