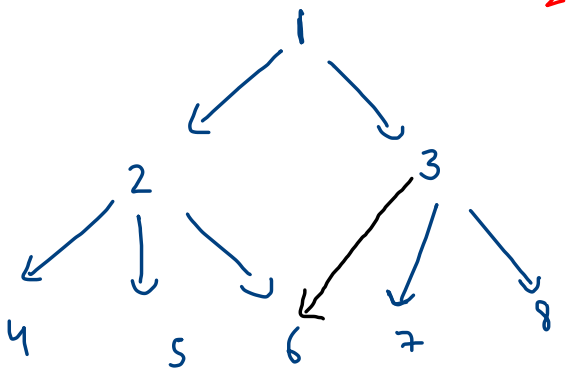
Redundant connection 2 : (graph -> directed)



2 parent

8th edge

1	2
1	3
2	4
2	5
2	6
3	7
3	6
3	8

redundant edge

ans -> 3,6

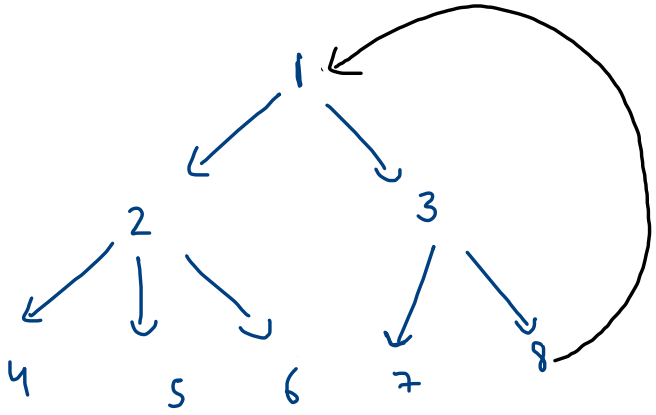
1	2	3	4	5	6	7	8
-1	0 ~~1~~	2 ~~1~~	2 ~~1~~	3 ~~1~~	4 ~~1~~	5 ~~1~~	-1

6th edge

e1 = 4

e2 = 6

cycle



2 – 4

2 – 5

2 – 6

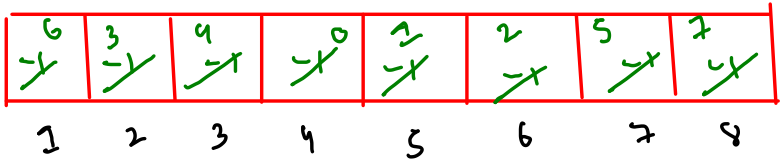
1 – 2

1 – 3

3 – 7

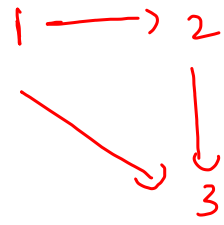
8 – 1

∴ 3 – 8



DSU

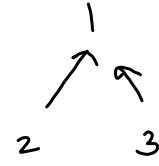
cycle detection fails directed graph.



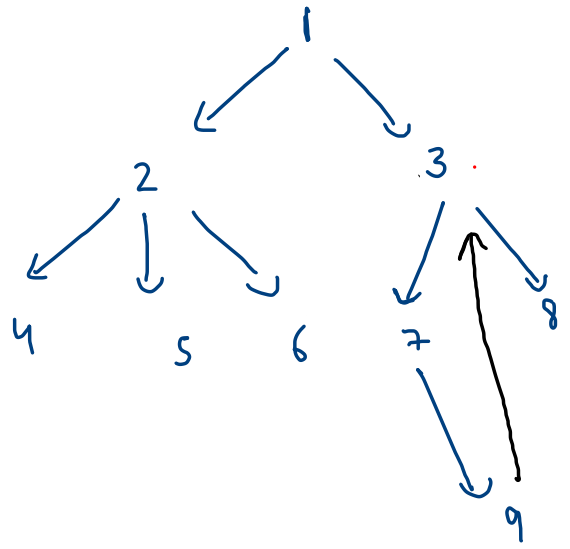
1 – 2

2 – 3

1 – 3



2 parent, cycle



1 - 2

2 - 4

2 - 5

2 - 6

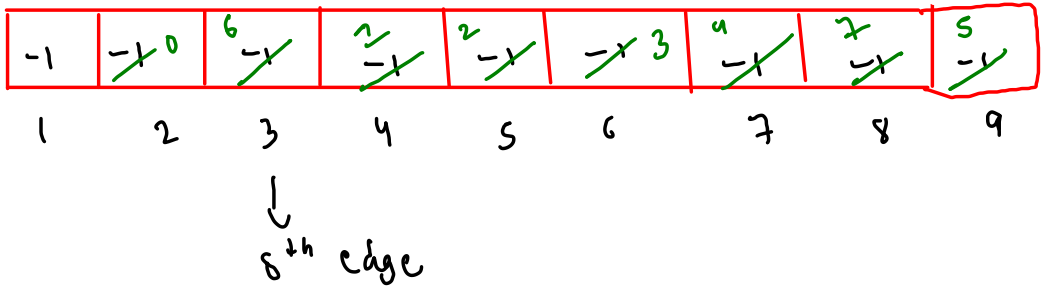
3 - 7

7 - 9

9 - 3 → redundant

3 - 8

1 - 3 → redundant X



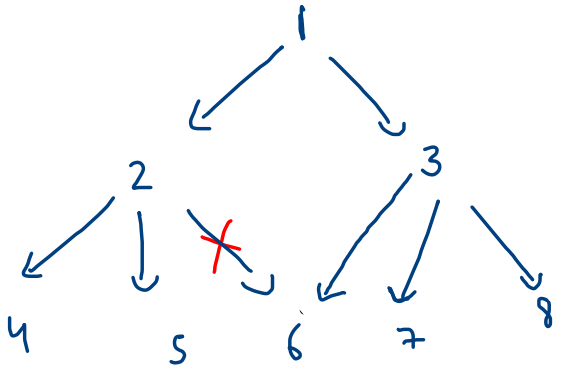
e1 = 6

e2 = 8

```
for(int i=0; i < edges.length;i++) {
    int u = edges[i][0];
    int v = edges[i][1];

    if(indeg[v] == -1) {
        //nothing to do
    }
    else {
        e1 = indeg[v];
        e2 = i;
    }

    indeg[v] = i;
}
```

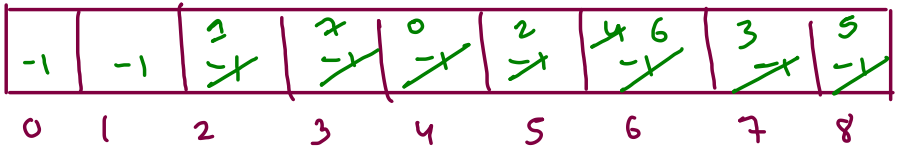
ei = -1

```
int ei = dsu(edges,e2); //restrict e2 and then do dsu


if(e1 == -1 && e2 == -1) {
    //case 2
    return edges[ei];
}

//case 1,case 3
if(ei == -1) {
    return edges[e2];
}
else {
    return edges[e1];
}
}
```



Case 1

2 parent



0 2 - 4 ✓

1 1 - 2 ✓

2 2 - 5 ✓

3 3 - 7 ✓

4 3 - 6 ✓

5 3 - 8 ✓

6 2 - 6 ✓

7 1 - 3 ✓

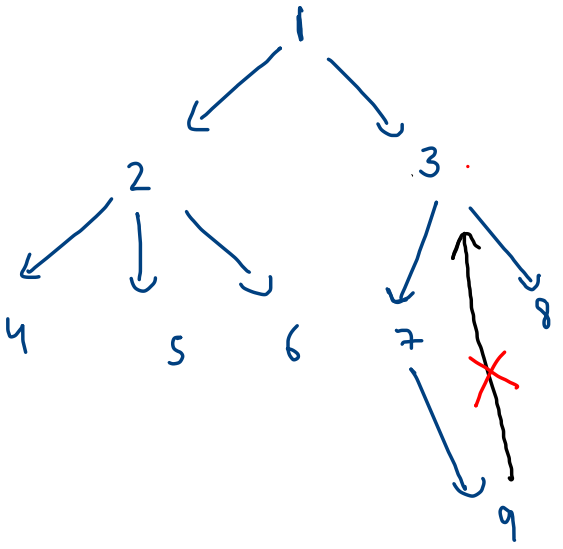
2 - 6

e1 = 4

e2 = 6

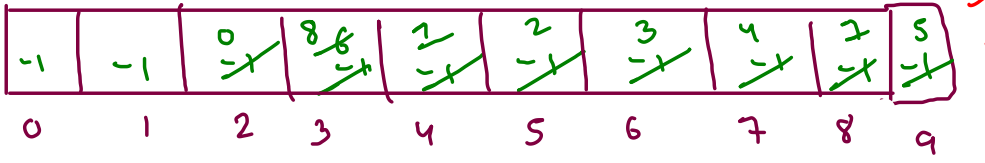
```
    for(int i=0; i < edges.length;i++) {
        int u = edges[i][0];
        int v = edges[i][1];

        if(indeg[v] == -1) {
            //nothing to do
        }
        else {
            e1 = indeg[v];
            e2 = i;
        }

        indeg[v] = i;
    }

    int ei = dsu(edges,e2); //restrict e2 and then do dsu


    if(e1 == -1 && e2 == -1) {
        //case 2
        return edges[ei];
    }

    //case 1,case 3
    if(ei == -1) {
        return edges[e2];
    }
    else {
        return edges[e1];
    }
}
```

ei = 6



2 parent, cycle

case 3



0 1 - 2 ✓

1 2 - 4 ✓

2 2 - 5 ✓

3 2 - 6 ✓

4 3 - 7 ✓

5 7 - 9 ✓

6 9 - 3 ✓

7 3 - 8 ✓

~~8 - 1 - 3~~ ✓

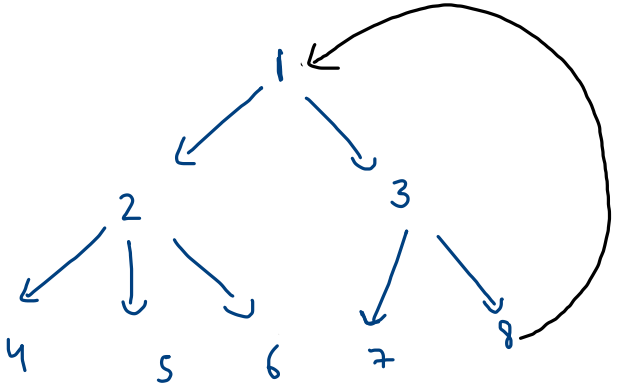
e1 = 6

e2 = 8

```
for(int i=0; i < edges.length;i++) {
    int u = edges[i][0];
    int v = edges[i][1];

    if(indeg[v] == -1) {
        //nothing to do
    }
    else {
        e1 = indeg[v];
        e2 = i;
    }

    indeg[v] = i;
}
```

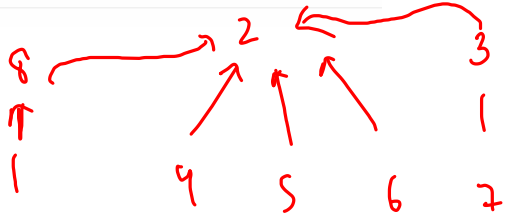
ei = 7

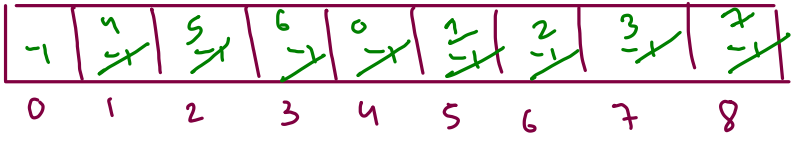
```
int ei = dsu(edges,e2); //restrict e2 and then do dsu


if(e1 == -1 && e2 == -1) {
    //case 2
    return edges[ei];
}

//case 1,case 3
if(ei == -1) {
    return edges[e2];
}
else {
    return edges[e1];
}
}
```



case

cycle





e1 = -1

e2 = -1

0 2-4 ✓

1 2-5 ✓

2 2-6 ✓

3 3-7 ✓

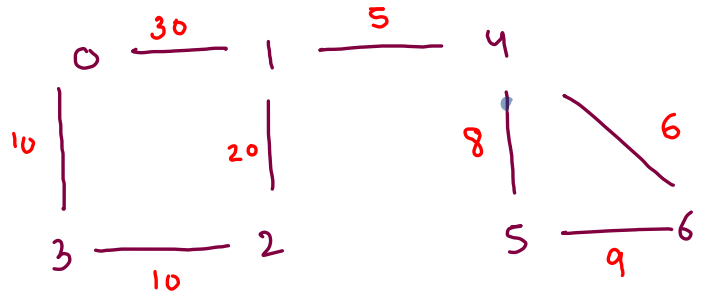
4 8-1 ✓

5 1-2 ✓

6 1-3 ✓

7 3-8 ✓

MST → kruskal algo



Arr [edges]

sort on wt. basis

1 - 4 @ 5 ✓

4 - 6 @ 6 ✓

4 - 5 @ 8 ✓

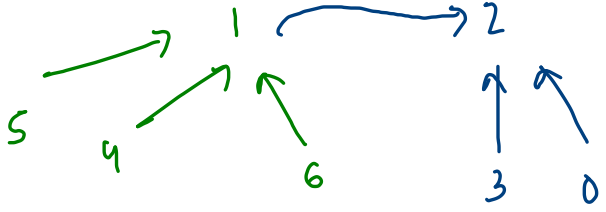
5 - 6 @ 9 ✓ don't use

2 - 3 @ 10 ✓

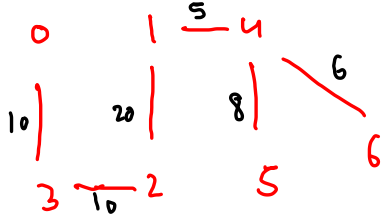
0 - 3 @ 10 ✓

1 - 2 @ 20 ✓

0 - 1 @ 30 ✓ don't use



wt = 5 + 6 + 8 + 10 + 10 + 20



equations satisfy

a == b

b == d

c != a

hold

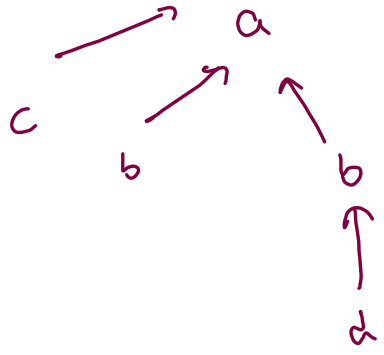
a == d

e != j

hold

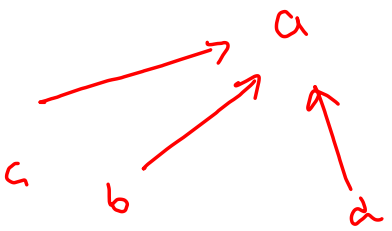
c == a

false



$a == b$ $b == d$ $b != e$ true holds

$a == d$ $e != j$ $c == a$ true holds

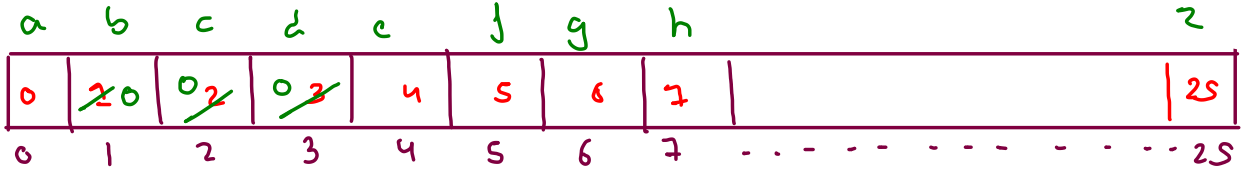


a -> 0

b -> 1

1 -> 25

a == b b == d b != e (true) a == d e != j (true) c == a




```
//perform dsu on == equations
for(int i=0; i < equations.length;i++) {
    String eq = equations[i];

    if(eq.charAt(1) == '=') {
        char op1 = eq.charAt(0);
        char op2 = eq.charAt(3);

        int l1 = find(op1-'a');
        int l2 = find(op2-'a');

        if(l1 != l2) {
            //merge

            if(rank[l1] < rank[l2]) {
                parent[l1] = l2;
            }
            else if(rank[l1] > rank[l2]) {
                parent[l2] = l1;
            }
            else {
                parent[l1] = l2;
                rank[l2]++;
            }
        }
    }
}
```

```
for(int i=0; i < equations.length;i++) {
    String eq = equations[i];

    if(eq.charAt(1) == '!') {
        char op1 = eq.charAt(0);
        char op2 = eq.charAt(3);

         int l1 = find(op1-'a');
         int l2 = find(op2-'a');

         if(l1 == l2) {
             return false;
         }
    }
}
```