Part 1
Graph Representation

What is Graph?

- In mathematics and computer science, graph theory is the study of graphs, which are mathematical structures used **to model pairwise relations between objects.**

Why study graph?

- Thousands of practical applications.
- Hundreds of graph algorithms known.
- Interesting and broadly useful abstraction.
- Challenging branch of computer science and discrete math

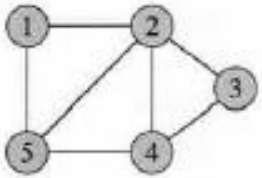
Important Terms

- Edges
- Vertices/Nodes
- Un/directed graph
- Un/weighted graph
- Dis/connected Graph
- Path, Cycle

Refer to your TR1313 Mathematics 1 notes

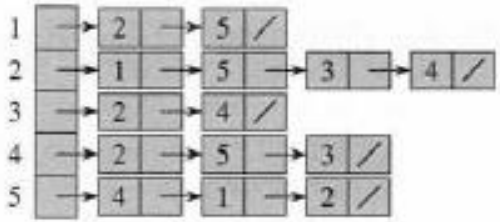
How to represent Graph?

Graph



(a)

List

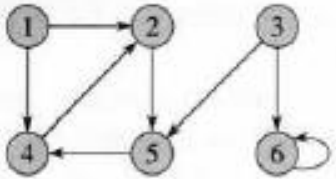


(b)

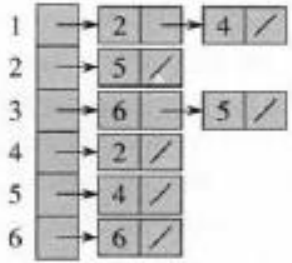
Adjacency Matrix

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)



(a)

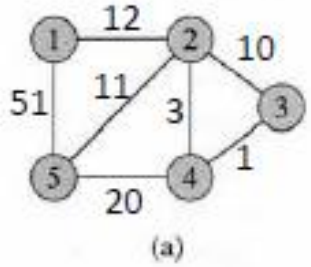


(b)

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

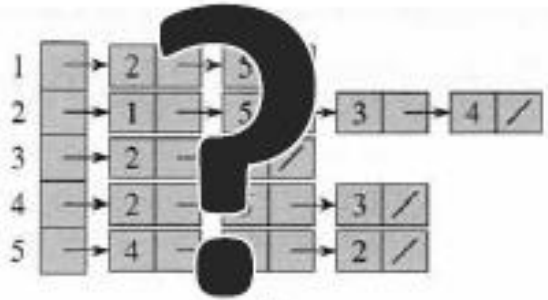
(c)

Graph



(a)

List



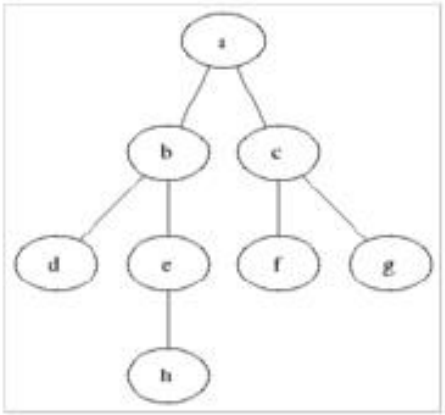
(b)

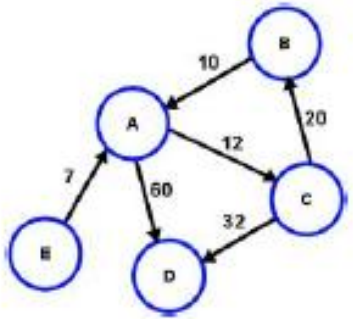
Adjacency Matrix

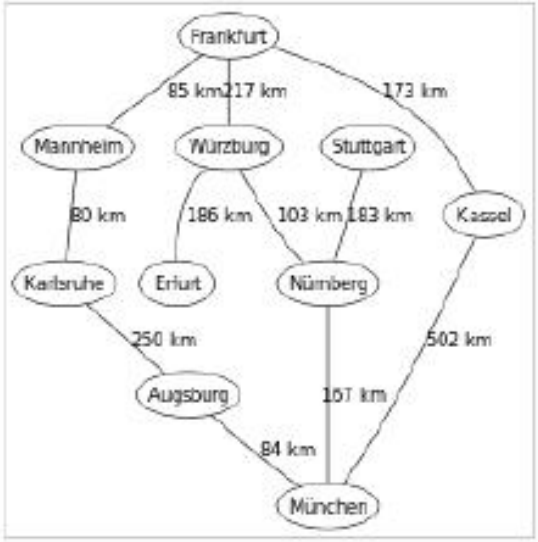
	1	2	3	4	5
1	0	12	0	0	51
2	12	0	10	3	11
3	0	10	0	1	0
4	0	3	1	0	20
5	51	11	0	20	0

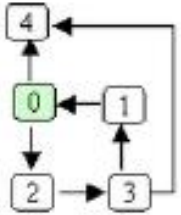
(c)

How to represent these?



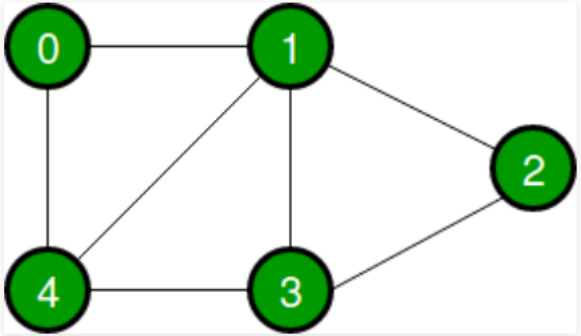






Graph Representation: Undirected Graph

Following is an example of an undirected graph with 5 vertices.



Full Java Codes: Graph and its representations
(https://www.geeksforgeeks.org/graph-and-its-representations/)

```
Output:

 Adjacency list of vertex 0
 head -> 1-> 4

 Adjacency list of vertex 1
 head -> 0-> 2-> 3-> 4

 Adjacency list of vertex 2
 head -> 1-> 3

 Adjacency list of vertex 3
 head -> 1-> 2-> 4

 Adjacency list of vertex 4
 head -> 0-> 1-> 3
```

Class Graph

```
// Java code to demonstrate Graph representation using ArrayList in Java
import java.util.*;
class Graph {

      Static void addEdge(ArrayList<ArrayList<Integer> > adj, int u, int v) {
          adj.get(u).add(v);
          adj.get(v).add(u);
      }

      // A utility function to print the adjacency list representation of graph
      static void printGraph(ArrayList<ArrayList<Integer> > adj)
      {
           for (int i = 0; i < adj.size(); i++) {
                System.out.println("\nAdjacency list of vertex" + i);
                      for (int j = 0; j < adj.get(i).size(); j++) {
                                System.out.print(" -> "+adj.get(i).get(j));
                      }
                      System.out.println();
           }
       }

}
```

Main method

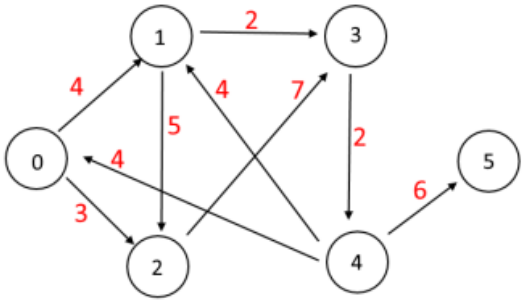
```
public static void main(String[] args) {
      // Creating a graph with 5 vertices
      int V = 5;
      ArrayList<ArrayList<Integer> > adj = new ArrayList<ArrayList<Integer> >(V);

       for (int i = 0; i < V; i++)
           adj.add(new ArrayList<Integer>());

           // Adding edges one by one
           addEdge(adj, 0, 1);
           addEdge(adj, 0, 4);
           addEdge(adj, 1, 2);
           addEdge(adj, 1, 3);
           addEdge(adj, 1, 4);
           addEdge(adj, 2, 3);
           addEdge(adj, 3, 4);
           printGraph(adj);
       }
}
```

Graph Representation: Weighted Graph

A Graph is called weighted graph when it has weighted edges which means there are some cost associated with each edge in graph.



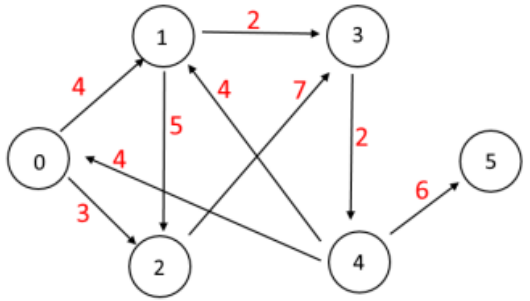
Weighted Graph

Refer to

https://algorithms.tutorialhorizon.com/weighted-graph-implementation-java/

Implementation of Weighted graph

- **Each edge** has an associated numerical value, called a ***weight*** (nonnegative integers)
- Weighted graphs can be either directed or undirected.
- The weight of an edge is often referred to as the "cost" of the edge.
- Will create an ***Edge class*** to put weight on each edge



Weighted Graph

```
<terminated> WeightedGraph [Java Application] C:\Progran
vertex-0 is connected to 2 with weight 3
vertex-0 is connected to 1 with weight 4
vertex-1 is connected to 2 with weight 5
vertex-1 is connected to 3 with weight 2
vertex-2 is connected to 3 with weight 7
vertex-3 is connected to 4 with weight 2
vertex-4 is connected to 5 with weight 6
vertex-4 is connected to 1 with weight 4
vertex-4 is connected to 0 with weight 4
```

Class Edge

```
static class Edge {
    int source;
    int destination;
    int weight;

    public Edge(int source, int destination, int weight){
        this.source = source;
        this.destination = destination;
        this.weight = weight;
    }
}
```

```
static class Graph {
        int vertices;
        LinkedList<Edge> [] adjacencylist;

        Graph(int vertices) {
              this.vertices = vertices;
              adjacencylist = new LinkedList[vertices];
               for (int i = 0; i <vertices ; i++) {  //initialize all the vertices
                    adjacencylist[i] = new LinkedList<>();
               }
          }

        public void addEgde(int source, int destination, int weight) {
            Edge edge = new Edge(source, destination, weight);
            adjacencylist[source].addFirst(edge); //for directed graph
        }

        public void printGraph(){
             for (int i = 0; i <vertices ; i++) {
                LinkedList<Edge> list = adjacencylist[i];
                for (int j = 0; j <list.size() ; j++)
                    System.out.println("vertex-" + i + " is connected to " +
                            list.get(j).destination + " with weight " +  list.get(j).weight);
             }
         }
}
```

Main Method (Weighted graph)

```
public static void main(String[] args) {
            int vertices = 6;
            Graph graph = new Graph(vertices);
            graph.addEgde(0, 1, 4);
            graph.addEgde(0, 2, 3);
            graph.addEgde(1, 3, 2);
            graph.addEgde(1, 2, 5);
            graph.addEgde(2, 3, 7);
            graph.addEgde(3, 4, 2);
            graph.addEgde(4, 0, 4);
            graph.addEgde(4, 1, 4);
            graph.addEgde(4, 5, 6);
            graph.printGraph();
        }
```