DSU

Disjoint Set Union（併査集）

情境-蓋路問題

- 有一個國家有 n 座彼此不相連的城市，由於交通很不方便，所以這個國家打算開始一個企劃，企劃負責人會下 k 個指示，每個指示有兩種可能：
  - 詢問兩座城市 q, p 間是否有路徑相連，回答 YES、NO
  - 要求在兩座城市 q, p 間建一條公路
- 其中 n < 1e5、k < 3e5。
- 輸入第一行有 n、k 兩個數字，接下來每行有一個字串和兩個數字，代表操作類別和參數。
- 範測：

Input:

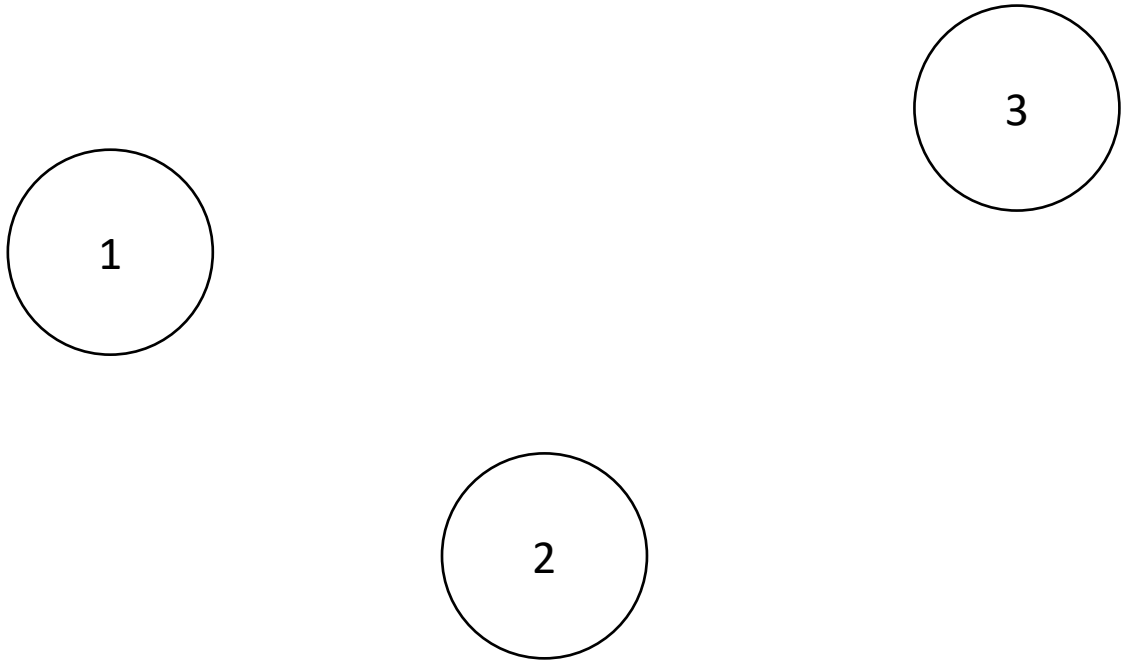
```
3 4
ASK 1 3
CONNECT 1 2
CONNECT 2 3
ASK 1 3
```

Output:

```
NO
YES
```

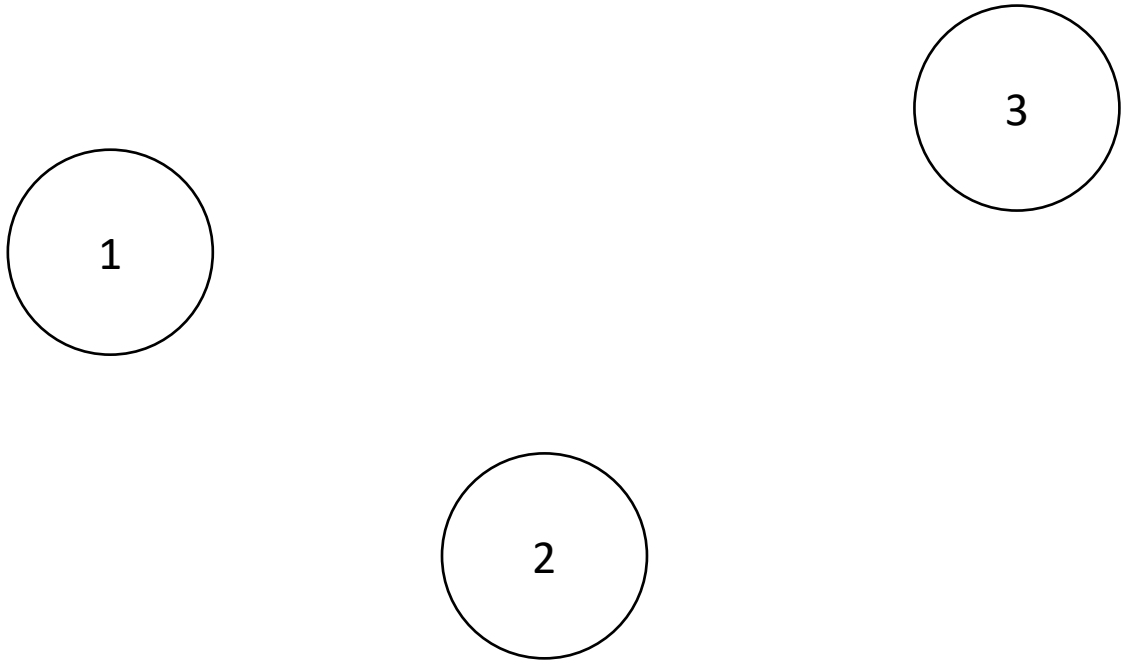
情境-蓋路問題-範測解釋

- 一開始有三個點（三座城市）



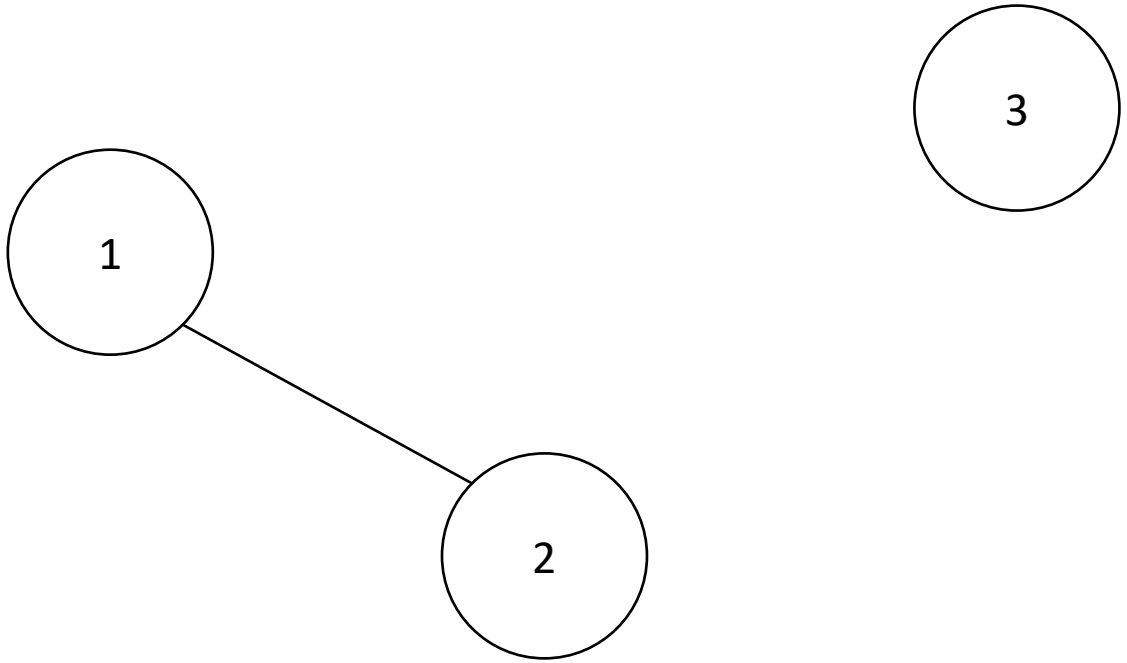
情境-蓋路問題-範測解釋

- 企劃負責人詢問是否 1、3 兩座城市聯通
  - 由於找不到任何一條路徑使得你可從 1 走到 3，因此回答 NO



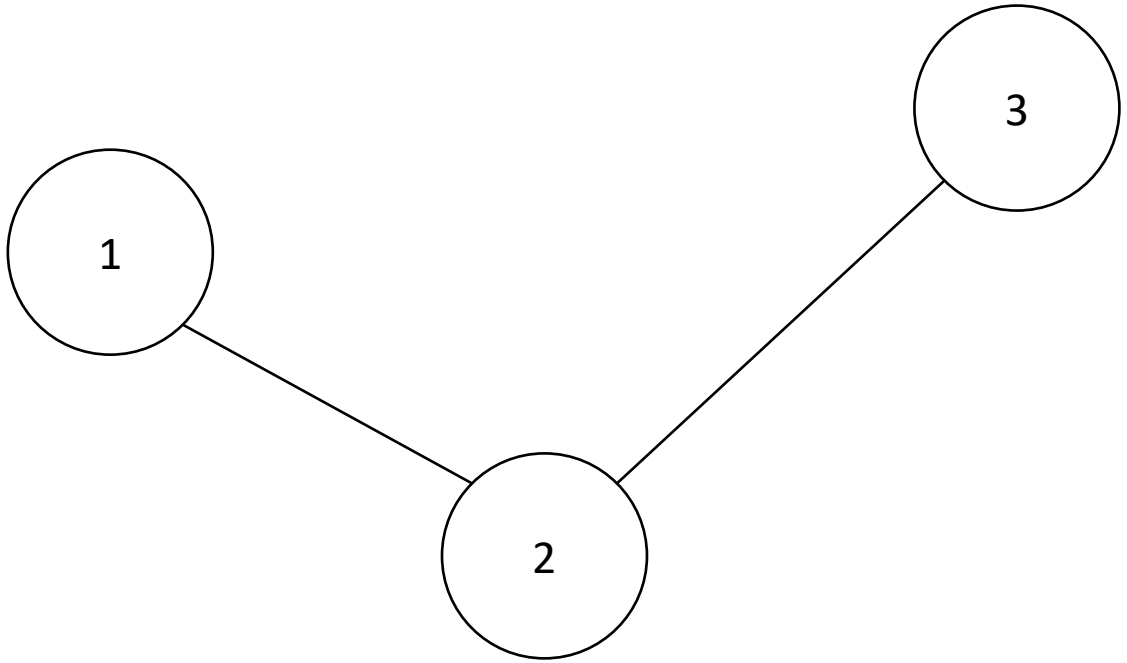
情境-蓋路問題-範測解釋

- 企劃負責人要求 1、2 之間蓋路



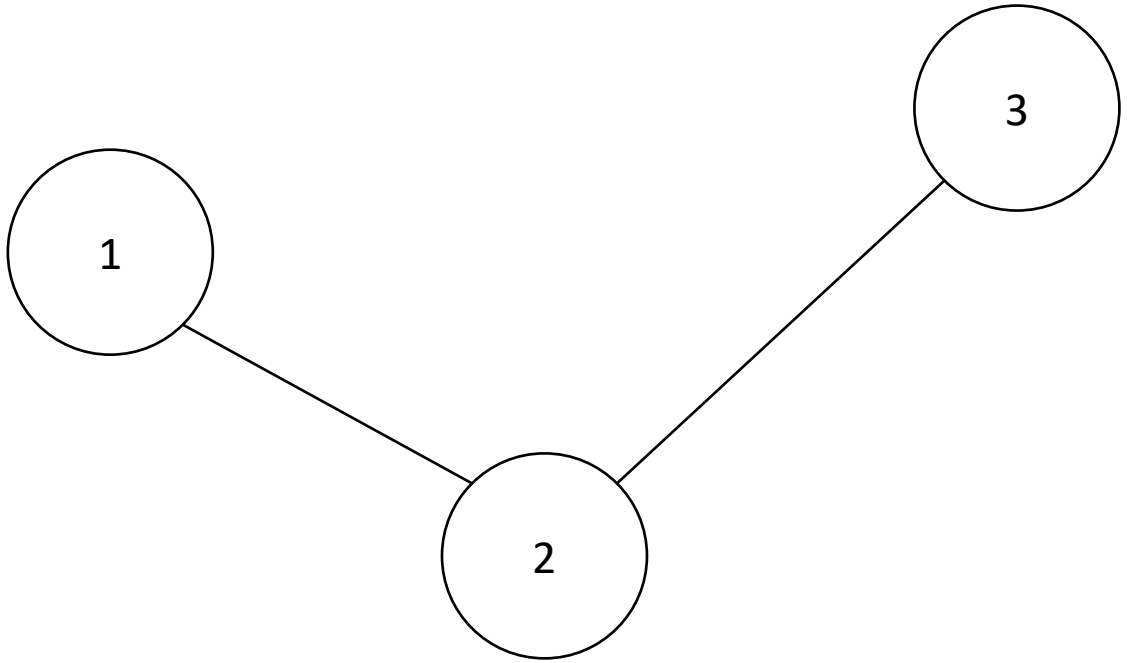
情境-蓋路問題-範測解釋

- 企劃負責人要求 2、3 之間蓋路



情境-蓋路問題-範測解釋

- 企劃負責人詢問是否 1、3 兩座城市聯通
  - 你回答 YES，因為你可以從 1 走到 2，再從 2 走到 3



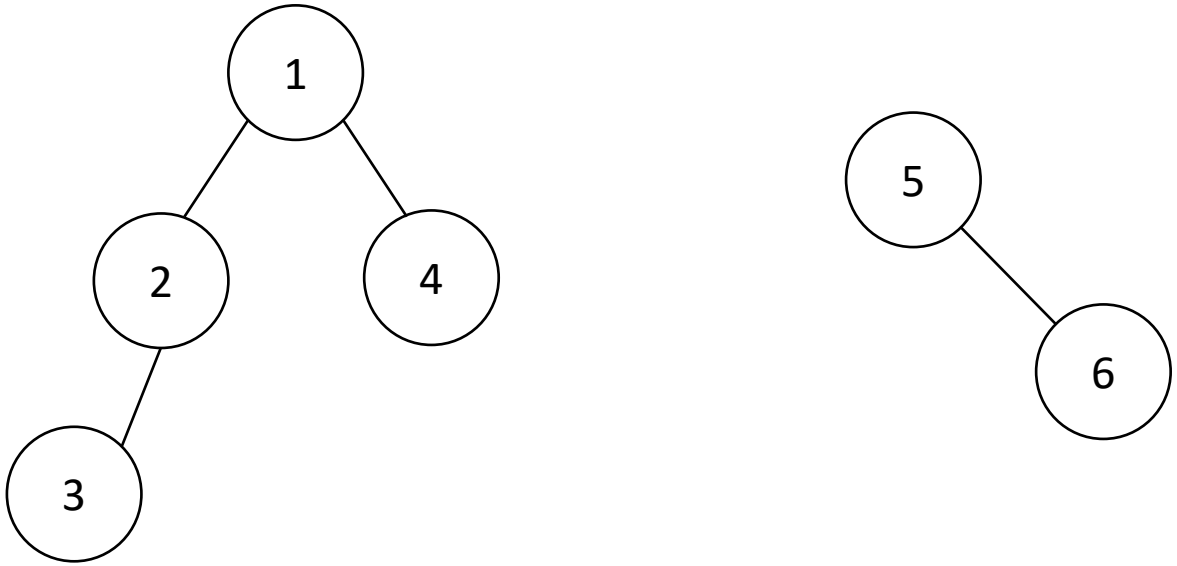
簡化模型

- 你有 N 個元素，一開始他們彼此都在不同的集合裡面，你每次有兩種操作：
  - 合併某兩個元素所在的集合
  - 查詢某兩個元素是否在同一個集合

- 你需要維護一個資料結構可以快速進行這兩個操作
  - DSU

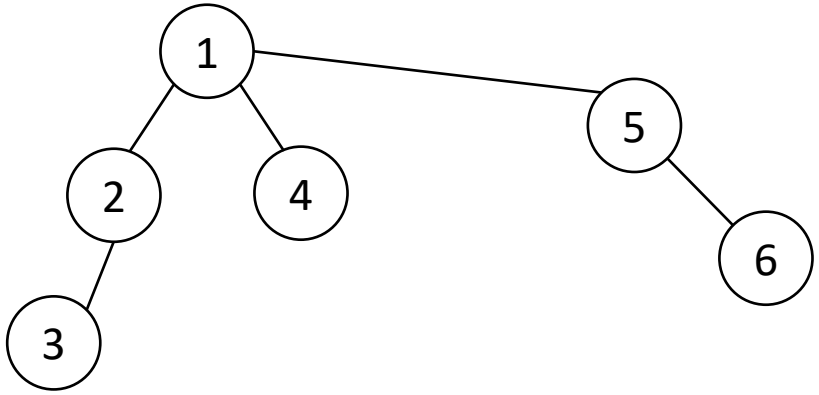
簡化模型-DSU

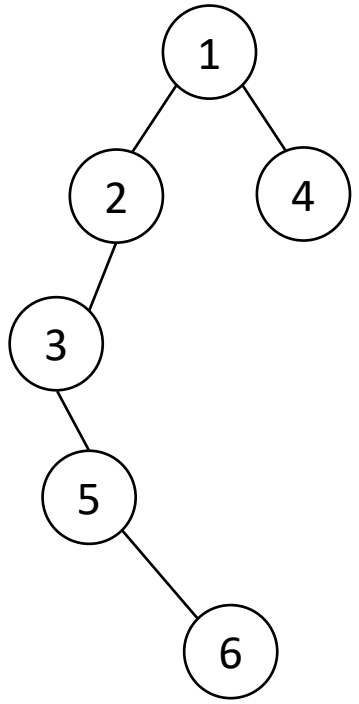
- 我們為每一個集合建一棵樹，然後選擇其中一個點當根節點
- 查詢兩個點是否在同一個集合 -> 檢查兩個點所在的樹的根節點是否相同
- 1、3 是否在同一個集合？4、5 是否在同一個集合？為什麼？



簡化模型-DSU

- 我們為每一個集合建一棵樹，然後選擇其中一個點當根節點
- 合併兩個點所在的集合 -> 合併兩棵樹
- 如果我要把 3、6 合併，左右哪個比較好？





實作-DSU-第一個版本

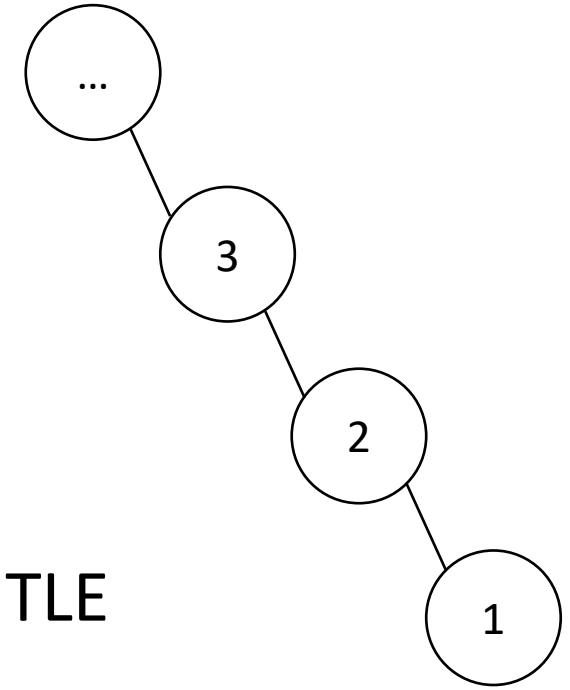
```cpp
#include<bits/stdc++.h>

using namespace std;

vector<int> arr;

void find_root(int node){
    if(arr[node] == -1)
        return node; // node is root
    return find_root(arr[node]);
    // we find the root of the parent of this node,
    // the depth(distance to root) will be decrease by 1
}

void merge(int p, int q){
    p = find_root(p);
    q = find_root(q);
    if(p == q)
        return; // we shouldn't merge a set with itself
    arr[q] = p;
    // We connect the root of q to the root of p
    // i.e. find_root(p) will be the root of find_root(q)
}

int main(){
    int n, k; cin >> n >> k;
    arr.resize(n); // parent of each node, the parent of root is -1
    for(int i = 0; i < n; i++)
        arr[i] = -1; // initially, each node is the root of the tree containing only it
    while(k--){
        string opr; cin >> opr;
        int q, p; cin >> q >> p;
        if(opr == "ASK"){
            if(find_root(p) == find_root(q)) cout << "YES" << endl;
            else cout << "NO" << endl;
        } else if(opr == "CONNECT") {
            merge(p, q);
        }
    }
}
```

```
┌─[linlee@MacBook] - [~/tmp] - [五 12 17, 17:39]
└─[$]> run dsu.cpp
3 4
ASK 1 3
CONNECT 1 2
CONNECT 2 3
ASK 1 3

NO
YES
[程式結束，耗時 56 毫秒]%
```

實作-DSU-第一個版本-Worst Case 複雜度

- 對於這樣子的測資，他的時間複雜度會是多少？

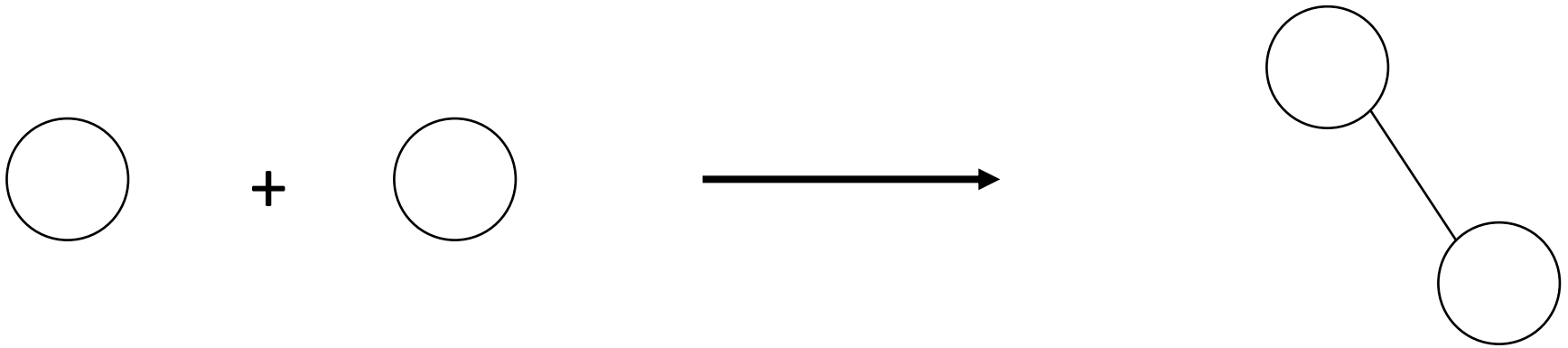
```
Input:

10000 9999
CONNECT 2 1
CONNECT 3 1
CONNECT 4 1
CONNECT 5 1
……
```



- 每次 find_root(1) 的複雜度是 O(k)
- 總共 k 次，因此複雜度是 O(k^2)，會 TLE
  - 要盡力讓樹不要太高 -> 讓樹盡力平衡

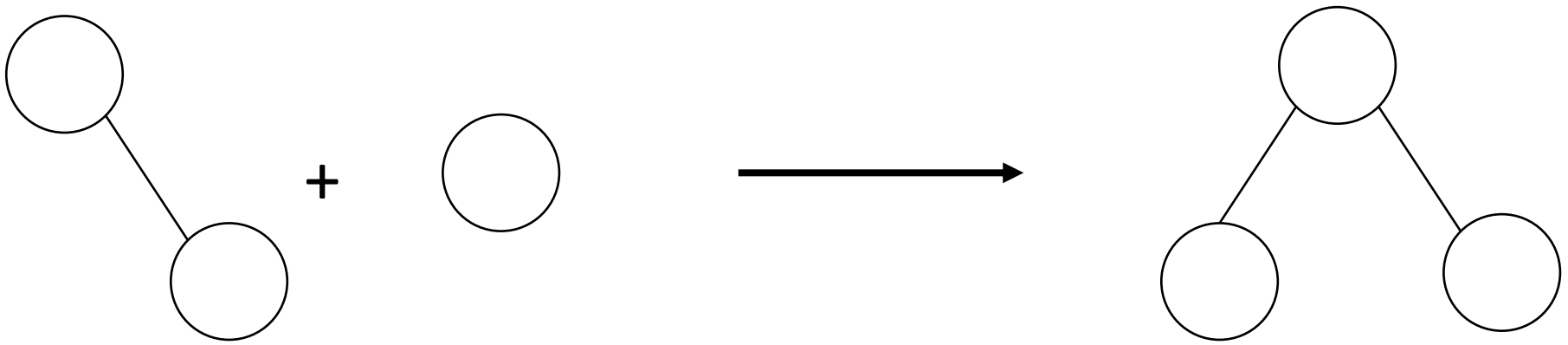
簡化模型-DSU-啟發式優化

- 倘若我們每次將兩個樹合併的時候，都將比較小的樹放到比較大的樹下面
- 會發生什麼？
  - 兩塊 size = 1 的 set 合併起來，高度會變成 2



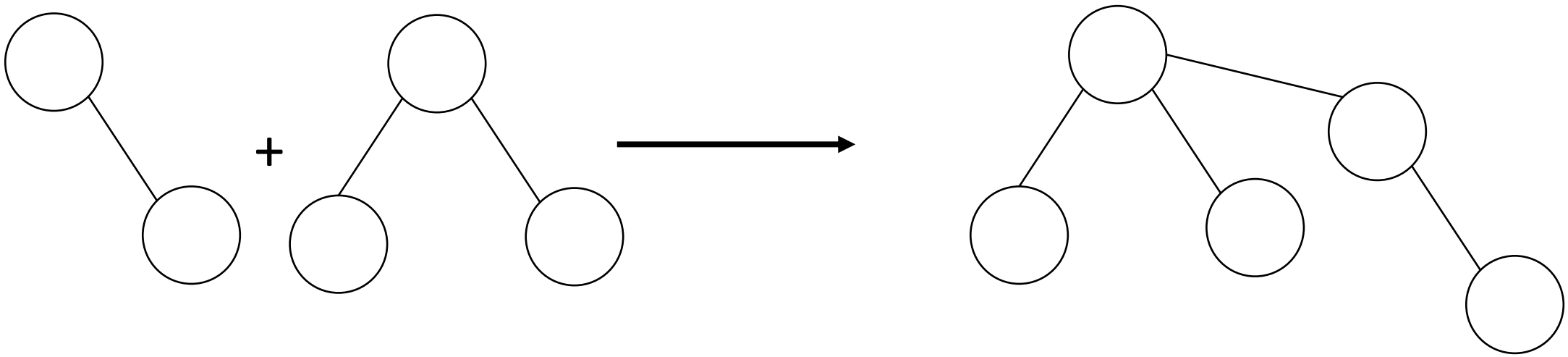
簡化模型-DSU-啟發式優化

- 倘若我們每次將兩個樹合併的時候，都將比較小的樹放到比較大的樹下面
- 會發生什麼？
  - 兩個高度不相同的，合併起來不會改變高度



簡化模型-DSU-啟發式優化

- 倘若我們每次將兩個樹合併的時候，都將比較小的樹放到比較大的樹下面
- 會發生什麼？
  - 兩個高度相同的，合併起來會讓高度加一



簡化模型-DSU-啟發式優化-複雜度分析

- 倘若我們每次將兩個樹合併的時候，都將比較小的樹放到比較大的樹下面
- 會發生什麼？
  - 想要構造出一個高度是 h 的，必須由兩個高度是 h – 1 的樹組成
  - 一個高度是 2 的樹，至少是 2 個點；
  - 一個高度是 3 個樹，至少是 2 * 2 = 4 個點；
  - 一個高度是 4 的樹，至少是 4 * 2 = 8 個點。
  - 一個高度是 h 的樹，至少是 2 ^ (h – 1) 個點；
  - 一個點數是 k 的樹，高度至多 O(log k)。

- 因此只要我們按照這種方式來合併，我們就可以保證每次操作複雜度至少在 O(log N) 以下（你不可能有一棵樹超過 N 個點）。

實作-DSU-第二個版本-啟發式合併

```cpp
#include<bits/stdc++.h>

using namespace std;

vector<int> arr, sz;

int find_root(int node){
    if(arr[node] == -1)
        return node; // node is root
    return find_root(arr[node]);
    // we find the root of the parent of this node,
    // the depth(distance to root) will be decrease by 1
}

void merge(int p, int q){
    p = find_root(p);
    q = find_root(q);
    if(p == q)
        return; // we shouldn't merge a set with itself
    if(sz[p] < sz[q]) swap(p, q);
    arr[q] = p;
    sz[p] += sz[q];
    // We connect the root of q to the root of p
    // i.e. find_root(p) will be the root of find_root(q)
}

int main(){
    int n, k; cin >> n >> k;
    sz.resize(n); // size of the `subtree` of each node
    arr.resize(n); // parent of each node, the parent of root is -1
    for(int i = 0; i < n; i++) {
        arr[i] = -1; // initially, each node is the root of the tree containing only it
        sz[i] = 1;   // initially, size of each tree is 1
    }
    while(k--){
        string opr; cin >> opr;
        int q, p; cin >> q >> p;
        if(opr == "ASK"){
            if(find_root(p) == find_root(q)) cout << "YES" << endl;
            else cout << "NO" << endl;
        } else if(opr == "CONNECT") {
            merge(p, q);
        }
    }
}
```

```
┌─[linlee@MacBook] - [~/tmp] - [五 12 17, 18:01]
└─[$]> run dsu.cpp
3 4
ASK 1 3
CONNECT 1 2
CONNECT 2 3
ASK 1 3
NO
YES
[程式結束，耗時 55 毫秒]%
```

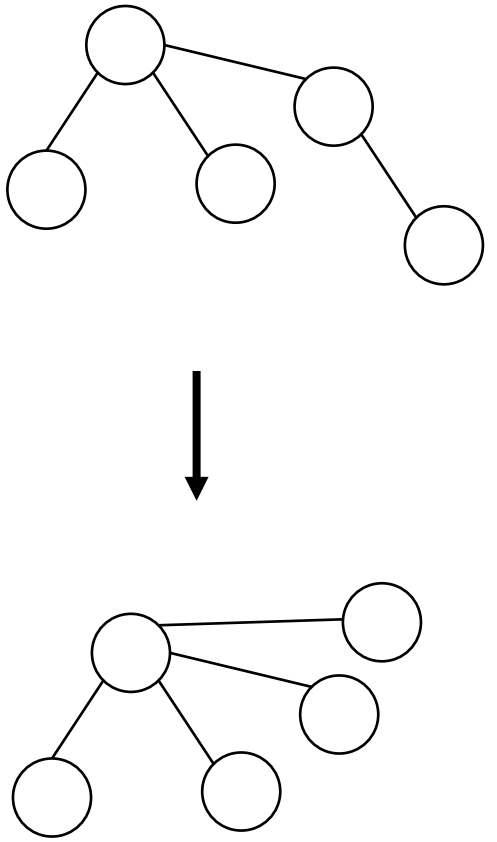
實作-DSU-第二個版本-複雜度分析

- 在 N 個點的情況
  - 查詢的複雜度最高是 O(logN)
  - 合併的複雜度 = 2次查詢 + O(1) = O(logN)

- 這個複雜度是好的，且絕大多數情況的複雜度都不超過如此。

實作-DSU-第三個版本-路徑壓縮

```cpp
#include<bits/stdc++.h>

using namespace std;

vector<int> arr, sz;

int find_root(int node){
    if(arr[node] == -1)
        return node; // node is root
    int root = find_root(arr[node]);
    arr[node] = root; // We make node directly connected to root
    return root;
    // we find the root of the parent of this node,
    // the depth(distance to root) will be decrease by 1
}

void merge(int p, int q){
    p = find_root(p);
    q = find_root(q);
    if(p == q)
        return; // we shouldn't merge a set with itself
    if(sz[p] < sz[q]) swap(p, q);
    arr[q] = p;
    sz[p] += sz[q];
    // We connect the root of q to the root of p
    // i.e. find_root(p) will be the root of find_root(q)
}

int main(){
    int n, k; cin >> n >> k;
    sz.resize(n); // size of the `subtree` of each node
    arr.resize(n); // parent of each node, the parent of root is -1
    for(int i = 0; i < n; i++) {
        arr[i] = -1; // initially, each node is the root of the tree containing only it
        sz[i] = 1;   // initially, size of each tree is 1
    }
    while(k--){
        string opr; cin >> opr;
        int q, p; cin >> q >> p;
        if(opr == "ASK"){
            if(find_root(p) == find_root(q)) cout << "YES" << endl;
            else cout << "NO" << endl;
        } else if(opr == "CONNECT") {
            merge(p, q);
        }
    }
}
```



模板题：https://loj.ac/p/109

维护一个 n 点的无向图，支持：

- 加入一条连接 u 和 v 的无向边
- 查询 u 和 v 的连通性

由于本题数据较大，因此输出的时候采用特殊的输出方式：用 0 或 1 代表每个询问的答案，将每个询问的答案依次从左到右排列，把得到的串视为一个二进制数，输出这个二进制数 $\bmod 998244353$ 的值。

请务必使用快读。

输入格式

第一行包含两个整数 n, m，表示点的个数和操作的数目。

接下来 m 行每行包括三个整数 op, u, v。

- 如果 $\mathrm{op} = 0$，则表示加入一条连接 u 和 v 的无向边；
- 如果 $\mathrm{op} = 1$，则表示查询 u 和 v 的连通性。

输出格式

一行包括一个整数表示答案。

模板題：https://loj.ac/p/109

```cpp
#include<bits/stdc++.h>

using namespace std;

vector<int> arr, sz;

const int MOD = 998244353;
int find_root(int node){
    if(arr[node] == -1)
        return node;
    int root = find_root(arr[node]);
    arr[node] = root;
    return root;
}

void merge(int p, int q){
    p = find_root(p);
    q = find_root(q);
    if(p == q)
        return;
    if(sz[p] < sz[q]) swap(p, q);
    arr[q] = p;
    sz[p] += sz[q];
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    int n, m; cin >> n >> m;
    sz.resize(n); arr.resize(n);
    for(int i = 0; i < n; i++) {
        arr[i] = -1; sz[i] = 1;
    }
    int ans = 0;
    while(m--){
        int op, u, v; cin >> op >> u >> v;
        if(op == 0) merge(u, v);
        else ans = (ans * 2 + (find_root(u) == find_root(v))) % MOD;
    }
    cout << ans << endl;
}
```