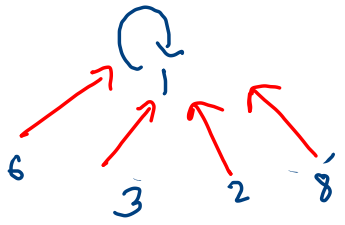
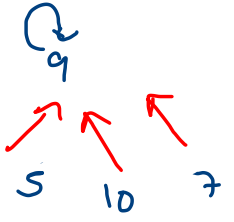
DSU -> disjoint set union

Parent

X	1	~~1~~ 2	~~1~~ 3	4	~~9~~ 5	~~1~~ 6	~~9~~ 7	~~1~~ 8	9	~~9~~ 10
0	1	2	3	4	5	6	7	8	9	10



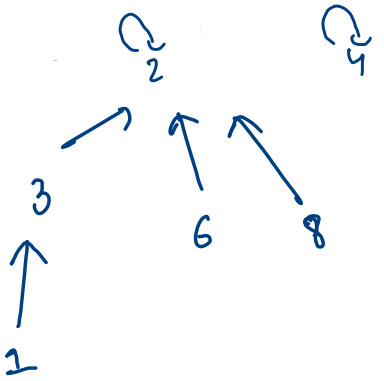


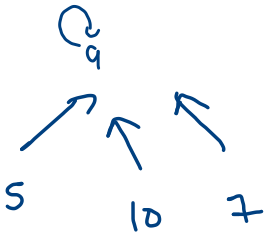
relation

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

Parent

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





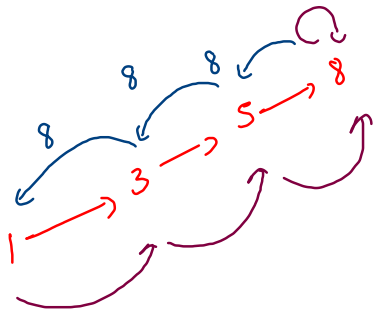
3 groups

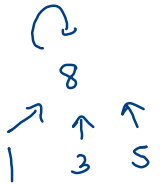
```
find (int x) {
    if( par[x] == x ){
        return x;
    }
    return find(par[x]);
}
union ( int x, int y){
    dx = find(x);
    dy = find (y);
    if(dx != dy){
       par[dx] = dy;
    }
}
```

relations

- 1 – 3
- 3 – 2
- 5 – 9
- 6 – 3
- 10 – 5
- 7 – 5
- 8 – 2

DSU optimisation → (i) path compression





parent

1 -> ~~3~~ 8

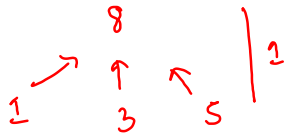
3 -> ~~5~~ 8

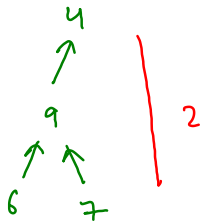
5 -> 8

8 -> 8

find(1)

(ii) union by rank:



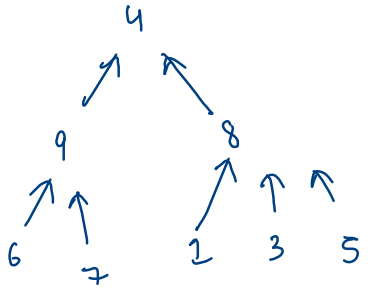


a) par[6] = 4;

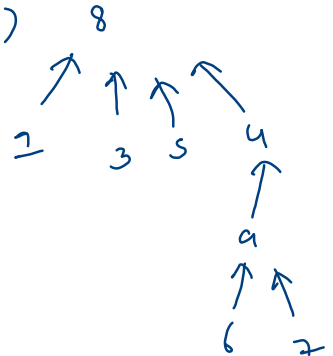
b) par[4] = 8;

c) both same

a)



b)



parent

X	1 → 3	2 → 3	3 → 9	4	5 → 9	6 → 3	7 → 9	8 → 3	9	10 → 9
0	1	2	3	4	5	6	7	8	9	10

rank

X	0	0	0 → 1	0	0	0	0	0	1 → 2	0
0	1	2	3	4	5	6	7	8	9	10

1 – 3

3 – 2

5 – 9

6 – 3

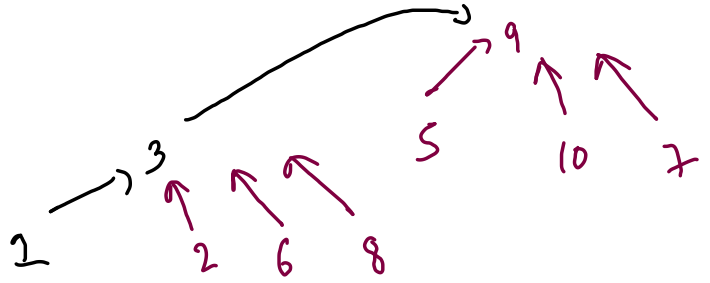
10 – 5

7 – 9

8 – 2

5 – 2

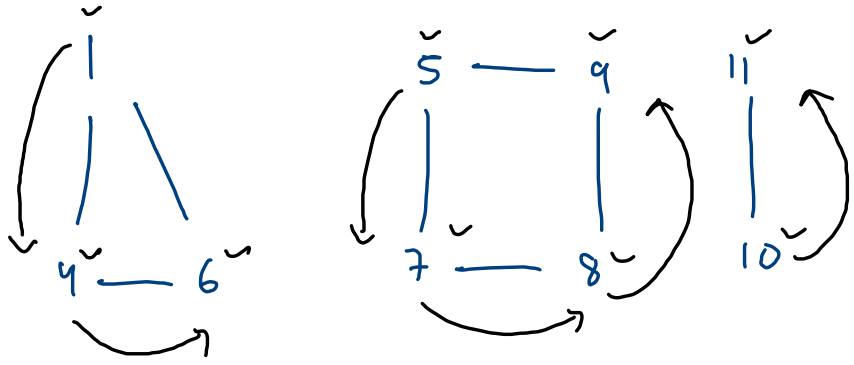
y



x = 5 rx = 9 rx = 1

y = 2 dy = 3 ry = 1

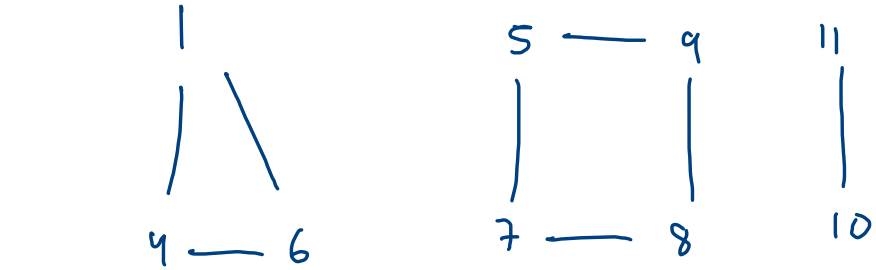
DFS

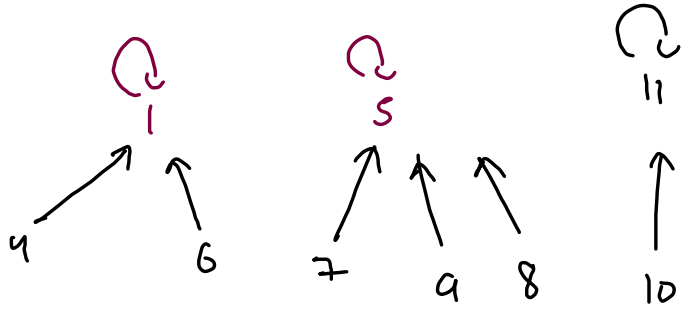


1 ✓ 5 ✓ 10 ✓

Count = ~~2~~ ~~2~~ 3

DSU





1 -> 4, 6

4 -> 1, 6

6 -> 1, 4

5 -> 7, 9

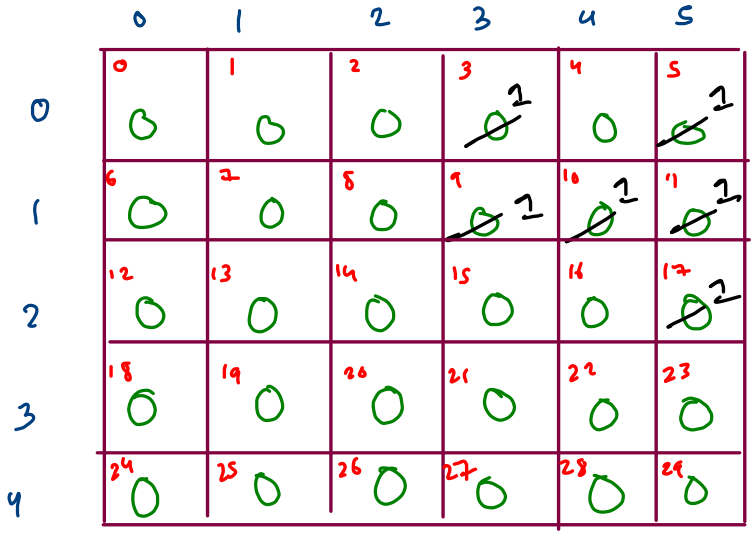
7 -> 5, 8

9 -> 5, 8

8 -> 7, 9

11 -> 10

10 -> 11



c = cols

i, j

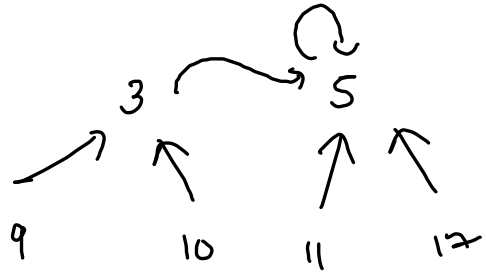
bno. = i * cols + j

updations :

(0,3) (0,5) (1,3) (2,5) (1,4) (1,5)

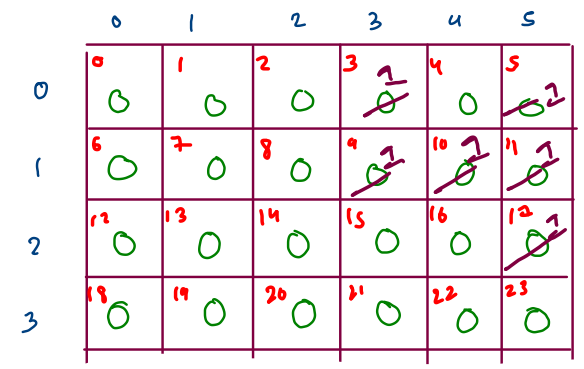
1 2 2 3 3 1

c = ~~0~~ ~~1~~ 2 ~~3~~ 2 ~~3~~ 4 ~~3~~ ~~4~~ ~~3~~ 2 1



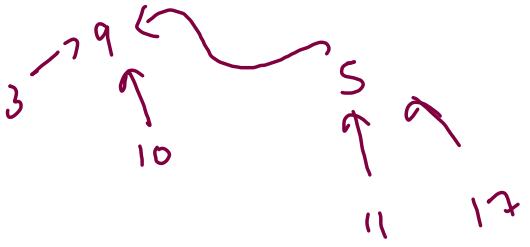
Conn. comps

graph dynamic → DSU



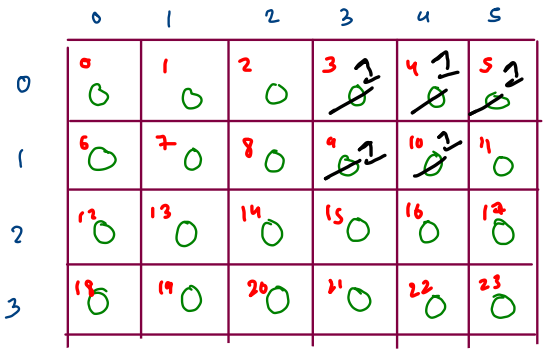
(0,3) (0,5) (1,3) (2,5) (1,3) (1,4) (1,5)

count = 1 2 3 4 3 4 3 2 1

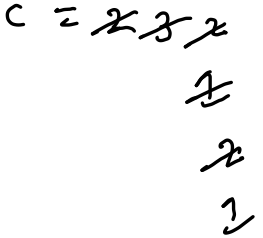


1 2 2 3 3 3 1

```
for(Point p : operators) {
    int px = p.x;
    int py = p.y;

    if(grid[px][py] == 1) {
        ans.add(count);
        continue;
    }

    grid[px][py] = 1;

    count++;

    //merge if required with nbr
    for(int[]d : dir) {
        int nx = px + d[0];
        int ny = py + d[1];

        if(nx >= 0 && nx < n && ny >=0 && ny < m && grid[nx][ny] == 1) {
            int bnp = px*m + py; //box number parent
            int bnn = nx*m + ny; //box number number

            //union
            int lp = find(bnp);
            int ln = find(bnn);

            if(lp != ln) {
                //merging is required
                if(rank[lp] < rank[ln]) {
                    parent[lp] = ln;
                }
                else if(rank[lp] > rank[ln]) {
                    parent[ln] = lp;
                }
                else {
                    parent[lp] = ln;
                    rank[ln]++;
                }
                count--;
            }
        }
    }

    ans.add(count);
}

return ans;
```



0,3 0,5 1,3 0,4 1,4



1 2 2 1 1

```
for(Point p : operators) {
    int px = p.x;
    int py = p.y;

    if(grid[px][py] == 1) {
        ans.add(count);
        continue;
    }

    grid[px][py] = 1;

    count++;

    //merge if required with nbr
    for(int[]d : dir) {
        int nx = px + d[0];
        int ny = py + d[1];

        if(nx >= 0 && nx < n && ny >=0 && ny < m && grid[nx][ny] == 1) {
            int bnp = px*m + py; //box number parent
            int bnn = nx*m + ny; //box number number

            //union
            int lp = find(bnp);
            int ln = find(bnn);

            if(lp != ln) {
                //merging is required
                if(rank[lp] < rank[ln]) {
                    parent[lp] = ln;
                }
                else if(rank[lp] > rank[ln]) {
                    parent[ln] = lp;
                }
                else {
                    parent[lp] = ln;
                    rank[ln]++;
                }
                count--;
            }
        }
    }

    ans.add(count);
}

return ans;
```