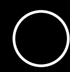
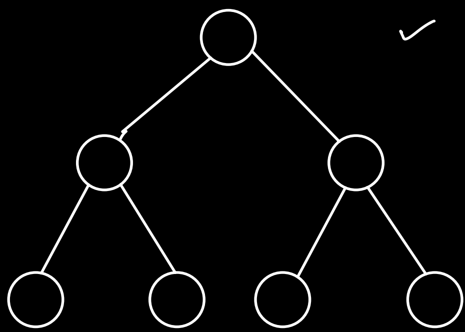
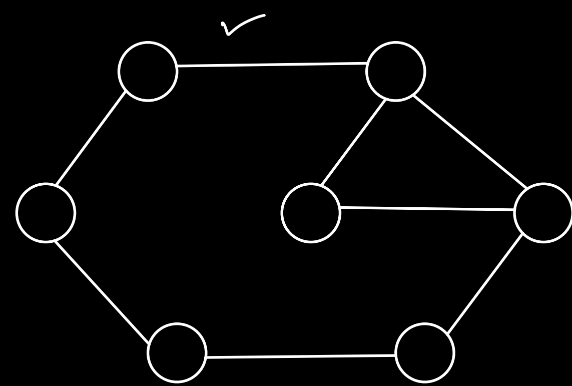
Graph: Bunch of nodes connected via edges.



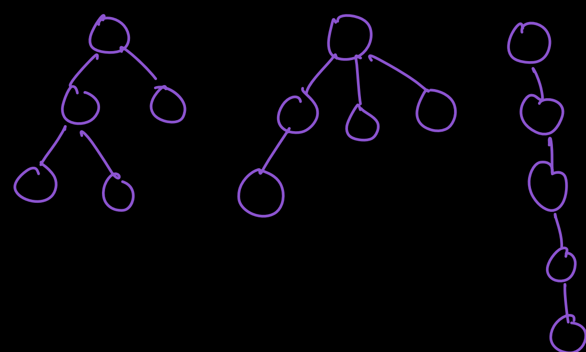


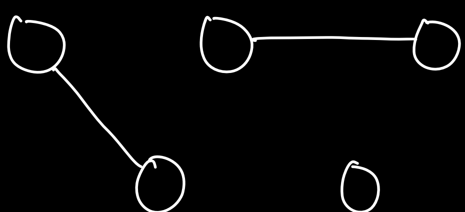


Tree:

- → Hierarchical DS unlike graph.
- → N nodes in a Tree ⇒ N-1 Edges.

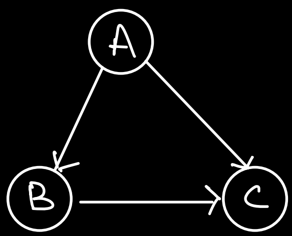
5 Nodes ⇒ Tree
⇒ N-1 Edges





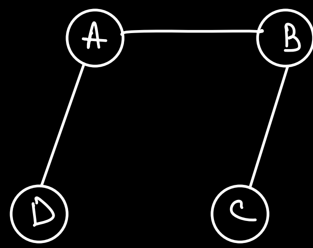
Classification of Graphs :

①



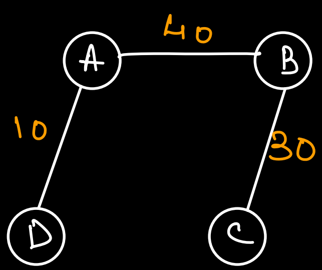
Directed

2)



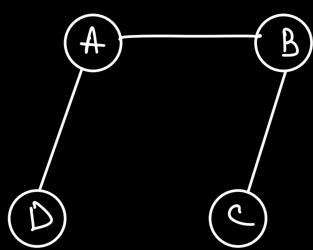
UnDirected

3)



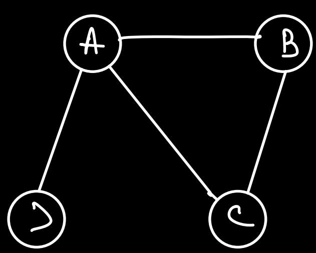
⇒ Weighted.

4)



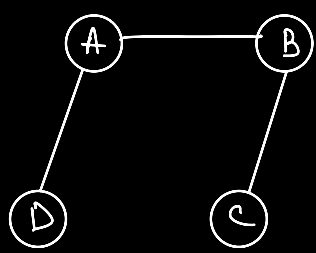
⇒ Unweighted.

5)



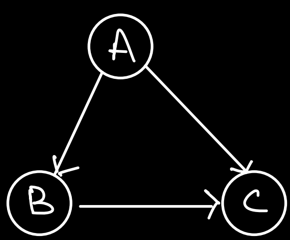
⇒ Cyclic

6)



⇒ Acyclic

ex



Unweighted Directed Acyclic graph.

FB

A —— B
friend

Instagram

A → B
follows

Undirected Graph

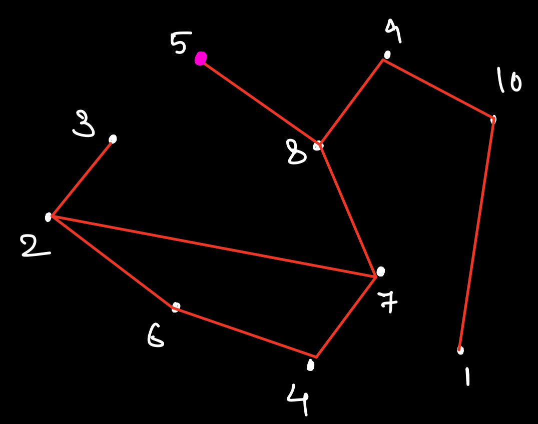
of Nodes (N), # of Edges (E)

N = 10, E = 10

u[ ], v[ ]

u[i] —— v[i]

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



In the problem statement:

→ Undirected vs directed

→ Unweighted vs weighted

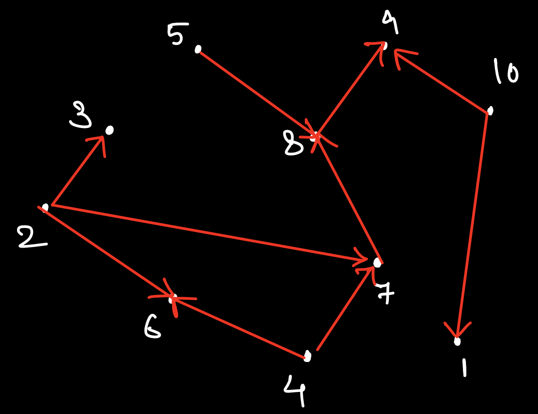
→ Cyclic vs acyclic ?
↳ Optional.

directed Graph

\# of Nodes (N), # of Edges (E)

N = 10, E = 10

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



Store a graph

1. Adjacency Matrix

u — v

⇒ Undirected.

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

int mat[6][6]

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

mat[i][j]
- 1 ⇒ Edge b/w i & j
- 0 ⇒ No Edge b/w i & j

N nodes ⇒ mat[N+1][N+1]

TC : $O(E)$

SC : $O(N^2)$

↳ N = 1000 | SC : 10^6 ⇒ Lot of space wastage.

→ E = 5000

$u \xrightarrow{w} v$

$(w \neq 0)$

	Unweighted	Weighted.
Undirected	m[u][v] = 1 m[v][u] = 1	m[u][v] = w m[v][u] = w
Directed	m[u][v] = 1	m[u][v] = w

Adjacency List

list<int> g[6];

↳ array of list of int of size = 6.

Undirected

N	E	w
5	7	
1	4	10
2	5	
3	2	
4	3	
2	4	
3	5	
1	2	

0

1 → 4, 2

2 → 5, 3, 4, 1

3 → 2, 4, 5

4 → 1, 3, 2

5 → 2, 3

g[i] ⇒ list of int

TC : O(E)

SC : O(E)

Undirected ⇒ # of Entries = 2E

directed ⇒ # of Entries = E

* Undirected Weighted Graph

$\text{List}\langle \text{pair}\langle \text{int}, \text{int} \rangle \rangle \; g[\underbrace{N+1}_{6}];$

N=5

$u \xrightarrow{w} v$

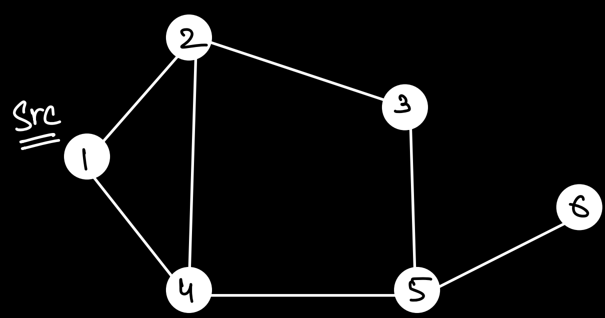
u	v	w
1	2	10
2	4	6
3	5	4
1	5	8

- 0
- 1 → {2,10}, {5,8}
- 2 → {1, 10} ,{4,6}
- 3 → {5,4}
- 4 → {2, 6}
- 5 → {3,4}, {1,8}

	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})

Q: Given an undirected graph, a source node & a destination node. Check if destination node can be visited from source node.

S : 1 , D : 6



N = 6, E = 7

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

⇒ Adjacency list

list<int> g[7]

0
1 → 2, 4
2 → 1, 4, 3
3 → 2, 5
4 g[4] → 1, 2, 5   g[4][0]
5 → 3, 6, 4
6 → 5

S = 1

DS: ~~1~~ ~~2~~ ~~4~~ ~~3~~ ~~5~~ 6

Operations
- insert from rear
- delete from front

⇒ QUEUE

bool Vis[N+1] = {false}

Vis[7]:

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

⇒ Breadth first Search

⇒

~~1~~ ~~2~~ ~~4~~ ~~3~~ ~~5~~ ~~6~~

0	
1	→ 2, 4
2	→ 1, 4, 3
3	→ 2, 5
4	→ 1, 2, 5
5	→ 3, 6, 4
6	→ 5

Vis[7]:

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

⇒ return vis[dest] → T / f

```
bool bfs(N, E, u[], v[], src, dest) {
    list<int> g[N+1];
    for(i=0; i<E; i++) {                    TC: O(E)
        // u[i] —— v[i]                     SC: O(E)
        g[u[i]].add(v[i]);
        g[v[i]].add(u[i]);
    }
    Queue<int> q;
    q.enqueue(src)
    bool vis[N+1] = {f};
    vis[src] = true;
    int level[N+1]; level[src] = 0
    int parent[N+1]; parent[src] = -1;
    while(q.size() > 0) {
        int cu = q.front();
        q.dequeue();
        // Iterate over Adj. list of cu.
        for(i=0; i<g[cu].size(); i++) {
            int cv = g[u][i];
            if (vis[cv] == false) {
                vis[cv] = true;
                q.enque(cv);
                level[cv] = level[cu] + 1;
            }   parent[cv] = cu;
        }
    }
    return vis[dest];
}
```

=

TC : O(E) | SC : O(E + N) E >> N

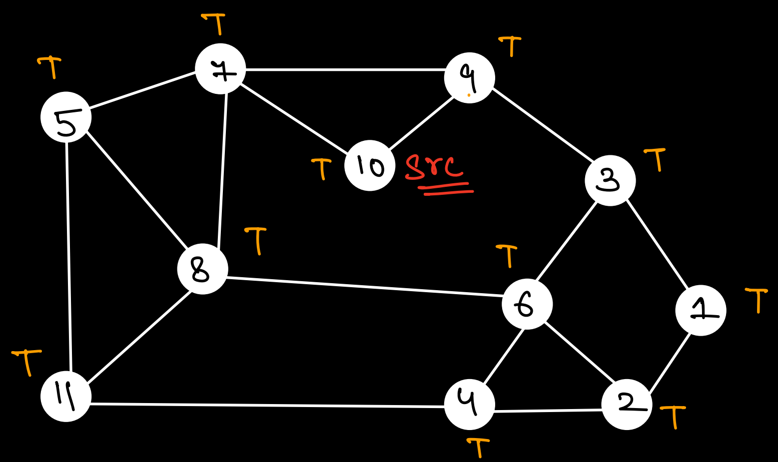
↳ ≃ O(E)

CN	# of iterations.
1	g[1]
2	g[2]
3	g[3]
⋮	⋮
N	g[N]

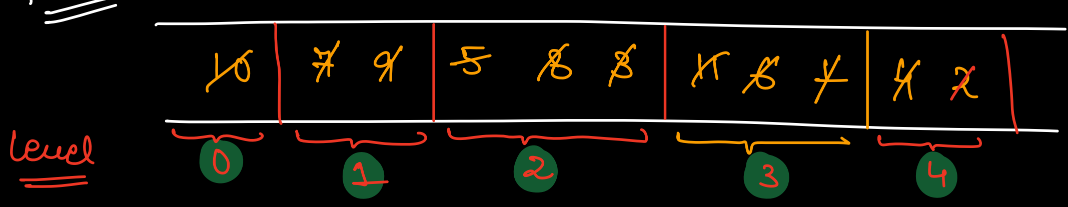
g[1] + g[2] + --- g[N] ⇒ O(E)

Ex:

S : 10 , D = 2



Queue



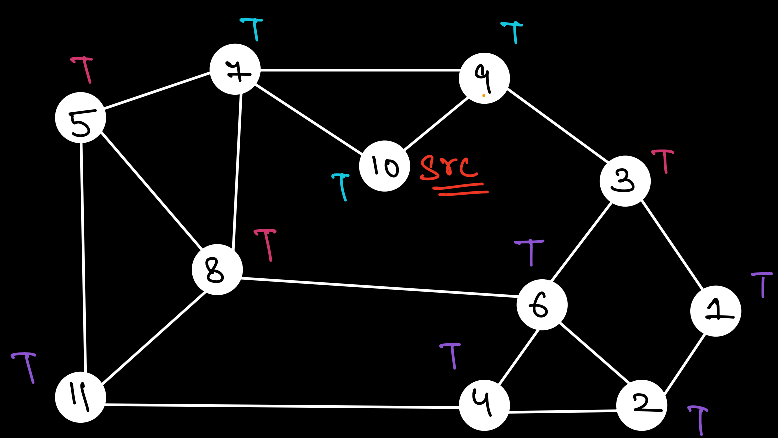
level ≡ Min path in terms # of Edges.

⇒ BFS algo also gives us the shortest path from src to dest in Unweighted graph.

```
int level[N+1];
level[src] = 0
```

Ex:

S: 10, D=2



int parent[12] =

0	1	2	3	4	5	6	7	8	9	10	11
×	3	6	9	11	7	8	10	7	10	-1	5

~~10~~ ~~7~~ ~~9~~ ~~5~~ ~~8~~ ~~3~~ ~~11~~ ~~6~~ ~~1~~ ~~2~~

dest = 2, src = 10

int parent[12] =

0	1	2	3	4	5	6	7	8	9	10	11
×	3	6	9	11	7	8	10	7	10	-1	5

dest = 2, src = 10

$2 \leftarrow 6 \leftarrow 8 \leftarrow 7 \leftarrow 10$

S: 10, D: 11.

$11 \xrightarrow{} 5 \xrightarrow{par[11]} 7 \xrightarrow{par[5]} 10 \xrightarrow{par[7]} -1$ par[10]

1. fill the parent[]
2.

```
list<int> path;
while(d != src){
    path.add(d)
    d = parent[d];
```

→ O(N)

3

— * —

Jun-2022

Announcem

↓