October 7, 2025

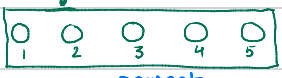
DSU & APPLICATIONS

* DSU is generally used to maintain connectivity (who is connected to whom)
It is a structure that supports :

1. Find(x) : Find which group of elements is 'x' a part of. $\leq O(\log N)$
2. Merge (x, y) : Merge the groups in which x & y are present. $\leq O(\log N)$

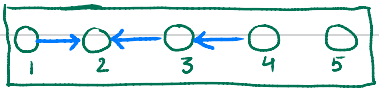
Example :

→ Starting : Individually Not Connected



→ Merge (1,2) / Merge (3,2) / Merge (4,3)

parent



→ Check if 1 & 4 are already connected.

Find(1)
⇒ 2
find(4)
⇒ 2

find(1) == find(4) means these 1 & 4 are connected

Ques 1. CHAIN - Strange Food Chain

https://www.spoj.com/problems/CHAIN/

There are 3 kinds of animals A, B and C. A can eat B, B can eat C, C can eat A. It's interesting, isn't it?

Now we have n animals, numbered from 1 to n. Each of them is one of the 3 kinds of animals: A, B, C.

Today Mary tells us k pieces of information about these n animals. Each piece has one of the two forms below:

- 1 x y: It tells us the kind of x and y are the same.
- 2 x y: It tells us x can eat y.

Some of these k pieces are true, some are false. The piece is false if it satisfies one of the 3 conditions below, otherwise it's true.

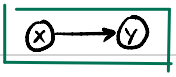
- X or Y in this piece is larger than n.
- This piece tells us X can eat X.
- This piece conflicts to some true piece before.

Whenever there is lots of information, and from that we have to deduce something, that's often a question of weighted DSU.

↓

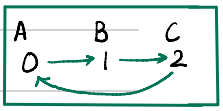
deducing from Knowledge graph, meaning if there are bunch of nodes in a graph and information from one to another is given then we can deduce the information about the path from A to B using DSU. We keep info in edges, making it weighted DSU.

Suppose there is a node 'x' and we say that x can eat y



Let's say we have 3 categories such as A is represented by 0, B by 1, C by 2 and we define the relation :

Category 0 can eat category 1
Category 1 can eat category 2.
Category 2 can eat " 0.



So, if 'x' belonging to some category is eating 'y' belonging to some category, Then what can we deduce from this categorization about values of x & y?

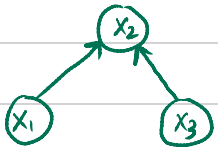
⇒ $(y-x)\%3 = 1$

If y can eat x, then what will be the relation : $(x-y)\%3 = 1$

We will use this logic to solve the given problem.

AlgoZenith

Here, given some relation between variables, we have to deduce the relation between some two unrelated variables which are not directly related.



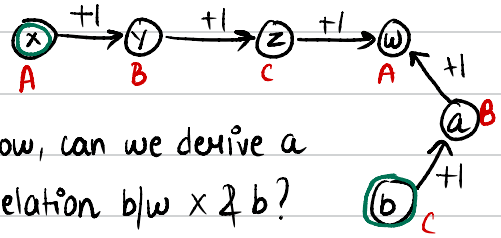
$(x_2 - x_1)\%3 = 1$

$-\ (x_2 - x_3)\%3 = 1$

$(x_3 - x_1)\%3 = 0$

x_3 & x_1 are of the same type.

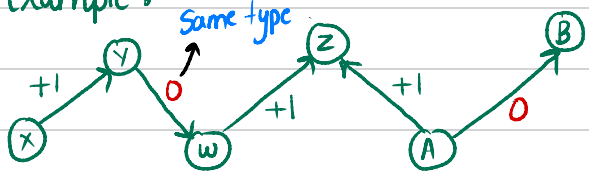
If x can eat y, let's add +1.



Now, can we derive a relation b/w x & b?

Yes. If they are somehow connected then we can derive info.

Example:



Relation b/w x & B?

(Final - Initial) % #val = ___

$(B - x)\%3 = 1$

* DSU

Test case:

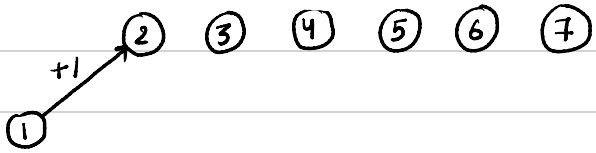
eat ← Node1 / Node2

2	1	2	(1 eats 2)
2	2	3	(2 eats 3)
2	3	4	(3 eats 4)
1	5	4	(5 and 4 are same)
2	6	5	(6 eats 5)
2	5	7	(5 eats 7)

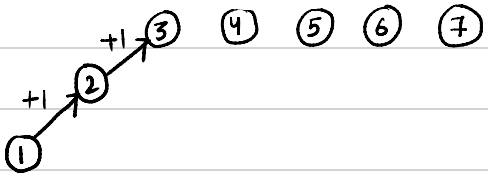
(1 are same)

All available nodes:

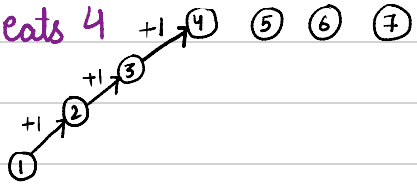
→ 1 eats 2



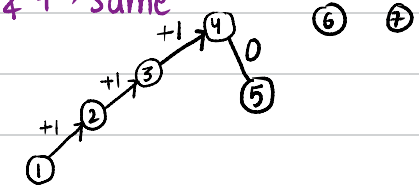
→ 2 eats 3



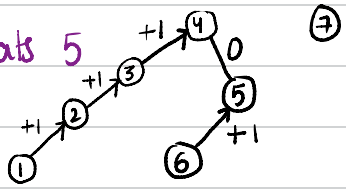
→ 3 eats 4



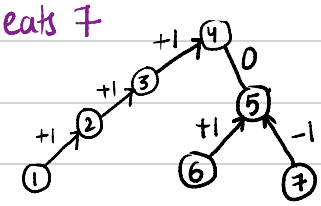
→ 5 & 4 → same



→ 6 eats 5



→ 5 eats 7

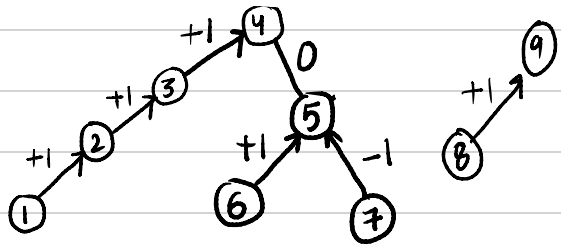


Since 7 can't be directly connected to 5 to maintain the tree structure

More test cases :

2 8 9
1 8 6

→ 8 eats 9



→ 8 & 6 are same

We know that there is 0 between the two but if its added directly then we will lose the tree structure. So,

Relation b/w x_4 (4 Node) & x_6 (6th Node)

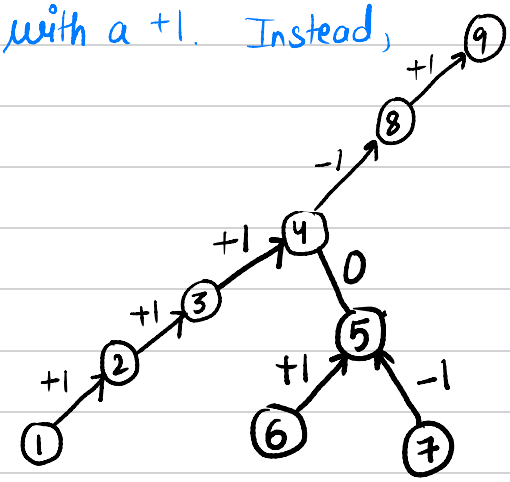
$$x_4 - x_6 = 1 \quad -①$$

Relation b/w x_6 & x_8

$$x_6 - x_8 = 0 \quad -②$$

From ① & ②, $x_4 - x_8 = 1$

Still can't directly attach 8 with 4 with a +1. Instead,



Why are we trying to keep it a tree?

Because it won't be possible to traverse the unique path otherwise.

Rank Compression Technique :

While merging the goal should be to keep the height of the tree minimum

So, always add the shorter height to longer one.

Size Compression :

Smaller one merges to larger one.

Both of them result in tree height of logN.

- Given these nodes at rank 0.

Rank 0 • • • • •

Find how many merges is needed to get:

Rank		Merges
Rank 1	=	1
Rank 2	=	$2 \times 1 + 1 = 3$
Rank 3	=	$2 \times 3 + 1 = 7$
Rank 4	=	15
⋮		

Rank $K = [2^K - 1]$ Bounded by no. of queries (Q)

$$2^K - 1 \leq Q$$

Maximum number of rank,

$$K \leq \log Q$$

```cpp
struct weighted_dsu{
    vector<int> par;
    vector<int> wt;
    vector<int> rank;
    void init(int n){//[1...n]
        par.assign(n+1,0);
        rank.assign(n+1,0);
        wt.assign(n+1,0);
        for(int i=1;i<=n;i++){
            par[i] = i;
            wt[i] =0;
        }
    }
    pair<int,int> find(int x){
        if(par[x]==x){
            return {x,0};
        }
        auto temp = find(par[x]);
        return {temp.F, temp.S + wt[x]};
    }
    int get_relation(int x,int y){
        auto xpath = find(x);
        auto ypath = find(y);
        if(xpath.F!=ypath.F){
            // NOT RELATED.
        }else{
            return xpath.S - ypath.S;
        }
    }
```

```cpp
    bool add_info(int x,int y, int c){
        auto xpath = find(x);
        auto ypath = find(y);
        if(xpath.F==ypath.F)return 0;
        if(rank[xpath.F]>=rank[ypath.F]){
            par[ypath.F] = xpath.F;
            wt[ypath.F] = xpath.S - c - ypath.S;
            if(rank[xpath.F]==rank[ypath.F]) rank[xpath.F]++;
        }else{
            par[xpath.F] = ypath.F;
            wt[xpath.F] =  c + ypath.S - xpath.S ;
        }
        return 1;
    }
};
```

```cpp
void solve(){
    int n,q;
    cin>>n>>q;
    weighted_dsu chain;
    chain.init(n);
    int falseinfo = 0;
    while(q--){
        int type,a,b;
        cin>>type>>a>>b;
        int c = 0;
        if(type==2)c=1;
        if(a>n||b>n||a<=0||b<=0){
            falseinfo++;
            continue;
        }
        if(chain.add_info(a,b,c)){

        }else{
            int curval = chain.get_info(a,b);
            if((curval-c)%3!=0)falseinfo++;
        }
    }
    cout<<falseinfo<<endl;
}
```

* Path compression