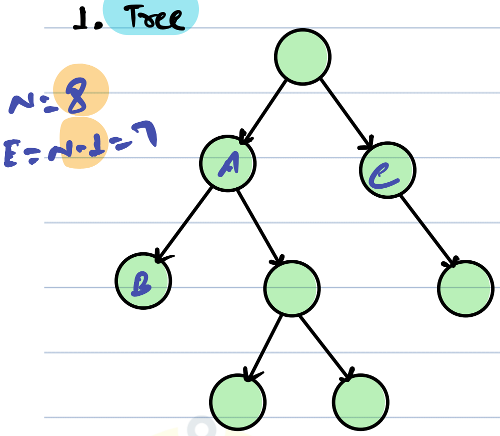
Today's agenda

- ↳ Intro
- ↳ Types of graph
- ↳ Storage
- ↳ BFS(level order) +1

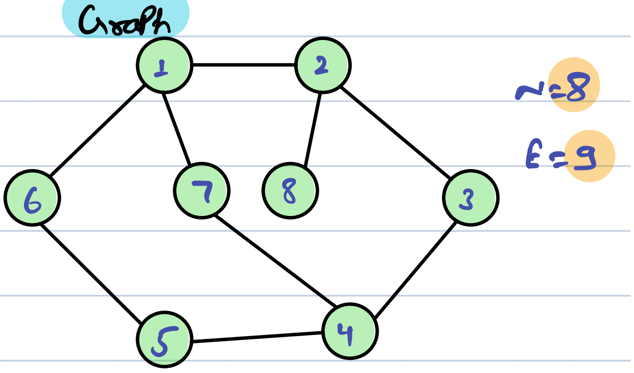
Intro

↳ Graph: Connection of nodes and edges

1. Tree



Graph



1 and 4 are nbrs of 7.

→ diff^n bet^n trees and graphs.

1. Nodes in graph can have any number of connections without hierarchy.

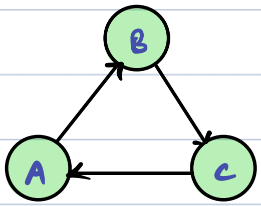
↓

↳ Graphs can have cycles.

2. Any directional movement is allowed in graph.

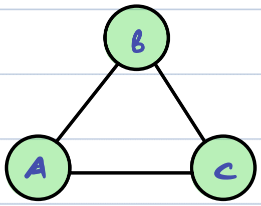
* Classification of graphs

Case I : Based on types of edges.



↳ directed graph

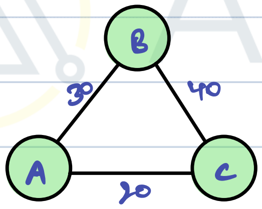
↳ insta followers



↳ undirected graph

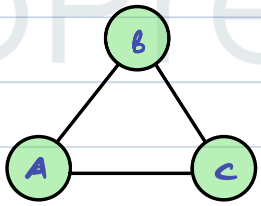
↳ facebook friend

Case II : Based on weight of edge



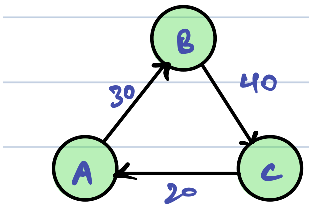
↳ weighted graph

↳ google maps



↳ unweighted graph

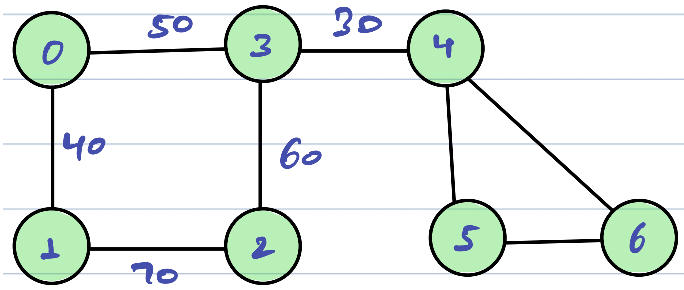
↳ ??



→ directed weighted graph

Storing a graph

undirected graph



input

N=7 → nodes

M=8 → no. of edges

3	4	30
1	2	70
2	3	
4	6	
0	1	
4	5	
5	6	
0	3	

① Adjacency matrix representation

Graph

Nodes

nodes

	0	1	2	3	4	5	6
0	0	~~0~~ 1	0	~~0~~ 1	0	0	0
1	1 ~~0~~	0	~~0~~ 1	0	0	0	0
2	0	1 ~~0~~	0	3 ~~0~~	0	0	0
3	~~0~~ 1	0	1 ~~0~~	0	30 ~~0~~ ~~1~~	0	0
4	0	0	0	~~1~~ 30 ~~0~~	0	~~0~~ 1	~~0~~ 1
5	0	0	0	0	~~0~~ 1	0	~~0~~ 6
6	0	0	0	0	1 ~~0~~	~~0~~ 1	0

7x7

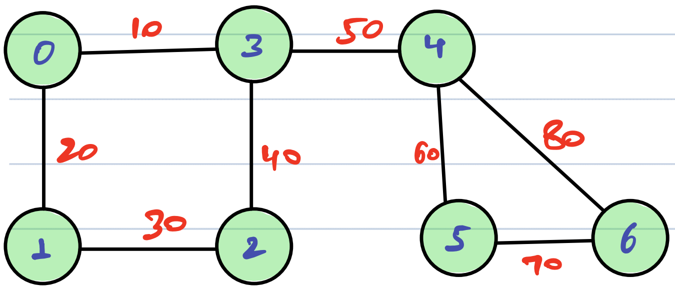
0 → disconnected

1 → Connected

issue:

↳ Space wastage

② adjacency list representation



N=7 M=8

edges

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

```
List<List<Integer>> graph= new Arraylist<>();
    for(int i=0; i<n; i++) {
        graph.add(new ArrayList<>());
    }
```

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

graph

//PSuedo Code

```
Class Pair{
    int v;
    int wt;
}

Construction (int n, int m, int edges[m][2]){
    List<List<~~Integer~~ Pair>>graph= new ArrayList<>();
    for(int i=0; i<n; i++){
        graph.add(new ArrayList<>());
    }

    for(int i=0; i<m; i++){
        int u= edges[i][0];
        int v= edges[i][1];
        graph.get(u).add(v);
        graph.get(v).add(u);
    }
    return graph;
}
```

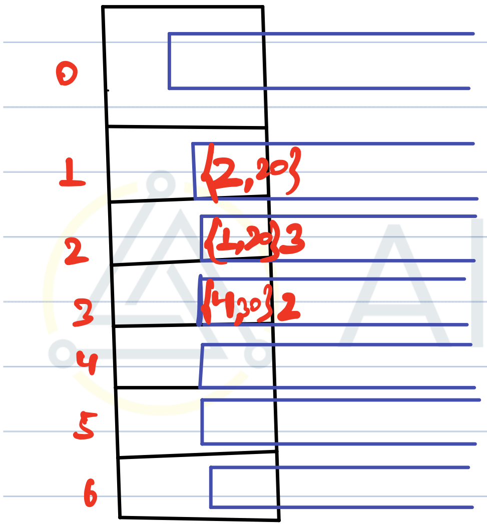
```
Construction (int n, int m, int edges[m][2]){

    List<List<Integer>> graph= new ArrayList<>();
    for(int i=0; i<n; i++) {
        graph.add(new ArrayList<>());
    }

    for(int i=0; i<m; i++){
        int u= edges[i][0];
        int v= edges[i][1];
        graph.get(u).add(v);
        ~~graph.get(v).add(u);~~
    }
}
```

u= 2
v= 3

N=7 m=8

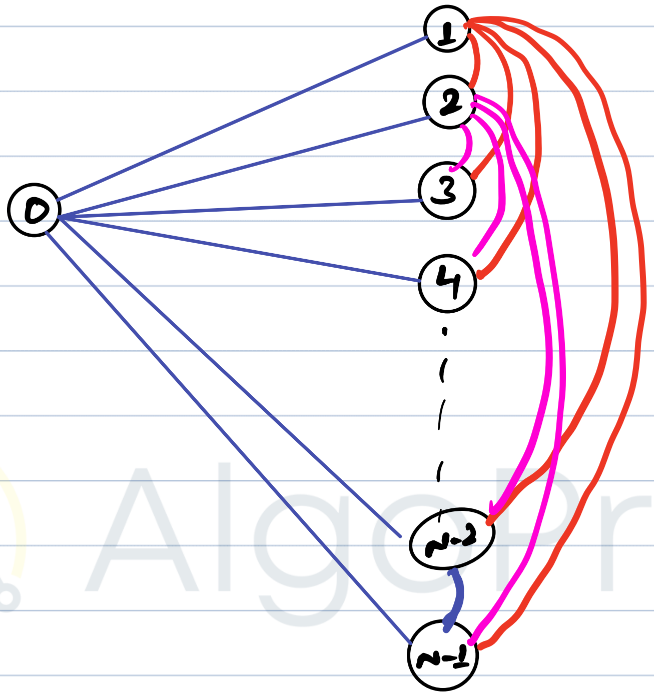
edges

	0	1	
0	3	4	30
1	1	2	20
i → 2	2	3	
3	4	6	
4	0	1	
5	4	5	
6	5	6	
7	0	3	



graph

Q) Find max no. of edges possible if N nodes are present in Graph.



$$(N-1) + (N-2) + (N-3) + \ldots \_\_\_ \quad \_\_\_ + 1$$

$$\frac{n*(n+1)}{2}$$

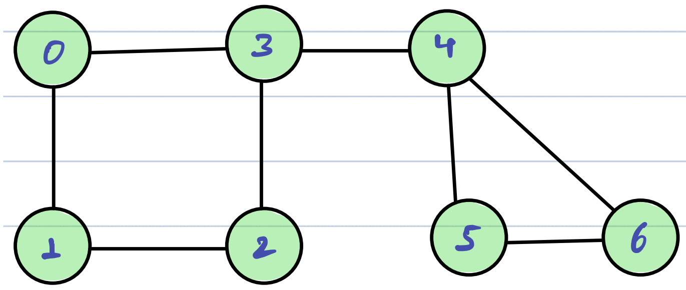
$$\frac{(N-1) * N\cancel{-1+1}}{2} = \frac{N * (N-1)}{2} \approx O(N^2)$$

$O(V^2)$

Break till 9:50 PM

→ Breadth first search

BFS traversal



→ no. of nodes

→ No. of edges

N=7 M=8

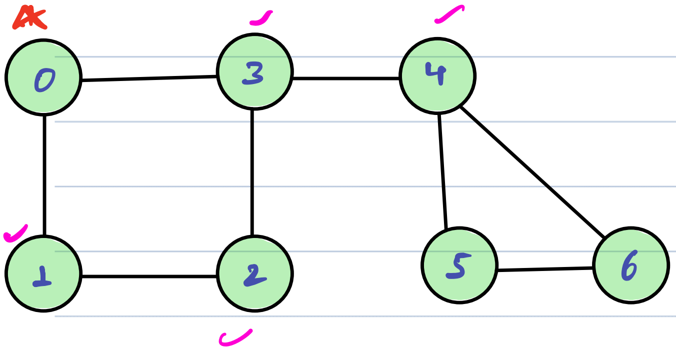
i=0

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

List<List<Integer>> graph = new ArrayList<>();

graph

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

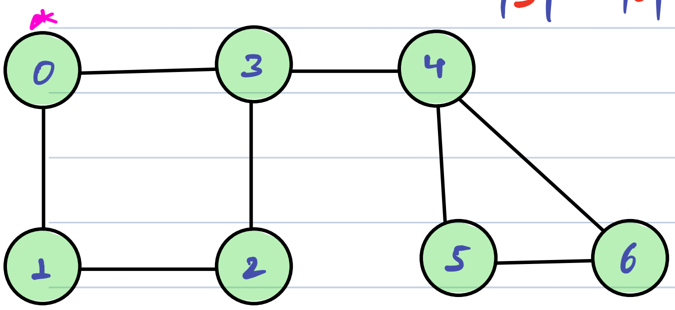


→ you will have to decide the start Point for bfs.

0 (1 3) (2 4) (5 6)

graph

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



queue : ~~0~~ ~~1~~ ~~3~~ ~~2~~ ~~4~~ ~~6~~ ~~5~~

vis :

0	1	2	3	4	5	6
~~F~~T	~~F~~T	~~F~~T	T~~F~~	T~~F~~	T~~F~~	~~F~~T

rem= 5

0 (1 3)(2 4)(6 5)

0 (1 3) (2 4) (5 6)

//Psuedo Code

```
void BFS (int N, int M, int[][] edges){

O(V+E) ← List<List<Integer>> graph = Construction(N,M,edges);

        Queue<Integer> q;
        boolean [] vis = new int[N];

        q.add(0);
        vis[0] = true;

        while (q.size() > 0 ){
            int rem = q.remove();
            S.O.P(rem);

            //add all unvisited nbrs.
            List<Integer> nbrs = graph.get(rem);

            for(int v: nbrs){
                if(vis[v] == false){
                    q.add(v);
                    vis[v] = true;
                }
            }
        }
}
```

T.C : $O(V+2E) \approx O(V+E)$
↓
$O(V^2)$

S.C : $O(V)$

O(v)