"You don't have to be a genius to code, you just have to be persistent."

Hello Everyone
Very Special Good Evening
to All of you ☺
We will Start
from 9:00 pm

Graphs 1

~~~~~~~~~~

Agenda :

~~~~~~~~~~

Introduction { 1. Introduction to Graph
2. Types of Graphs

creation { 3. How to store data in Graph

traversal { 4. BFS (Breadth First Search)

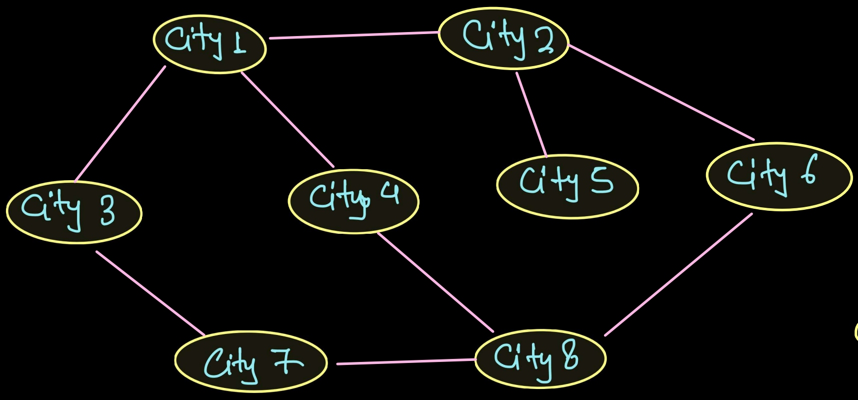
Searching in Graph { 5. Is Path Available from Source to Destination

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Introduction to Graph

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Want to store information of cities and their connectivity.



With the help of graph we can store this kind of information.

cities ≡ vertex

links ≡ Edge

No. of Edges = 9

No. of vertex = 8

Definition: Collection of vertex and edge is known as Graph.

Neighbour of 1 ⟶ city 2, city 3 and City 4

neighbour (nbr) of 8 ⟶ city 4, city 6, and city 7

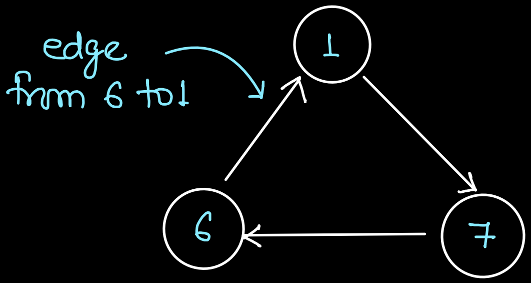
Direct connected vertex are known as Neighbours.

~~~~~~~~~~~~~~~~~~

Types of Graphs

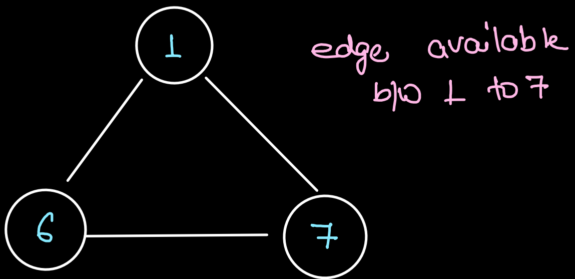
~~~~~~~~~~~~~~~~~~

1. Based on type of edges:



directed graph

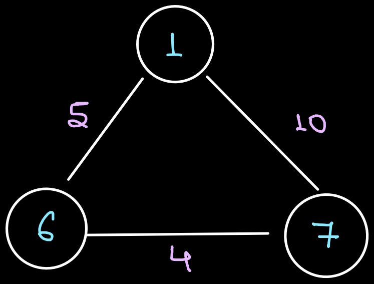
[instagram, youtube scriber]



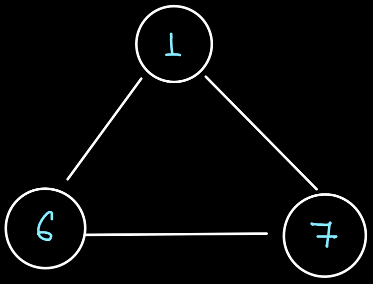
undirected graph

[LinkedIn, facebook]
connection

2. Based on Edge wt. present or not:

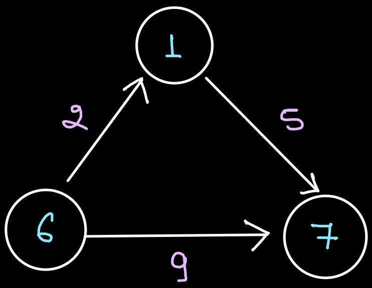


weighted graph



unweighted graph

3. Combination of above types are also possible:



directed weighted graph

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

How to store data in Graph

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

There is two famous implementation of graph is available :

① Adjacency matrix

② Adjacency list

1. Adjacency Matrix:

Vertex = 7 , Edges = 8

```
int[][] graph = new int[vtx][vtx];
```

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

Edge→

i →

u	v	
0	3	✓
0	1	✓
2	3	✓
3	4	✓
1	2	✓
4	5	✓
4	6	✓
5	6	✓

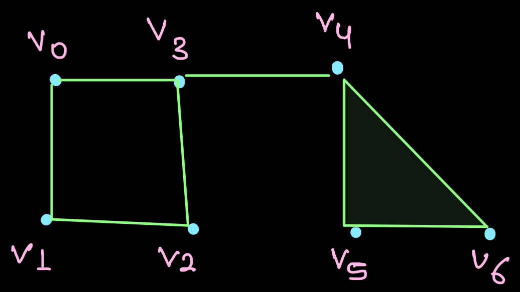
```
int u= edge[i][0];
int v= edge[i][1];
// Edge b/w u & v
graph[u][v] =1
graph [v][u] = 1
```

undirected graph

```
int u = edge[i][0] ;        u=2
int v = edge[i][1] ;        v=3

graph[u][v]=1; → graph[2][3]=1
grap[v][u]=1 ; → graph [3][2]=1
```

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

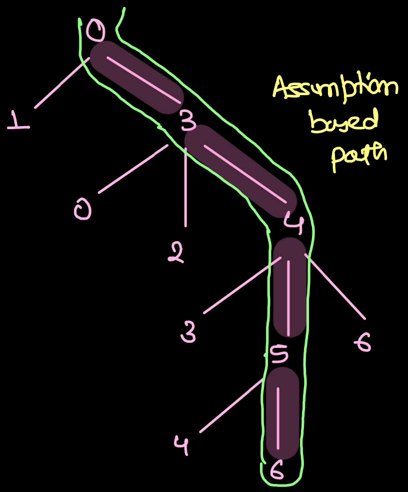


unweighted undirected graph

undirected graph.

$$\begin{array}{c|ccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 5 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 6 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}$$

source = 0 ; destination =6



directed Weighted Graph:

$$\begin{array}{c|ccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 0 & 0 & 5 & 0 & 10 & 0 & 0 & 0 \\ 1 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 9 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 7 & 20 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 & 15 \\ 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

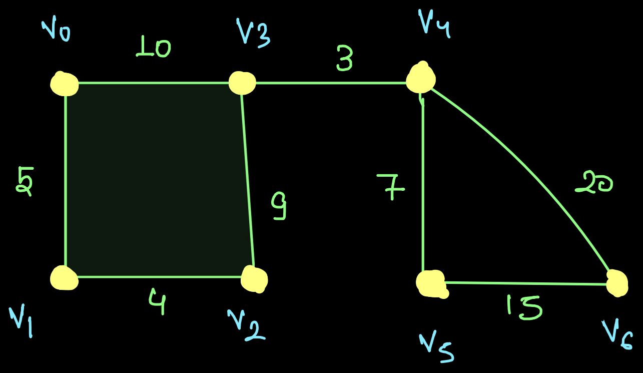
Vertex = 7 , Edges = 8

Edge→

U	v	wt	
0	3	10	✓
0	1	5	✓
2	3	9	✓
3	4	3	✓
1	2	4	✓
4	5	7	✓
4	6	20	✓
5	6	15	✓

```
int u = edge[i][0];
int v = edge[i][1];
int wt = edge[i][2];
// Edge from u to v
   with weight wt'
graph[u][v] = wt;
```

	0	1	2	3	4	5	6
0	0	5	0	10	0	0	0
1	0	0	4	0	0	0	0
2	0	0	0	9	0	0	0
3	0	0	0	0	3	0	0
4	0	0	0	0	0	7	20
5	0	0	0	0	0	0	15
6	0	0	0	0	0	0	0



Major disadvantge of Adjacency matrix:

major disadvantge is wastage of Space. that 's why. most of the tim we will deal with adjacency list problem