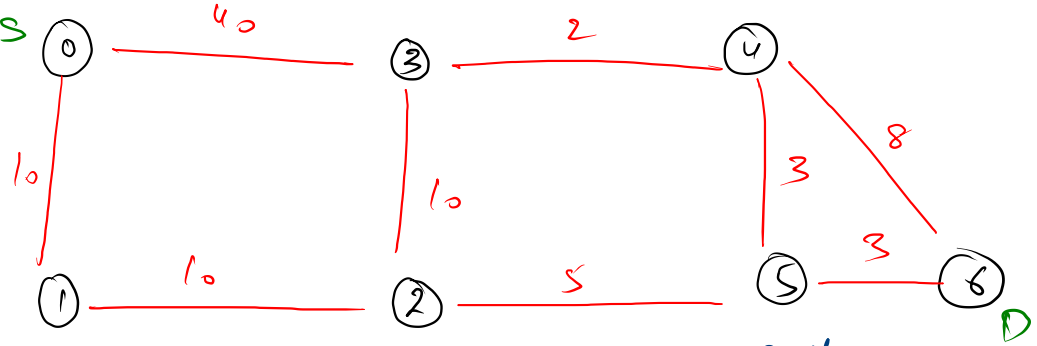
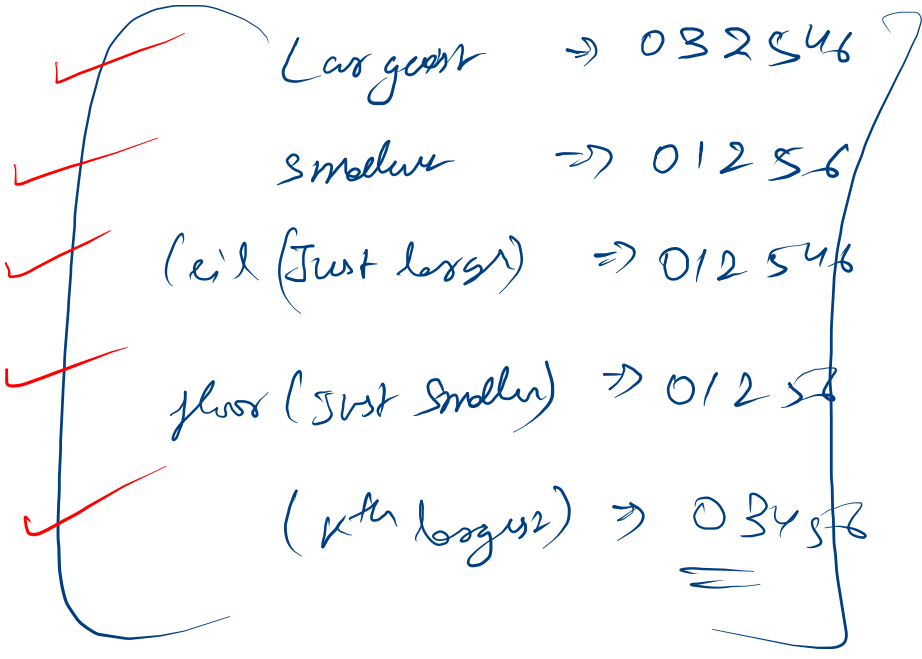
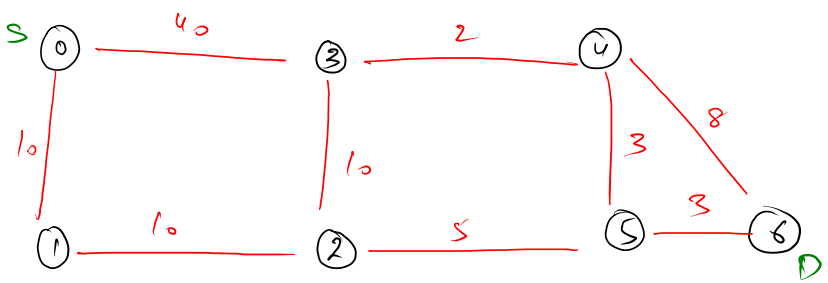
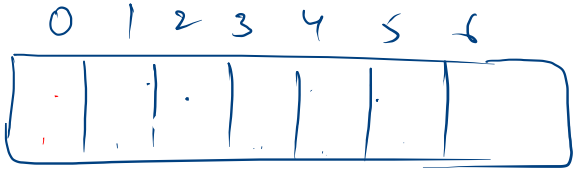


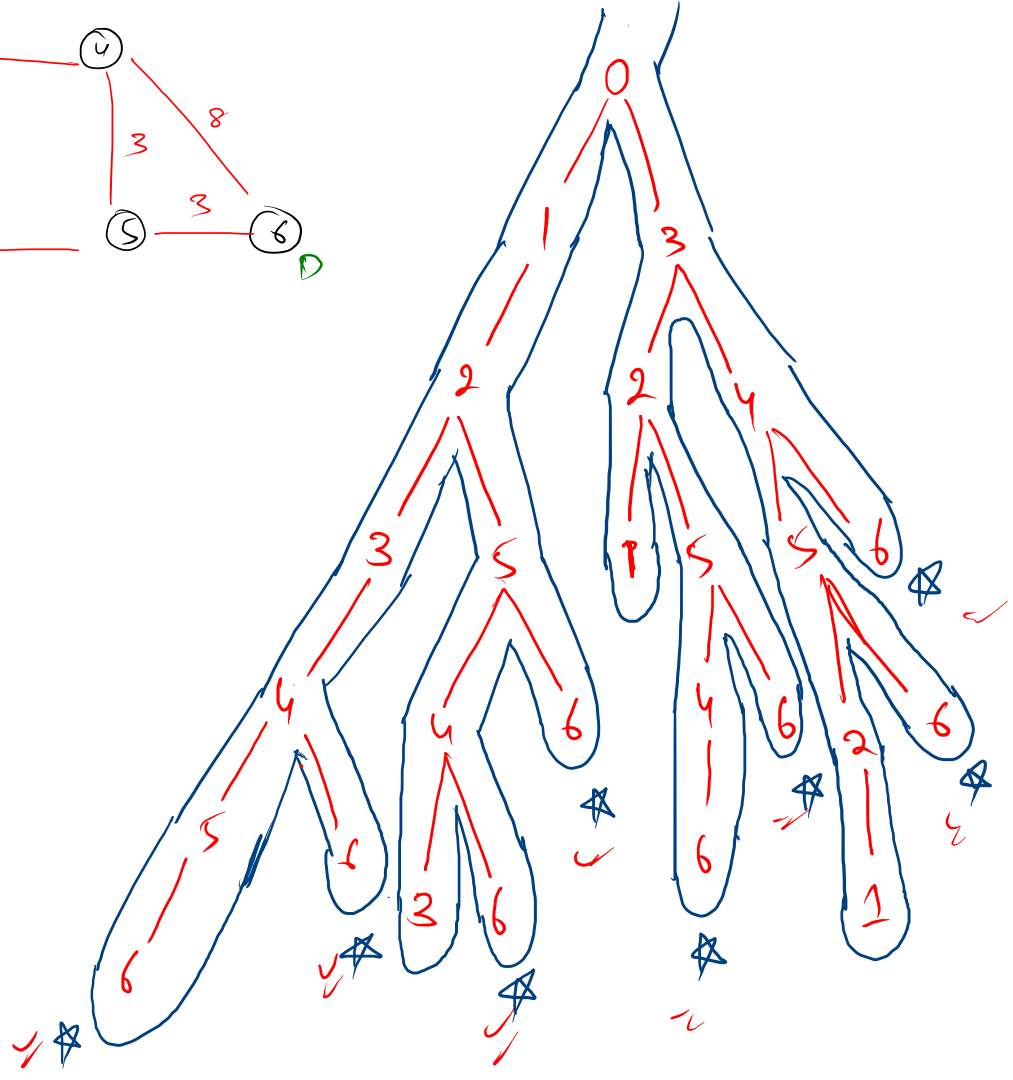

```
7
9
0 1 10
1 2 10
2 3 10
0 3 40
3 4 2
4 5 3
5 6 3
4 6 8
2 5 5
0 → Src
6 → dest
30 → Criteria
4 → K
```

Path		Cost
→ 0 3 4 6	⇒	50
→ 0 3 4 5 6	⇒	48
→ 0 1 2 5 6	⇒	28
→ 0 1 2 5 4 6	⇒	38
→ 0 1 2 3 4 6	⇒	40
→ 0 1 2 3 4 5 6	⇒	38
→ 0 3 2 5 6	⇒	58
→ 0 3 2 5 4 6	⇒	46









0123456@38
012346@40
012546@36
01256@28
032546@66
03256@58
03456@48
0346@50

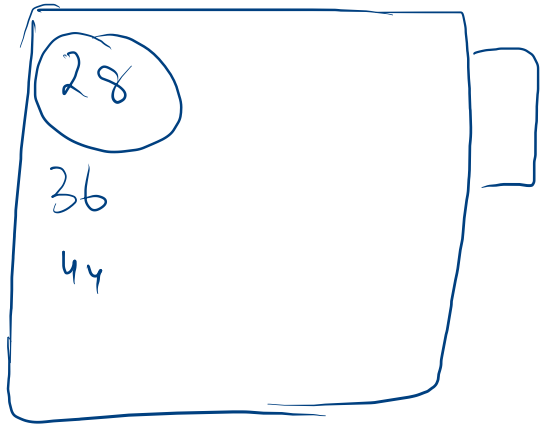
```
if(src == dest){
    // System.out.println(psf+"@"+wsf);

    if(wsf > lpathwt){ // largest
        lpathwt = wsf;
        lpath = psf;
    }

    if(wsf < spathwt){ // smallest
        spathwt = wsf;
        spath = psf;
    }

    if(wsf > criteria){ // larger
        if(wsf < cpathwt){ // min
            cpathwt = wsf;
            cpath = psf;
        }
    }

    if(wsf < criteria){ // smaller
        if(wsf > fpathwt){ // max
            fpathwt = wsf;
            fpath = psf;
        }
    }

    if(pq.size() < k){
        pq.add(new Pair(wsf,psf));
    }else{
        if(pq.peek().wsf < wsf){
            pq.remove();
            pq.add(new Pair(wsf,psf));
        }
    }
    return;
}
```

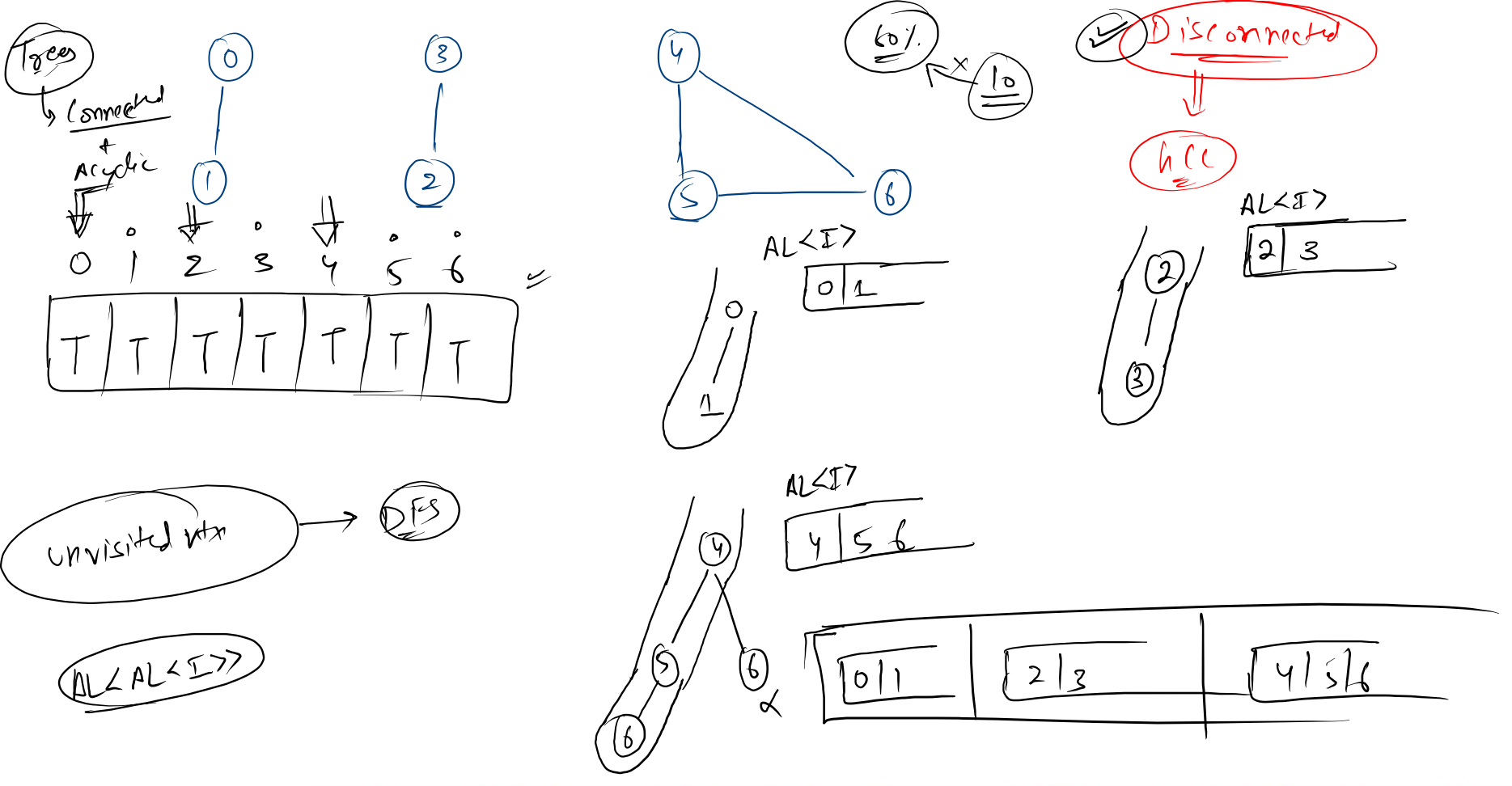
```
vis[src] = true; // mark

for(Edge e : graph[src]){
    if(vis[e.nbr] == false){
        multisolver(graph,e.nbr,dest,vis,criteria,k,psf+e.nbr,wsf+e.wt);
    }
}
vis[src] = false; // unmark
```

10 5 44 6 8 3 24 15 19 28 36

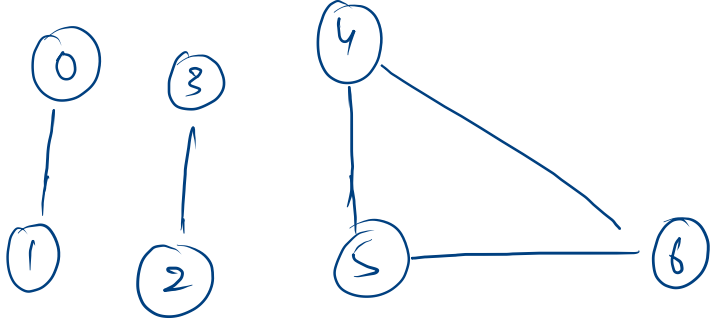
3rd

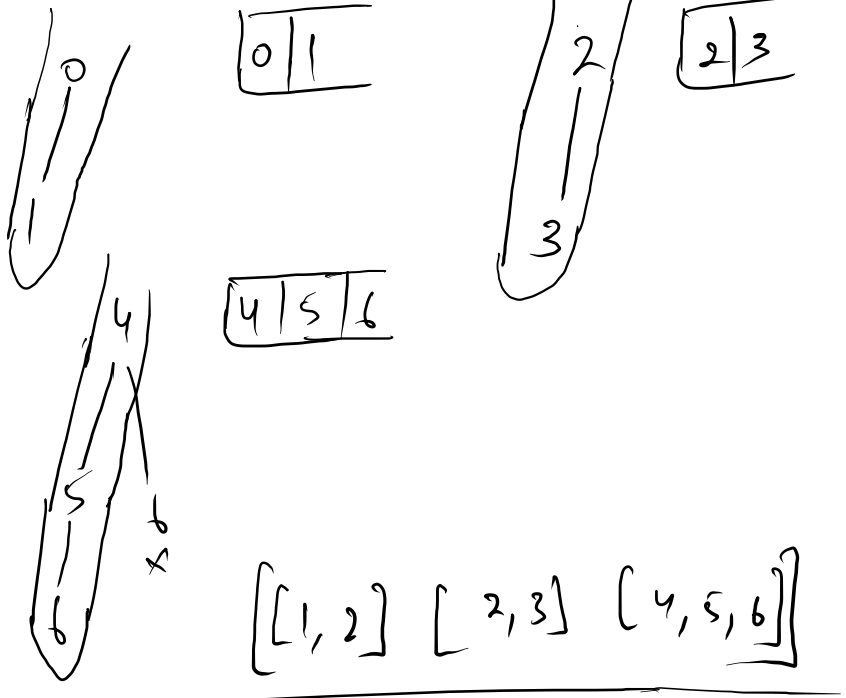
28






```java
public static ArrayList<ArrayList<Integer>> GCC(ArrayList<Edge>[] graph){

}

public static void DFS(ArrayList<Edge>[] graph,int vtx,boolean vis[],ArrayList<Integer> res){

}
```

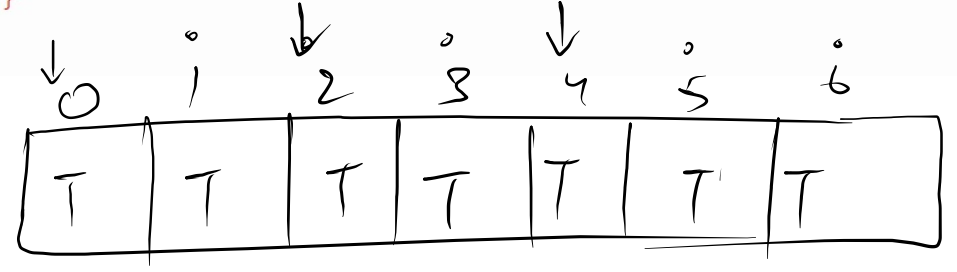




[[1, 2] [2, 3] [4, 5, 6]]

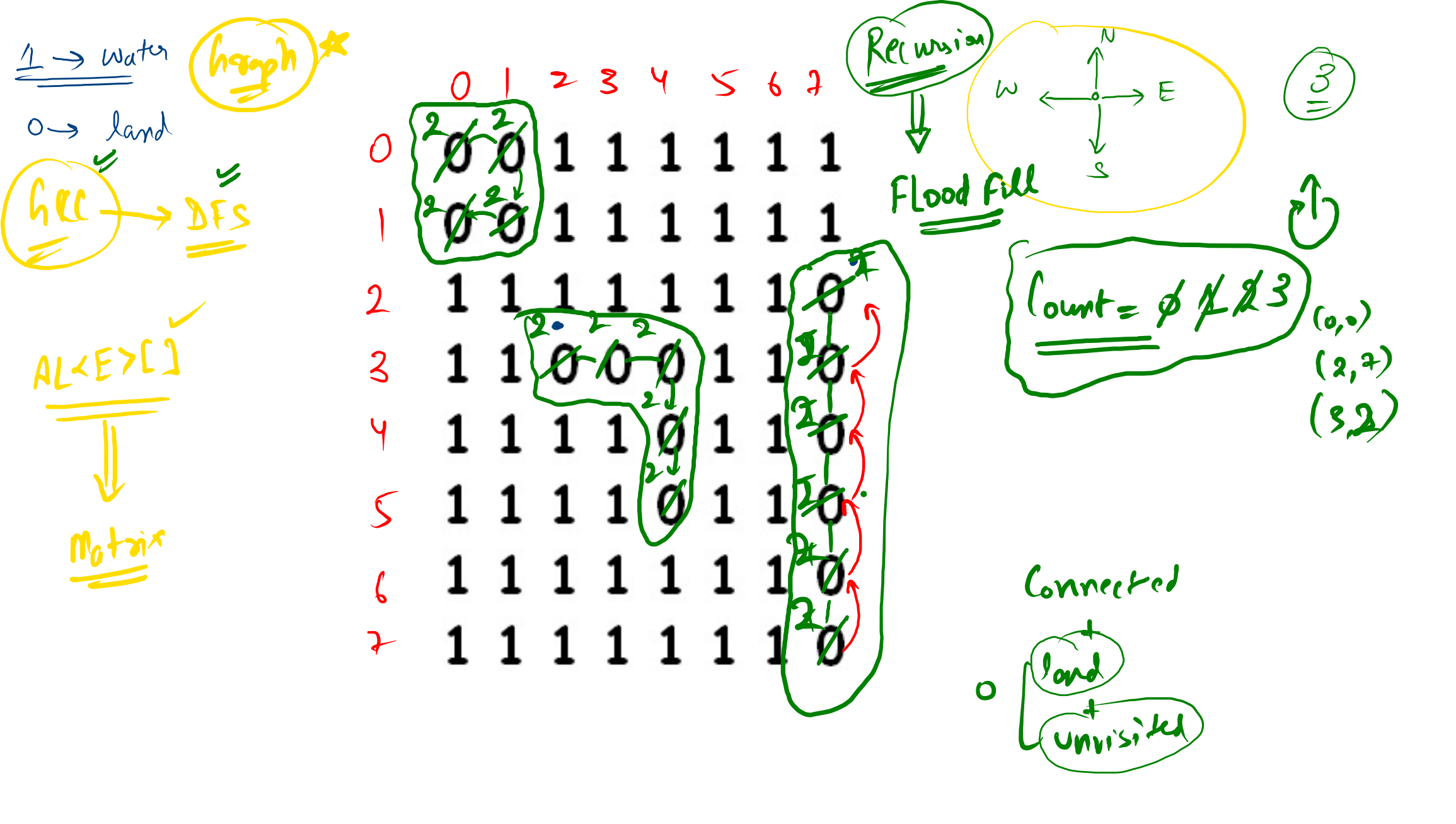
ans = | 1 2 | 2 3 | 4 5 6 |

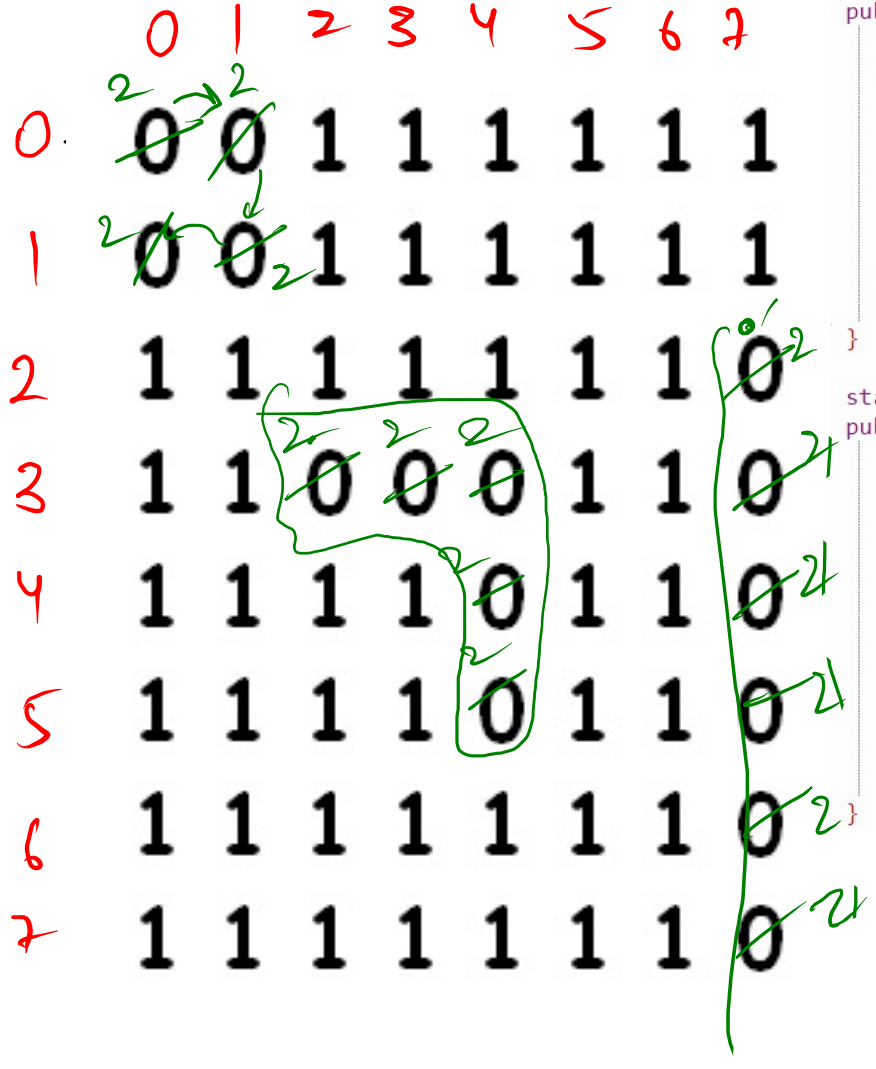
```java
public static ArrayList<ArrayList<Integer>> GCC(ArrayList<Edge>[] graph){
    ArrayList<ArrayList<Integer>> ans = new ArrayList<>();
    boolean vis[] = new boolean[graph.length];

    for(int i = 0 ; i < graph.length ; i++){
        if(vis[i] == false){
            ArrayList<Integer> res = new ArrayList<>();
            DFS(graph,i,vis,res);
            ans.add(res);
        }
    }
    return ans;
}

public static void DFS(ArrayList<Edge>[] graph,int vtx,boolean vis[],ArrayList<Integer> res){
    vis[vtx] = true;
    res.add(vtx);

    for(Edge e : graph[vtx]){
        if(vis[e.nbr] == false){
            DFS(graph,e.nbr,vis,res);
        }
    }
}
```



main

```java
ArrayList<ArrayList<Integer>> comps = GCC(graph);
System.out.println(comps);
```


1 → water
0 → land
graph
GCC
DFS
AL<E>[]
Matrix
Recursion
Flood Fill
N
W
E
S
3
count = 0 1 2 3
(0,0)
(2,7)
(3,2)
Connected
0
land
unvisited

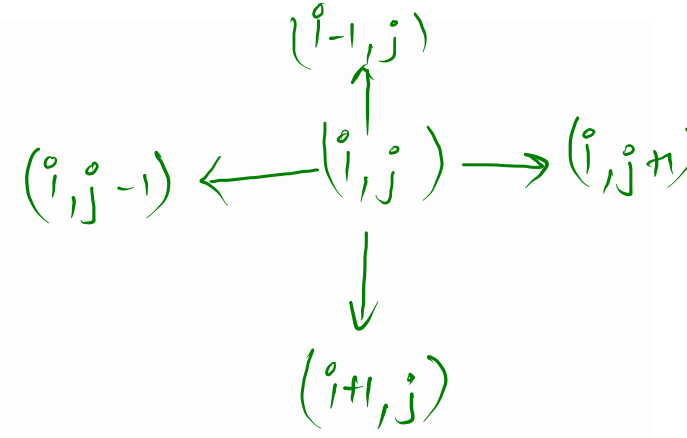


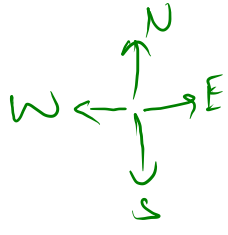

```java
public static int numOfIslands(int[][] board){
    int count = 0;
    for(int i = 0 ; i < board.length ; i++){
        for(int j = 0 ; j < board[0].length ; j++){
            if(board[i][j] == 0){
                count++;
                DFS(board,i,j);
            }
        }
    }
    return count;
}

static int dir[][] = {{-1,0},{0,+1},{+1,0},{0,-1}};
public static void DFS(int [][]board,int row,int col){
    board[row][col] = 2; // mark visited

    for(int d = 0 ; d < 4 ; d++){
        int rd = row+dir[d][0];
        int cd = col+dir[d][1];

        if(rd < 0 || cd < 0 || rd >= board.length || cd >= board[0].length || board[rd][cd] != 0){
            continue;
        }

        DFS(board,rd,cd);
    }
}
```

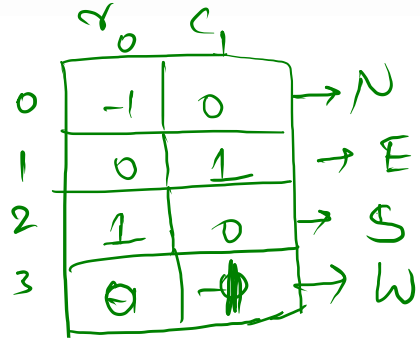
N E S W (0,-1)

Orange + unvisited land

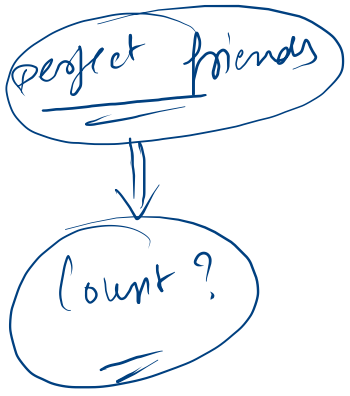
count = 3

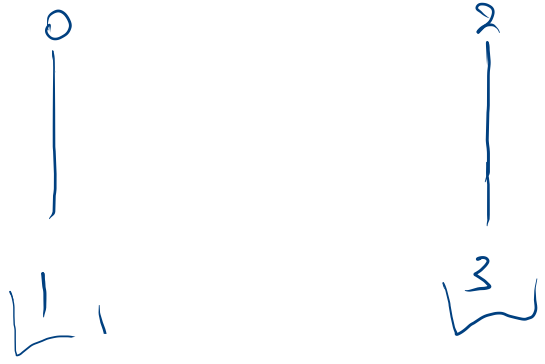


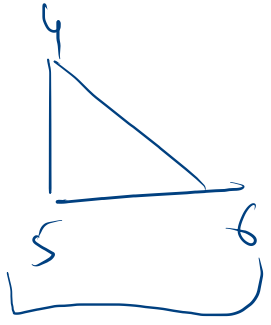




no. of perfect friends



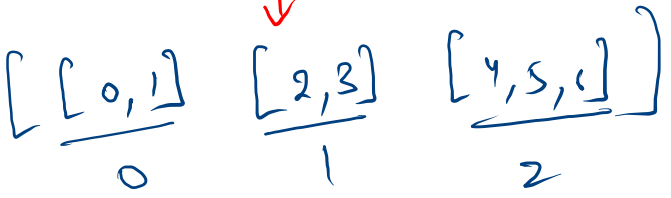




$${}^{n}C_{r} = \frac{n!}{(n-r)!\,r!}$$

(u, v)

hcc →



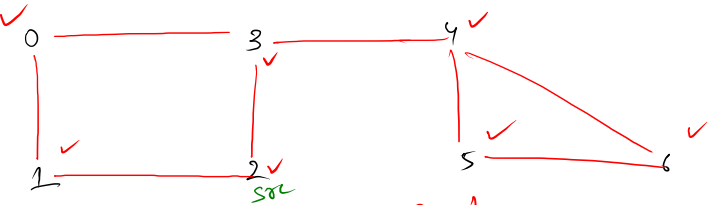
Count = $(h_0 \cdot h_1) + (h_0 \cdot h_2) + (h_1 \cdot h_2)$

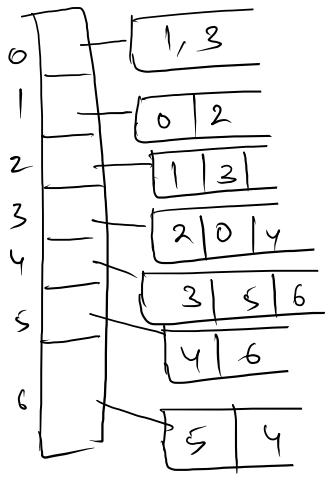
$= (2 \cdot 2) + (2 \cdot 3) + (2 \cdot 3)$

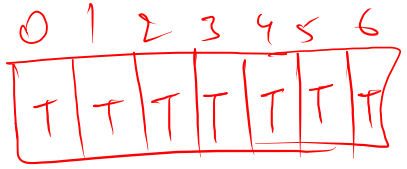
$\Rightarrow 4 + 6 + 6 \Rightarrow 16$

Count

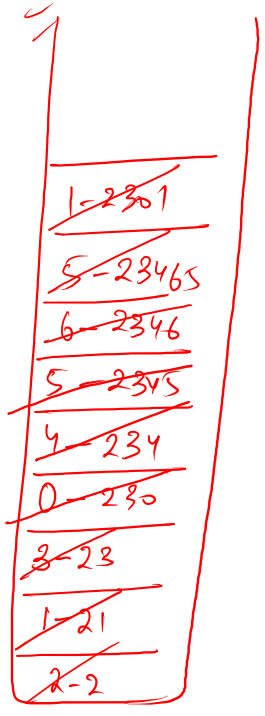
```
for(i=0 ; i<len; i++){
    for(j=i+1; j<len; j++){
        Count ____
    }
}
```

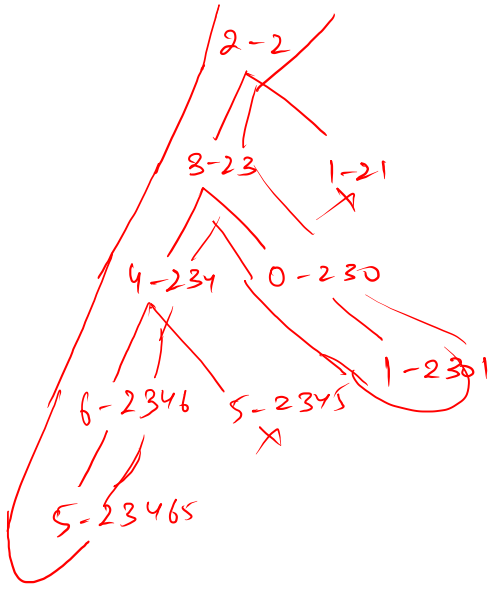




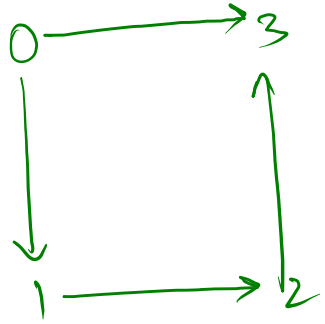


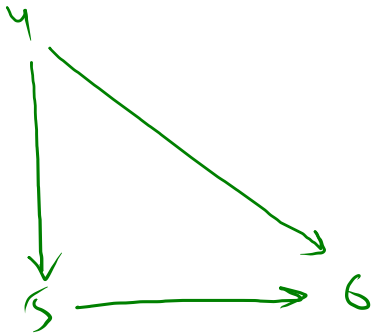
2@2
3@23
4@234
6@2346
5@23465
0@230
1@2301






```
        int src = Integer.parseInt(br.readLine());
        iDFS(graph,src);
    }
    public static class Pair{
        int vtx;
        String  psf;
        Pair(int vtx,String psf){
            this.vtx = vtx;
            this.psf = psf;
        }
    }
    public static void iDFS(ArrayList<Edge>[] graph,int src){
        Stack<Pair> st = new Stack<>();
        boolean vis[] = new boolean[graph.length];

        st.push(new Pair(src,""+src));

        while(st.size() > 0){
            Pair rem = st.pop();

            if(vis[rem.vtx] == true){
                continue;
            }else{
                vis[rem.vtx] = true;
                System.out.println(rem.vtx+"@"+rem.psf);

                for(Edge e : graph[rem.vtx]){
                    if(vis[e.nbr] == false){
                        st.push(new Pair(e.nbr,rem.psf+e.nbr));
                    }
                }
            }
        }
    }
}
```





Graph → multiple Topological sort

0 1 2 3 4 5 6

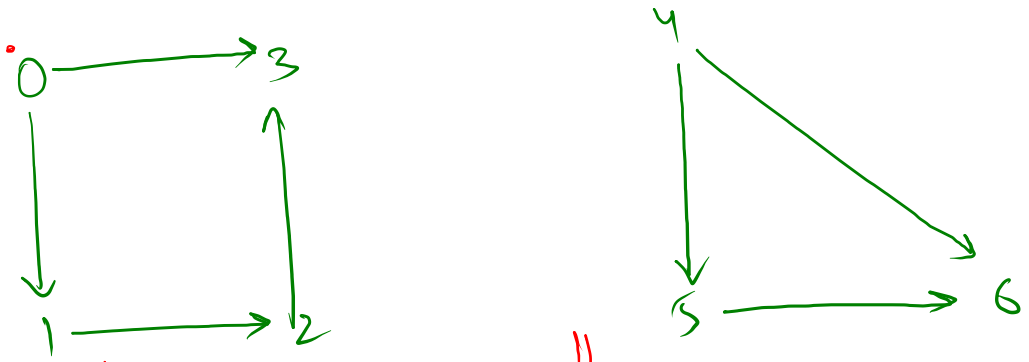
0 4 1 2 3 5 6

0 1 2 4 3 5 6

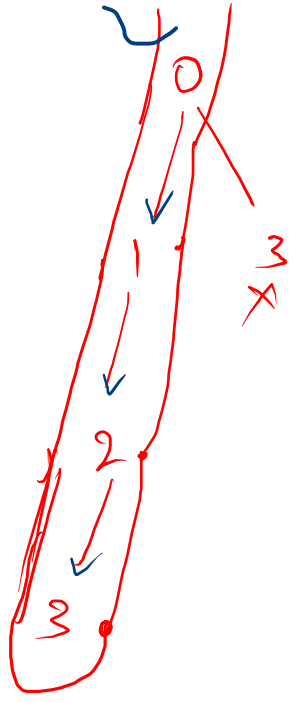
Topological sort ⇒ Vtx Linear Order + Condition

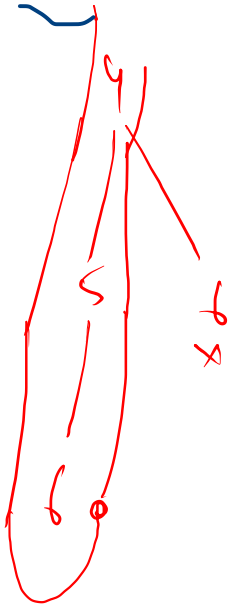
(u → v)

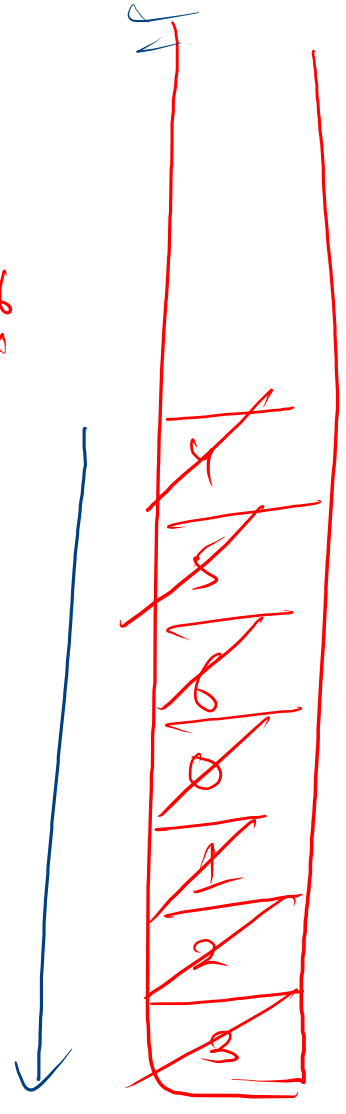
(u v)

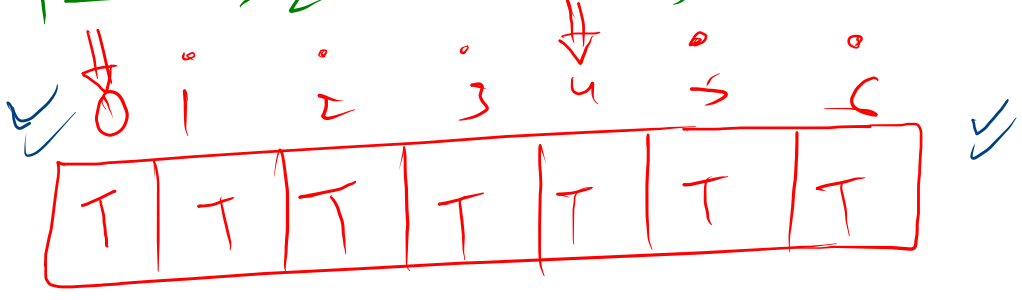


DFS









4 5 6 0 1 2 3

T.S. ⇒ DFS