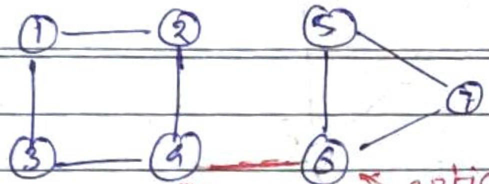
Graphs :

~~1~~ . ~~2~~ Non-Connected



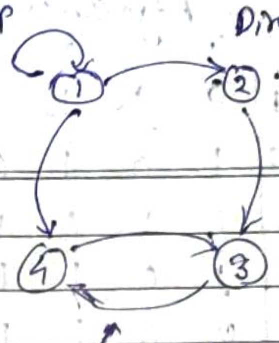
articulation point

2 components

If we add this edge then it will become connected graph.

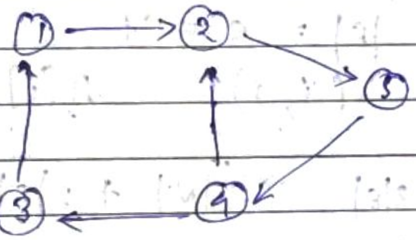
self loop

Directed graph



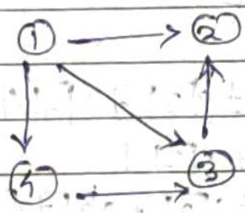
indegree 2
out degree 1

Parallel edges



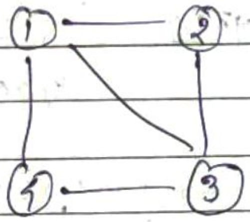
Strongly Connected

~~for~~ from every vertices we can ~~read~~ reach all the other vertices so it is called Strongly connected



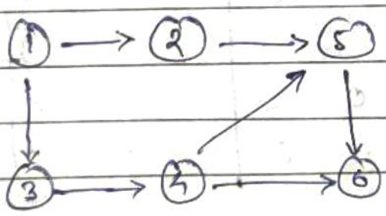
simple di graph

degree 3



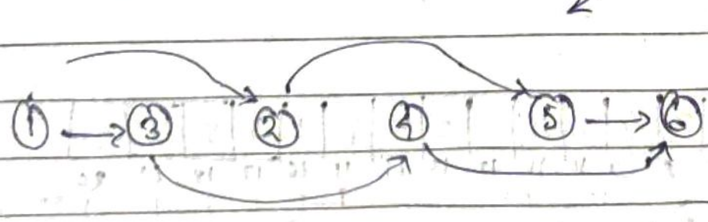
Graph / non directed ~~gra~~graph

Directed Asyclic graph (DAG)



If we are able to arrange the DAG such that the edges are going in forward direction then it is called as Topologically ordering of vertices.

Topologically ordering



~~Adjacency~~ Adjacency Matrix : (for nondirected graph)

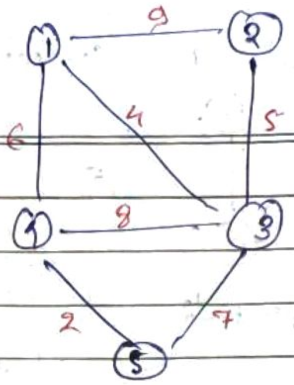
$$A = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 & 1 & 1 \\ 4 & 1 & 0 & 1 & 0 & 1 \\ 5 & 0 & 0 & 1 & 1 & 0 \end{array}$$

5×5

$n \times n = n^2$

$O(n^2)$

space



Adjacency list

A:

1 → [2] → [3] → [4/]

2 → [1] → [3/]

3 → [1] → [2] → [4] → [5/]

4 → [1] → [3] → [5/]

5 → [3] → [4/]

$G = (V, E)$

$|V| = n = 5$

$|E| = e = 7$

(i, j) $A[i][j] = 1$

$|V| + 2|E|$ → $n + 2e$

Space → $O(n + 2e)$

$|V| + 2|E| + 1$

$5 + 2 \times 7 + 1 = 20$

$20 + 1 = 21$

for compact list

Cost adjacency matrix:

$$A = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 9 & 4 & 6 & 0 \\ 2 & 9 & 0 & 5 & 0 & 0 \\ 3 & 4 & 5 & 0 & 8 & 7 \\ 4 & 6 & 0 & 8 & 0 & 2 \\ 5 & 0 & 0 & 7 & 2 & 0 \end{array}$$

A, 1 → [2|9] → [3|4] → [4|6/]

2

3

4

5

Compact list :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
/	7	10	12	16	19	/	2	3	4	1	3	1	2	3	4	1	3	5	3	4

← 21

$\underbrace{0 \ldots 5}_{\text{vertices}}$ → 1 vertice (7–9), 2 (10–11), 3 (12–15), 4 (16–18), 5 (19–20)

The vertices 1 starting from index 7

The vertices 2 starting from index 10

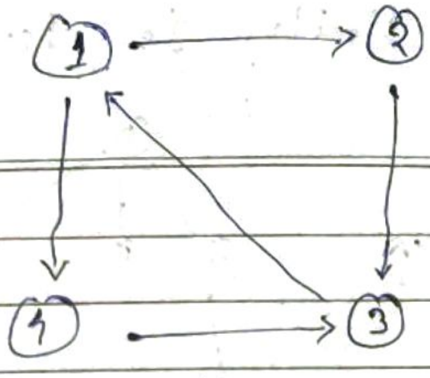
Space $|n| + 2|E|$

$n + 2e$

$n + 2n = 3n$

$= O(n)$

→ Representation of Directed graph :-



Adjacency matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

4×4

$G = (V, E)$

$|V| = 4$

$|E| = 5$

Adjacency List : (out going)

A

1 → [2|] → [4|/]

2 → [3|/]

3 → [1|/]

4 → [3|/]

$|V| + |E|$

$n + e$

$n + n = O(n)$

edges.

Inverse A.L. : (in comming)

1 → [3|/]

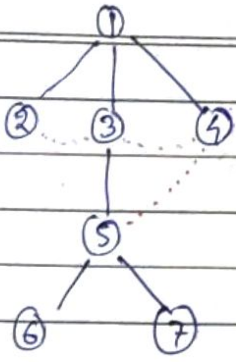
2 → [1|/]

3 → [2|] → [4|/]

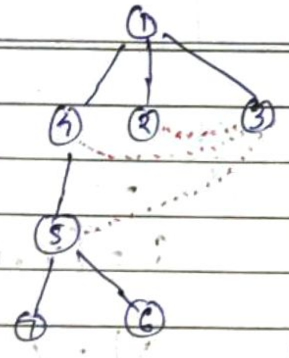
4 → [1|/]

⇒ Breadth First search (BFS):

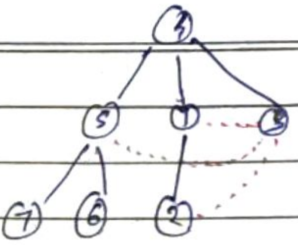
→ It is similar to level order traversal of (Binary) tree.



BFS: 1, 2, 3, 4, 5, 6, 7

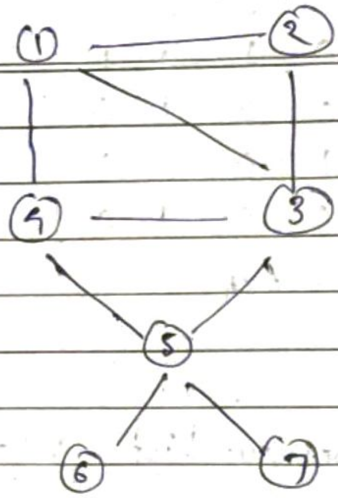


BFS: 1, 4, 2, 3, 5, 7, 6



BFS: 4, 5, 1, 3, 7, 6, 2

different starting vertices

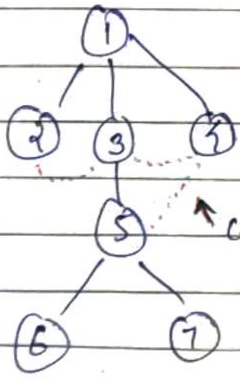


1. visiting
2. Exploring → (explor vertices adjecent to it)

BFS : 1, 2, 3, 4, 5, 6, 7

Queue : ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7

visiting vertices is importa[nt], order can be anything.



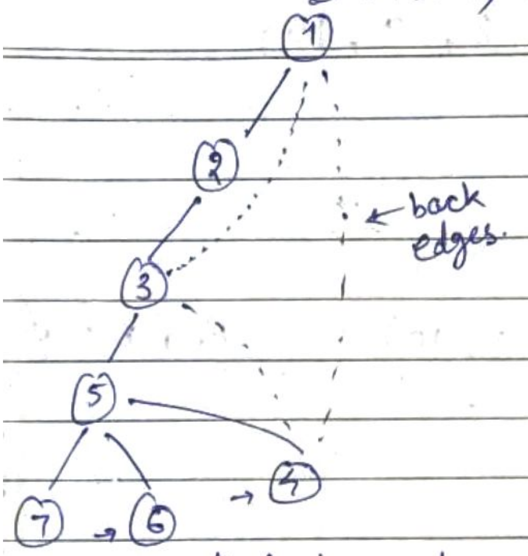
cross edges

Analytical time O(n)

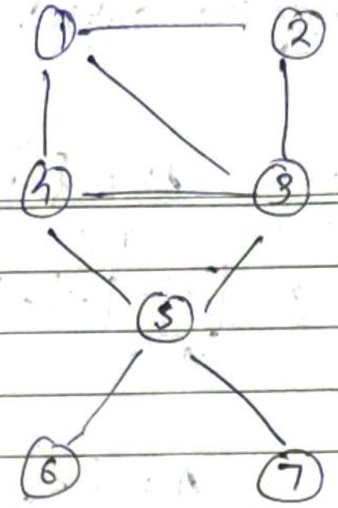
BFS spanning tree

Depth First Search (DFS) :-

DFS: 1, 2, 3, 5, 7, 6



Depth first search spanning tree



~~you~~ (we can start from any vertices)

T → O(n) (depends which DS using)

→ 1. visiting
2. Exploring

→ first ~~push to~~ visit ~~all~~ the node then explore, then push next vertices and suspend exploration on current vertices and push to stack, ~~and~~ start exploration next vertices. and go on.

→ Program ~~of~~ for DFS :

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

visited

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

```
Void DFS (int u)
{
    if (visited [u] ==0)
    {
        Printf ("%d", u);
        visited [u] = 1;
        for ( v=1; v<=n; v++)
        {
            if (A[u][v] ==1) && (visited[v] ==0)
                DFS (v);
        }
    }
}
```

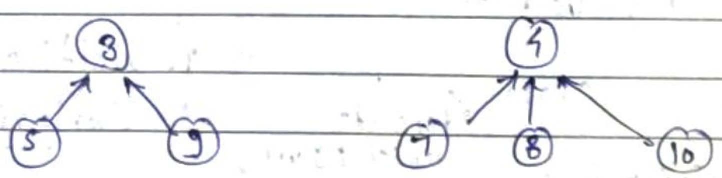
→ Disjoint subset: This is usefull in detecting the cycles in the graphs.

$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

S

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

$A = \{3, 5, 9\}$ $B = \{4, 7, 8, 10\}$ $A \cap B = \emptyset$



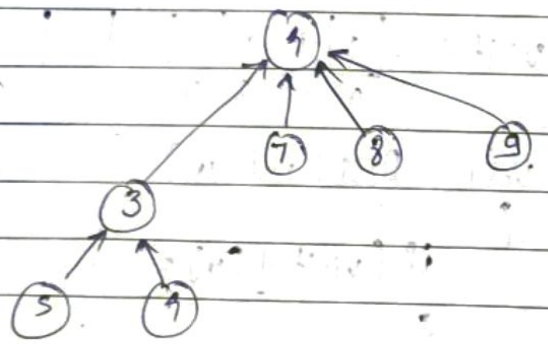
S

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

- union
- find

$A \cup B = \{3, 4, 5, 7, 8, 9, 10\}$

select ~~Root~~ Parent as the node who has more element



S

	-1	-1	4	-7	3	-1	4	4	3	4

← Parents

4 has total 7 node

10 → This Represent 10 has parent 4

```
void union (int u, int v)
{ if (S[u] < S[v])
  {
     S[u] = S[u] + S[v];
     S[v] = u;
  }
  else
  { S[v] = S[u] + S[v]
    S[u] = v;
  }
}
```

```
int find (int u)          ← find the parent.
{ int x = u;
  while ( S[x] > 0)                  u = 10   → x = 10
  {                                  → S[10] = 4
     x = S[x];              ←           S[4] = -7
  }                                          ↑ parent.
  return x;
}
```

⇒ How to find the circle :-

let's say 5 belong to set 4 and 10 belongs to set 4, if 5, and 10 have and edge between them then it will form an cycle.

So if two nodes belong to same set then don't connect them otherwise it will form the cycle.