2nd Dec 2019

(1) Two Sum.

```
public int[] twoSum (int[] nums, int target)
{
    Map < Integer, Integer > m = new HashMap <>();
    for(int i=0; i< nums.length, i++)
    {
        int complement = target - nums[i];
        if (m.containsKey (complement))
        {
            return new int[] { m.get(complement), i};
        }
    }
throw new IllegalArgumentException ("No two sum solution");
}
```

nums = [2, 7, 11, 15] target = 9.

return [0, 1] 2+7 = 9.

Reverse Integer

(2)

```
public int reverse (int x)
{
    int num=0;
    int sign = x<0 ? -1 : +1;

    if (x<0)
        x= x * -1;

    int max= Integer.MAX_VALUE /10;

int digit= sign>0 ? Integer.MAX_VALUE - max*10 :
        -((Integer.MIN_VALUE + max*10));

    while (x>0)
    {
        int dig= x%10;

        if (num > max || num==max && dig >= digit)
            return 0;

        num = num *10 +dig;
        x= x/10;
    }

    return sign * num;

}
```

32 bit signed integer.

120 → 21

-123 → 321

123 → 321

Lulp

(3) Reverse bits.

Input 00110.

output 01100.

>>1 → to make shift.

```
00110      00011      00001      00000      00000
 & 1        & 1        & 1        & 1        & 1
   0.         1.         1          0          0.
```

→

insert 0,1.

```
101 << 1     1010          1010
          or    0       or    1
             1010          1011
```

```
int times = 32;
List<Integer> l = new ArrayList<>();

while (times > 0)
{
    int t = n & 1;
    l.add(t);
    n = n >> 1;
    times--;
}
```

populating array by taking and each last bit and shifting it to right

```
int res = 0;
for (int i = 0; i < 32; i++)
{
    res = res << 1;
    res = res || l.get(i);
}
return res;
}
```

by left shift adding "0" at end & taking OR by element present at i index.

00000101001010000011110100111100 → 43261596.

00111001011110000010100101000000 0. → 964176192

(4) Number of 1 bits

```
public int hammingWeight(int n)
{
    int times = 32;
    int count = 0;

    while (times > 0)
    {
        if ((n & 1) == 1)
            count++;

        n = n >>> 1;
        times--;
    }
    return count;
}
```

11111111111111111111111111111101

Ans = 31.

}

(5) Count bits

2 (10) 4(100) 8(1000) 16(10000.) O(n).

No. of 1's is 1.

No. is converted into $2^m + x$. eg $10 = 8+2$, $9 = 8+1$.

No. of 1 is $\boxed{1 + \#\ 1\text{'s in } x.}$

Number	# of 1's
1	1.
2	1.
3 = 2+1	2.
4	1.
5 = 4+1	2.
6 = 4+2	2
7 = 4+3	3 i+2.
8	1
9 = 8+1	2.
10 = 8+2	2.

```
public int[] countbits(int num)
{
    int[] result = new int[num+1];
```

eg 11=8+3 p would be 3.

```
    int p=1;   // tracks the index of no. x.
    int pow=1;
    for(int i=1; i<=num; i++)
    {
        if( i == pow)
        {
            result[i] = 1;
            pow <<= 1; // multiplying by 2.
            p=1;
        }
        else{
            result[i] = result[p] + 1;
            p++;
        }
    }
```

10 = 8 + 2
1 + 1 = 2

(6) Power of Two.

```
public boolean isPowerOfTwo (int n)
{
    if (n<=0)
        return false;

    int times = 32;
    int count = 0;

    while (times > 0 )
    {
        if ((n &1) == 1)
        {
            count++;
            if (count > 1)
                return false;
        }
        n = n >>1;          // right shift.
        times--;
    }
    return true;
}
```

-2147483648 → false.

(7) Palindrome. Number.

```
public boolean isPalindrome(int x)
{
    if (x<0 || (x%10==0 && x!=0))
    {
        return false;
    }
    int rev=0;
    while (x>rev)
    {
        rev = rev *10 + x%10;
        x /= 10;
    }

    return x== rev || x== rev/10;
}
```

(8) Roman To integer.

```
public int romanToInt(String s)
{
    Map<character, Integer> m = new HashMap<>();
    m.put('I', 1);
    m.put('V', 5);
    m.put('X', 10);
    m.put('L', 50);
    m.put('C', 100);
    m.put('D', 500);
    m.put('M', 1000);

    int prev=0, curr=0, res = 0;

    for(int i = s.length()-1; i>=0; i--)
    {
        curr = m.get(s.charAt(i));

        if(curr < prev)
        {
            res = res - curr;
        }
        else
        {
            res = res + curr;
            prev = curr;
        }
    }
    return res;
}
```

LVIII.

I ① curr = 1 prev = 0
res = 0+1 = 1
prev = 1

I ② curr = 1 prev = 1
res = 1+1 = 2.
prev = 1

I ③ curr = 1 prev = 1
res = 2+1 = 3
prev = 1.

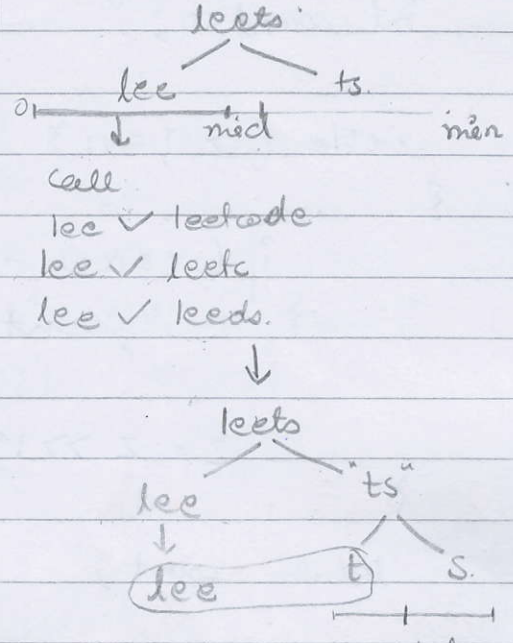
V ④ curr = 5 prev = 1.
res = 3+5 = 8
prev = 5;

L ⑤ curr = 50 prev = 5
res = 8+50 = 58
prev = 50;

Longest common prefix.

```
public String longestCommonPrefix(String[] strs) {
    if (strs == null || strs.length == 0) { return ""; }
    int lo = 1; int hi =
    int minl = Integer.MAX_VALUE;
    for (String s: strs)
        minl = Math.min(minl, s);
    int hi = minlen;

    while (lo <= hi)
    {
        int mid = (lo+hi)/2;
        if (isCommon(strs, mid))
        {
            lo = mid+1;
        }
        else{
            hi = mid-1;
        }
    }

    return strs[0].substring(0, (lo+hi)/2);
}

public static boolean isCommon(String[] strs, int mid)
{
    String s = strs[0].substring(0, mid);

    for (int i=1; i < strs.length; i++)
    {
        if (!str[i].startsWith(s))
        {
            return false;
        }
    }
    return true;
}
```

{leets, leetcode, leetc, leeds}



Hamming Distance

```
public int HammingDistance (int x, int y)
{
    int z = x ^ y;
    int count=0;

    while ( z != 0)
    {
        if ( (z&1) == 1)
        {
            count++;
        }
        z = z >> 1;
    }
    return count;
}
```

1 0 0 0 1

4 0 1 0 0

Distance = 2.

8th Jan 2020

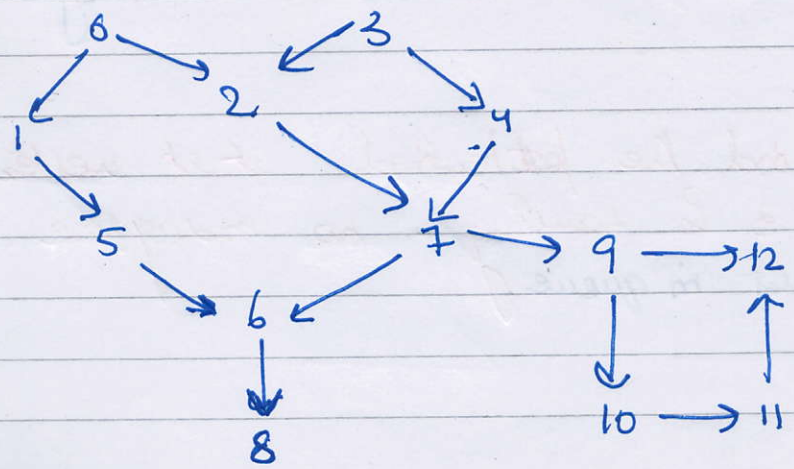
% detect cycle

% Topological Sort ~ Graph

```
bool topological_sort ( src, vec<> &vis, vec<bool> &
                                        cycledetection)
{
    vis[src] = true;
    cycledet[src] = true;
    bool res = false;
    for ( Edge *e : graph[src] )
    {
        if ( vis[e->v] )
            res = res || topological ( e->v, vis, stack );
        else if ( cycledetection[e->v] )
            return true;
    }
    stack.push_back (src);
    cycleDet[src] = false;
    return res;
}
```

If the cycle is present in graph we won't be able to do topological sort.

Khan's Algorithm → tells about cycle & give order
→ work on BFS

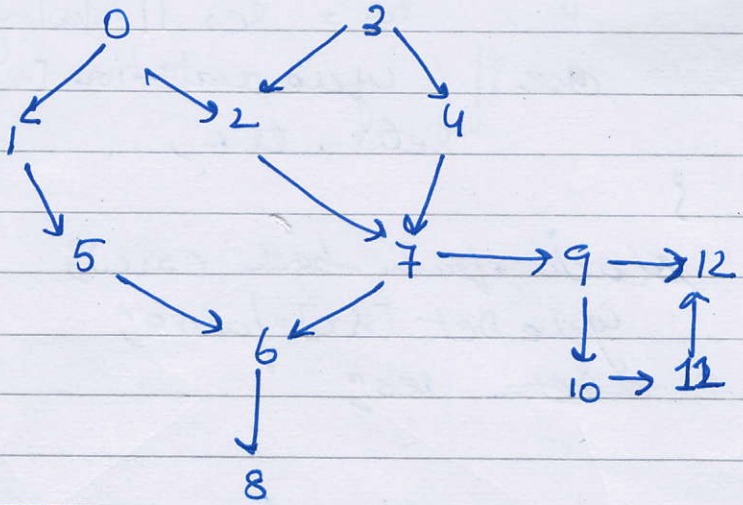


Indegree. → No. of edges entering.

Outdegree → No. of edges leaving from here

If Indegree is 0. it would be a leading probability of being the last one.

Calculating out degree.



0	1	2	0	1	1	1	2	1	1	1	1	2
0	1	2	3	4	5	6	7	8	9	10	11	12

do BFS and add the value +1 on neighbours of node.

→ Take a loop find the potential start vertex in graph.
→ here 0 & 3 is having no indegree.
put in queue.

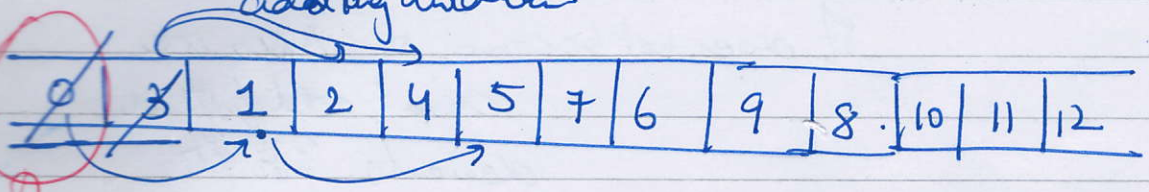
jiska indegree usko stack mein add krna hai kyu ki

0	3

Now we do get remove add child

if child of 0 is having some frequency decrement, the child whose frequency becomes 0 is to be added in queue.

Anything which is popped is to be added in stack



stack.

0	3	1	2	4	5	7	6	9	8	12

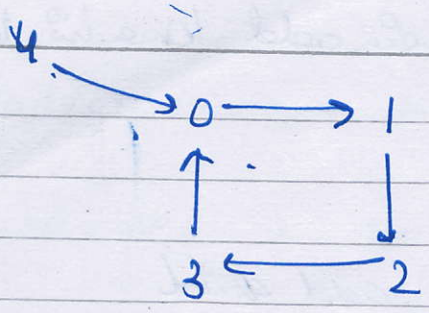
jab indegree zero hoga tab dalegi vertex

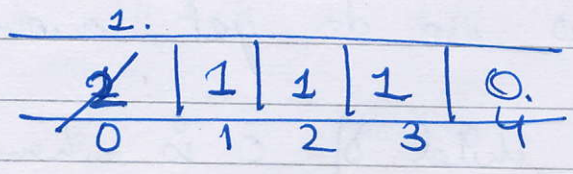
take out 0.

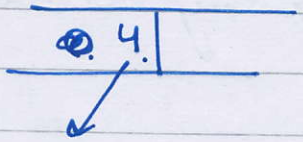
dec frequency of child.

1 frequency becomes 0. - add in queue.

2 frequency becomes 1 - not to be added in queue.

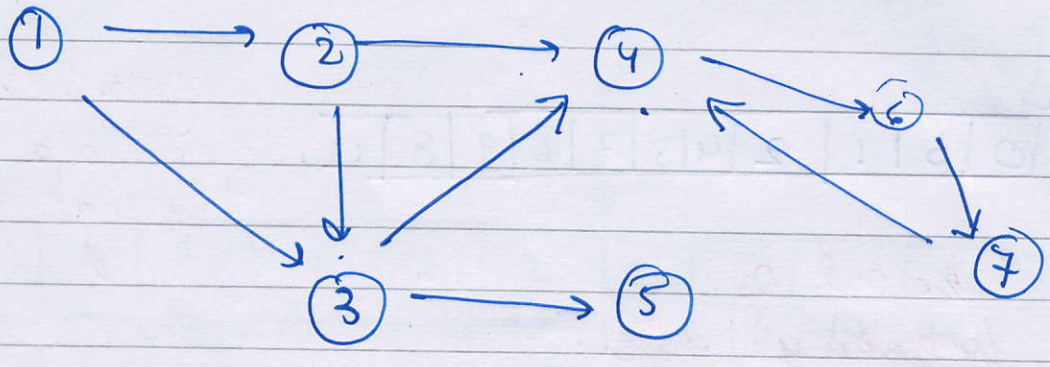


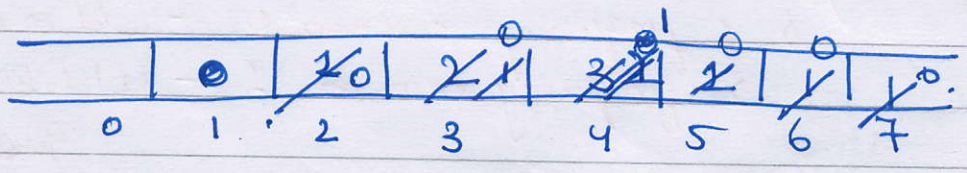


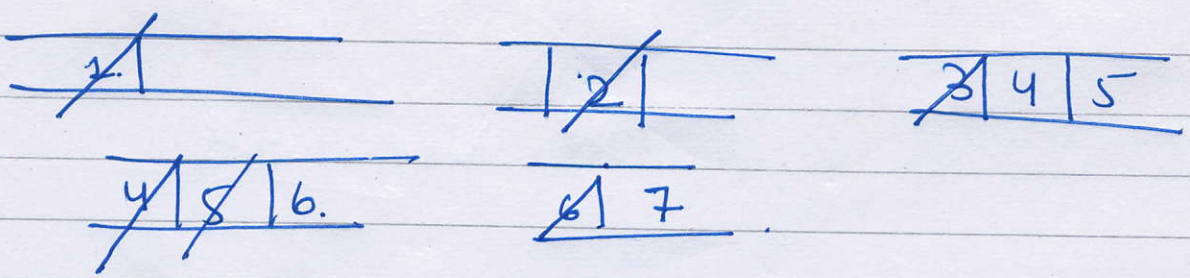


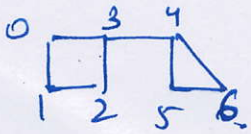
takeout 4 decrement frequency of child.

if does not become 0. then we cannot add further elements into it





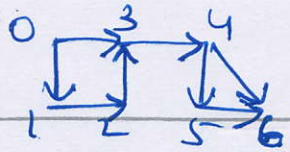




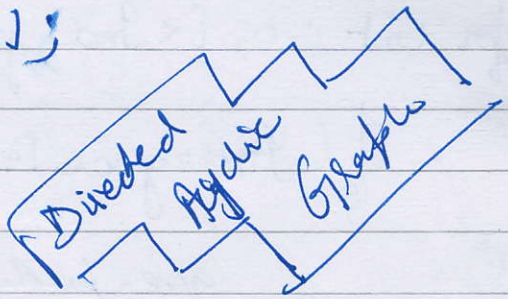
Khan Algo.

Apni outdegree nikal lo → it is indegree for other node.

```
void KhanAlgo()
{
    vector<int> indegree(graph.size(),0);

    for(int i=0; i<graph.size(); i++)
    {
        for(Edge *e: graph[i])
        {
            indegree[e->v]++;
        }
    }
    queue<int> que;

    for(int i=0; i<indegree.size(); i++)
    {
        if(indegree[i]==0)
        {
            que.push_back(i);
        }
    }
    vector<int> ans;                    // putting in stack
    while(que.size()!=0)
    {
        int node = que.front();
        que.pop();
        ans.push_back(node);

        for(Edge *e: graph[node])
        {
```

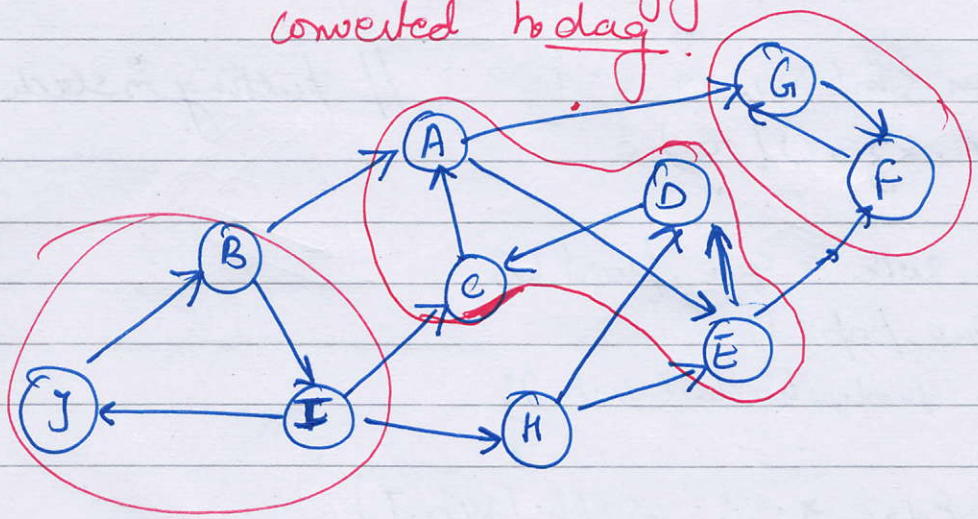



```
Indegree[e->v]--;
if(Indegree[e->v]==0)
{
    que.push(e);
}
}
if(ans.size()!=graph.size())
{
    cout<<"cycle";
}else
{
    for(int i=0;i<ans.size();i++)
    {
        cout<<ans[i];
    }
}
}
```

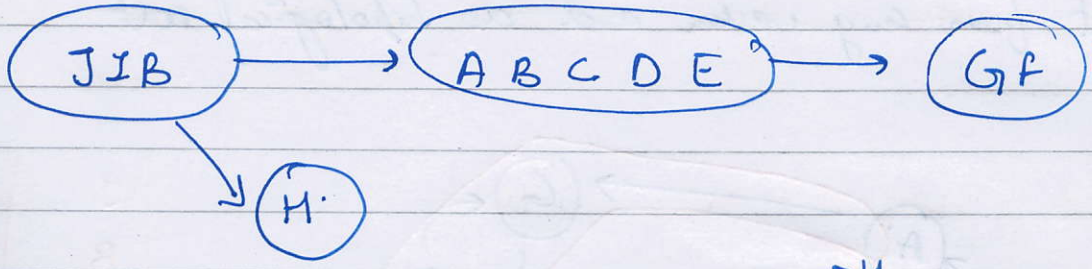


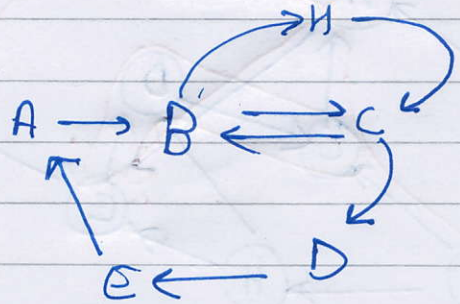
SCC — strongly connected component.

connected to dag.

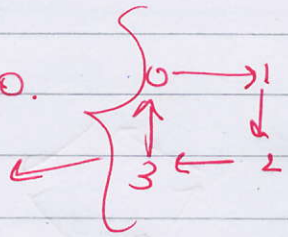


This is strongly connected component because it is possible to reach the starting node again.





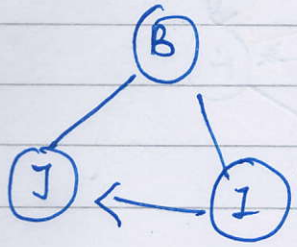
(1) In Khan Algo
If there is a cycle
it would not have any indegree 0.



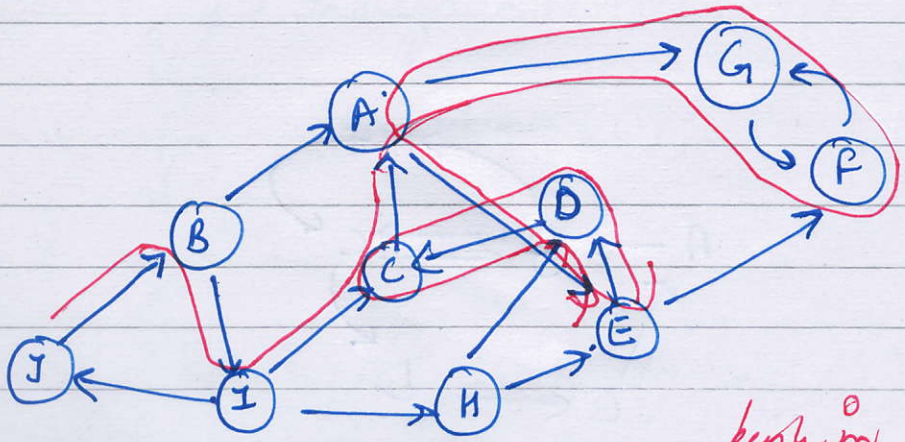
No node with
0 indegree.

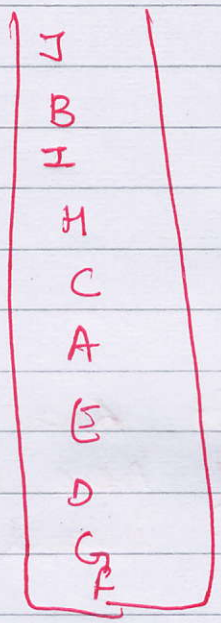
(2) Then we reduce the dependency as we make calls.

Kosaraju



start from any vertex and do topological sort



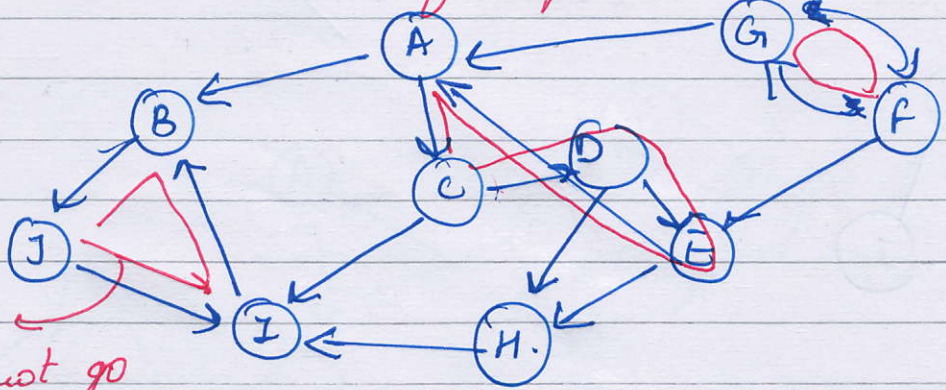


push in post call

when we get strongly connected components we can also get cycle when the edges are being reversed.

Take Transpose of edges.

DFS Topological



It cannot go anywhere else thus strongly connected.

If I was able to reach A → B before & I am able to do the same now, then this implies that there is a strongly connected component & 2 way path exist

Now start popping stack & perform DFS on the transpose graph.

start with 4, it make sure that it's part of scc.

pop 6, 1.

* from 4cc : no. of times call is made is no. of strongly connected components

1. Topological sort, return stack *
2. call DFS according to stack
3. call DFS (src, vis)
   - 3.1 vis ← src.
   - 3.2 add unvisited nbr
   - 3.3. DFS (nbr, vis)

1.1 Inverse of graph

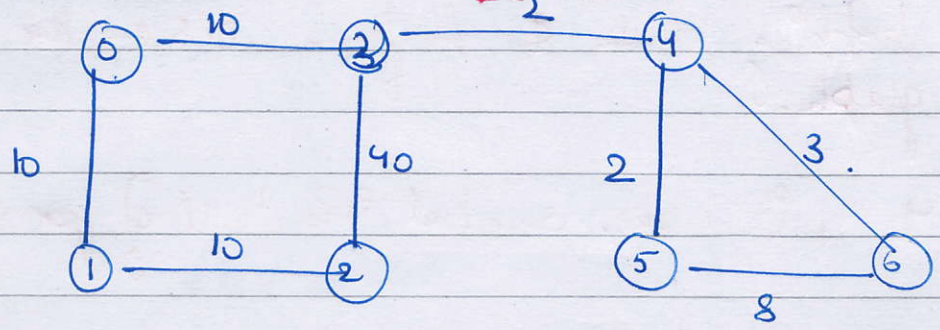
Important

21st Jan 2020

single source every vertex min cost path

s/s. → jesa source vesa answer.

Dijkstra → It can never give MST



source to destination m → minimum cost path

priority queue. → at top keep min element.

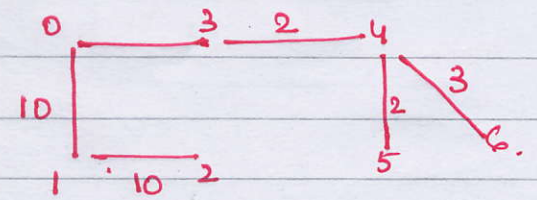
5, 4, 14, 03245.
2, 3, 50, 032.
4, 3, 12, 034.
2, 1, 20, 012

6, 4, 15, 0346.
6, 5, 22, 03456

0
1
3
4
5
6.

1 | 0 | 10 | 01 → src, dest, cost, path.
3 | 0 | 10 | 03
0 | 0 | 0

priority queue.



If element coming out is having non-negative parent value on edge between them.

6, 4, 3, 15, 0346
5, 4, 2, 14, 034, 5
2, 3, 40, 50, 032.
(4). 4, 3, 2, 12, 034
2, 1, 10, 29012.
(3) 3, 0, 10, 10, 03
(2) 1, 0, 10, 10, 01
(1) 0, -1, 0, 0, 0.

taking element out on the basis of weight so far.

any of two can be come out.

vertex, parent, cost, path, edge weight

1. queue ← pair(src, 0)
2. while(que.size() != 0)
3. 2.1 vis ← pair.src.
2.2 for (all unvisited nbr)
2.3. que ← pair (nbr , pair.wt + wtf) ;

vector <int vector< Edge*>> dGraph (n, vector < Edge*>());

class dpair (→ u will be parent
{ public
① int vtx = 0 ;
int parent u = 0 ;
int wt = 0 ;
int wsf = 0;

② dpair(int v, int u, int wt, int wsf)
{
this → vtx = v tx
this → u = u parent
this → wt = wt;
this → wsf = wsf;
} ;
} ;

void dijikstra (int src)
{
int dest = 6 ;
priority_queue < dpair > que ;
vector < bool > vis (n, false);
que.push (dpair (0, -1, 0, 0));

while (que.size () != 0)
{
① [dpair rp = que.top();
que.pop() ;
③ if (!vis [rp.vtx]) { addEdge (dGraph, rp.vtx, rp.parent, rp.wt)}
dshortestpath.[rp.vtx] = rp.wsf
④ { vis [rp.vtx] = true ;
for(Edge * e : graph[rp.vtx])
{ if (! vis[e → v])
que.push (e → v, rp.vtx, e.wt, rp.wsf + e.

② if (vis[rp.vtx]) { continue; }. // for cycle.

③ bool operator < (dpair const & o) const
{ return this.wsf > o.wsf;
}

when we are writing greater
we will be using greater operator ?
else
lesser ✗

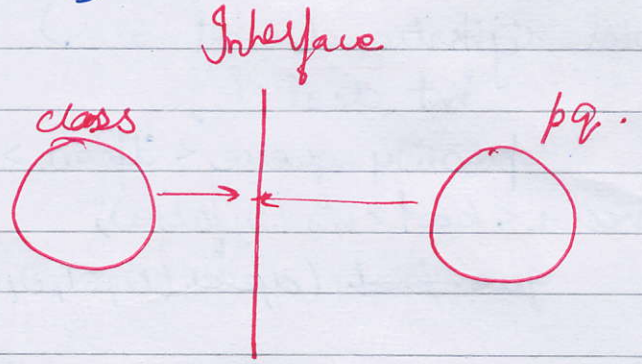
C++ → max pq

Java priority queue → by default min priority queue.

```
class dpair implements Comparable<dpair>
{
    int wt;
```

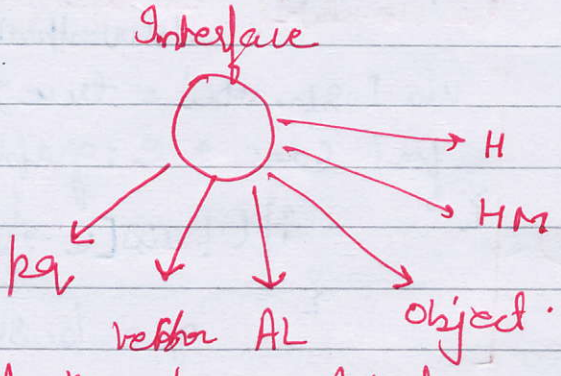
implements → interface
It is saying mujhe override karo.

```
public int compareTo(dpair o)
{
    return this.wt - o.wt;
}
```



Apne aap ko kuresha. strong banaunga aur apne aap ko interface ke through guide karunga

→ class interacts with interface.
→ priority queue implements interface.



All these knows Interface
∴ If we are able to implement interface we will be able to use them.

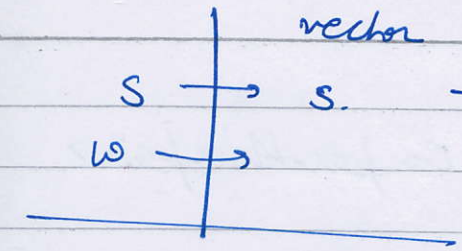
vector.

strong → takes it ahead

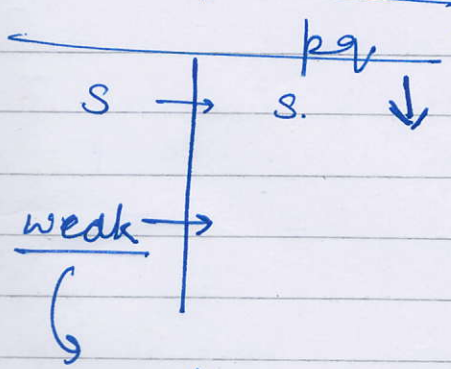
element itself as strong

weak →

K stops k flights Leetcode



bina le jaaiga // sort-
ahead //reverse ke



→ keeps min at top.

strong ko upper rakh diya → min. max p

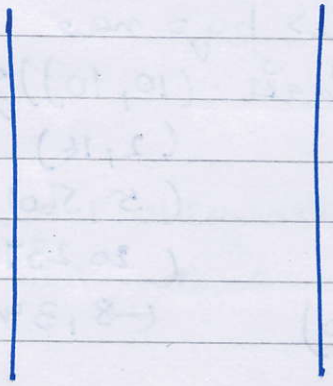
it will take it to top → max pq.

an element which is strong is pretending to be weak

Array List ka behaviour in terms of sorting.

C++ There are two interfaces. one written by user other by C++.

In C++ whatever interface we will write C++ will do opposite of it.



priority_queue <int, vector<int>, greater<int>> q - qp ;

In priority queue if we write this it will create min heap.

```
{ public static class pair implements Comparable<pair>
    int age
    int wt.

    pair ( )
    {

    }

                                        // override.
    public int compareTo ( pair o).
    {
        return mis.age - o.age;  // forman.
                                 // o.age - mis.age;
    }
}
```

return	forman.
-8	20
2	10
5	5
10	2
20.	-8

main.

```
PriorityQueue <pair> pq = new PriorityQueue<>();
pq.~~push~~ add ( new pair (10, 10));
                  (2, 16) ;
                  (5, 560)
                  (20, 2340)
                  (-8, 340)

while (pq.size()!=0)
{
    spso ( pq.peek().age + " " + pq.peek().wt );
    pq.remove();
```

```
class pair
{   public:
        age ;
        wt ;
    pair ( , )
    {
```

user operator with here greater will be passed through interface

when we send value it passes through 2 interface which in order to make sure there is no change in value we use (const)

```
    bool operator (<) ( pair const &p1, pair const &p2) const
    {
        Return this->age > p1.age ;
    }
```

function constant because the

as these are passed by & which makes the value being treated as local variable.

```
priority_queue <pair> pq ;
```

for min.

return this->age < p1

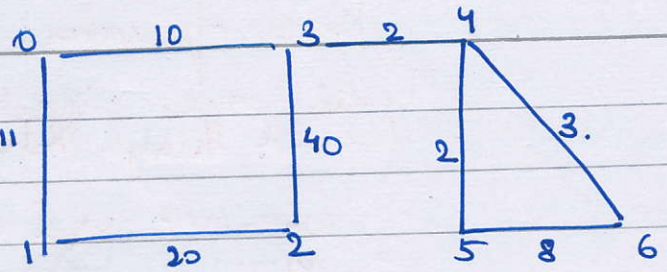
Prims.

MST → in which sum of all edges is min.

a graph in which each edge is having min value.

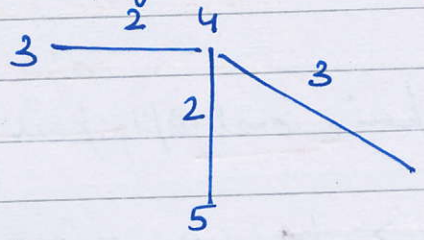
[greedy kahe take min value]

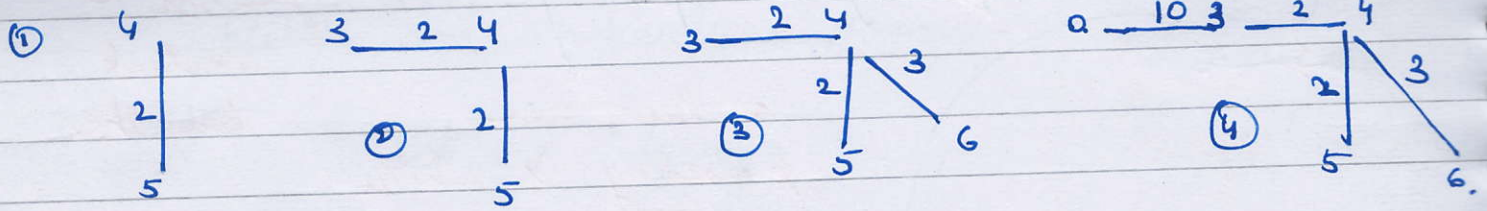
pure tree ka minimum weight wala path. no matter where you start from.



① all the edges in priority queue
② take min edge

keep cycle in mind.





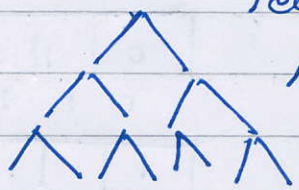
dijikstra → prim change operation. wt → wt.

to return his.wt > 0.wt;

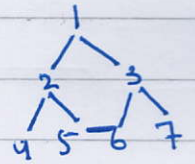
Khan's topological sort tells whether cycle is present or not but does not tell which place it takes place

Redundant connection Leetcode.

There is one cycle in graph.

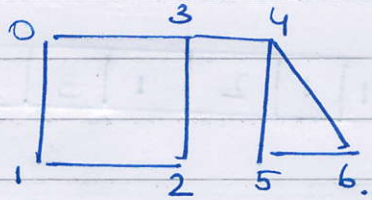


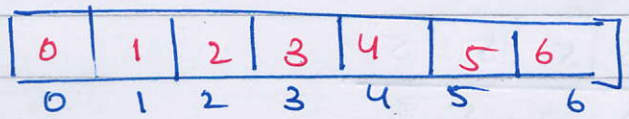
(1,2) (2,3) (3,1)



Union - Find

find
to cycle - best.

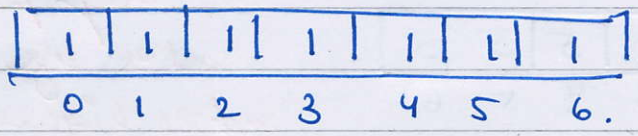


① 

Parent array

har kisi ka parent khud ko bnade.

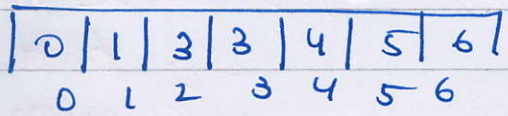
② size array which tells is wale bande ki size kitni hai.

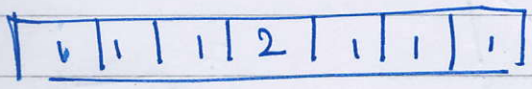


1 take edge 3–2.

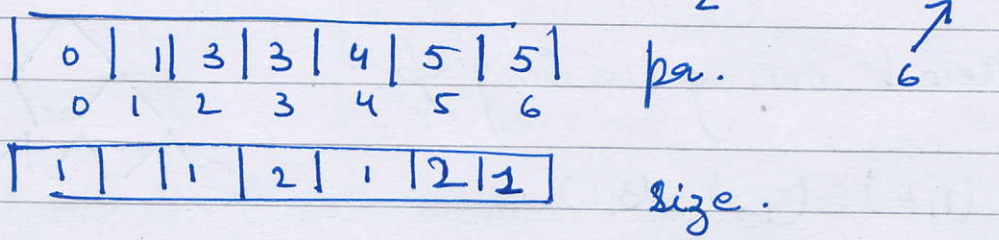
checked their parents & sizes. → same hue. join either
mein.

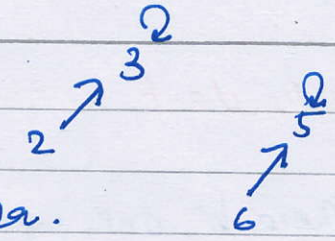
diff parents means different set elements have been encountered



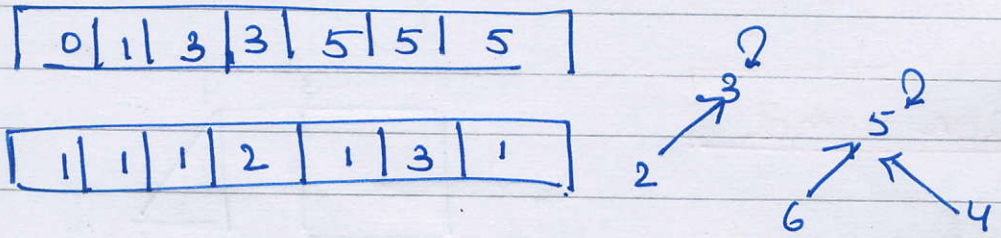


[2] edge. 5—6.

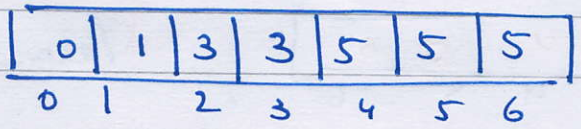




[3] 4-5

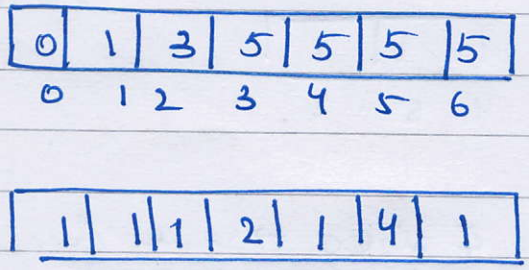


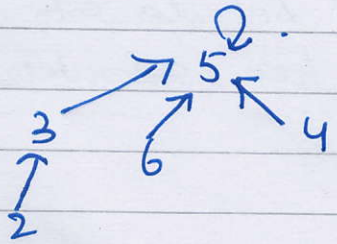
[4] 4—6



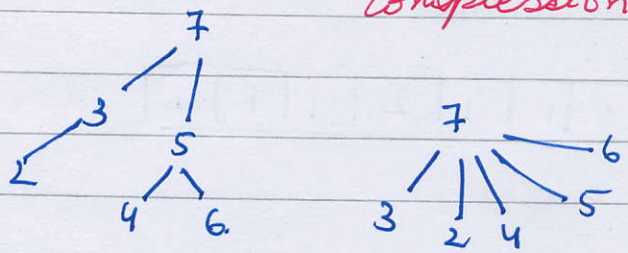
parent of 4 is 5, parent of 6 is 5
Thus 4-6 is edge is responsible for cycle.

[5] 3-4





we always put smaller size parent under higher size. parent in order to reduce search efforts
- This can be handled using path compression.



union find

684

L

1061

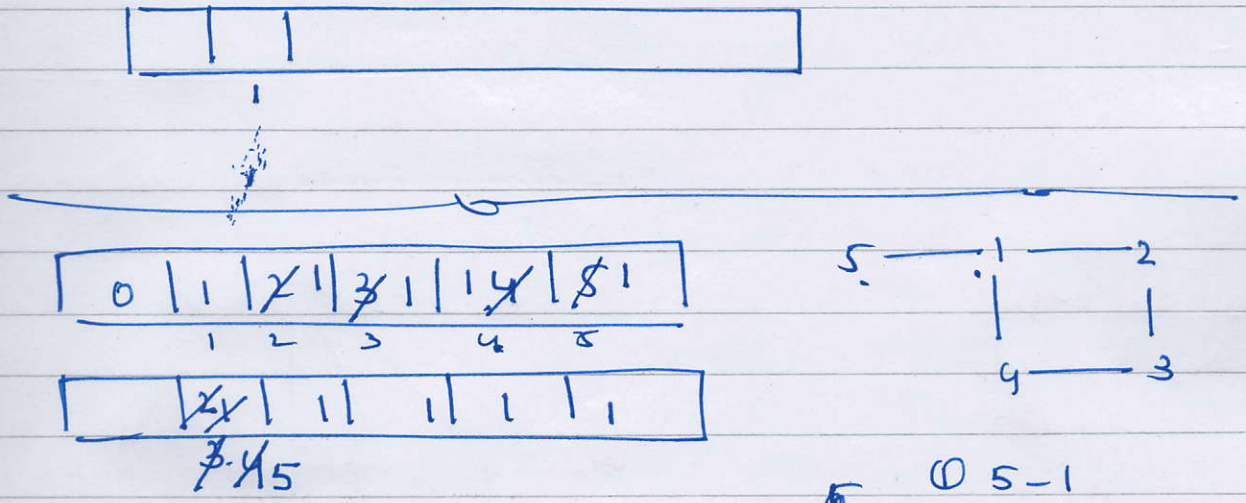
lexicographically.

(6) 0 – 1.

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

2	1	1	2	1	4	1

(7) 0 – 3 different parents



parent of 2 & 3 are same (1).

(1) 5 – 1
(2) 1 – 4
(3) 4 – 3.
(4) 1 – 2.
(5) 2 – 3

14th Feb 2023

DSU

Kruskal.

Rotten Oranges.

X-O.

Island.

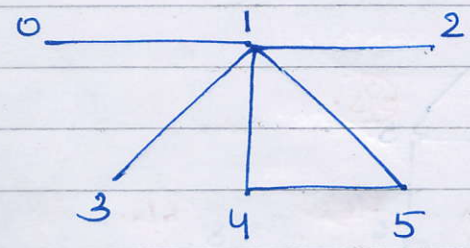
Bricks

2nd/Feb/2020.

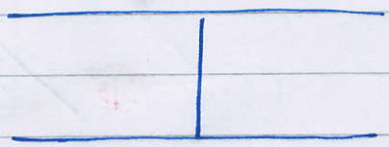
Articulation point.

Points in graph due to which no. of components increases. if we remove it.

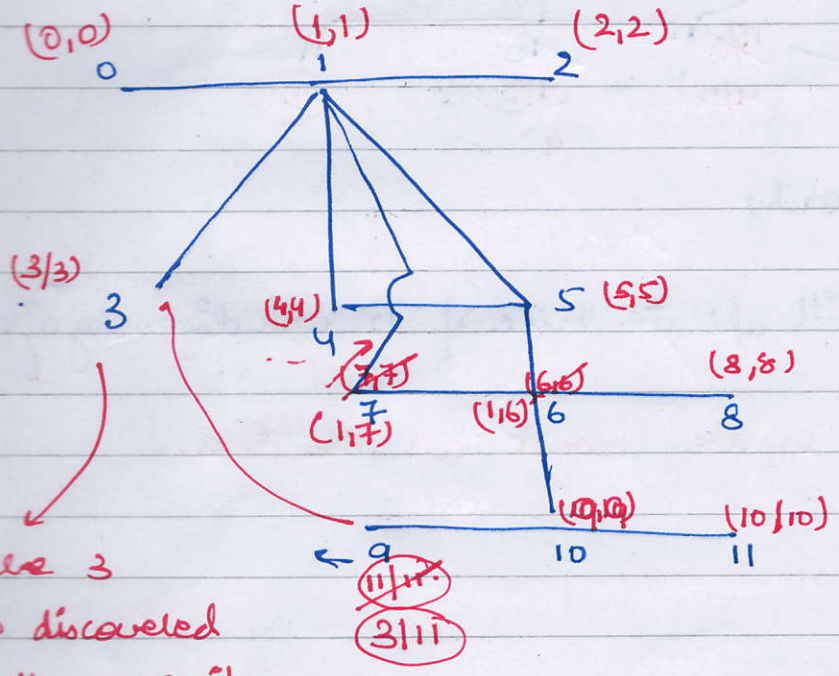
(1)



(2)



at time



here 3 is discovered after 1 i.e. it is dependent on 1.

3 ∄ 9
(3/3) 11/11

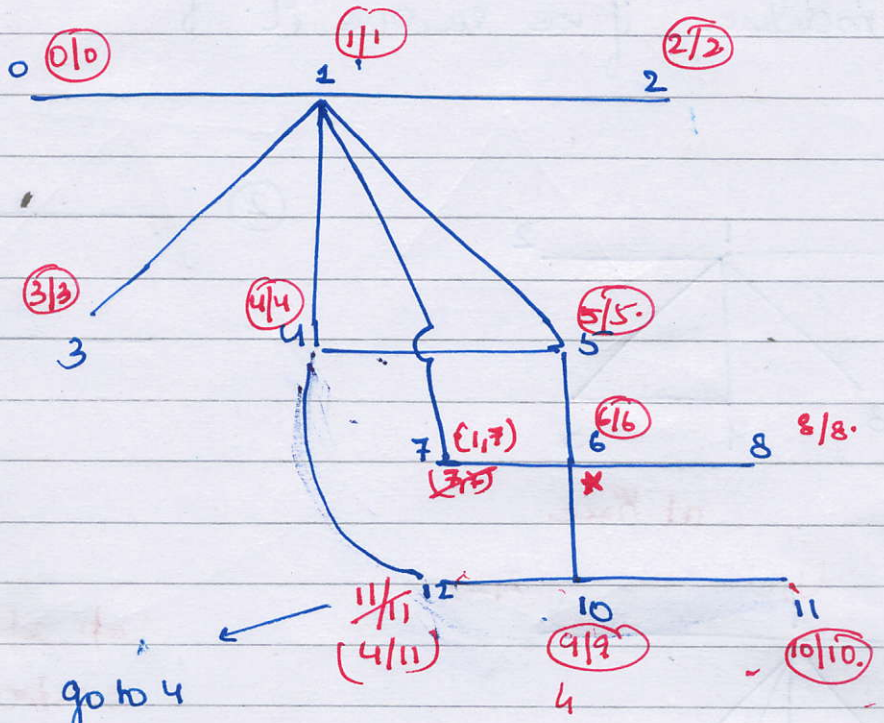
we update 9.

aap apne component mein beta dete time powerful logo ko jaante ho.

aage powerful ko jaante hohi keep it uchipoly if not remove

low:

discover.

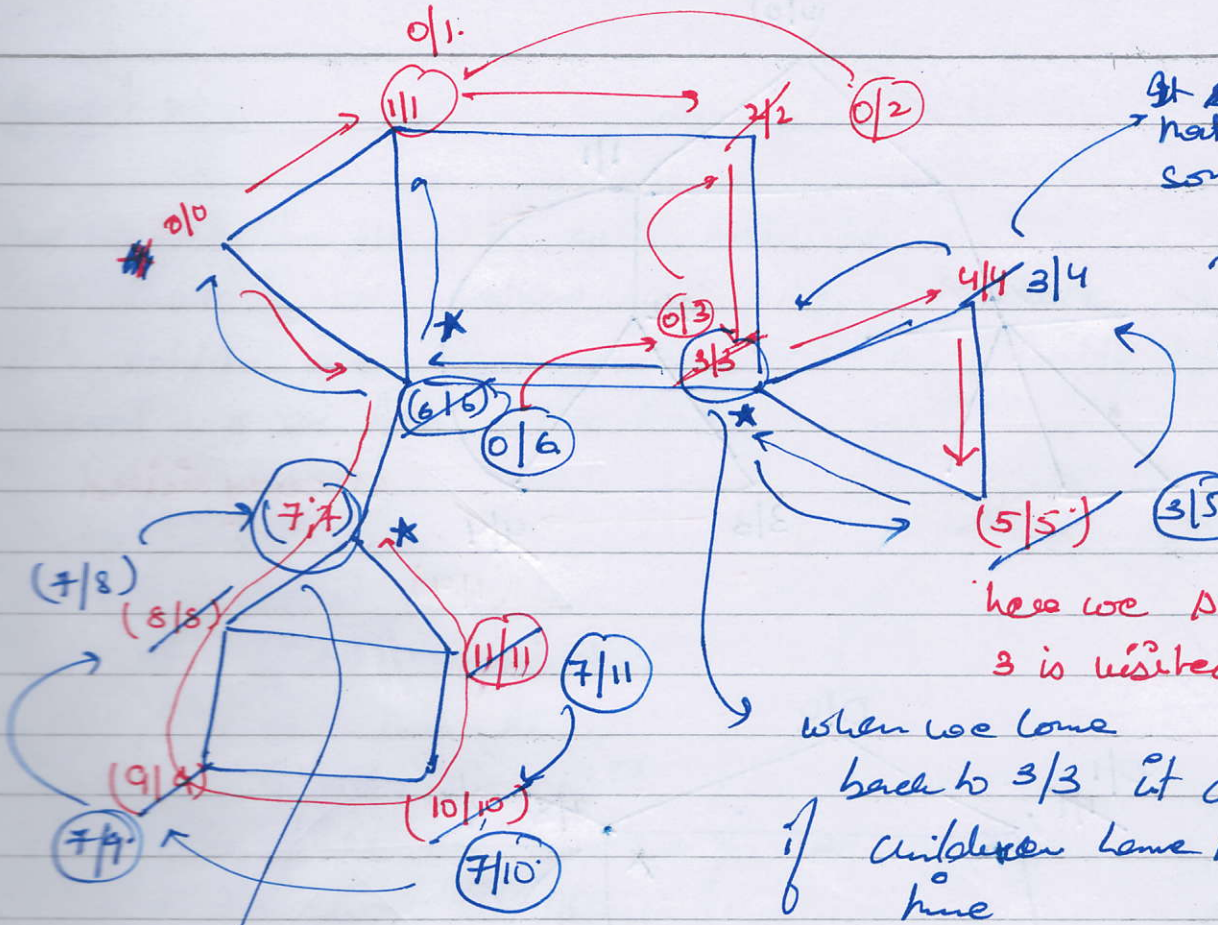


go to 4

it is already visited

we will update value of 12 with discovery of 4.

update lowest all the time.



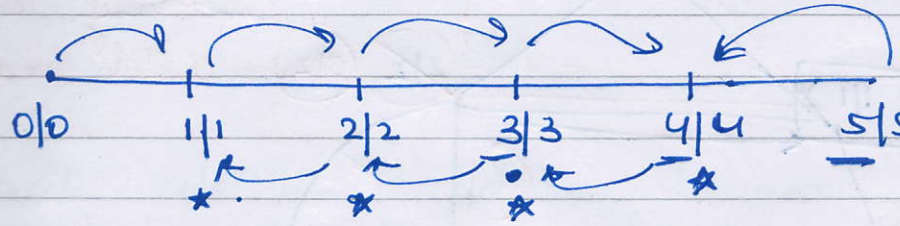
It says that it knows somebody lowest ∴ I cannot be articulation point for it

here we see 3 is visited.

when we come back to 3|3 it need if children have lowest time

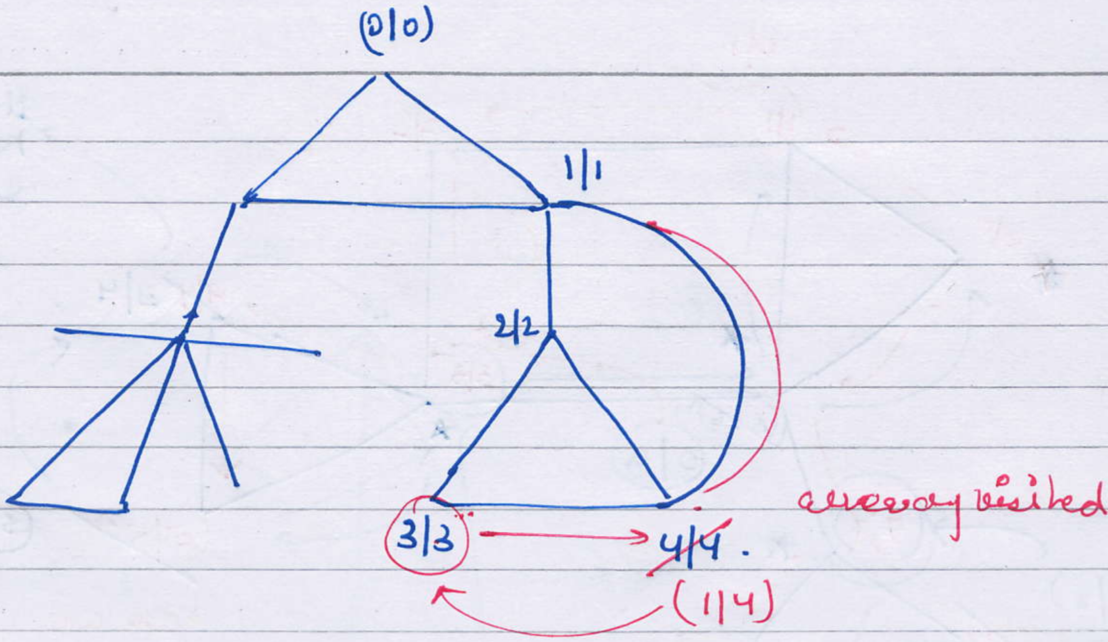
NB all have lowest & 3. If 3 is removed then it's articulation point

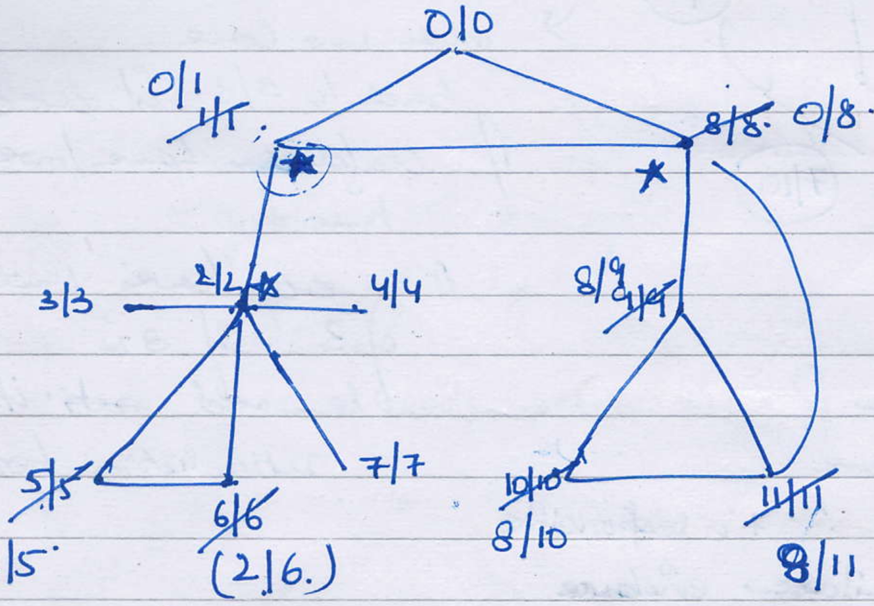
all children have same lowest time ∴ 7 is responsible for children existance

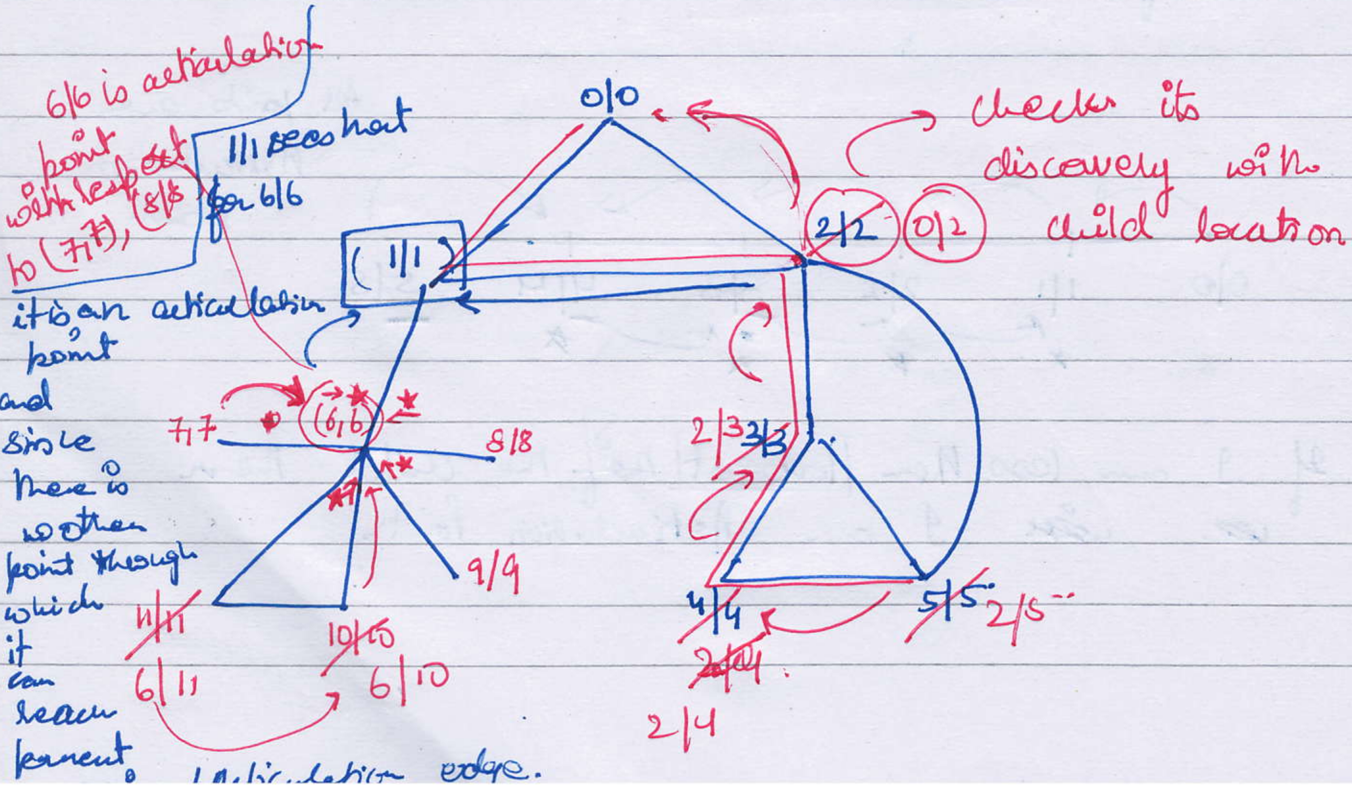


All points are Articulation point.

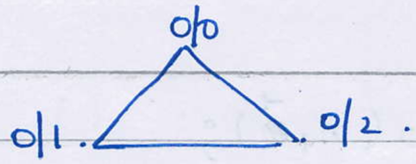
If I am less than lowest then me child mean I am Articulation Point.


(0|0)
1|1
2|2
3|3
4|4
(1|4)
already visited


0|0
1|1
0|1
8|8
0|8
2|2
3|3
4|4
8|9
5|5
2|5
6|6
(2|6.)
7|7
10|10
8|10
11|11
8|11

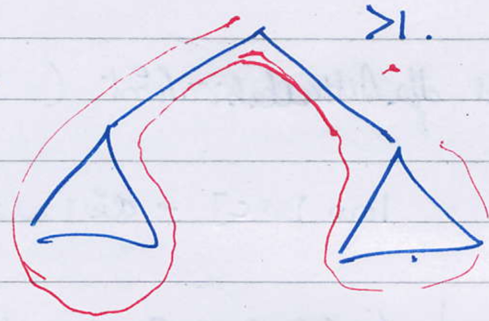

6|6 is articulation point with respect to (7|7), (8|8)
1|1 sees that for 6|6
it is an articulation point
and since there is no other point through which it can reach parent
0|0
(1|1)
2|2
0|2
checks its discovery with child location
(6,6)
7|7
8|8
9|9
11|11
6|11
10|10
6|10
2|3
3|3
4|4
2|4
2|4
5|5
2|5
articulation edge.

Root →



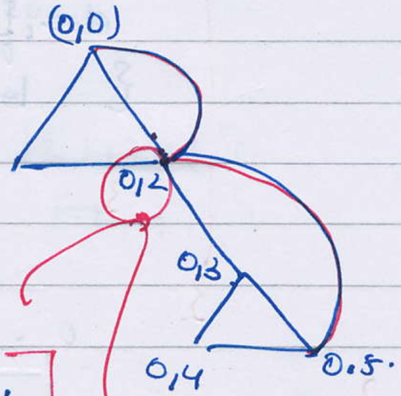
we need to handle this call individually.
if there is no vertices having edge between themselves which are connected to 0 this imply that root is an Articulation Point.

For root
if No. of calls from root is more than one then it means in order to visit other part we have to go through root.
∴ Root is Articulation Point.



Discovery se hi lowest ko kyun update kete hai ?

Mere chale jaane se farak padta hai ya nahi padta is dependent on the discovery because it will also determine the hierarchy.



1. Discovery se update karenge lowest ko because we if we donot here child might say it knows stronger edge directly but it need not be true here

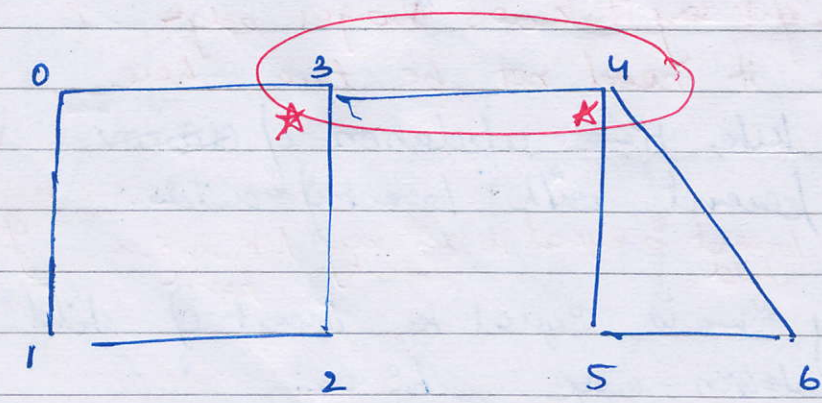
0,5 is going through 2 first

2. Backtrack kete time updation of ~~discov~~ lowest of parent with lowest of childs.
our agar lowest child is equal to discovery parent Articulation point

③ If discovery time is equal to lowest of child ~~we~~ need two articulation edge nahi hai

```
vector<int> low(n, 0);
vector<int> discovery(n, 0);
vector<bool> APpoints(n, false);
vector<bool> visited(n, false);
int time = 0;
int callsFromRoot = 0;

void dfsArticulationPoint(int src, int par, int osrc)
{
    low[src] = dis[src] = time++; // marking self with time.

    for(Edge* e : graph[src])
    {
        if(discovery[e->v] == -1)          // unvisited.
        {
            ① if(src == osrc){callsFromRoot++;}.
            ② dfs(e->v, src, osrc);
            ③ if(dis[src] <= low[e->v]){APpoint[src] = true;}.
            ④ if(dis[src] < low[e->v]){cout << "AP edge" << src << "to"
                                       << e->v << endl;
            ⑤ low[src] = min(low[src], low[e->v]);
        }
        else if(e->v != par)   // visited but non parent logo se
        {
            low[src] = min(low[src], discovery[e->v]); check
            // region for backEdge                    karenge
        }
    }
}
```

// APoint
// edge AP



Point 3, 4
bridge 3-4

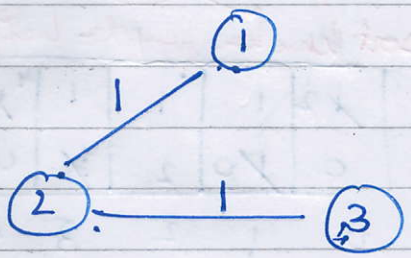
```
void AP_pointsAndBridges()
{
    dfsAP(0,-1,0);
    if(callsForRoot > 1)
    {
        cout << 0 << endl;
    }

    for(int i=0; i<APpoint.length; i++)
    {
        if(APpoint[i] == true && i!=0)
        {
            cout << i << endl;
        }
    }
```

1168 Optimize water distribution in Village

Leetcode.

① well creation ② pipe building

n = 3 wells = [1,2,2] pipes [[1,2,1], [2,3,1]]



O(nlogn)

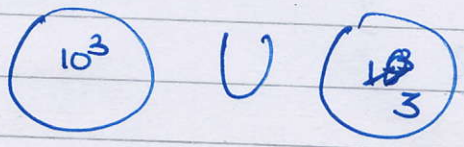
1	2	2
1	2	3

graph utheo

0

DSU Disjoint Set.

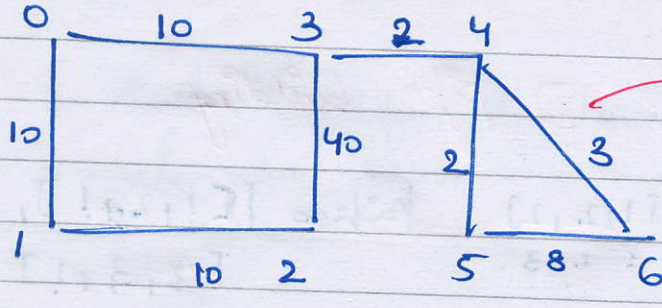
Set 1 Set 2



kisko kiske saath union kre.

set 2 ka union set 1

edge | wein...



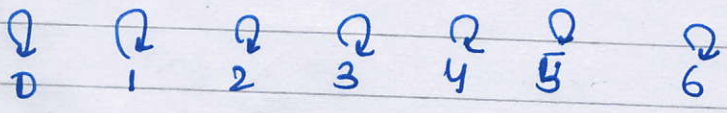
put in priority queue or put in vector & sort

pair <int, int> p<i, i>>

ispe sort krneka fayda hai because it is already implemented

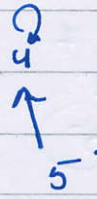
size./rank.	~~2~~	1	1	1	~~4~~ ~~3~~	1	1
par	0	~~0~~	2	~~4~~	4	~~4~~	~~4~~
	0	1	2	~~3~~	4	5	6

mi

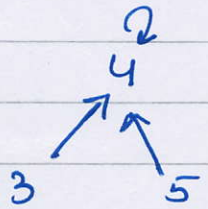


(size , 4)

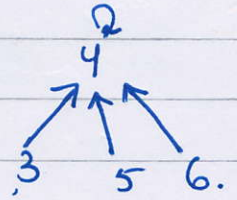
edge 4–5.



edge 3–4

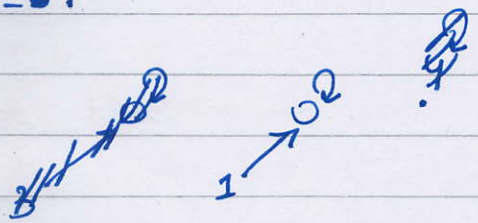


edge 4–6.

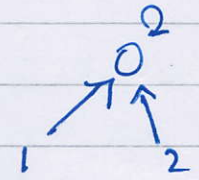


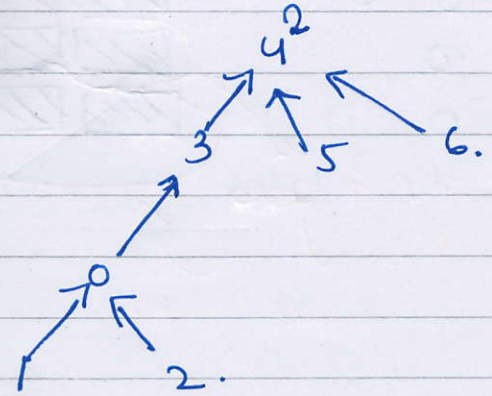
jiska size chota
usko merge
karenge bade
ke sath.

edge 0–1

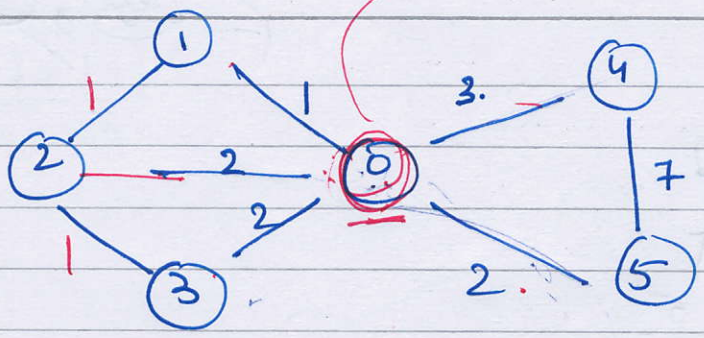


edge 1–2



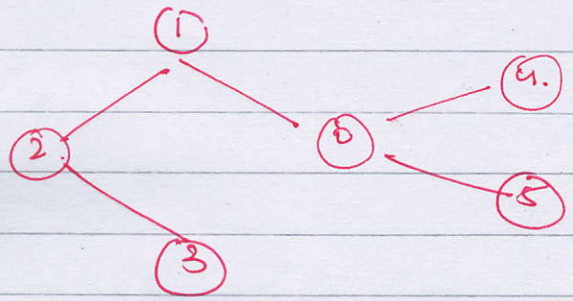


well bana the kahi in such a way to have min cost.

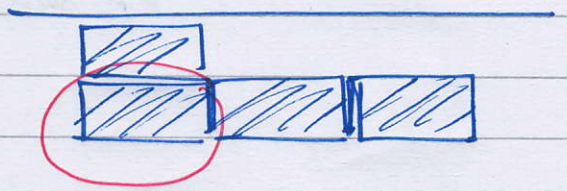


DSU.

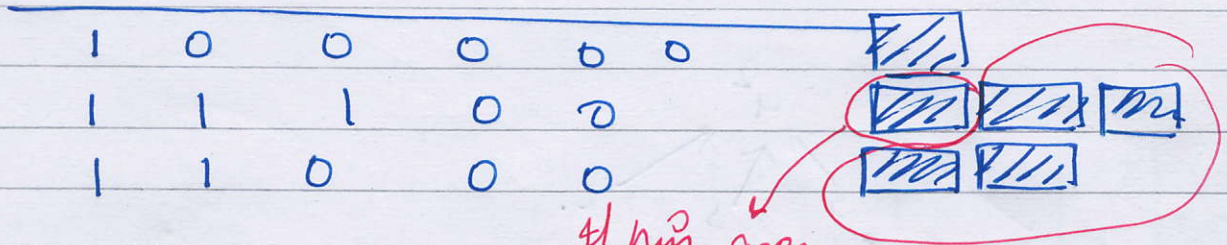
Only one well is to be dug



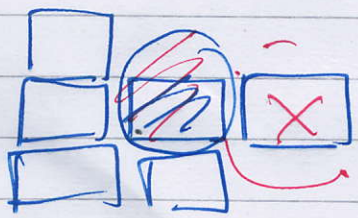
Brick wall



If this breaks. all breaks.



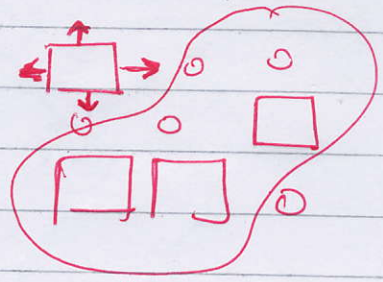
If this goes all fall.



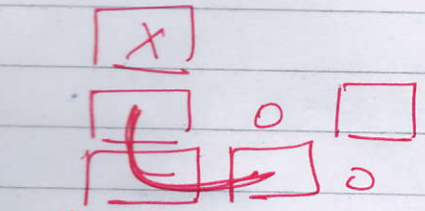
publu ko kaha attack ko sakta hai
→ go more & put 0 at that place.

→ base wall se dfs ki 4 calls ← ↑ → ↓

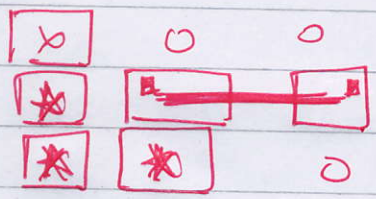
→ put block back
and we wont be making
call from base wall as
it is has been marked.



will be left unmarked.



will be marked.



All which are connected with main wall will not fall.

Entire matrix is traversed maxma
∴ all the vertices are traversed only once.

(1)

1 0 1 1
1 1 1 1 1 0 0
1 0 1 0 0 1 0

blocks to be removed

put block back.

(2)

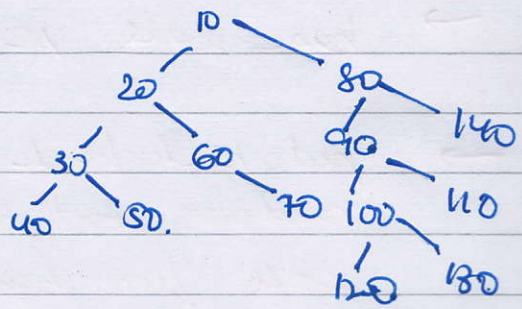
X 0 X X
X 0 X 0 1 0 0
X 0 X 0 0 1 0

(3)

X 0 X X
X 1 X 1 1 0 0
X 0 X 0 0 1 0

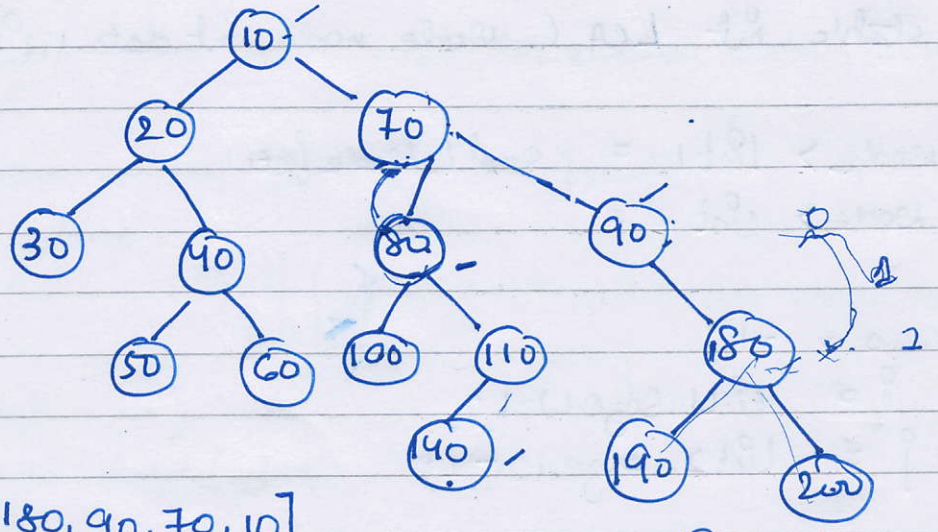
all 4 will fall.

Trees Day 1.



4th Feb 2020

LCA.



[190, 180, 90, 70, 10]
[10]

[50, 20, 10]

[60, 40, 20, 10]

equal incoming both

unequal last common node will be LCA

AL<Node> RootToNodePath(node, data)

① if(node == null)

② if(node.data == data)
AL<N> base = new AL<>();
base.add(node);
return base

③ AL<N> left = new AL<>();
if(left.size() > 0)
{ left.add(node);
return left;
}

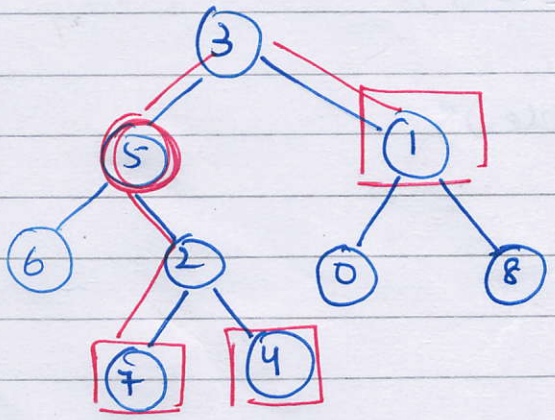
④ AL<N> right =
if{

}

⑤ return new ArrayList<>();

```
public static int LCA ( Node node, int data1, int data2)
{
    AL <Node> list1 = rootToNode(a);
    AL < Node > list2 =

    int ans = -1;
    int i = list1.size()-1;
    int j = list2.size()-1;

    while ( i>=0 && j>=0)
    {
        if ( list1.get(i) != list2.get(j))
        {
            break;
        }
        else,   ans = list1.get(i).data;
                i--;
                j--;
    }
```

Kth distance Nodes.



Target = 5
k = 2

Ans [7, 4, 1]

Never make call to previous node.

5 3

k=2. 0 1

k-idx

prev = (idx-1)

level

```
public static void kDown (Node node, Node pnode, int k, int level)
{
① if ( node == null // node != pnode)
    return
② if ( pnode != null && node == pnode)
    return;
③ if (level == 0) {
    sopo (node.data + " ");

  kDown ( node.left, pnode, level - 1);
  kDown ( node.right, pnode, level - 1);
}

public static void kAway (Node node, int k).
{
  AL < Node > list1 = root.toNode (
  Node pnode = null;

  for ( int i = 0 ; i < list1.size; i++)
  {
      kDown ( list1.get(i), pnode, k - i);
      pnode = list1.get(i);
  }

}
```

(k>2)

```
public static int kAway_02 ( Node node, int data, int k)

    if ( node == null)
{       return -1;
}

    if ( node.data == data )
{
        kdown ( node, null, k);
        return 1;
}

int ld = kAway_02( node.left, data, k);
    if ( ld != -1)
{
        kDown( node.right, node.left, k-ld);
        return ld+1;
}

int rd = kAway_02 ( node.right, data, k);
    if ( rd != -1)
{
        kDown ( node.left, node.right ; k-rd);
        return rd+1;
}

return -1;
}
```