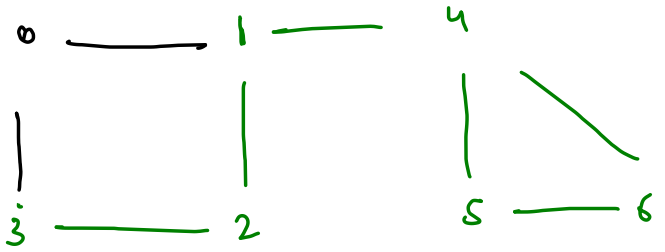
Articulation point :



ap :

1

4

0 -> 1, 3

1 -> 0, 2, 4

2 -> 1, 3

3 -> 0, 2

4 -> 1, 5, 6

5 -> 4, 6

6 -> 5, 6

vtx = 0

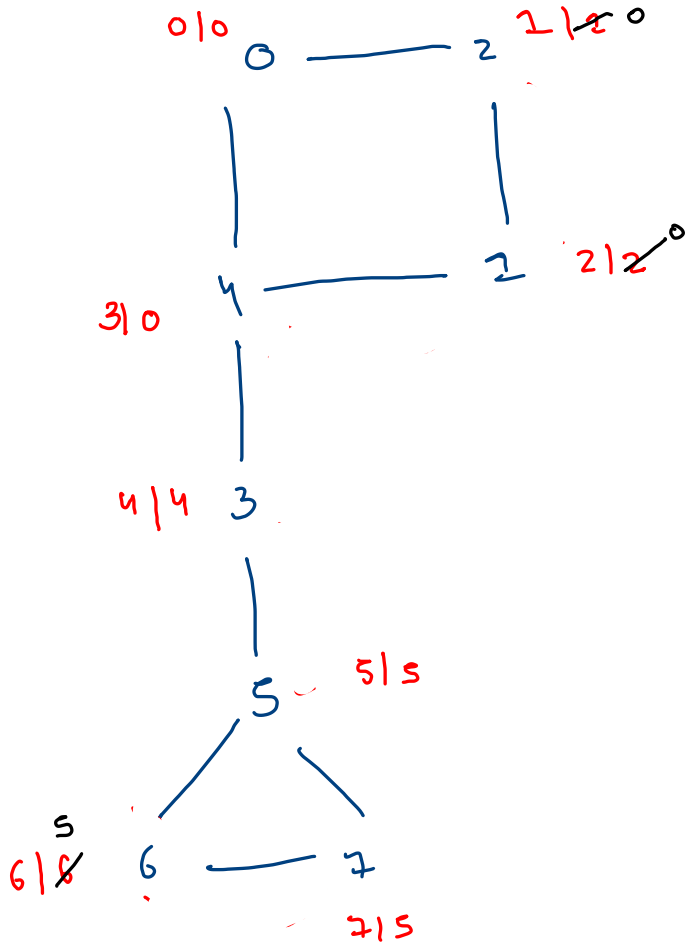
remove(0)

back_track (add (0))

cc = 1

cc = 1

Articulation point



DFS

par[ ]

disc[ ]

low[ ] 4^v

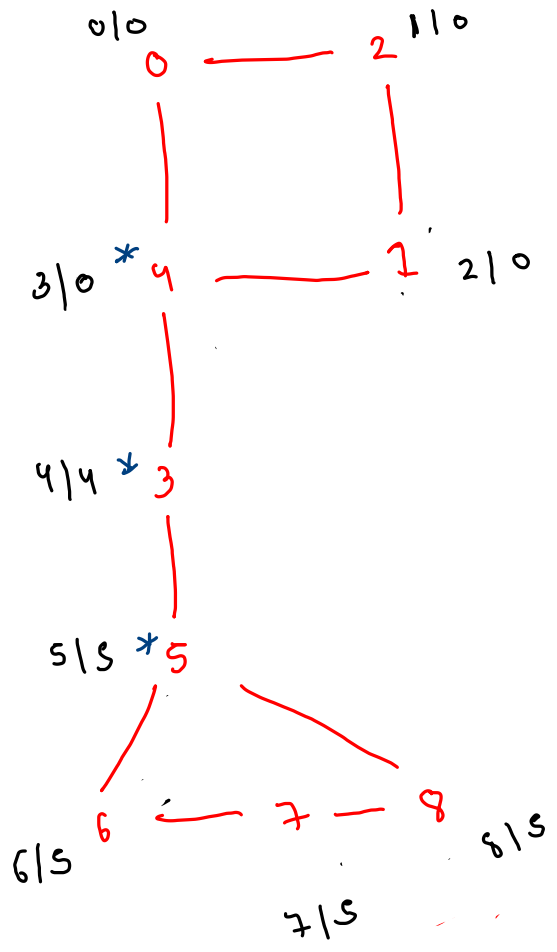
u — v 3^v

deciding whether u is an ap.

if(disc[u] ≤ low[v]){

u → ap

}

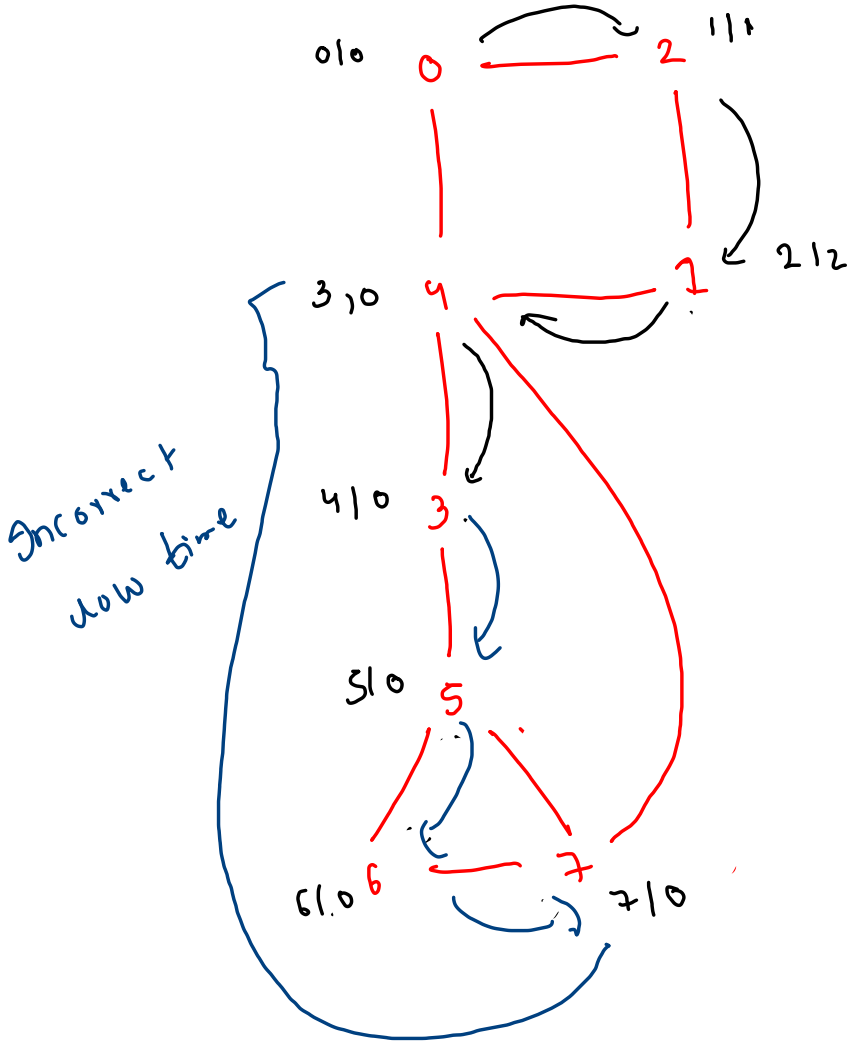


nbr

```
if (nbr == par) {
    // ignore;
}
else if (nbr == unvisit){
        dfs , solve
        low[src] = Math.min(low[src],
                            low[nbr]);
}
else if (nbr == visited)
    low[src] = Math.min(low[src],
                        disc[nbr])
}
if (disc[src] <= low[nbr]){
        src -> ap
}
```

u -> src

v -> nb



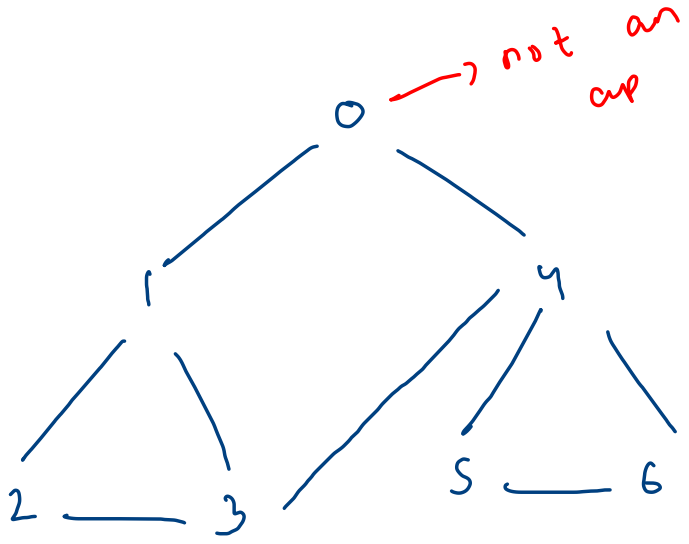
loop

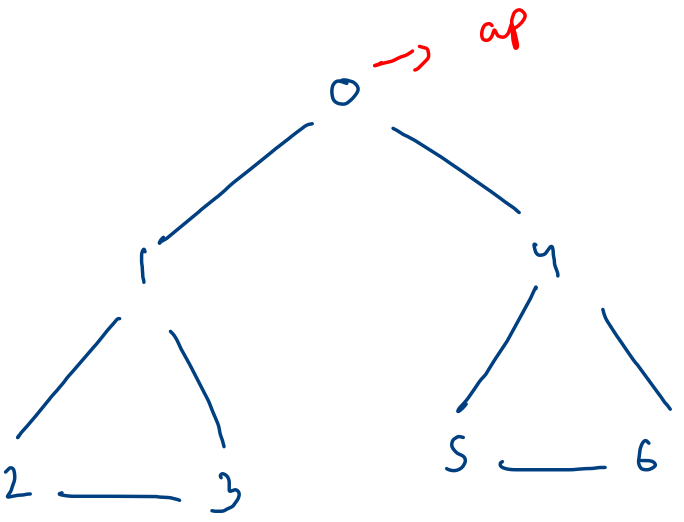
```
if (nbr == par) {
    // ignore;
}
else if (nbr == unvisit){
        dfs , solve
    low[src] = Math.min(low[src], low[nbr]);
}
else if (nbr == visited)
    low[src] = Math.min(low[src], disc[nbr])
}
```

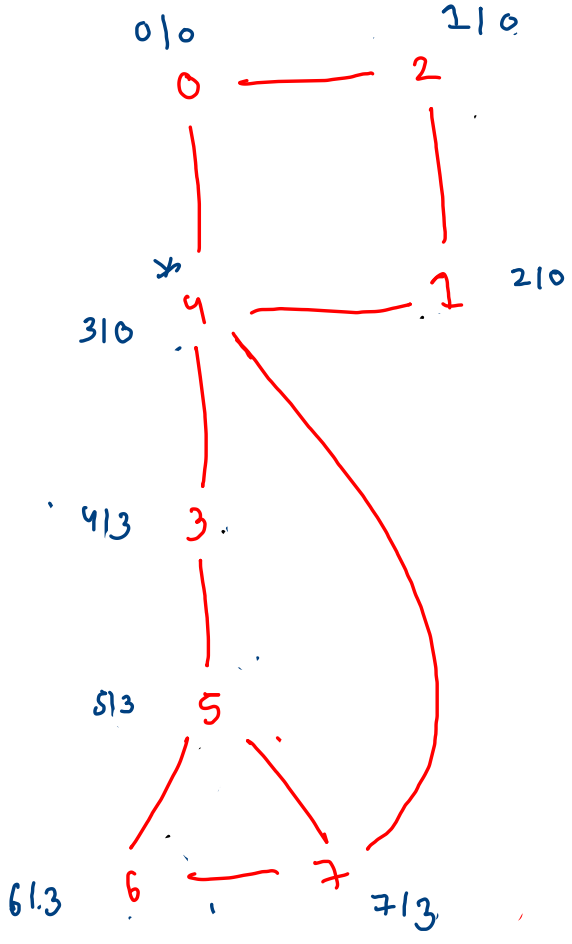
u -> src
v -> nb

~~disc~~ low

```
if (disc[src] <= low[nbr]){
        src -> ap
}
```


0
not an ap
1
4
2
3
5
6


0
ap
1
4
2
3
5
6



d | l

dfsc = 0

(0)

par

0 -> -1

1 -> 2

2 -> 0

3 -> 4

4 -> 1

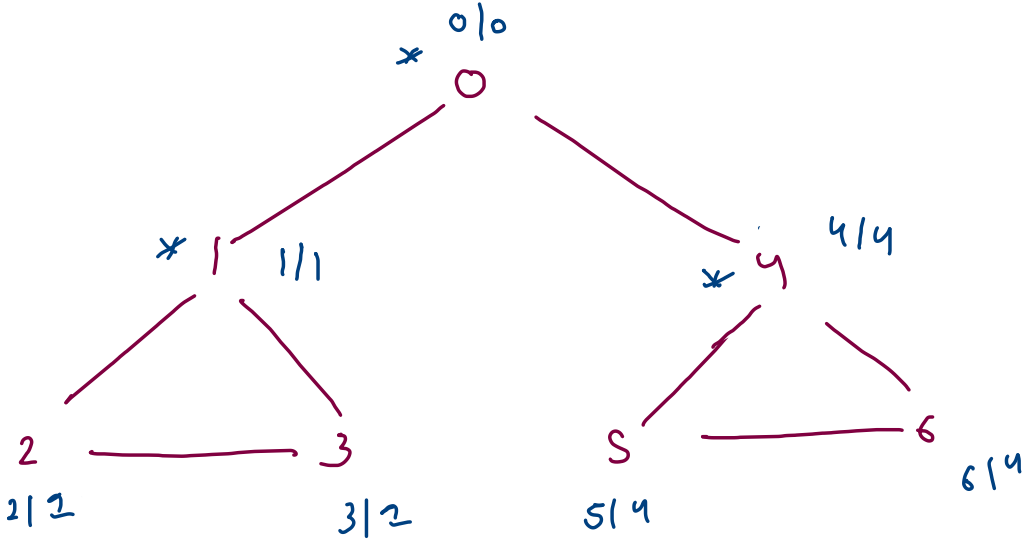
5 -> 3

6 -> 5

7 -> 6

```
vis[src] = true;

disc[src] = low[src] = time;
time++;

for(int nbr : graph[src]) {
    if(par[src] == nbr) {
        //nothing to do
    }
    else if(vis[nbr] == true) {
        //visited nbr
        low[src] = Math.min(low[src],disc[nbr]);
    }
    else if(vis[nbr] == false) {


        //unvisited nbr
        par[nbr] = src;
        articulation_point(nbr,graph,vis);
        low[src] = Math.min(low[src],low[nbr]);


        if(par[src] == -1) {
            //original src
            dfsc++;

            if(dfsc >= 2) {
                ap[src] = true;
            }
        }
        else if(disc[src] <= low[nbr]) {
            ap[src] = true;
        }
    }
}
```

dfsc = 1



dfsc = ~~x~~ 2

0 , 1, 4

```
vis[src] = true;

disc[src] = low[src] = time;
time++;

for(int nbr : graph[src]) {
    if(par[src] == nbr) {
        //nothing to do
    }
    else if(vis[nbr] == true) {
        //visited nbr
        low[src] = Math.min(low[src],disc[nbr]);
    }
    else if(vis[nbr] == false) {


        //unvisited nbr
        par[nbr] = src;
        articulation_point(nbr,graph,vis);
        low[src] = Math.min(low[src],low[nbr]);


        if(par[src] == -1) {
            //original src
            dfsc++;

            if(dfsc >= 2) {
                ap[src] = true;
            }
        }
        else if(disc[src] <= low[nbr]) {
            ap[src] = true;
        }
    }
}
```

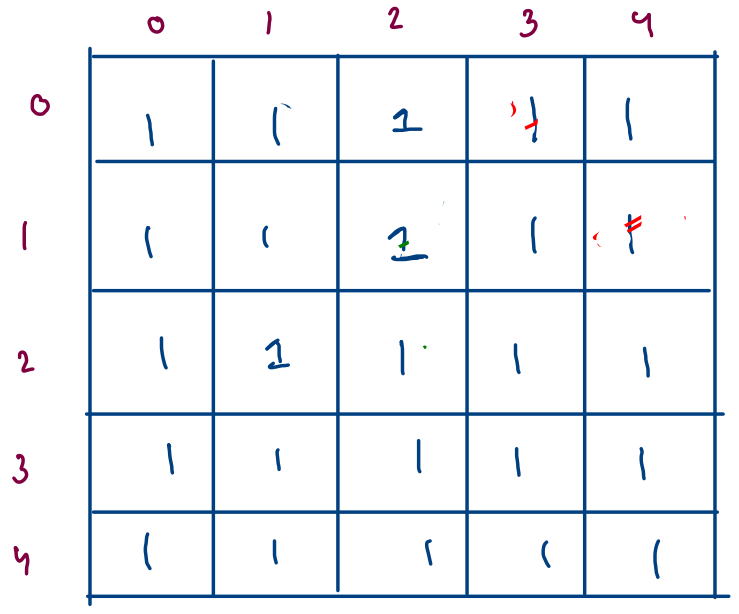
bridge ->

disc[src] < low[nbr] {

src to nbr;

}

```
vis[src] = true;

disc[src] = low[src] = time;
time++;

for(int nbr : graph[src]) {
    if(par[src] == nbr) {
        //nothing to do
    }
    else if(vis[nbr] == true) {
        //visited nbr
        low[src] = Math.min(low[src],disc[nbr]);
    }
    else if(vis[nbr] == false) {


        //unvisited nbr
        par[nbr] = src;
        articulation_point(nbr,graph,vis);
        low[src] = Math.min(low[src],low[nbr]);


        if(par[src] == -1) {
            //original src
            dfsc++;

            if(dfsc >= 2) {
                ap[src] = true;
            }
        }
        else if(disc[src] <= low[nbr]) {
            ap[src] = true;
        }
    }
}
```

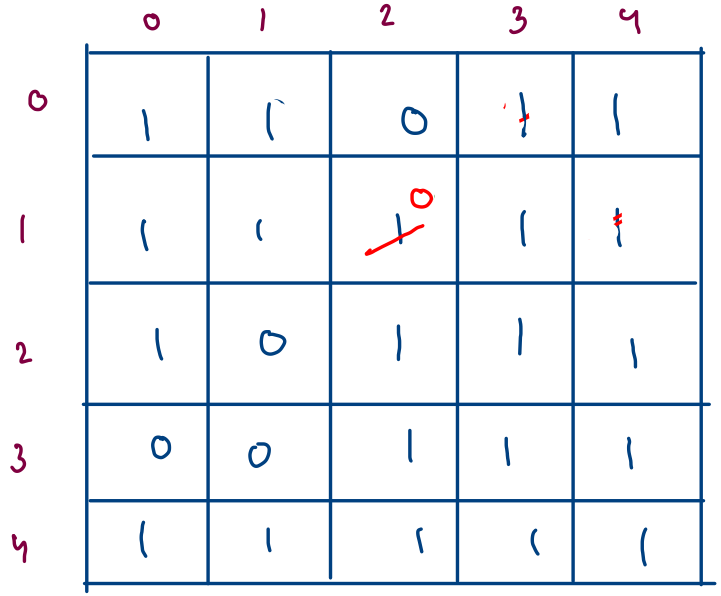


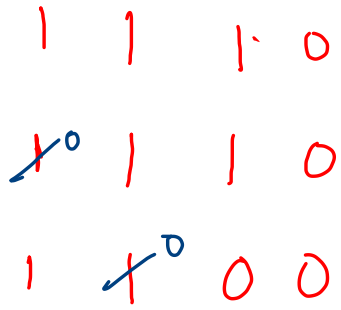
islands = 2 2 [

return 0;

]

min days -> either 1 or 2





ans = 1

islands

```
if (islands >= 2) {
    // grid is already disconnected
    return 0;
}
else {  // ans 1 or 2
    [ convert each one -> zero
      if(cc > prev-cc) {
          return 1;
      }
    ]
    return 2;
}
```