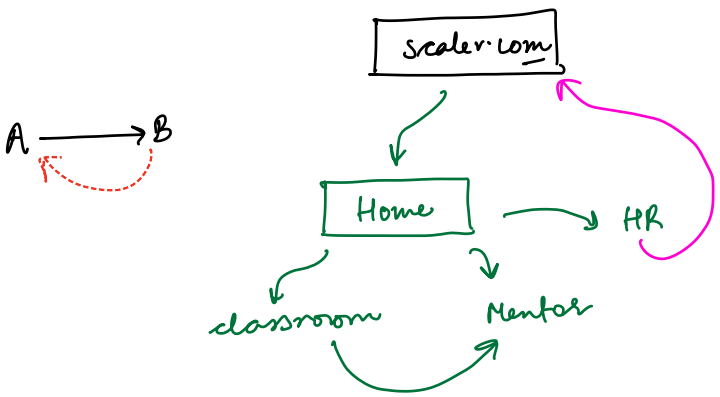
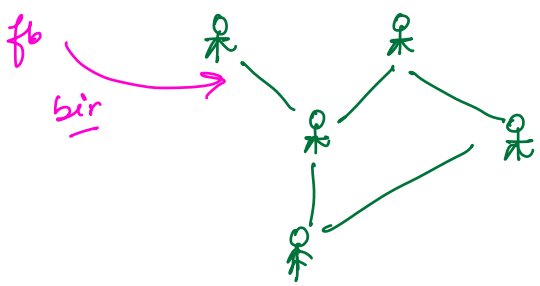
graphs ? network

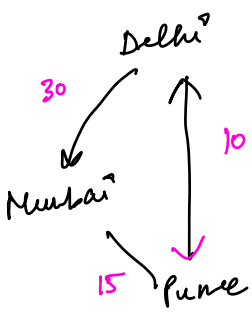
collection of nodes and edges

tree → graph



social media

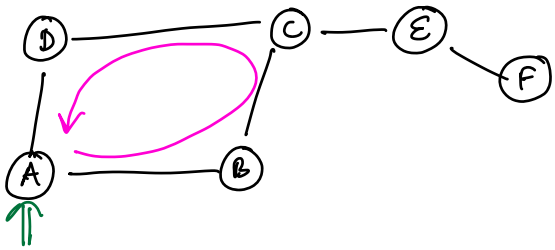


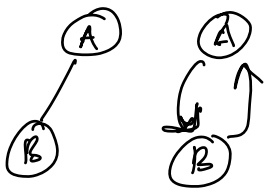


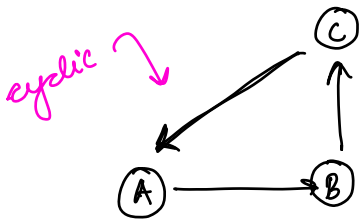
weighted / unweighted

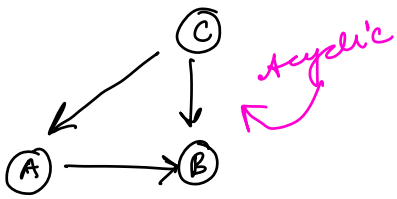
A → B insta

directed ⟹ undirected

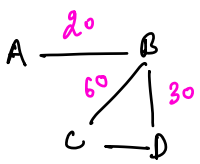


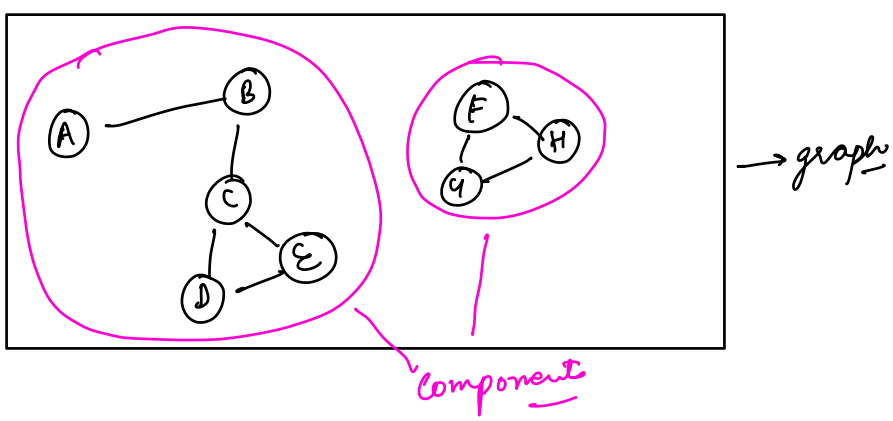




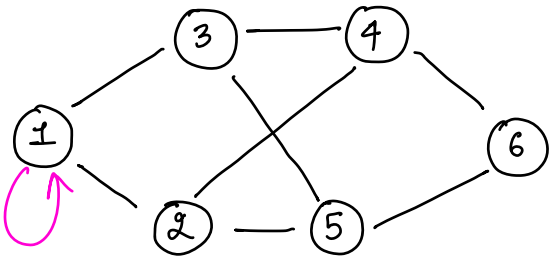


If you start from a node you reach that node again without visiting any edge twice.





- How to store the graph?



nodes :- 6
edges :- 8

1 — 3 1-2 1→6
2 — 4 2-5
3 — 5 5-6
4 — 6 3-4

1) Adjacency matrix

no of nodes ~ n * n

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

mat[i][j] → 1 → i-j
mat[i][j] → 0 → X

undirected
i → j
j → i

directed
i → j

1) access the conn ✓
2) updating the conn ✓

advantage

1 → n

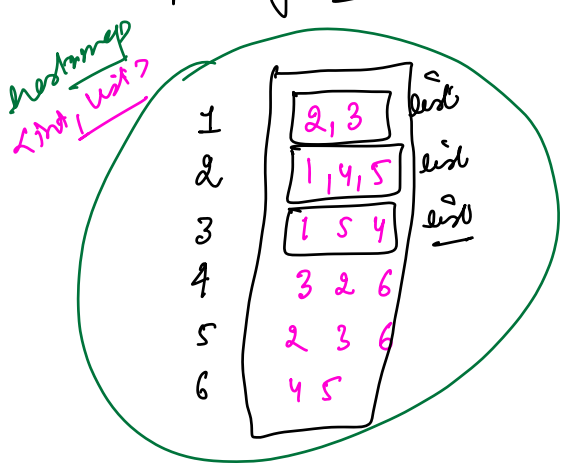
mat[n+1][n+1];

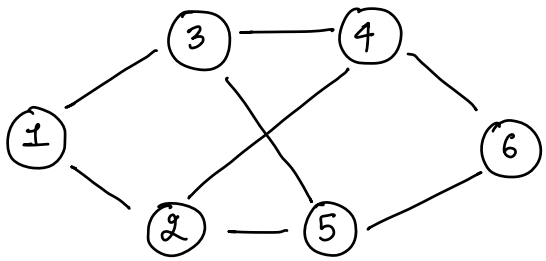
Input 1→3
n = 3
m = 3
1 3
2 3
1 2

```
int n;  input(n)
int m;  // edges
for( i=0; i<m; i++)
{
    input( u, v);  // there is an edge b/w u & v
    mat[u][v] = 1;
    mat[v][u] = 1;
}
```

directed → do acc to que

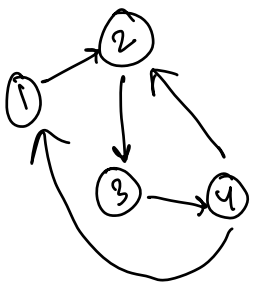
Adjacency list





1 → n

list <int> graph [n];



1	2
2	3
3	4
4	1, 2

set <int> graph[n+1];

access - O(1)

```
int n;
int m;
list <int> graph [n+1];
for( int i=0; i<m; i++)
{
    input(u,v);
    graph[u].insert(v);
    graph[v].insert(u);
}
```

1 → n

u, v, weight

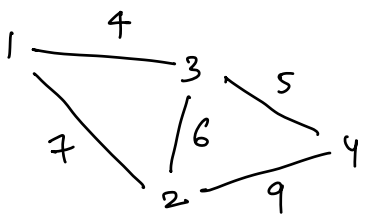
u → v

undirected

directed

```
graph[u].insert(v);
```

weighted

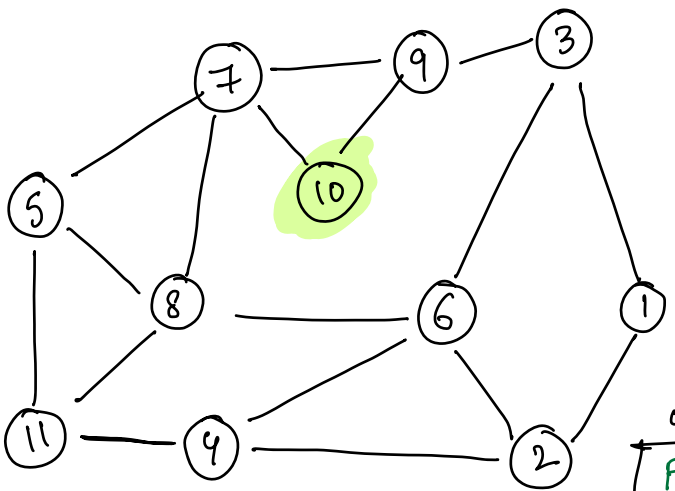


1	{3,4} {2,7}
2	{1,7} {3,6} {4,9}
3	{1,4} {2,6} {4,5}
4	{2,9} {3,5}

list < pair > graph[n+1];

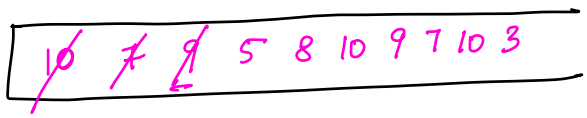
Traversals → BFS
→ DFS

1) BFS : Breadth first search (level order Traversal)



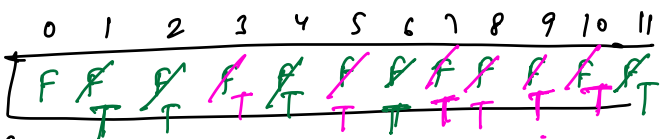
1 → n

queue

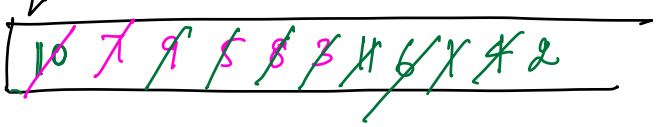


track the visited nodes

bool visited [n+1];



q



10 7 9 5 8 3
11 6 1 4 2

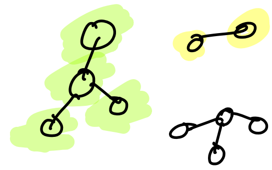
code

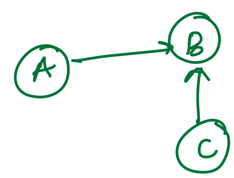
1 → n

```
list<int> graph[n+1];

queue <int> q;
bool visited [n+1];        // initially everything is false

// source
for( source=1; source<=n; source++)
{   if( visited[source] ) continue;
    q.push(source);
    visited[ source] = true;

    while( !q.empty() )
    {
        int x = q.front();
        print(x);
        q.pop();
        for( i=0; i< graph[x].size(); i++)
        {   int v= graph[x][i];
            if( !visited [v])
            {
                visited[v]=true;
                q.push(v);
            }
        }
    }
}
```

1) not connected

2) for directed





T.C: n + m
n → node, m → edges

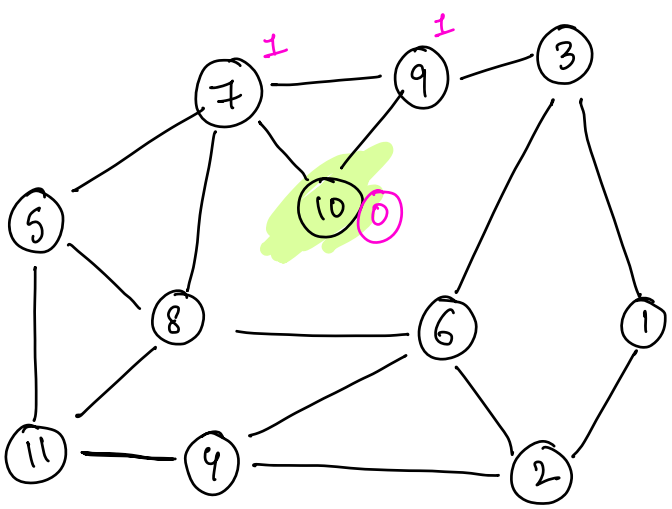
$m \approx n^2$

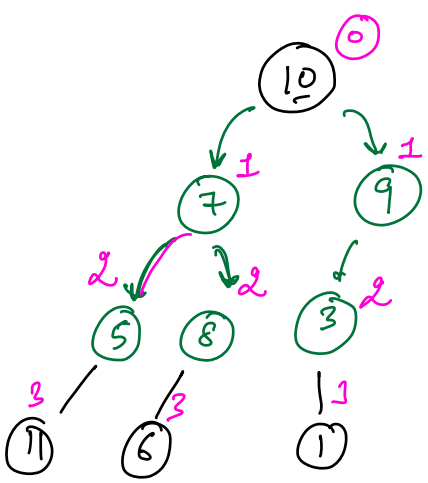
source — destination

Is it possible to go from source to destnat?

BFS(source) → if visited[destination] = true
go you can visit.

shortest distance from source to destnation





style

When we do BFS from source, it's kind of visiting every node from source through its shortest path.

#unweighted

dist[6] = dist[8] + 1
dist[8] = dist[7] + 1
dist[7] = dist[10] + 1
dist[10] = 0

```
list<int> graph[n+1];

queue <int> q;
bool visited[n+1];

// source
```

1) not connected
2) for directed

```
q.push(source);
visited[source]=true;
dist[source]=0;

while( !q.empty() )
{
    int x = q.front();
    print(x);
    q.pop();

    for( i=0; i< graph[x].size(); i++)
    {
        int v= graph[x][i];

        if( !visited[v]
        {
            dist[v] = dist[x]+1;
            visited[v]=true;

            q.push(v);
        }
    }
}
```

distance[n+1]; // ∞

// initially everything is false

Time: n + m (n: node, m: edge)