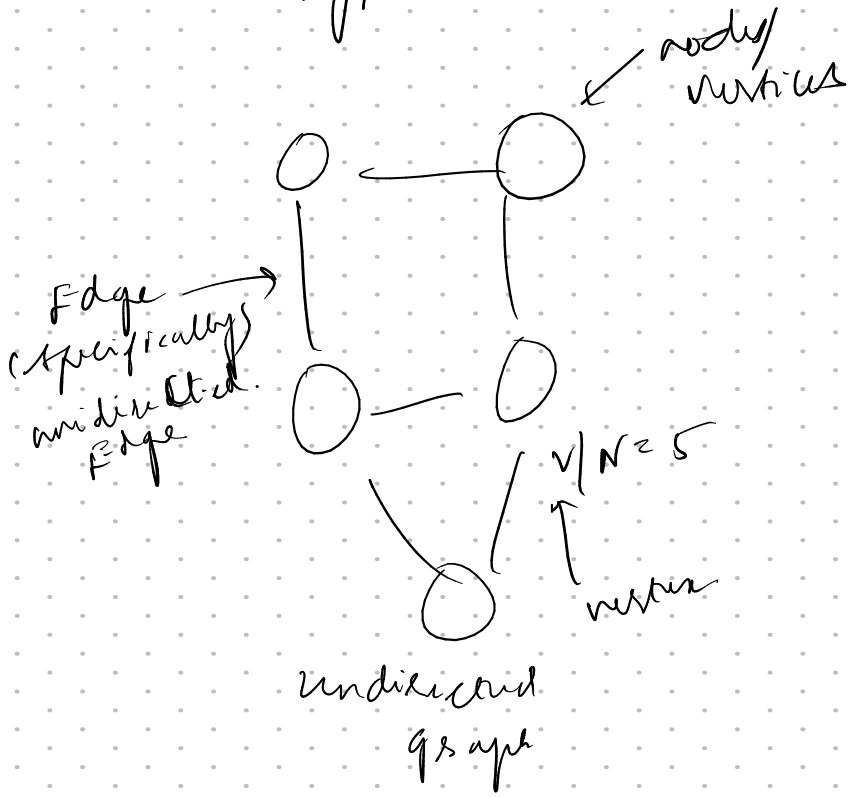
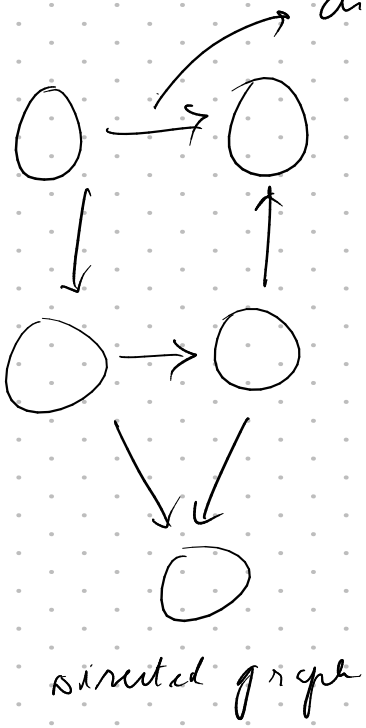
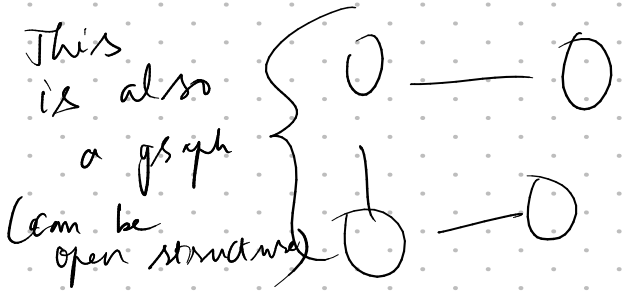
Graph

Types conventions used





Cycles in a Graph.



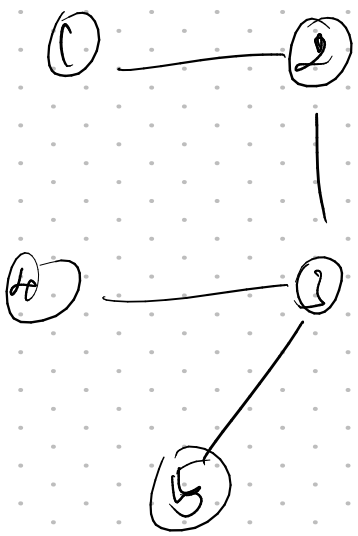
cycle: start from a node
end at that node itself.

Undirected Edge → path / edge
in both
the ways
u → v
v → u

- If there is a single cycle in the graph
- Then it is called as the undirected cyclic graph.
- If the cycle can't be formed then it is a acyclic graph. (DAG) → directed Acyclic graph

path → contain a lot of nodes (vertices) and each of them are reachable

Eg:



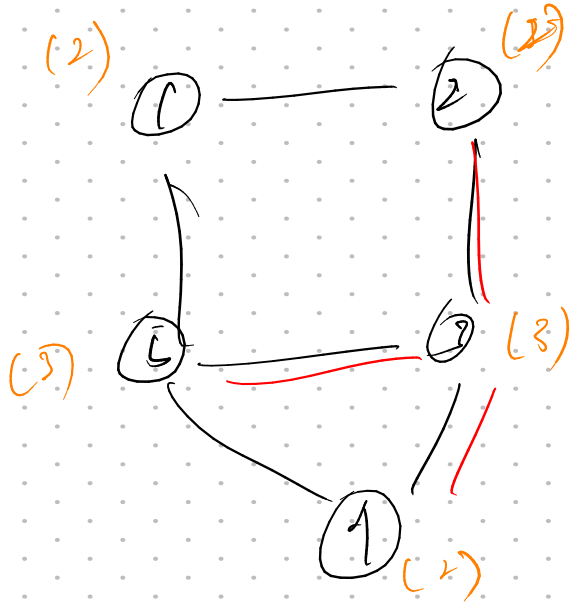
Path: $1 \to 2 \to 3 \to 5$

But $1 \to 2 \to 3 \to 2 \to 1$ is not a path as the 2 is appearing more than once.

- Node cannot appear twice in a path.
- The adjacent nodes must have the edge.

Degrees in Graph

- Degree is the number of incoming & outgoing edges of the graph. (undirected graph)
- Eg:



$D(3) = 3$ (In this case)

(i)

Property :- Total Degree of the graph is $2 \times E$ of a graph.

Reason :- Because every edge is associated with 2 nodes

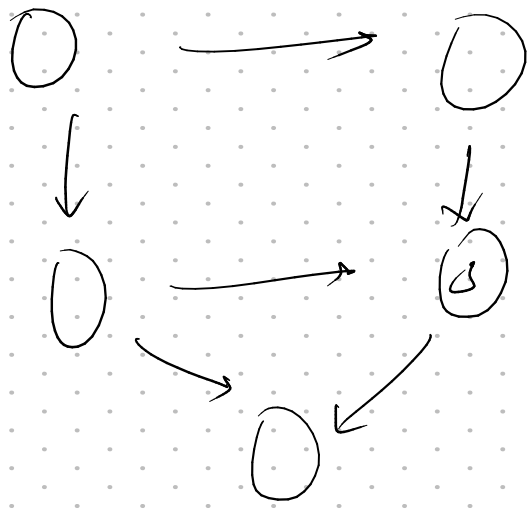
Total degree manually : $2+2+3+3+2 = 12$

Total degree using $2 \times E \Rightarrow 2 \times 6 = 12$ $\therefore E = 6$

Directed graph

Indegree: No. of incoming edges

outdegree: No. of outgoing edges.



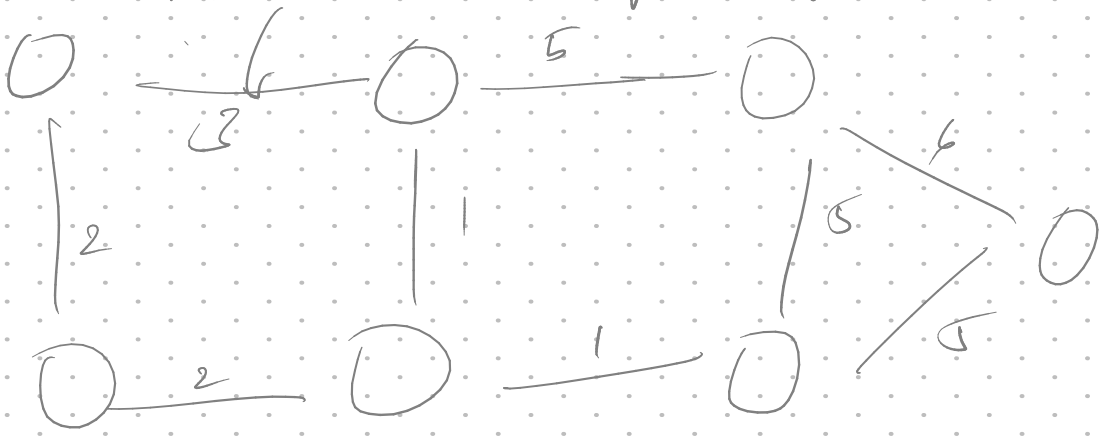
Indegree (3) = 2

outdegree (3) = 1

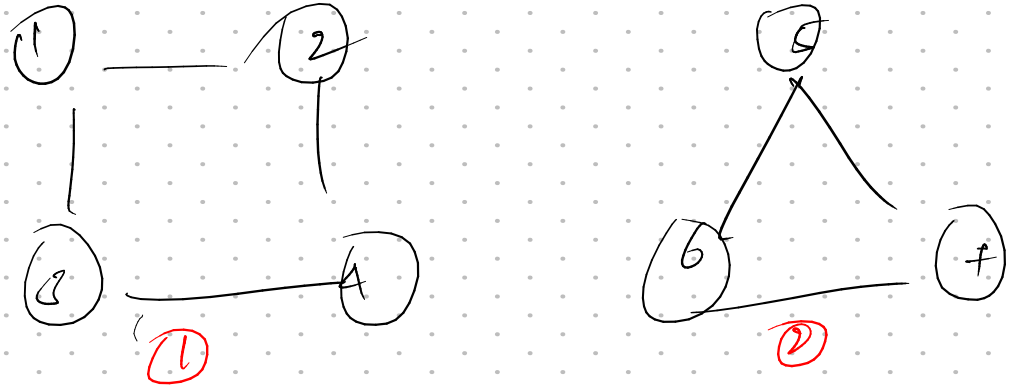
Edge weights:

- Every edge will have the weights.
- If the weights aren't assigned then we assume Edge weight = 1 (unit weights).

These are the edge weights.



Connected Components of the graph.



N = 10 E = 8 Edges.

- This the graph having the 4 components (but they are separate not connected).
- we will use the visited array for the graph traversal.

vis = | F | F | F | . . . . . | F | N = 11

0 1 2 10

```
for i: 1 ⟶ 10
{
    if (!visited (vis [i])
        traversal (i)
}
```

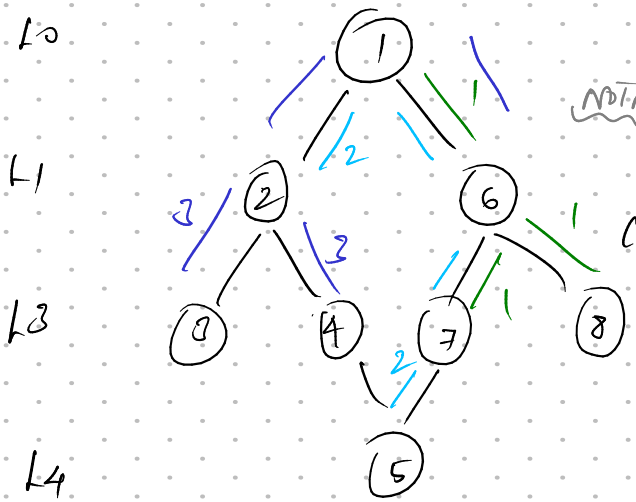
Just calling the traversal () doesn't visit all the vertices if the graph has the components that aren't connected.

Graph Traversal Techniques.

i) BFS
ii) DFS.

[NOTE: There can be 0-based indexing or 1-based indexing]

BFS (Breadth first search):



Breadth → Level-wise

NOTE: The nodes that are chosen in each level doesn't matter as long as we visit every node.

Traversal: 1, 6, 2, 8, 7, 4, 3, 5
(starting node: 1)

(starting node: 6)

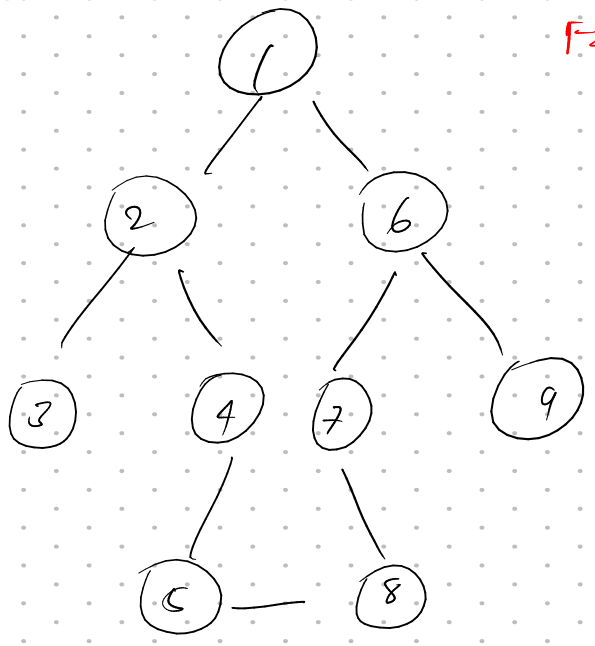
Traversal: 6, 1, 7, 8, 2, 5, 3, 4

Example:

BFS (use Queue)

↓

FIFO



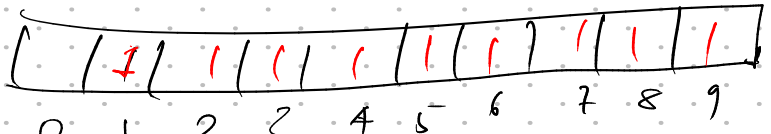
Adjancy List

0 → { }
1 → {2, 6}
2 → {1, 3, 4}
3 → {2}
4 → {2, 5}
5 → {4, 8}
6 → {1, 7, 9}
7 → {6, 8}
8 → {5, 7}
9 → {6}.

Starting node = 1

Create an visited array of n+1

Visited



Traversal.

1, 2, 6, 3, 4, 7, 9, 5, 8

Levels: 1 2 0 4.

Took out 1, go to its neighbour then mark as visited.

Complexity Analysis.

i) Space complexity → O(3n) ≈ O(N)

adj list, visited, Queue

ii) Time Complexity

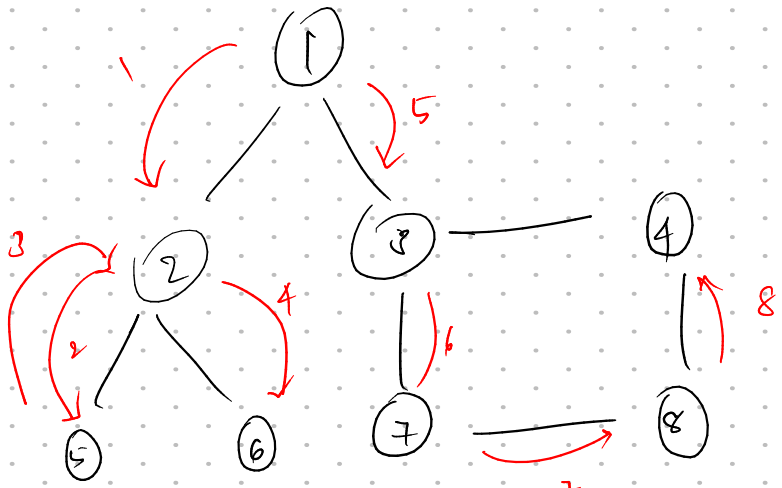
Everytime a node goes into the BFS.

for queue O(N) + O(2E) → Total degree.

N → no. of vertices/nodes — Edges.

DFS - Depth First Search.

NOTE: Use Recursion for DFS



Starting = 1

Inversal = 1, 2, 5, 6, 3, 7, 8, 4

Starting = 3.

Inversal = 3, 4, 8, 7, 1, 2, 5, 6

Adj List

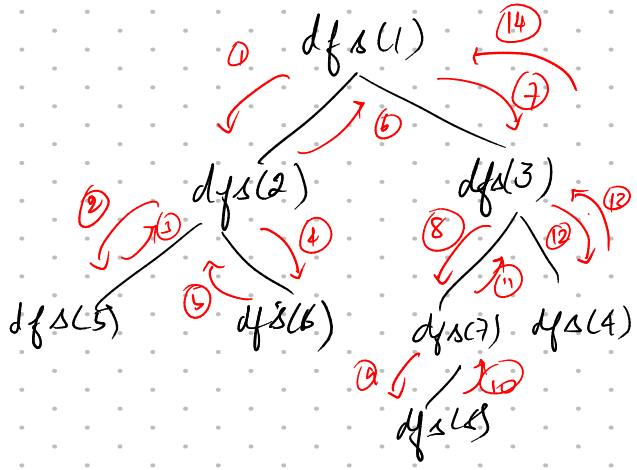
1 → { 2, 3 }
2 → { 1, 5, 6 }
3 → { 1, 7, 4 }
4 → { 3, 8 }
5 → { 2 }
6 → { 2 }
7 → { 3, 8 }
8 → { 4, 7 }

Initial steps:

visited

0	1	2	3	4	5	6	7	8
0	1	0	0	0	0	0	0	0

```
dfs( node)
{
   vis[node] = 1
   list.add(node)
   for (auto it : adj[node])
   {
      if (!visited[it])
         dfs[it]
   }
}
```



DFS Inversal: 1, 2, 5, 6, 3, 7, 8, 4.

Analysis:

space: $O(N) + O(N) + O(N)$ ← worst case for the recursion if the tree is skewed.

- $O(N)$ → dfs_res
- $O(N)$ → visited array

≈ Approx $(O(N))$

Time: $O(N) + O(2E) \simeq O(2E)$

- $O(N)$ → no. of vertices
- $O(2E)$ → edges.