# Welcome ☺

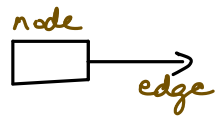
Agenda: Graph
Basics
Store
Traverse
1 quesⁿ

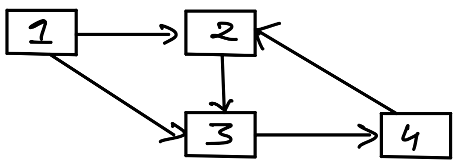
Contest Topic → until DP

---

## Graphs

⇒ Graph is a collection of nodes & edges.



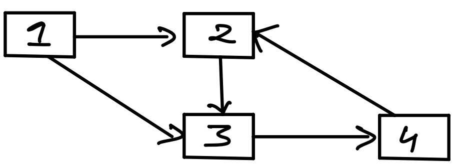




# nodes = 4

# edges = 5

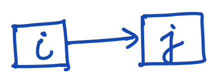
How graphs are stored ?

1) Adjacency Matrix



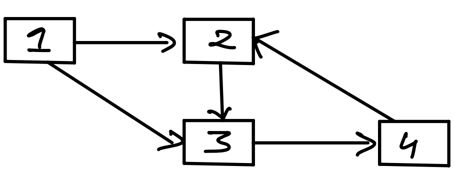

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 |

$S.C = O(N^2)$

A[i][j] → 1 , 


→ 0 , no edge.

2) Adjacency List



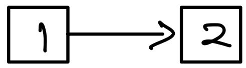

1 ⟶ { 2, 3 }
2 ⟶ { 3 }
3 ⟶ { 4 }
4 ⟶ { 2 }

SC $O(N+E)$

List of list
Array of list

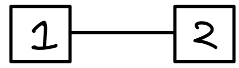
# Properties of Graph

1) Directed

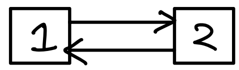


travel only from 1 to 2

Undirected.

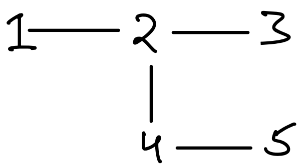


Same as.

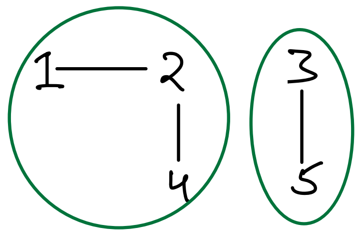


travel from 1 to 2 & from 2 to 1

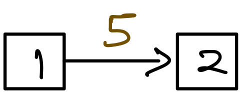
2) Connected



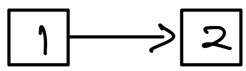
Disconnected.



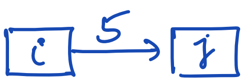
3) Weighted



Unweighted.



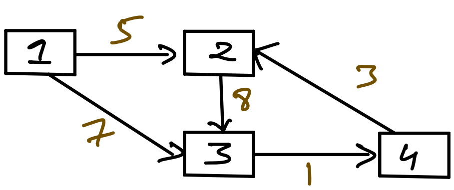
A[i][j] → 5 (>0)



→ 0 no edge

A[i][j] => weight over edge i → j.

Adj[i] → list of pairs ( j, wt)



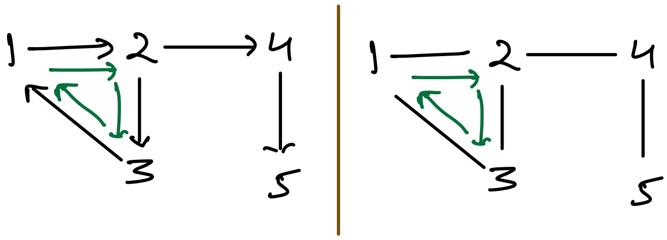
1 → { (2,5) , (3,7) }

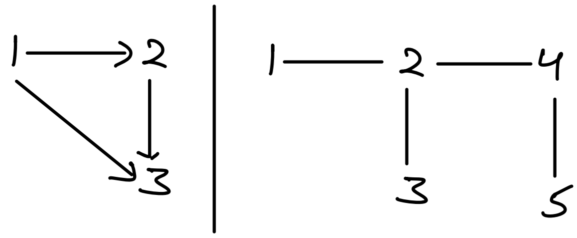
2 → { (3,8) }

3 → { (4,1) }

4 → { (2,3) }

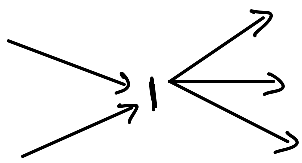
4) Cyclic Acyclic

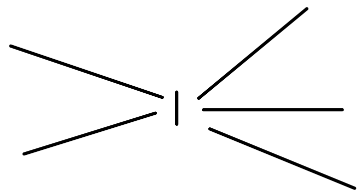



Undirected graphs → cycle of min. 3 nodes will be considered.

5) Indegree / Outdegree. Degree

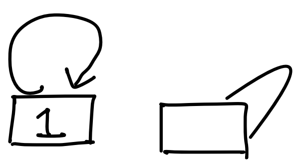


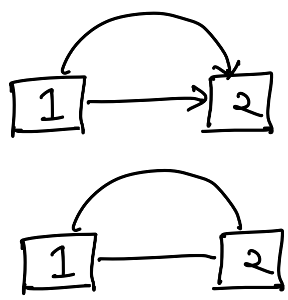




in[1] = 2
out[1] = 3

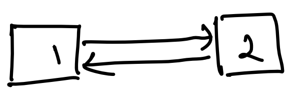
degree[1] = 5

6) Simple Graph → connected graph without self edge. & multiedges.



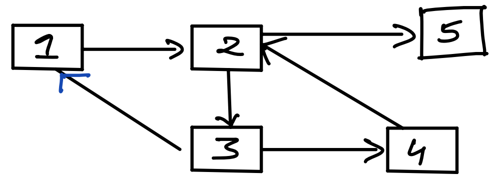




not multiedge



# Traversal

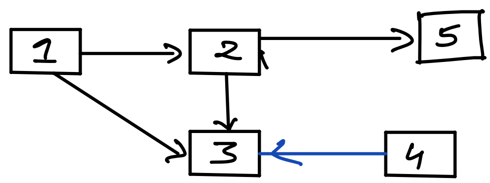
1) Depth First Search.



→ Go deep till it is possible. Once a path is completed, backtrack and try alternate path.

1) Travel all the nodes only once.
2) Keep track of visited nodes.
3) Check if all nodes are travelled before exit.

```
∀i  vst[i] = false.
for ( i → 1 to N)
{
    if ( ! vst[i])   dfs(i)
}
```



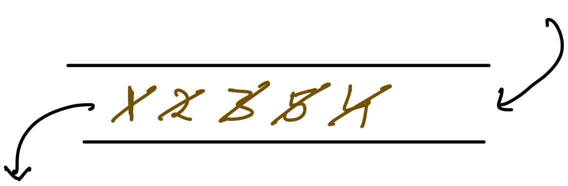
```
void dfs ( u)
{
    vst[u] = True
    print(u)
    for ( v : adj[u])..
    {
        if ( ! vst[v])   dfs(v)
    }
}
```

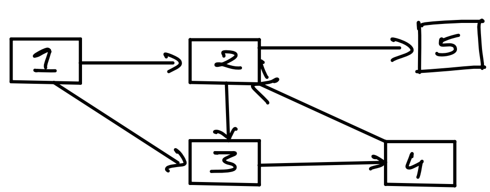
visited    recursion

S.C → O(N + N)

T.C → O(N + E)

2) Breadth First Traversal (BFS)





O/p   1 2 3 5 4

$\forall i$ vst[i] = false.
for ( i → 1 to N)
{
    if ( ! vst[i]) bfs(i)
}

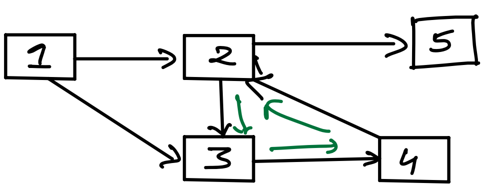
1) Travel all the nodes only once.
2) Keep track of visited nodes.
3) Check if all nodes are travelled before exit.

```
void bfs(u)
{
    vst[i] = True
    q.enqueue(u)
    while (!q.isEmpty())
    {
        x = q.dequeue()
        print(x)
        for ( v : adj[x])
        {
            if ( ! vst[v]) {
                q.enqueue(v)
                vst[v] = true
            }
        }
    }
}
```

T.C → O(N + E)

S.C → $O(\underbrace{N}_{\text{visited}} + \underbrace{N}_{\text{queue}})$

**Q** Check if a simple directed graph has a cycle or not ?




Ans = True

If a visited node is travelled again ⟹ cycle ✗

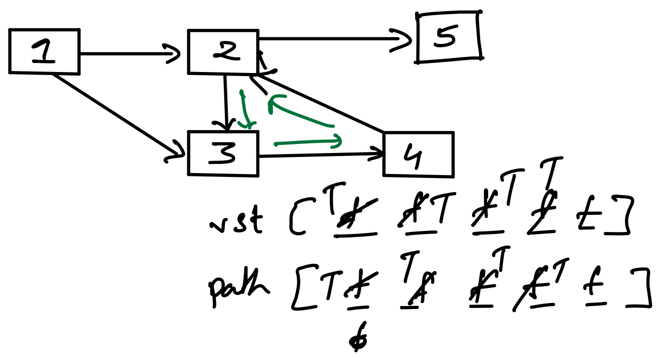
If a visited node in same path is travelled again ⇒ cycle ✓

travel a path ⇒ <u>DFS</u>

<u>Code</u>

```
∀i vst[i] = false.
for ( i -> 1 to N)
{
    if ( !vst[i] && dfs(u) ) return True.
}

bool dfs (u)
{
    vst[u] = True
    path[u] = True
    for ( v : adj[u] )..
    {   if (path[v] == True) return True.
        if ( !vst[v]) {
            if (dfs(v))   return True.
        }
    }
    path[u] = False.
    return false.
}
```




T.C → $O(N+E)$

S.C → $O(N+N+N)$