- Introduction
- Terminology
- DFS
- BFS

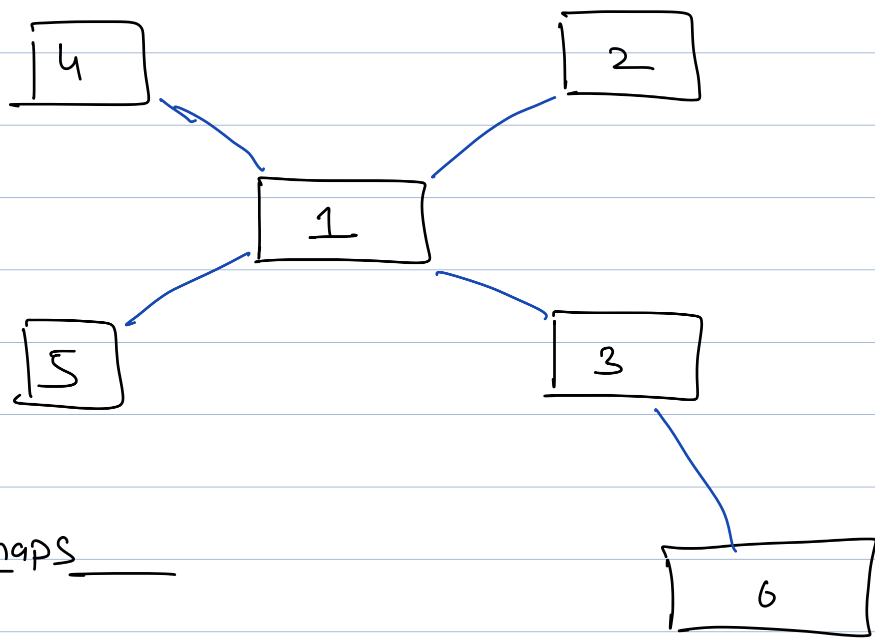
Graphs ⇒ Network of entities.

Nodes → Vertices

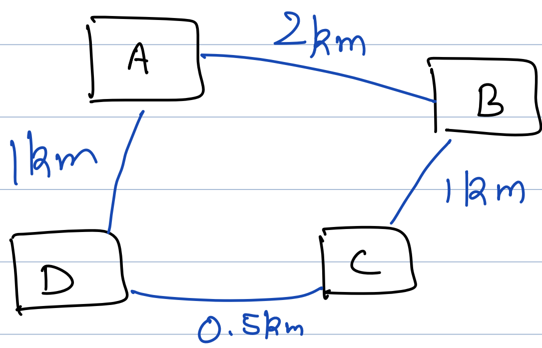
Edges.

Nodes ⇒ A physical/virtual Entity.
Edges ⇒ Depicts connections between nodes

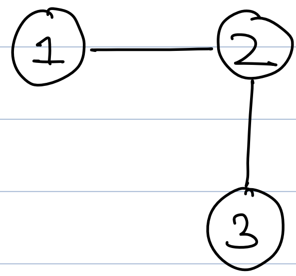
Social Media



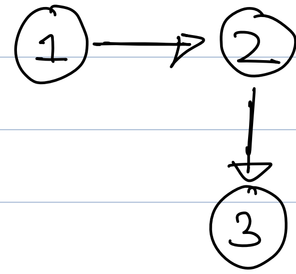
Google maps



Directed & Undirected Edges

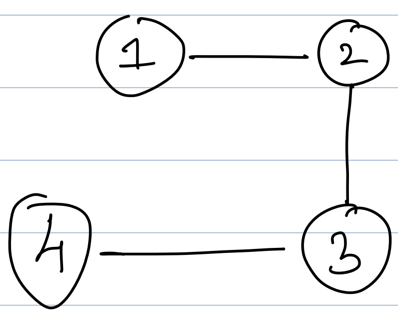


Undired Edges

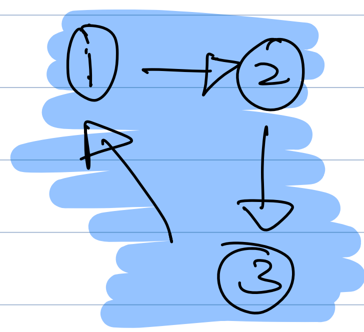


Directed Edges

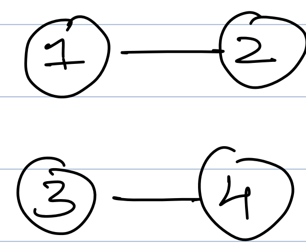
Connected & Disconnected Graphs



Connected graphs



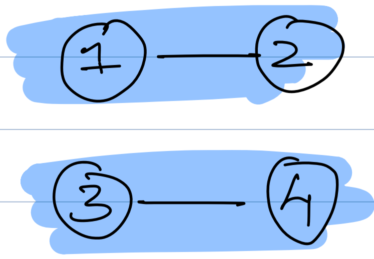
Connected



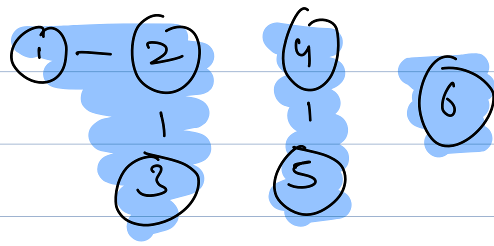
Disconnected

Every node should reachable from every othe node.

Connected Components (Undirected Graphs)

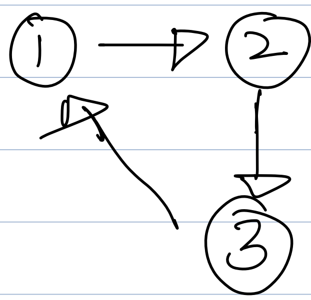


⇒ 2 Connected Components.

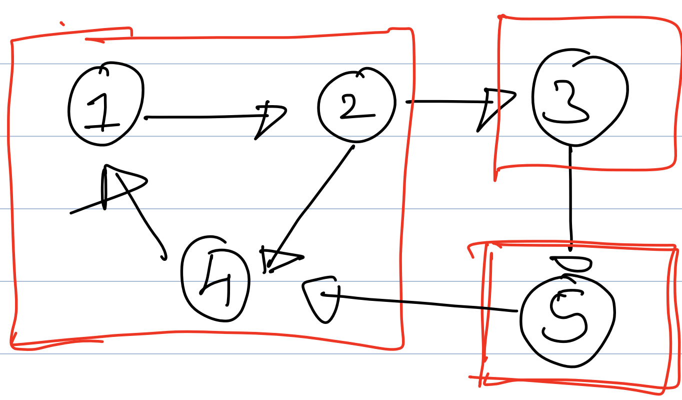


⇒ 3 Connected Components

Strongly Connected Components (Directed Graphs)

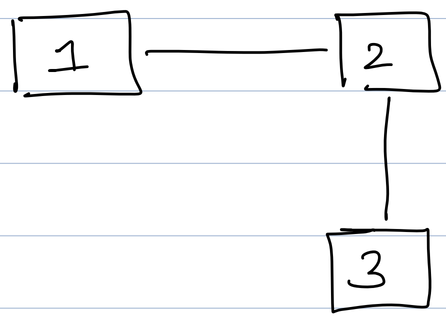


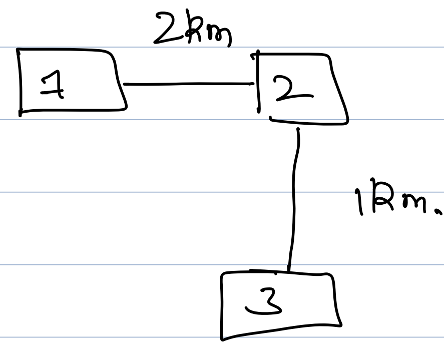
⇒ 1 SCC



⇒ 3 SCC

Weighted & Unweighted Edges

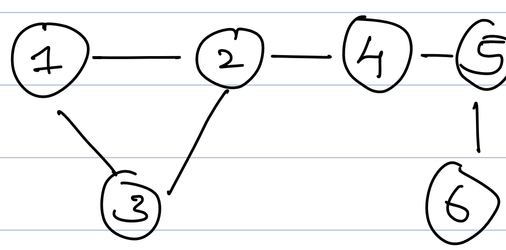


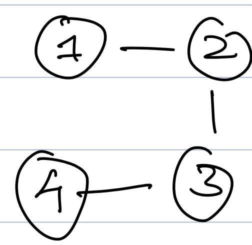


Unweighted Edges

Weighted Edges

Cyclic & Acyclic graphs



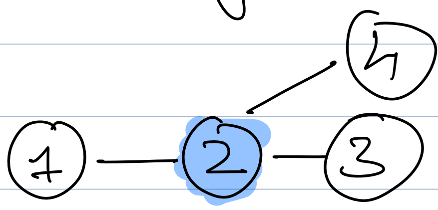


Cyclic graphs

Acyclic graphs

Degree of a node

Undirected graph

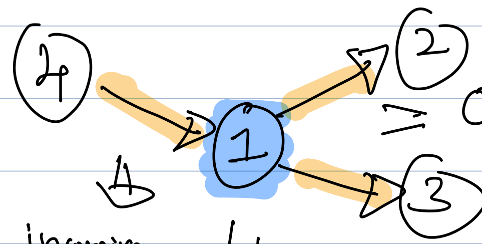


$\Downarrow$

deg = 3

Degree of a node is the number of neighbouring nodes it is connected to

Directed graph



Outgoing edge.

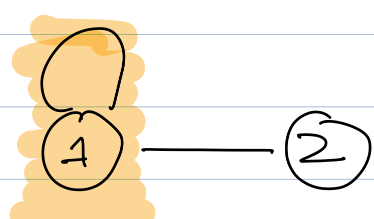
incoming edge

$\Downarrow$

Indegree = 1

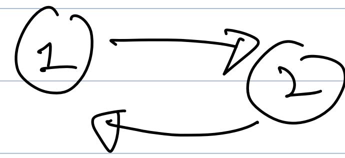
Outdegree = 2

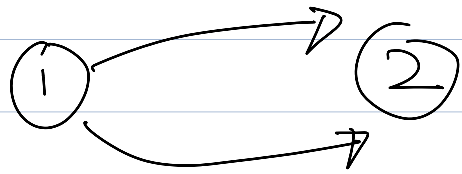
Self loop



Multi edges

⇒ multiple edges doing the exact same job.

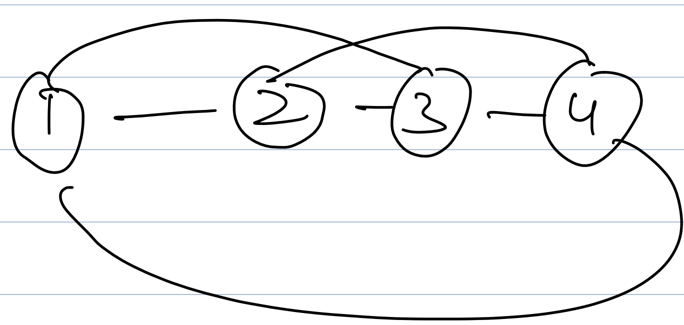




Not a multi-edge

Simple Graphs

Graphs which have no self loop and multi edges



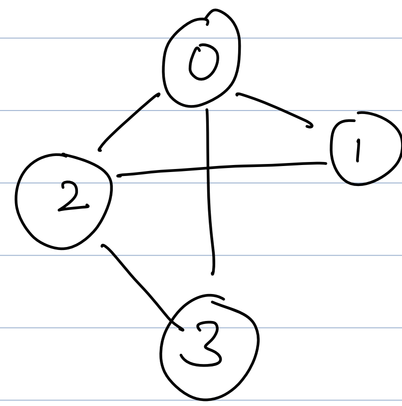
Input Structure for graph Question.

1) No of nodes = 4 = {0, 1, 2, 3}

2) No of edges = 5

3) Definition of edges.

0	1
2	3
1	2
0	3
0	2



Adjacency Matrix

int adj [n] [n]

$n \Rightarrow 4$
$e \Rightarrow 5$

	0	1	2	3
0		1		
1	1			
2				
3				

0	1
2	3
1	2
0	3
0	2

if there is an edge between n_1 & n_2.

$mat[n_1][n_2] = 1$
$mat[n_2][n_1] = 1$

Undirected

$mat[n_1][n_2] = 1$

directed

$SC: O(n^2)$

Pros

1) We can check whether 2 nodes are connected in $O(1)$;
2) Insertion is $O(1)$

Cons.

$1 \rightarrow 2 \rightarrow 3 \qquad 4 \rightarrow 5$

Not space efficient

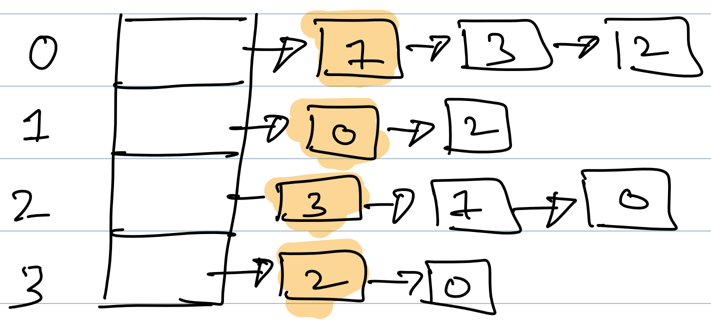
Adjacency list

list <int> adj [n];

$n \Rightarrow 4$

$e \Rightarrow 5$

0	1
2	3
1	2
0	3
0	2



↳ Total Space = $N + 2E$

if there is an edge between n_1 & n_2.

adj [n_1].push (n_2);
adj [n_2].push (n_1)

Undirected

adj [n_1].push (n_2);

directed

SC: $O(N + E)$

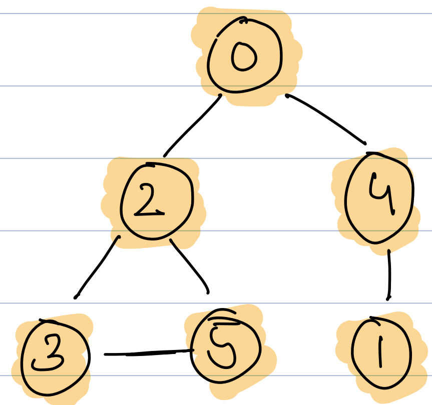
: $O(V + E)$

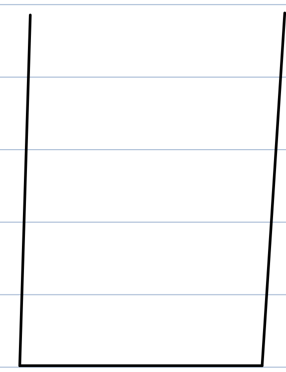
SC: $O(N + E)$

$O(V + E)$

Pros	Con
1) Space efficient	1) Direct connection b/w 2 nodes cannot be identified in O(1).
2) O(1) insertion.

DFS [Depth first Search]





0, 2, 3, 5, 4, 1

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

int visited[V];

Pseudo Code !

```
void dfs (int node) {

    visited [node] = 1
    print (node);

  for (int i=0 ; i< adj (node].size ; i++) {

        int neigh => adj [node] [i];
        if ( ! visited (neigh] )
                    dfs (neigh);
  }

}
```

Total method calls = V.

$N_1 + N_2 + N_3 + N_4 \ldots N_V = 2$

no of iteration $= O(V + 2E)$ [Undirected Graph]

no of iteration $= O(V + E)$ [Directed Graph]

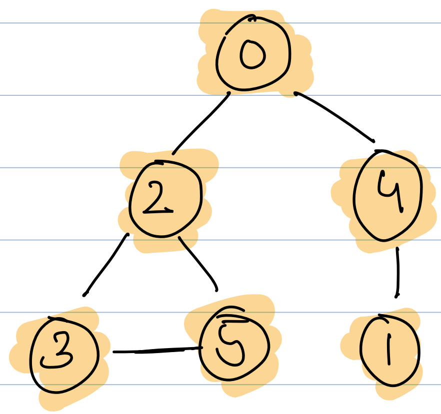
TC : $O(V+E)$

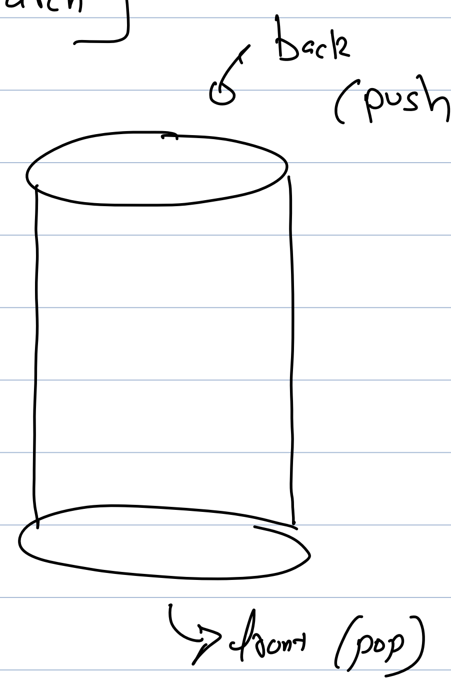
SC : Stack space + Adjacency list + Visited array

$O(V)$ $\quad$ $O(V+E)$ $\quad$ $O(V)$

$$SC = O(V+E)$$

BFS [Breadth First Search]





0, 2, 4, 3, 5, 1

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

Pseudo Code !

```
void bfs (int node) {

    Queue <int> q;
    q.push (node);
    visited [node] = 1;

    while (!q.empty()) {

        int n => q.front();
        q.pop();
        print (n);

        for (int i=0 ; i< adj [n].size(); i++) {
            int neigh => adj [n] [i];

            if (!visited [neigh])
                q.push (neigh);
                visited [neigh] = 1;
        }
    }
}
```

TC : $O(V+E)$

SC : $O(V+E)$ → Queue = $O(V)$; Adjacency list = $O(V+E)$; visited = $O(V)$

$$\boxed{n_1 \rightarrow n_2}$$

$$n_1 - x_1 - x_2 - n_2$$