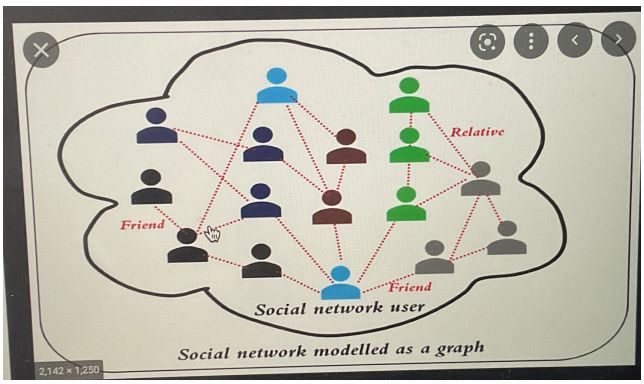
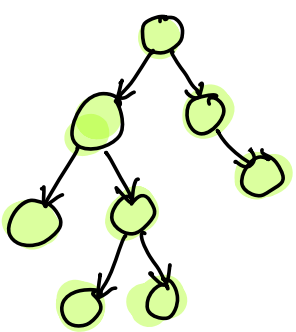


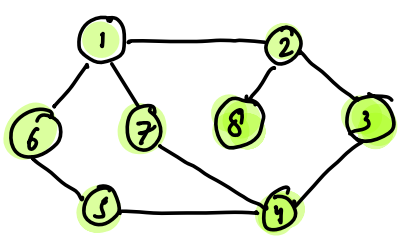
Introduction to Graph

Graph : It is bunch of nodes connected via Edges

Ex 1: Trees



Graph



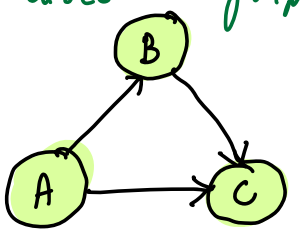
Main differences between trees & Graphs

1) Trees is hierarchical data structure, unlike graph
2) No. of Edges in N nodes Tree = N-1

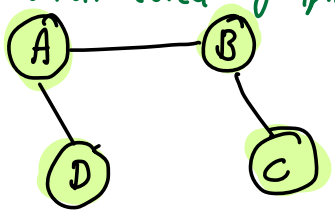
Classification of Graph

Case I

directed graph

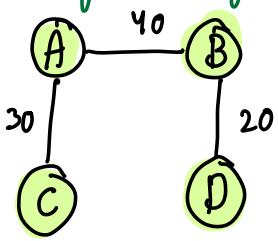


undirected graph

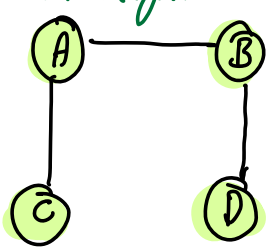


Case II

weighted graph

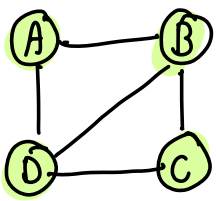


unweighted graphs

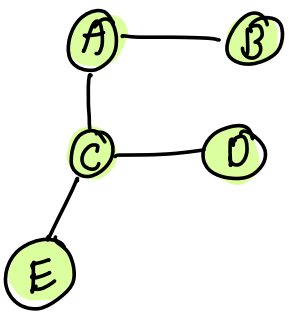


Case III

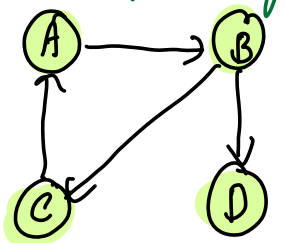
undirected cyclic graph



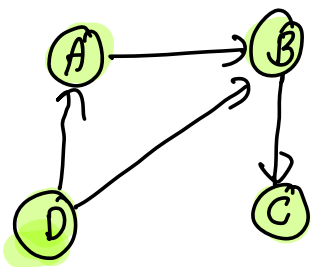
undirected acyclic graph



directed cyclic graph



directed acyclic graph



How Graph is given as Input?

Q1 ⇒ Given an undirected graph with N nodes & M edges

Input Format:

1st line

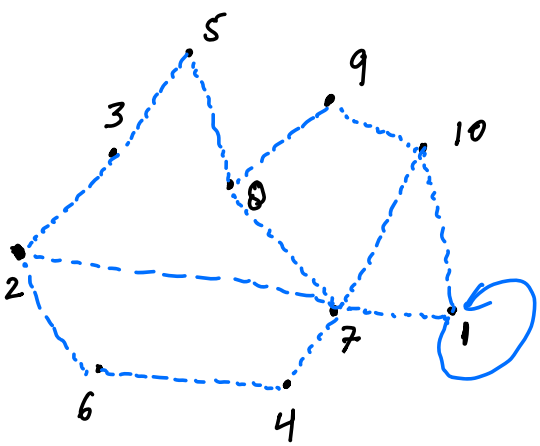
Node # Edges

N E

Followed by E lines

u v

N	E
10	14
2	3
4	7
8	9
2	7

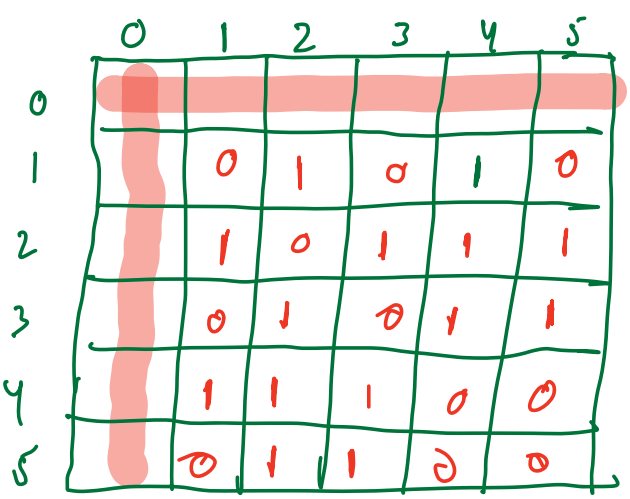


7 8
10 1
4 6
5 8
2 6
10 9
7 10
3 5
7 1
1 1

Storing a Graph

Input

N	E
5	7
1	4
2	5
3	2
4	3
2	4
3	5
1	2

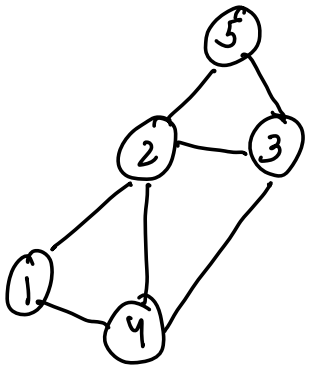


SC: $O(N^2)$

N nodes ⇒ g[N+1][N+1] ✓ TC: O(E)

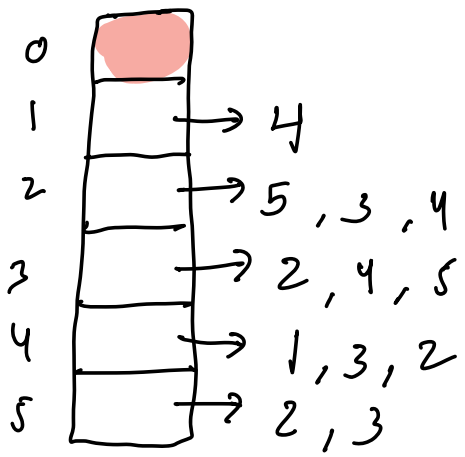
	unweighted	weighted
undirected	g[u][v] = 1 g[v][u] = 1	g[u][v] = w g[v][u] = w
directed	g[u][v] = 1	g[u][v] = w

1



Input :

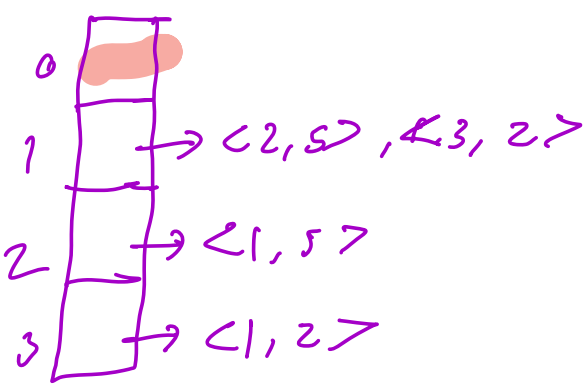
N	E
5	6
1	4
2	5
3	2
4	3
2	4
3	5



array of list

list < pair < int, int >> g(4)

3	5	
1	2	5
1	3	2
—	—	—
—	—	—



SC : O(E)

TC : 2E ⇒ O(E)

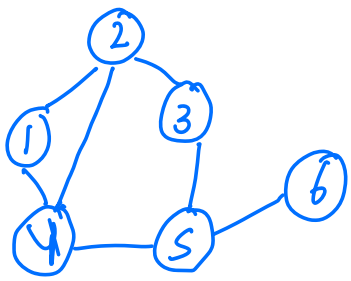
	unweighted	weighted
undirected	g[u].add(v) g[v].add(u)	g[u].add({v, w}) g[v].add({u, w})
directed	g[u].add(v)	g[u].add({v, w})

unwght → list<int> g[N+1]

wght → list<pair<int, int>> g[N+1]

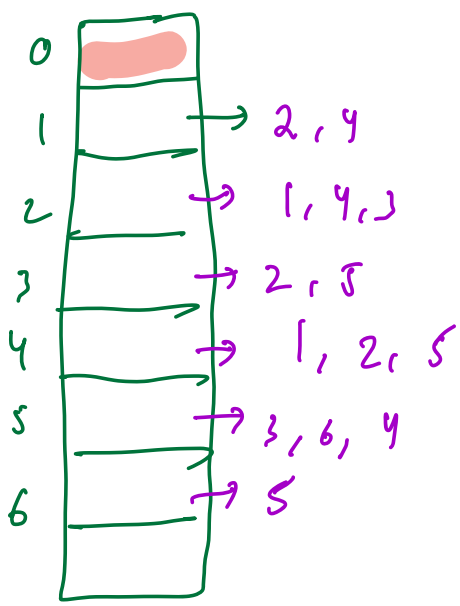
Q⇒ Given an undirected graph. Source Node & destination - Check if node can be visited from source node-

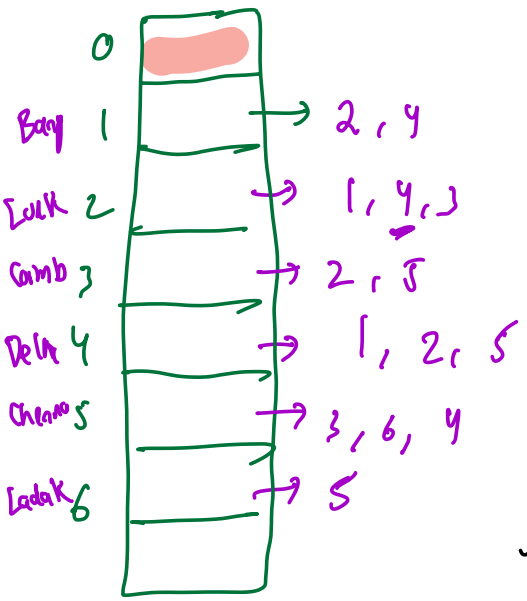
S D
1 → 6



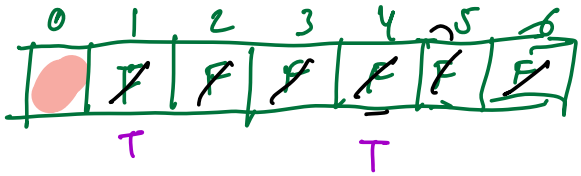
Input :

N E
6 7
1 2
1 4
2 4
2 3
3 5
5 6
4 5

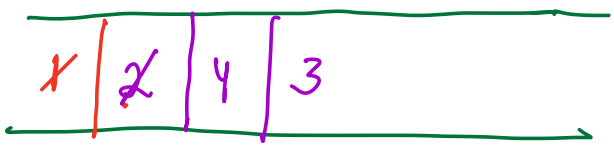




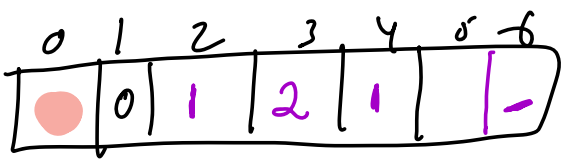
s=1 , d=6



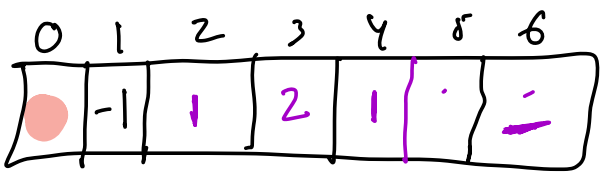
Q



level



parent



```
bool BFS( int N, int E, int u[], int v[], int s, int d
{
  list <int> g[N+1]
  for( int i = 0; i < E; i++)
    {
      g[u[i]].add(v[i]);
      g[v[i]].add(u[i]);
    }

  Queue < int > q;
    q.insert(s);
  bool vis[N+1]=F;
  vis[s]=T;
  int lev[N+1]=-1 ; lev[s] = 0
  int par[N+1]=-1 ; par[s]=-1

while( q.size() > 0)
 {
    int cv = q.front();
      q.delete()

   for( i=0; i < g[cv].size(); i++)
      {
        int cv = g[cv][i]
```

```
            if( vis [cv] == false)
              {
                    vis [cv] = true;
                    q.add (cv);
                    lev [cv] = lev[cu] +1;
                    par [cv] = cu;
              }
        }
    }

  return  vis [d];
}
```