Lab 9
Graph

Quang D. C.
dungcamquang@tdtu.edu.vn

November 13, 2022

Note

In this lab, we will learn how to represent a graph on the computer:

- Adjacency Matrix
- Adjacency List
- Edge List

And two ways to traverse a graph:

- Breadth First Search (BFS)
- Depth First Search (DFS)

Part I
Classwork

In this part, lecturer will:

- Summarize the theory related to this lab.
- Instruct the lesson in this lab to the students.
- Explain the sample implementations.

Responsibility of the students in this part:

- Students practice sample exercises with solutions.
- During these part, students may ask any question that they don't understand or make mistakes. Lecturers can guide students, or do general guidance for the whole class if the errors are common.

1. What is Graph?

Graph is the data structure that included vertices and edges. In other words, graph is a set of vertices where some pairs of vertices are connected by edges.

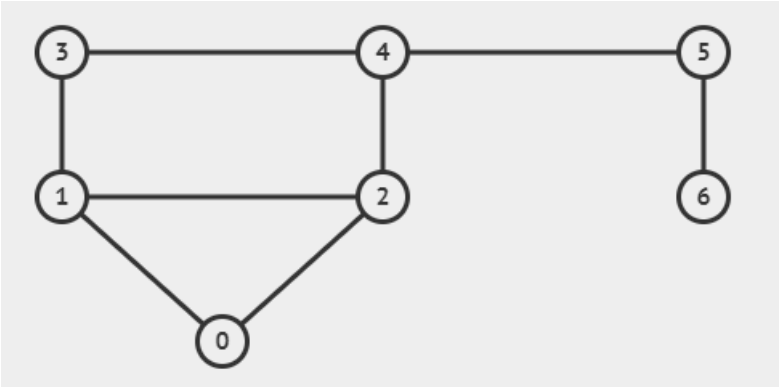


Figure 1: Graph

We have three options to represent a graph:

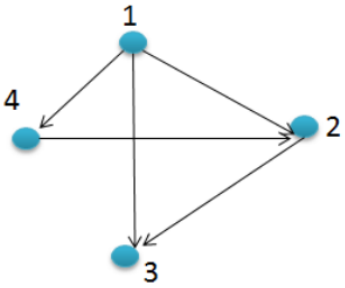
- Adjacency Matrix
- Adjacency List
- Edge List

2. Representations of Graph

2.1. Adjacency Matrix (AM)

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.

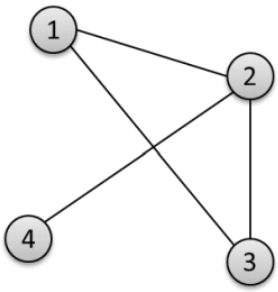
Let the 2D array be $adj[][]$, a slot $adj[i][j] = 1$ indicates that there is an edge from vertex i to vertex j, otherwise $adj[i][j]$ contains 0.



	1	2	3	4
1	0	1	1	1
2	0	0	1	0
3	0	0	0	0
4	0	1	0	0

Figure 2: Directed Graph

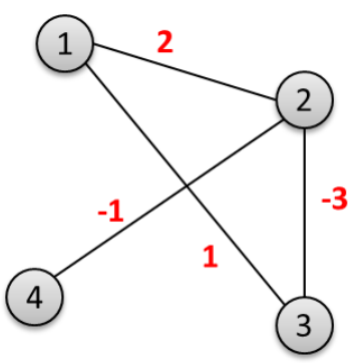
Adjacency matrix for undirected graph is always symmetric.



	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	0
4	0	1	0	0

Figure 3: Undirected Graph

Adjacency Matrix is also used to represent weighted graphs. If $adj[i][j] = w$, then there is an edge from vertex i to vertex j with weight w.



	1	2	3	4
1	0	2	1	0
2	2	0	-3	-1
3	1	-3	0	0
4	0	-1	0	0

Figure 4: Weighted Graph

You can create an AdjacencyMatrix class and define the code like this:

```
public class AdjacencyMatrix{
    private int[][] adj;
    private final int NUMBER_OF_VERTICES;

    public AdjacencyMatrix(int vertices){
        NUMBER_OF_VERTICES = vertices;
        adj = new int[NUMBER_OF_VERTICES][NUMBER_OF_VERTICES];
    }

    public void addEdge(int vertexSource, int vertexDestination,
    int weight){
        try {
            adj[vertexSource][vertexDestination] = weight;
            adj[vertexDestination][vertexSource] = weight;
        } catch (ArrayIndexOutOfBoundsException indexBounce){
            System.out.println("The vertex is invalid");
        }
    }

    public void printGraph(){
        for(int i = 0; i < NUMBER_OF_VERTICES; i++){
            for(int j = 0;j < NUMBER_OF_VERTICES; j++){
                System.out.print(adj[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

2.2. Adjacency List (AL)

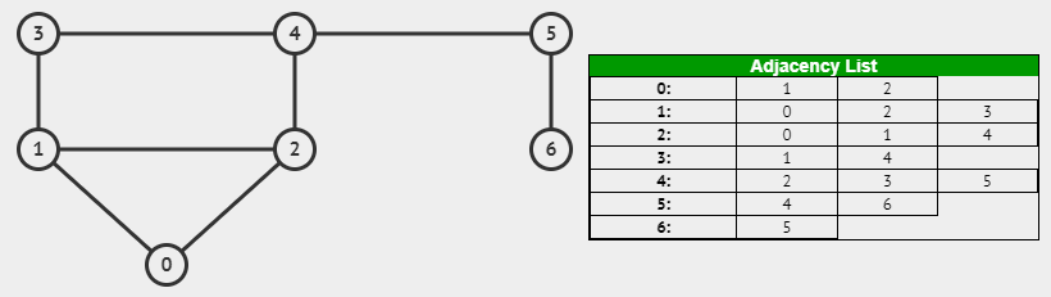


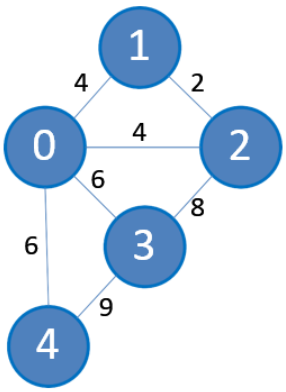
Figure 5: Adjacency List

In Java, there are many collections that you can use to implement Adjacency List like Vector, LinkedList, ArrayList, Map, ... Remember **import java.util.*** if you want to use Java collections.

We use LinkedList to implement an AdjacencyList class (this code for the graph without weights. If you want to handle weights, you can define a new class called IntegerPair to store two Integer as the attributes):

```
import java.util.*;

public class AdjacencyList{
    private int V;   // No. of vertices
    private ArrayList<LinkedList<Integer>> adj;

    public AdjacencyList(int v)
    {
        V = v;
        adj = new ArrayList<LinkedList<Integer>>();
        for (int i=0; i < v; ++i)
            adj.new LinkedList<Integer>());
    }

    public void addEdge(int u, int v)
    {
        adj.get(u).add(v);
    }

    public void printGraph(){
        for(int i = 0; i < V; i++){
            System.out.print("Vertex " + i + ": ");
            System.out.print("head");
            for(Integer v: adj.get(i)){
                System.out.print("->" + v);
            }
            System.out.println();
        }
    }
}
```

2.3. Edge List (EL)



i	w	u	v
0	2	1	2
1	4	0	1
2	4	0	2
3	6	0	3
4	6	0	4
5	8	2	3
6	9	3	4

Figure 6: Edge List

There are many data structures to implement Edge List, this code use Vector and class IntegerTriple, which is defined by user:

```
class IntegerTriple{
    private Integer weight;
    private Integer source;
    private Integer dest;

    public IntegerTriple(Integer w, Integer s, Integer d){
        weight = w;
        source = s;
        dest = d;
    }

    public Integer getWeight(){
        return weight;
    }

    public Integer getSource(){
        return source;
    }

    public Integer getDest(){
        return dest;
    }

    @Override
    public String toString(){
        return weight + " " + source + " " + dest;
    }
}

public class EdgeList{
    private Vector<IntegerTriple> edges;

    public EdgeList(){
        edges = new Vector<IntegerTriple>();
    }
```

```
36
37      public void addEdge(int w, int u, int v){
38          edges.add(new IntegerTriple(w,u,v));
39      }
40
41      public void printGraph(){
42          for(int i = 0; i < edges.size(); i++){
43              System.out.println(edges.get(i));
44          }
45      }
46  }
```

3. Graph traversal algorithms

3.1. Breadth First Search (BFS)

Review the BFS algorithm that you learned in your theory class. Mapping it to the following code (this code is implemented for Adjacency Matrix):

```
1  public void BFS(int s){
2      boolean visited[] = new boolean[NUMBER_OF_VERTICES];
3
4      Queue<Integer> queue = new LinkedList<Integer>();
5
6      visited[s] = true;
7      queue.offer(s);
8
9      while(!queue.isEmpty()){
10         int x = queue.poll();
11         System.out.print(x + " ");
12
13         for(int i = 0; i < NUMBER_OF_VERTICES; i++){
14             if(adj[x][i] != 0 && visited[i] == false){
15                 queue.offer(i);
16                 visited[i] = true;
17             }
18         }
19     }
20 }
```

3.2. Depth First Search (DFS)

Review the recursion DFS algorithm that you learned in your theory class. Mapping it to the following code (this code is implemented for Adjacency Matrix):

```
1  public void DFS_recur(int v, boolean[] visited){
2      visited[v] = true;
3
4      System.out.print(v + " ");
5
6      for(int i = 0; i < NUMBER_OF_VERTICES; i++){
7          if(adj[v][i] != 0 && visited[i] == false){
8              DFS_recur(i, visited);
9          }
10     }
```

```
}

public void DFS(int s){
    boolean[] visited = new boolean[NUMBER_OF_VERTICES];
    DFS_recur(s, visited);
}
```

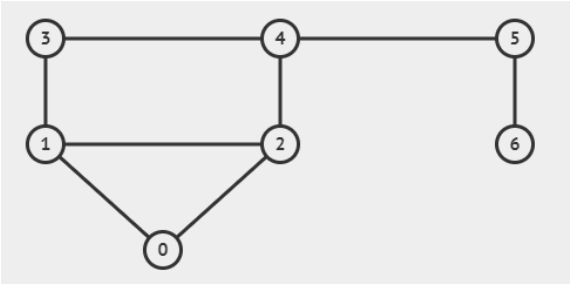
Part II
Excercise

Responsibility of the students in this part:

- Complete all the exercises with the knowledge from **Part I**.
- Ask your lecturer if you have any question.
- Submit your solutions according to your lecturer requirement.

Exercise 1

A graph is saved to file in AM format: (see the picture below)




```
7
0 1 1 0 0 0 0
1 0 1 1 0 0 0
1 1 0 0 1 0 0
0 1 0 0 1 0 0
0 0 1 1 0 1 0
0 0 0 0 1 0 1
0 0 0 0 0 1 0
```

(a) Read the graph from the file and print the AM on the screen.

(b) Count the number of vertices.

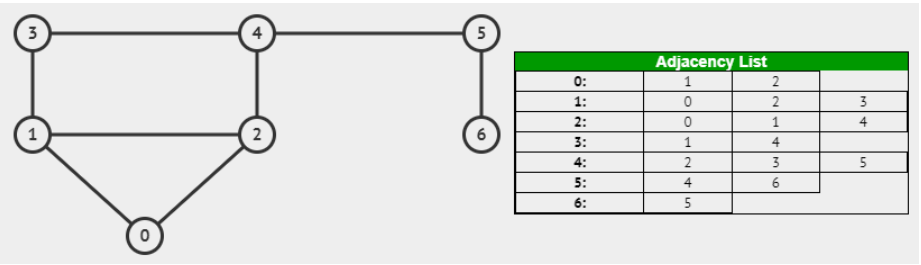
(c) Count the number of edges.

(d) Enumerate neighbors of a vertex u.

(e) Check the existence of edge (u, v).

Exercise 2

A graph is saved to file in AL format: (see the picture below)



Adjacency List			
0:	1	2	
1:	0	2	3
2:	0	1	4
3:	1	4	
4:	2	3	5
5:	4	6	
6:	5		

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

(a) Read the graph from the file and print the AL on the screen.

(b) Count the number of vertices.

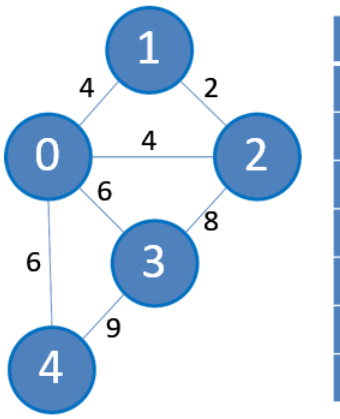
(c) Count the number of edges.

(d) Enumerate neighbors of a vertex u.

(e) Check the existence of edge (u, v).

Exercise 3

A graph is saved to file in EL format: (see the picture below)



i	w	u	v
0	2	1	2
1	4	0	1
2	4	0	2
3	6	0	3
4	6	0	4
5	8	2	3
6	9	3	4

```
2 1 2
4 0 1
4 0 2
6 0 3
6 0 4
8 2 3
9 3 4
```

(a) Read the graph from the file and print the EL on the screen.

(b) Count the number of vertices.

(c) Count the number of edges.

(d) Enumerate neighbors of a vertex u.

(e) Check the existence of edge (u, v).

Exercise 4

Write a method in the AdjacencyMatrix class to convert a graph from Adjacency Matrix to Adjacency List.

```
public AdjacencyList convertToAL(){
...
}
```

Exercise 5

Choose a graph, and use Adjacency Matrix to represent it. Implement the functions for the below requirements:

(a) Traverse the graph by using BFS. Print the traversal result on the screen.

(b) Traverse the graph by using DFS. Print the traversal result on the screen.

(c) Implement the DFS method without recursion. Print the traversal result on the screen.*(Hint: using Stack)*

(d) Implement the *IsReachable* method to test whether vertex v is reachable from vertex u.

THE END