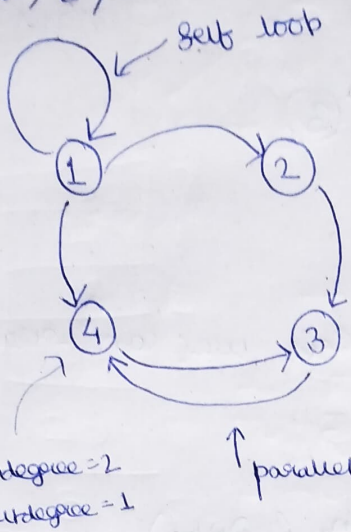
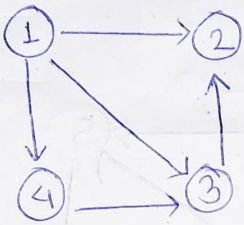
Graph

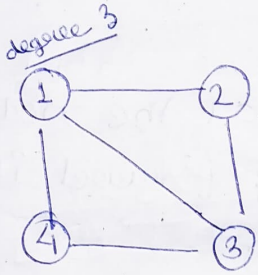
$G = (V, E)$

① 

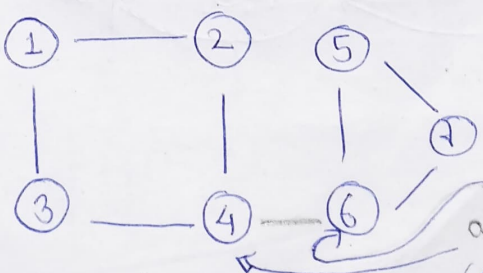
directed graph.

② 

simple graph.

③ 

graph / non-directed graph.

④ 

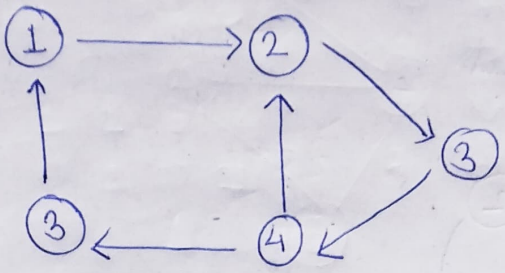
Non connected

articulation point

It is the vertex after ~~going~~ removing of which will seperate it.

2 component

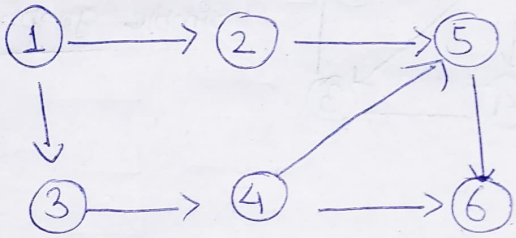
Strongly connected



- when from every vertices we can reach every other vertices.

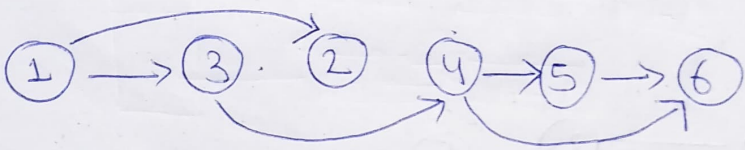
Directed acyclic graph (DAG)

It is a digraph with no cycle.



Topological ordering

If you can arrange all the nodes with pointer only pointing forward it is called Topological ordering



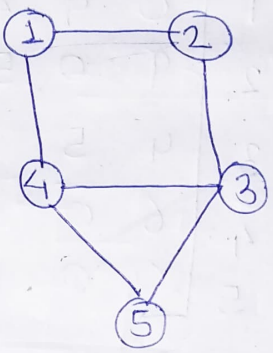
Representation of undirected graph

1. Adjacency matrix
2. Adjacency list
3. compact list

Adjacency matrix

$A =$

	1	2	3	4	5
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



5x5

$G(V, E)$

$|V| = n = 5$

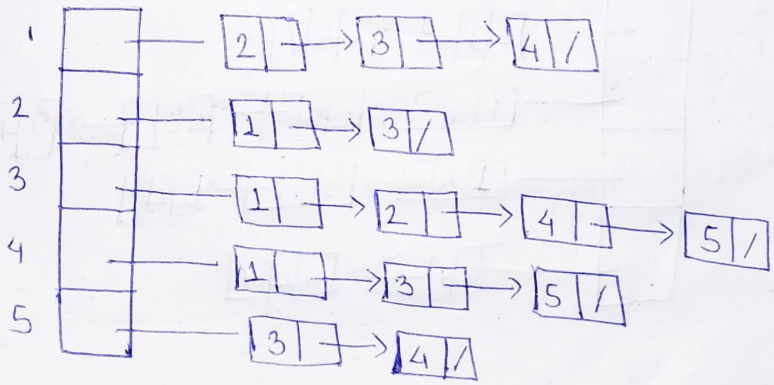
$|E| = e = 7$

$n \times n = n^2$

$O(n^2)$

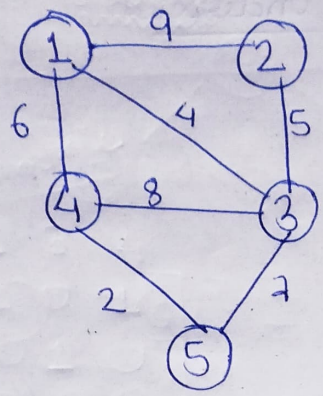
Adjacency list

A



$|V| + 2|E|$

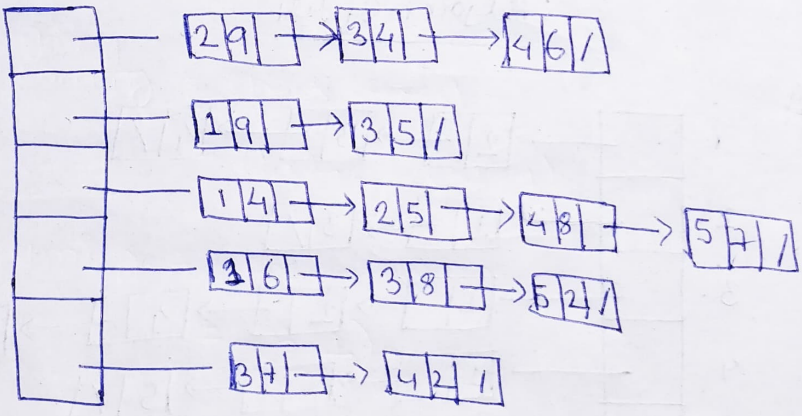
$n + 2e$



Cost adjacency matrix

	1	2	3	4	5
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

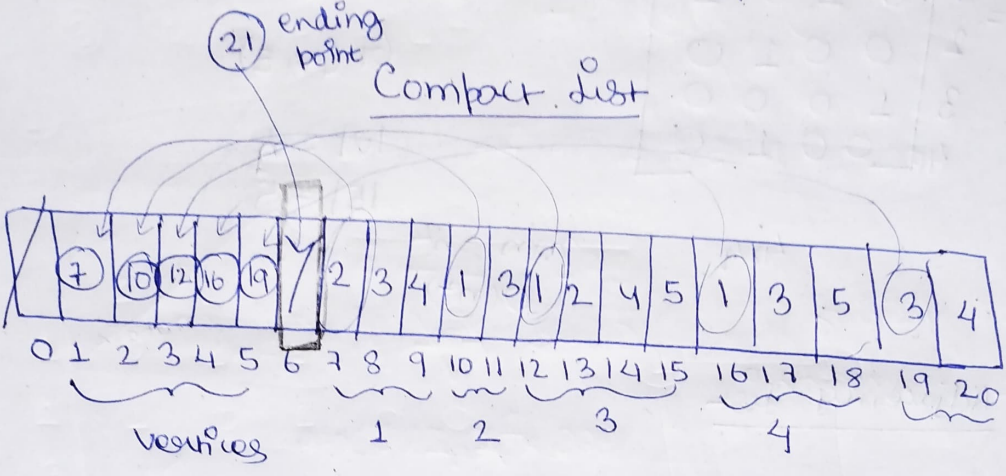
Cost adjacency ~~DD~~ list



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

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

$= 21$



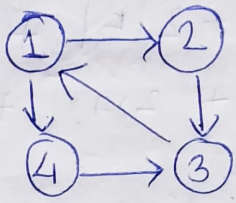
$|V| + 2|E|$

$n + 2e$

$n + 2n = 3n = O(n)$

Graphs

Adjacency matrix



G1 (V, E)

|V| = 4

|E| = 5

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

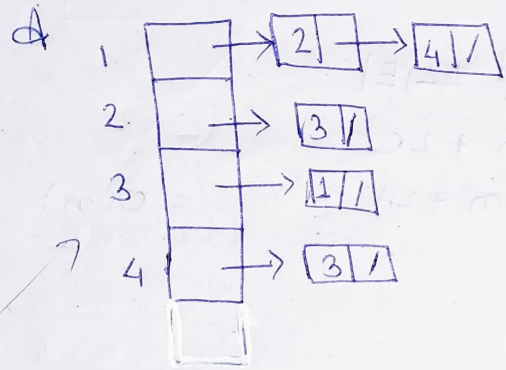
4x4

$n \times n = n^2$

$O(n^2)$

4x4 matrix.

Adjacency list



|V| + |E|

n + e

n + n = O(n)

edges going ~~coming~~ = check row

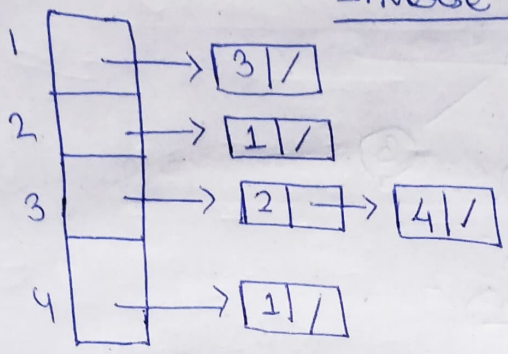
= (1, 2), (1, 4)

edges coming = check column

= (3, 1)

edges going out

Inverse adjacency list



edges comming in

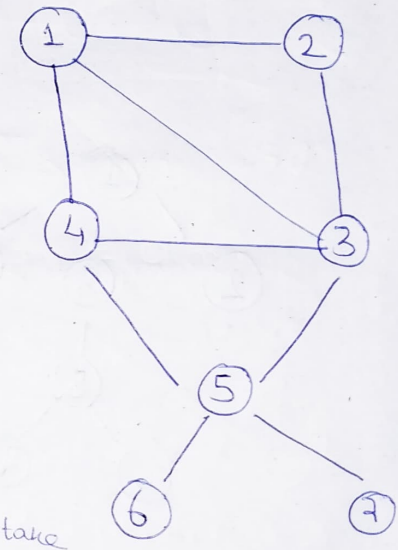
Breadth First Search (BFS)

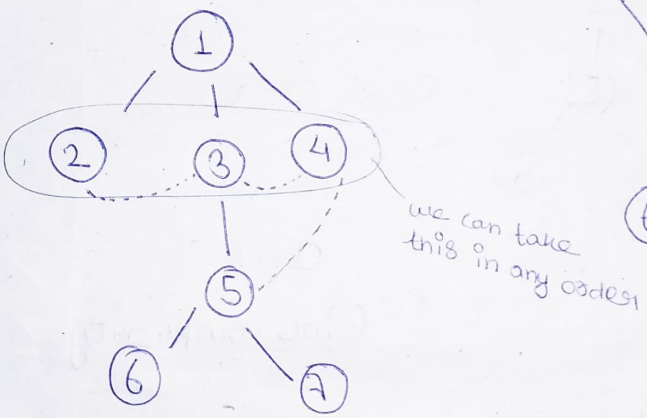
like level order of a binary tree

In BFS we use Queue data structure

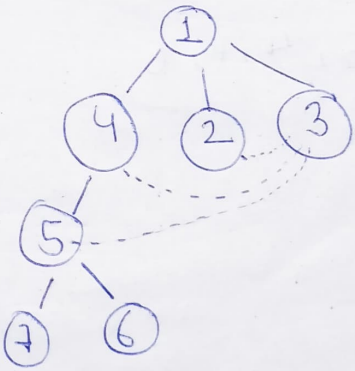
Queue

{graph without any cycle or closed edges is a tree}

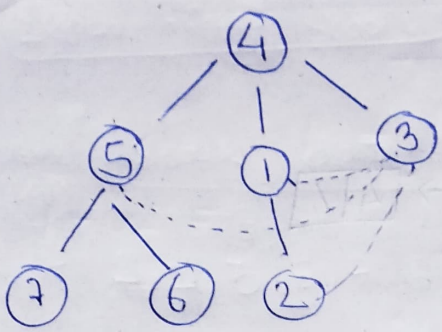




BFS: 1, 2, 3, 4, 5, 6, 7



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



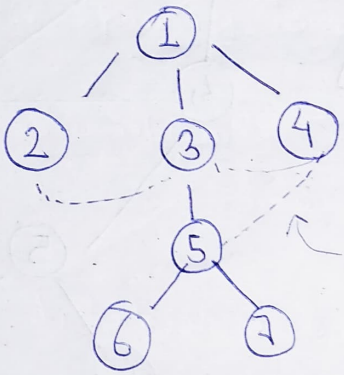
BFS : 4, 5, 1, 3, 7, 6, 2

1. Visiting
2. Exploring

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

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

~~BFS 1 2 3 4~~



BFS Spanning tree

Cross edges

a) n)

time complexity

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

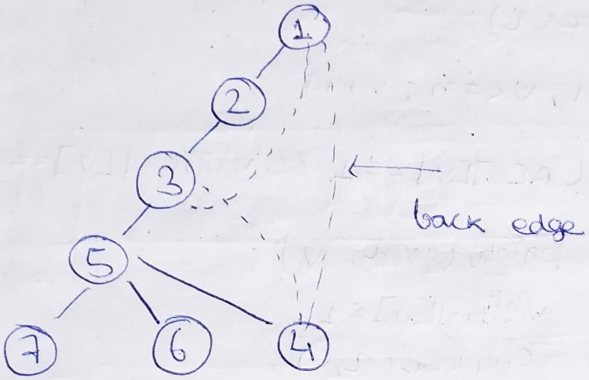
```
void BFS (int i)
    int u;
    printf ("%d", i);
    visited [i] = 1;
    enqueue (q, i);
    while (!isEmpty(q))
    {
        u = dequeue (q);
        for (v=1; v<=n; v++)
        {
            if (A[u][v]==1 && visited[v]==0)
            {
                printf ("%d", v);
                visited[v] = 1;
                enqueue (q, v);
            }
        }
    }
```

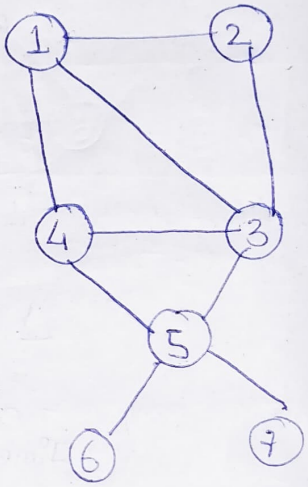
Time complexity → $O(n^2)$

Depth first Search

It uses stack data structure.

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



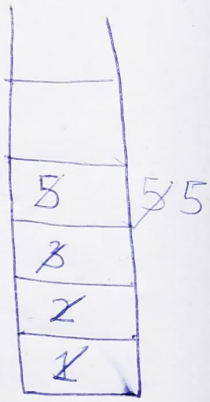


DFS Spanning Tree

Time complexity O(n)

analytical ~~time~~ time

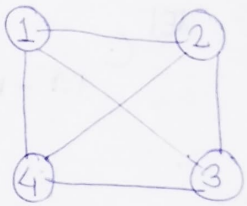
1. 1, 3, 5, 4, 7, 6, 2
2. 1, 2, 3, 4, 5, 6, 7
3. 1, 4, 5, 7, 3, 2, 6
4. 4, 1, 3, 5, 6, 7, 2



```
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);
        }
    }
}
```

Spanning tree

1. Spanning tree
2. Prims MST
3. Kruskal MST



$G = (V, E)$

$n = |V|$

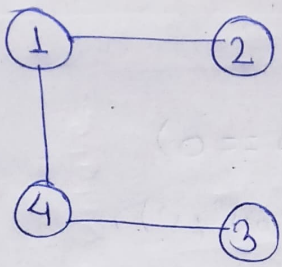
$e = |E|$

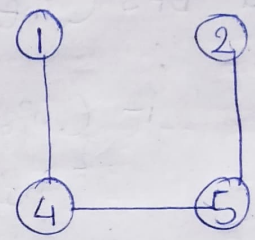
$S \subseteq G$

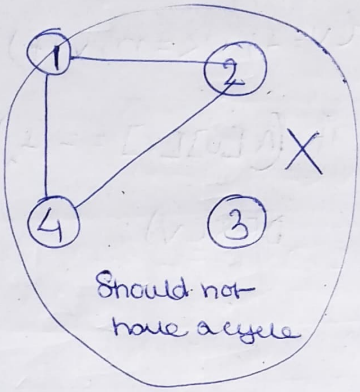
$S = (V', E')$

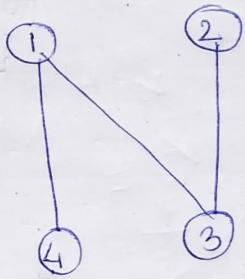
$|V'| = |V|$

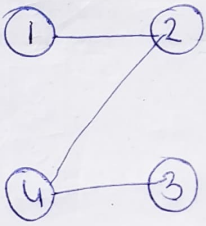
$|E'| = |V| - 1$

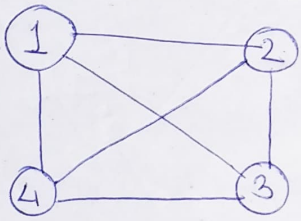










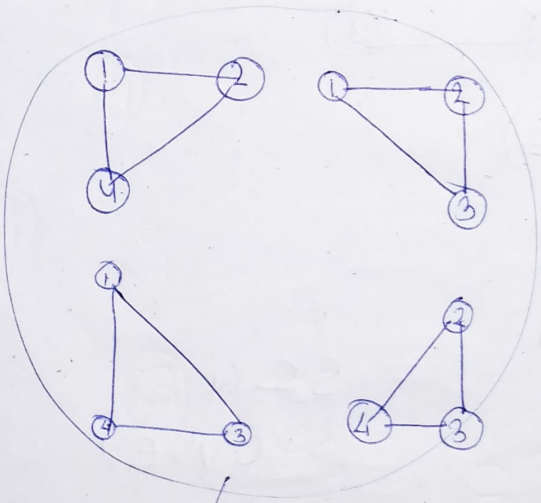


$|V| = 4$

$|E| = 6$

$$^{|E|}C_{|V|-1} = {}^{6}C_{3}$$

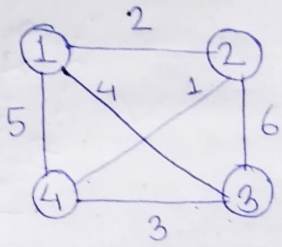
no. of spanning trees with cycle.

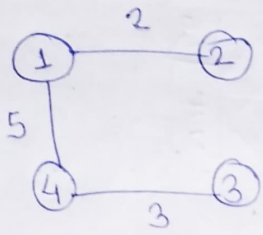


rings which are not valid spanning tree

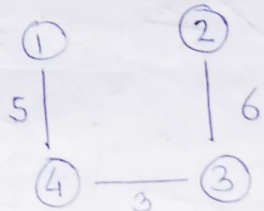
$|E|$ $C_{|V|-1}$ - Cycle

Minimum cost spanning tree

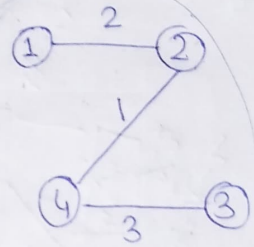




10



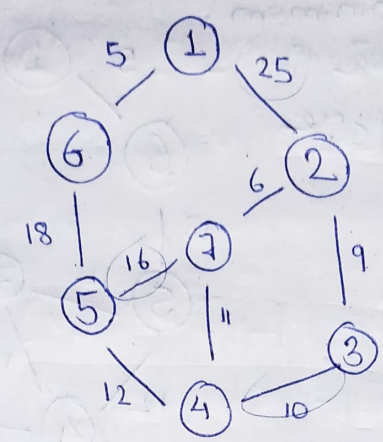
14



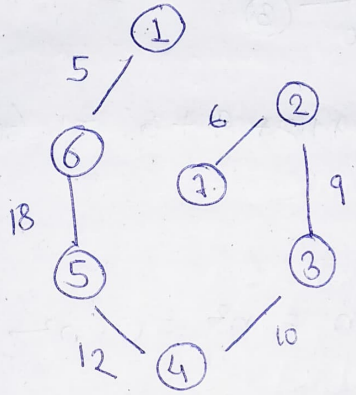
6

minimum

Prim's minimum Cost Spaning Tree



Spanning → Tree will be connected and it will not have cycles.



$(|V|-1)|E|$

$ne = n \times n$

$= O(n^2)$

$5 + 18 + 12 + 10 + 9 + 6 = 60$

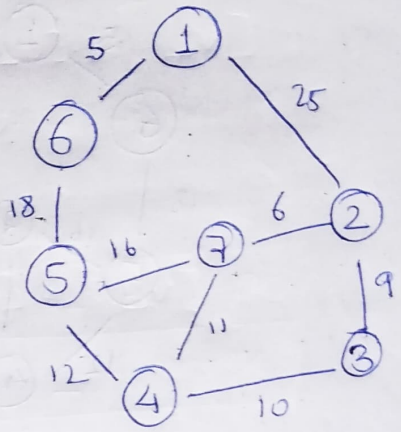
If we use heap tree finding minimum edge,

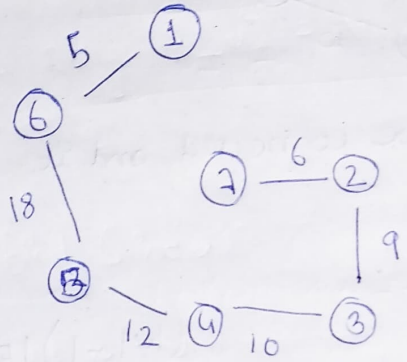
$(|V|-1)\log|E|$

$O(n \log n)$

kruskal's

Always select the minimum edge but make sure it does not form a cycle





Cost = 5 + 18 + 12 + 10 + 9 + 6 = 60

$(|V| - 1)\,|E|$

$nc = nn = n^2 = O(n^2)$

By using heap,

$O(n \log n)$

Disjoint Subset

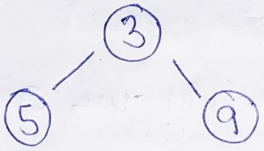
$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

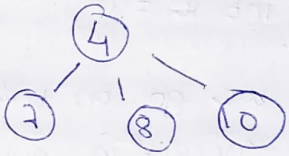
If you take the intersection there is no common element among them.

S

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

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





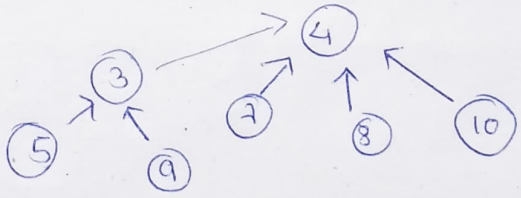
S

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

Two operations

- union
- find

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



S

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

```
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)
{
    int x = u,
```

we go on taking value till we reach a negative number.

```
    while ( s[x] > 0 )
    {
        x = s[x];
    }
```