**Graph**

A graph is a data structure that consists of the following two components:

**1.** A finite set of vertices also called as nodes.

**2.** A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not the same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, and locale. See [this](http://en.wikipedia.org/wiki/Graph_theory#Applications) for more applications of graph.

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



The following two are the most commonly used representations of a graph.

**1.** Adjacency Matrix

**2.** Adjacency List

There are other representations also like, Incidence Matrix and Incidence List. The choice of graph representation is situation-specific. It totally depends on the type of operations to be performed and ease of use.

**Adjacency Matrix:**

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. Adjacency matrix for undirected graph is always symmetric. 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.

In case of an undirected graph, we need to show that there is an edge from vertex i to vertex j and vice versa. In code, we assign adj[i][j] = 1 and adj[j][i] = 1

In case of a directed graph, if there is an edge from vertex i to vertex j then we just assign adj[i][j]=1

The adjacency matrix for the above example graph is:

Adjacency Matrix Representation



*Pros:* Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).

*Cons:* Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time. Computing all neighbors of a vertex takes O(V) time (Not efficient).

Please see [this](https://ide.geeksforgeeks.org/9je5j6jJ13) for a sample Python implementation of adjacency matrix.

| import java.util.\*;    public class Main {  public static void main(String[] args)  {  Scanner sc = new Scanner(System.in);    // n is the number of vertices  // m is the number of edges  int n = sc.nextInt();  int m = sc.nextInt();  int[][] adjMat = new int[n + 1][n + 1];  for (int i = 0; i < m; i++) {  int u = sc.nextInt();  int v = sc.nextInt();  adjMat[u][v] = 1;  adjMat[v][u] = 1;  // for a directed graph with an edge pointing from u to v,we just assign  // adjMat[u][v] as 1  }  }  }    // This code is contributed by Tapesh(tapeshdua420) |
| --- |

**Adjacency List:**

An array of lists is used. The size of the array is equal to the number of vertices. Let the array be an array[]. An entry array[i] represents the list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is the adjacency list representation of the above graph.

Adjacency List Representation of Graph

| // Java code to demonstrate Graph representation  // using ArrayList in Java    import java.util.\*;    class Graph {    // A utility function to add an edge in an  // undirected graph  static void addEdge(ArrayList<ArrayList<Integer> > adj,  int u, int v)  {  adj.get(u).add(v);  adj.get(v).add(u);  // for a directed graph with an edge pointing from u to v,  // adj.get(u).add(v);  }    // 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);  System.out.print("head");  for (int j = 0; j < adj.get(i).size(); j++) {  System.out.print(" -> "  + adj.get(i).get(j));  }  System.out.println();  }  }    // Driver Code  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);  }  } |
| --- |

**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

*Pros:* Saves space O(|V|+|E|). In the worst case, there can be C(V, 2) number of edges in a graph thus consuming O(V^2) space. Adding a vertex is easier. Computing all neighbors of a vertex takes optimal time.

*Cons:* Queries like whether there is an edge from vertex u to vertex v are not efficient and can be done O(V).

In Real-life problems, graphs are sparse(|E| <<|V|2). That’s why adjacency lists Data structure is commonly used for storing graphs. Adjacency matrix will enforce (|V|2) bound on time complexity for such algorithms.