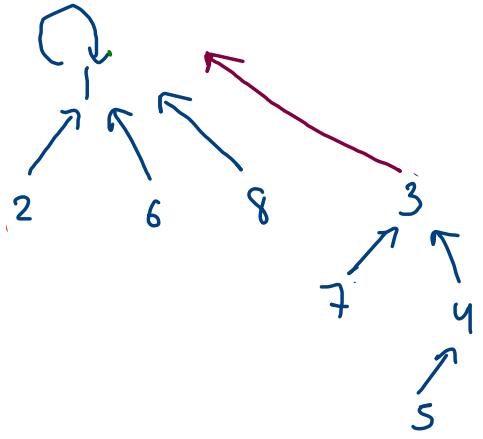
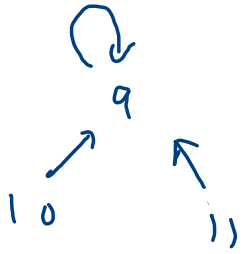
DSU : Disjoint set union

Parent

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





relations:

- ✓ 1 - 2
- ✓ 4 - 5
- ✓ 2 - 6
- ✓ 3 - 7
- ✓ 9 - 10
- ✓ 6 - 8
- ✓ 3 - 5
- ✓ 10 - 11
- ✓ 8 - 5

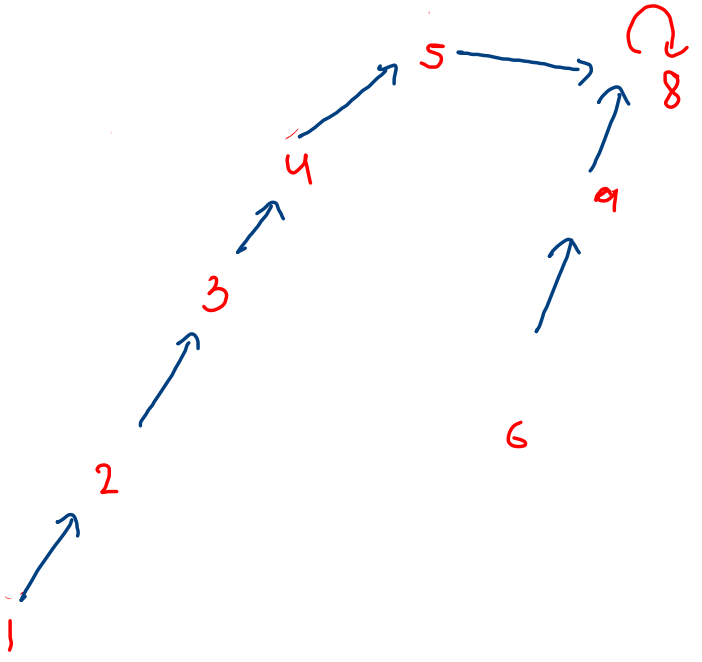
Parent

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

```
union (int x, int y) {
    dx = find (x);
    dy = find (y);
    // now dx -> dy
    par[dx] = dy;
}

int find (int x) {
    if (par[x] == x) {
        return x;
    }
    else {
        return find (par[x]);
    }
}
```

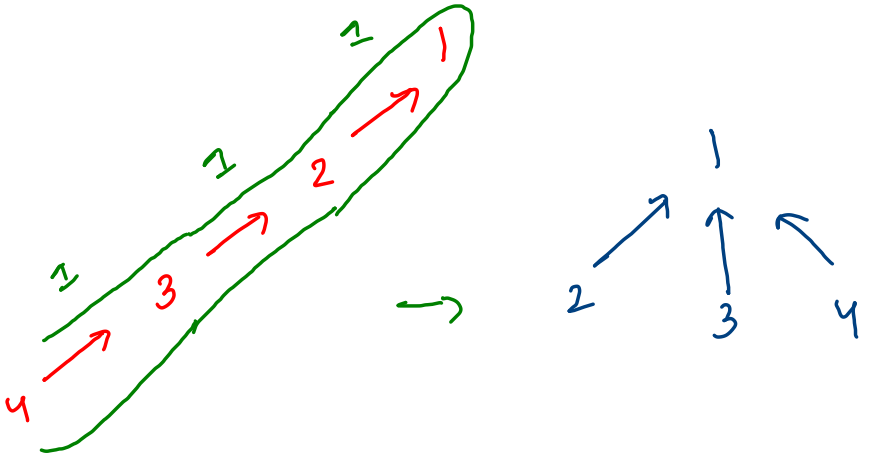
x = 3 y = 6

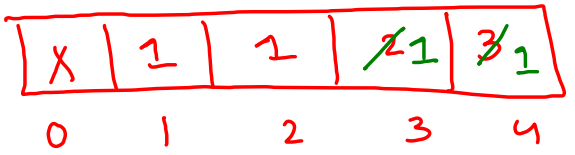
dx = 5 dy = 8



- 1 - 2
- 2 - 3
- 4 - 4
- 4 - 5
- 6 - 9
- 9 - 8
- 3 - 6

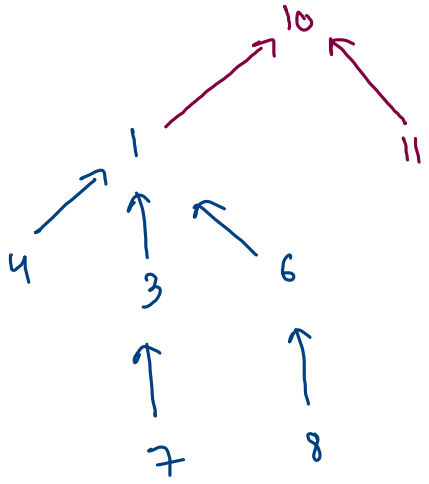
optimisation : i) path compression

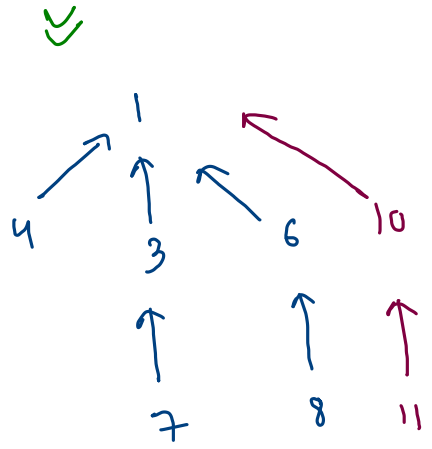





```
public static int find(int x) {
    if(par[x] == x) {
        return x;
    }
    else {
        return find(par[x]);
    }
}
```

(ii) union by rank






```
public static void union(int x,int y) {
    int lx = find(x);
    int ly = find(y);

    par[lx] = ly;
}
```

7 11

$l_x = 1$ rank : 2

$l_y = 10$ rank : 1

does rank should point to higher rank.

```java
for(int i=1; i <= n;i++) {
    par[i] = i;
    rank[i] = 0;
}

for(int i=0; i < relations.length;i++) {
    int x = relations[i][0];
    int y = relations[i][1];

    union(x,y);
}
```

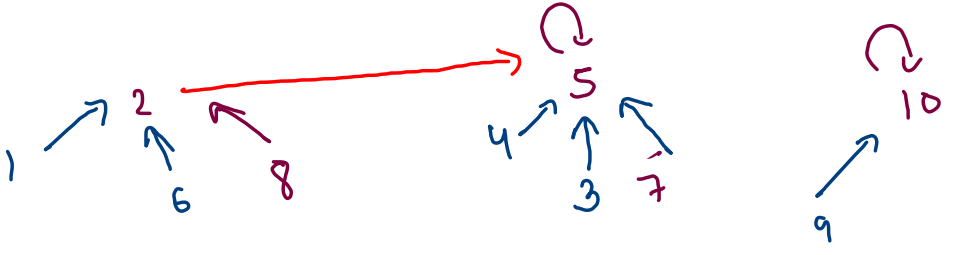
```java
public static void union(int x,int y) {
    int lx = find(x);
    int ly = find(y);

    //merging
    if(lx != ly) {
        if(rank[lx] < rank[ly]) {
            par[lx] = ly;
        }
        else if(rank[lx] > rank[ly]) {
            par[ly] = lx;
        }
        else {
            par[lx] = ly;
            rank[ly]++;
        }
    }
}

public static int find(int x) {
    if(par[x] == x) {
        return x;
    }
    else {
        int ans = find(par[x]);
        par[x] = ans; //path compression
        return ans;
    }
}
```

par

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

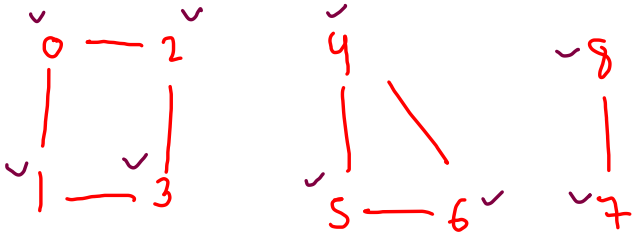
rank

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



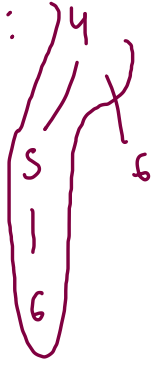
- 1 - 2
- 4 - 5
- 2 - 6
- 3 - 7
- 9 - 10
- 6 - 8
- 3 - 5
- 3 - 8

DFS



0:

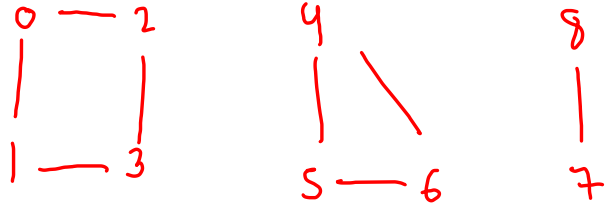
4:

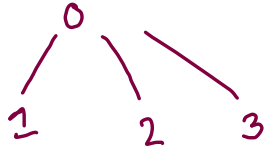


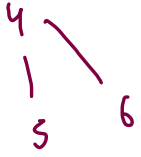
Comp = ~~0~~ ~~1~~ ~~2~~ 3

7:

DSU (Dynamic graph)







0 — 1

0 — 2

1 — 3

2 — 3

4 — 5

5 — 6

8 — 7

4 — 6

Number of Islands II

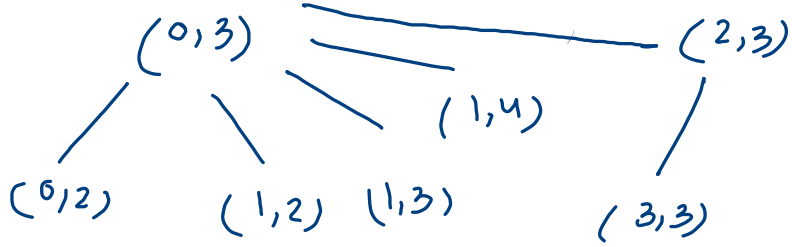
Dynamic graph

	0	1	2	3	4
0	0	0	1	1	0
1	0	0	1	1	1
2	0	0	0	1	0
3	0	0	0	1	0
4	0	0	0	0	0

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

ans: 1 2 1 2 3 3 1

c : 0 1 2 3 4 3 4 3 2 1



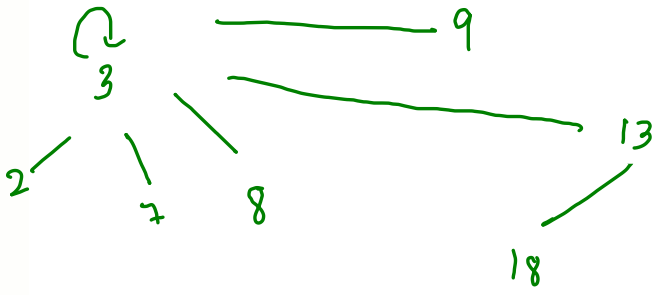
```java
for(int k=0; k < operators.length;k++) {
    int i = operators[k].x;
    int j = operators[k].y;

    if(mat[i][j] == 1) {
        ans.add(count);
        continue;
    }

    mat[i][j] = 1;
    count++;

    for(int d = 0; d < 4;d++) {
        int ni = i + dir[d][0];
        int nj = j + dir[d][1];

        if(ni >= 0 && ni < n && nj >= 0 && nj < m && mat[ni][nj] == 1) {
            int sc = i * m + j; //source cell no.
            int nc = ni * m + nj; //nbr cell no.

            //union
            int ls = find(sc); //leader of source
            int ln = find(nc); //leader of nbr

            if(ls != ln) {
                //merging is posssible
                if(rank[ls] < rank[ln]) {
                    par[ls] = ln;
                }
                else if(rank[ls] > rank[ln]) {
                    par[ln] = ls;
                }
                else {
                    par[ls] = ln;
                    rank[ln]++;
                }
                count--;
            }
            else {
                //do nothing
            }
        }
    }

    ans.add(count);
}
```

ans :

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

1 2 1 2 3 3 1

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

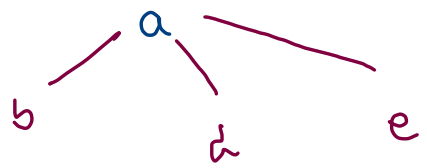


	0	1	2	3	4
0	0	0	1	1	0
1	0	0	1	~~0~~ 1	1
2	0	0	0	1	0
3	0	0	0	1	0
4	0	0	0	0	0

Satisfiability of Equality Equations

✓ F

a == b c != b d == b a == e e != b a == d



c

ch = 'a'