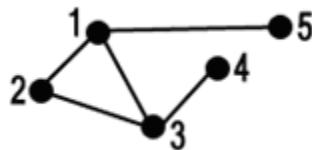


Introduction to Graphs

A graph consists of a set of vertices, usually known as V and a set of edges, usually known as E . Sometimes, vertices are known as nodes, while edges are known as arcs, but they generally mean the same thing. The diagram below is a graph – the dots represent the vertices of the graph and the lines connecting the dots represent the edges of the graph. A vertex is said to be **incident** to an edge if it is one of the end-points of the edge. Two different vertices are said to be **adjacent** if both vertices are **incident** to the same edge. For example, in the diagram below, vertex 1 is **adjacent** to vertex 5. The **degree** of a vertex is the number of edges that are **incident** to that vertex. For example, vertex 5 has a degree of 1.



In the above graph,

$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1, 2), (1, 3), (1, 5), (2, 3), (3, 4)\}$$

A graph can be **weighted** or **unweighted**. In a **weighted** graph, each edge in the graph has a weight associated with it. Likewise in an **unweighted** graph, each edge in the graph does not have a weight associated with it.

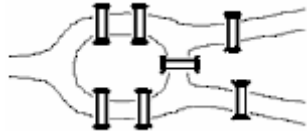
A graph is said to be **sparse** if the number of edges in the graph is much less than the number of vertices in the graph squared ($|V|^2$). Similarly, if the number of edges in the graph is close to the number of vertices in the graph squared ($|V|^2$), the graph is said to be **dense**.

A graph can also be **undirected** or **directed**. In an **undirected** graph, one can travel from one vertex to another in any direction as long as the 2 vertices are adjacent. For a **directed** graph, one can only travel from one vertex to another in one direction if the 2 vertices are adjacent. Taking the below diagram as an example of a directed graph, one can only travel from vertex 3 to vertex 4, but not from vertex 4 to vertex 3.



It is normal for people to wonder “what the heck can I do with the concept of graphs”. Generally, the vertices could be used to represent objects, while the edges could be used to represent the relationship between the objects.

- Roads between cities. The vertex could be used to represent the cities and that there is an edge connecting vertex i and j if and only if there is a road connecting city i and city j . This graph is weighted as each edge has a weight associated with it to represent the length of the road. The graph is also directed as for some roads, you can only travel along it in one direction (Unless you wish to meet an accident or be caught by the traffic policemen).
- Knights' Tour problem. The problem is to find the tour of a knight through a n by m chessboard. Only the valid moves of a knight can be used and every single square of the chessboard have to be visited. In the problem, the vertices of a graph can be used to represent every square on the chessboard, while the edges represent the valid moves of the knight. The graph is both undirected and unweighted. This is because knight can travel in both directions and there is no value associated with each valid move of the knight.
- Konigsberg's 7 Bridges problem. In the town of Konigsberg, the river Pregel separates the town into four separate land masses. The town is connected by seven bridges at various parts of the town (Refer to diagram below). Many people have wondered if it is possible to cross all seven bridges in one journey without having to cross any bridge twice. All of them failed. In this problem, the land masses can be the vertices of a graph and the bridge the edges of the graph. The graph is both unweighted and undirected.



Representation of graph

The choice of representation of graph is important, as different graph representations have different time and space costs. Therefore, it is important to choose the right representation when solving a graph problem. There is no such thing as the best kind of representation of graph as different graph algorithms might require you to use different representation of graph. There are basically just 4 main representation of graph – edge list, adjacency list, adjacency matrix and implicit representation.

Edge list

The most obvious way to represent the graph is to store a list with a pair of vertex representing an edge. If the graph is weighted, then the weight of the edge has to be stored together with the pair of vertices.

This representation is simple to use as it is very easy to code and also very space efficient. It is also easy to add an edge to the list. However, it is difficult to determine

whether 2 vertices are adjacent (as you would have to scan through the whole list) or to delete a specific edge (as you would have to scan through the whole list to find the edge to delete).



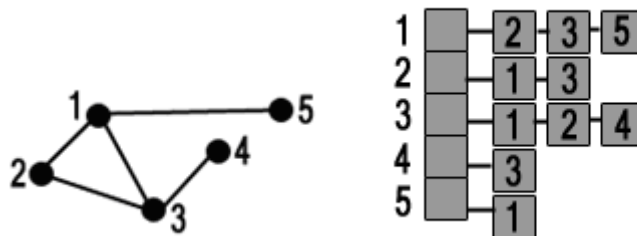
	V_1	V_2
e_1	1	2
e_2	1	3
e_3	1	5
e_4	2	3
e_5	3	4

Adjacency list

The second way to represent a graph is to keep track of all the edges that are incident to each vertex in the graph. This can be done by having an array of size N , where N is the number of vertex in the graph, to store the list of vertices that are adjacent to each vertex in the graph.

Adjacency list can be quite hard to code when there is no bound to the number of edges adjacent to each vertex. In such cases, linked lists have to be used or the lists have to be allocated dynamically. This representation uses as much memory as edge list. Finding all vertices that are adjacent to a particular vertex is easy, but one would need to search through all the edges in the list to determine whether 2 vertices are adjacent. Furthermore, addition of edges is easy, while deleting of edges is difficult as you need to determine the location of edges in the list first.

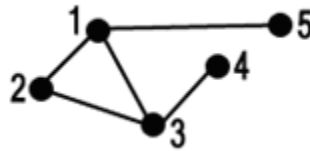
It is easy to extend this to weighted graph as all we need to do is to store the weight along with each adjacent vertex, instead of just the adjacent vertex, for each edge in the graph. Directed graph can also be easily maintained by either of these methods – storing the edges only in one direction, keeping a list of incoming and outgoing arcs or denoting the direction for each edge.



Adjacency matrix

The third way of representation of a graph is using adjacency matrix, which is actually a N by N array (N is the number of vertices in the graph). For an unweighted graph, if there exist an edge between vertex i and vertex j , let the entry i,j be 1. Otherwise it is zero or set to infinity (ie a very huge number), depending on the algorithm you are using. For a weighted graph, if there exist an edge between vertex i and vertex j , let the entry i,j be the weight of that edge. This adjacency matrix is symmetrical for a directed graph, while it is asymmetrical for an undirected graph.

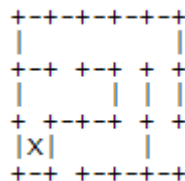
It is easy to insert edges and to delete edges using adjacency matrix. Also one can check whether 2 vertices are adjacent is easily. However, the problem with adjacency matrix is that it is not space efficient. This can be a problem if your graph is sparse and you adjacency matrix end up with a lot of zeros or infinity. Another problem with adjacency matrix is that it does not allow 2 edges to connect to the same 2 vertices – you have to choose which one you would want to store.



	V_1	V_2	V_3	V_4	V_5
V_1	0	1	1	0	1
V_2	1	0	1	0	0
V_3	1	1	0	1	0
V_4	0	0	1	0	0
V_5	1	0	0	0	0

Implicit Representation

For some graphs, it is not necessary to store the graph at all. This is because in such cases, it is easy to compute which vertex is adjacent to the current vertex. Examples would be the knights' tour (mentioned above) or finding the shortest path out of the maze (Refer to diagram below). It saves you the hassle of trying to store all the edges or using adjacency list or adjacency matrix.



Summary

Let E represent the number of edges, V represent the number of vertices in the graph and d_{\max} represent the maximum degree of a vertex. Then, the following table summarizes edge list, adjacency matrix and adjacency list.

	Edge List	Adjacency Matrix	Adjacency List
Space	$2 \times E$	V^2	$V + E$
Checking adjacency	E	1	d_{\max}
List of adjacent vertex	E	V	d_{\max}
Adding an edge	1	1	1
Deleting an edge	E	2	d_{\max}