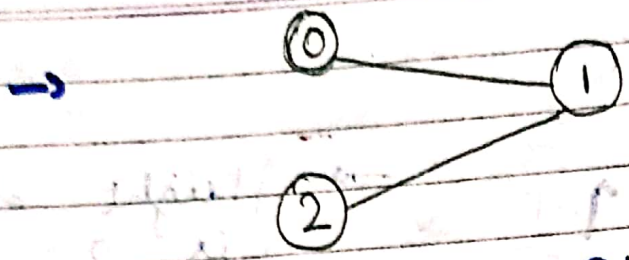
HOW TO REPRESENT TREES AND GRAPHS IN CODE

→ Adjacency Matrix : In this we create a V x V (V = Vertex) matrix.

→ Adjacency Matrix of Undirected graph:

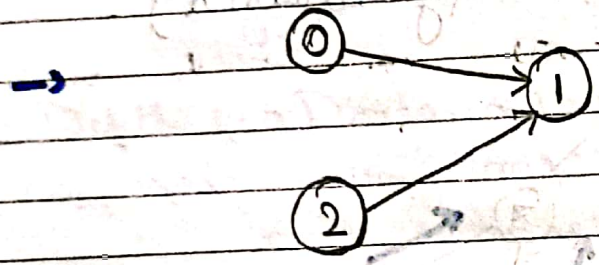


V = 3 , E = 2

	0	1	2
0	0	1	0
1	1	0	1
2	0	1	0

If i, j is connected:
a[i][j] = 1
else:
a[i][j] = 0

→ Adjacency matrix of directed graph:

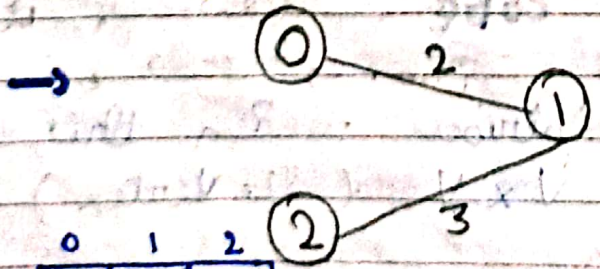


V = 3 , E = 2

	0	1	2
0	0	1	0
1	0	0	0
2	0	1	0

If i, j is connected:
a[i][j] = 1
else:
a[i][j] = 0

→ Adjacency matrix of undirected weighted graph:



V = 3 , E = 2

	0	1	2
0	0	2	0
1	2	0	3
2	0	3	0

If i, j is connected:
a[i][j] = weight
else:
a[i][j] = 0

-> General input we get in graph's questions:

First line contains n, m, where n = vertices, m = edges.
Then the following m lines contains configurations (connections) of edges.

e.g:

n	m
1	2
2	3
3	4

Here, n = 4 & m = 3
(It denotes an edge b/w 1 & 2)

ADJACENCY MATRIX

CODE

```
CONST INT N = 1e3 + 10;
```
-> Let's suppose we can have a max of 10^3 vertices.

-> Initializing an Adjacency matrix of N, N. And as it is declared globally, it will automatically be initialized with all 0 values.

```
INT GRAPH [N][N];

INT MAIN ()
{
    INT n, m;
```
-> vertices and edges resp.

```
    CIN >> n >> m;
```

```
FOR (INT i=1; i ≤ m; i++)
{
    INT V1, V2;
    CIN >> V1 >> V2;
    -> Undirected graph
    GRAPH [V1][V2] = 1;
    GRAPH [V2][V1] = 1;
}
```

→ But, there is an issue with this approach:

→ $O(N^2)$ space complexity.

→ And if we get a N with 10^5 then $N \times N$ will be 10^{10}, which is out of limit for an array, an array can store a maximum of $10^6 - 10^7$ capacity of values.

→ So, to represent large capacity of N, we use Adjacency lists.

```
    RETURN 0;
}
```

INPUT :

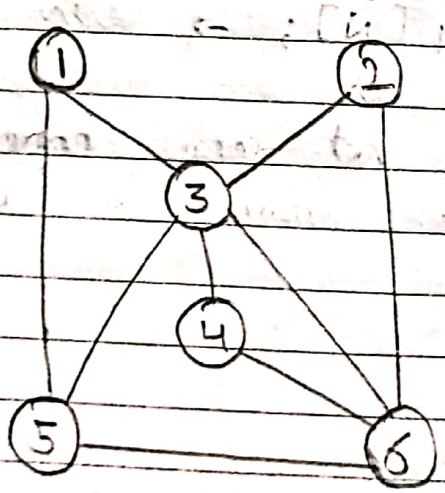
```
6 9
1 3
1 5
3 5
3 4
3 6
```

3 2
2 6
4 6
5 6

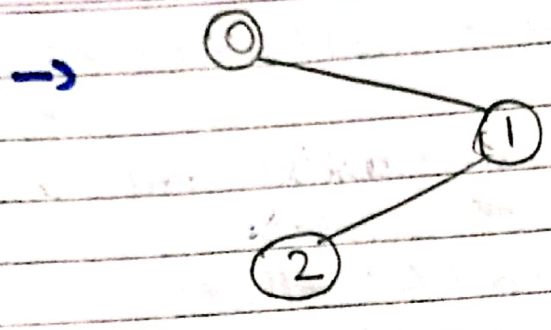
OUTPUT:

0 0 1 0 1 0
0 0 1 0 0 1
1 1 0 1 1 1
0 0 1 0 0 1
1 0 1 0 0 1
0 1 1 1 1 0

→ It will look something like this:



→ Adjacency list: In this if a graph has V vertices, then it will contain V no. of lists (e.g: If 3 vertices then 3 lists) And each list will contain its connected vertices (edges)).



Its adjacency list will be:

0 → 1
1 → 0, 2
2 → 1

(Its space complexity is O(V+E))

ADJACENCY LIST

CODE

```
CONST INT N = 1e3 + 10;
VECTOR<INT> GRAPH[N]; -> Vector of arrays
```

Here, GRAPH[0] is an array which will contain it's connected edges.

```
INT MAIN()
{
    INT n, m;
    CIN >> n >> m;

    FOR (INT i=1; i <= m; i++)
    {
        INT V1, V2;
        CIN >> V1 >> V2;

        GRAPH[V1].PB(V2);
        GRAPH[V2].PB(V1);
```

}

-> Time Complexity => O(N+M)

RETURN 0;
}

NOTE: If we have to store weighted undirected graph in Adjacency list, then we will do slight changes in our previous code:

-> Now we will take vector of pair instead of vector of arrays:

```
VECTOR < PAIR < INT, INT >> GRAPH [N];
```

-> And we will store pairs now with weights like this:

```
GRAPH [V1]. PB ( { V2, wt } );
GRAPH [V2]. PB ( { V1, wt } );
```