CHUYÊN ĐỀ

CẤU TRÚC DỮ LIỆU NÂNG CAO

DISJOINT – SET – UNION

(DSU)

PHẦN I: MỞ ĐẦU

Hệ thống các tập không giao nhau (disjoint-set-union – DSU) là một cách tổ chức dữ liệu đủ đơn giản nhưng rất hiệu quả để giải quyết nhiều loại bài toán khác nhau. Trong các năm gần đây các đề thi học sinh quốc gia, thi duyên hải … bài tập về đồ thị luôn chiếm một tỉ lệ lớn. Các bài tập ngày càng nâng cao về độ khó và đòi hỏi sử dụng tốt một số cấu trúc dữ liệu để giảm độ phức tạp. DSU là một cấu trúc rất hữu dụng và là nền tảng cho một số thuật toán như thuật toán Kruskal. Sau khi tìm hiểu tôi nhận thấy DSU là một cấu trúc dữ liệu hay và quan trọng. Do đó tôi quyết định chọn chuyên đề cấu trúc dữ liệu DSU.

Chuyên đề tổng hợp kiến thức về DSU đặc biệt tôi cũng đưa ra luôn hai kỹ thuật cải tiến cho phép thực hiện một số phép xử lý với độ phức tạp xấp xỉ O(1).

PHẦN II: NỘI DUNG

I. LÝ THUYẾT

1. Định nghĩa

Trong khoa học máy tính, Cấu trúc dữ liệu Disjoint – Set - Union (hợp của các tập hợp không giao nhau) là một cấu trúc dữ liệu quản lý các phần tử đã được phân chia thành các tập con không giao nhau. DSU cung cấp các phép toán: thêm các tập con mới, kết hợp các tập con lại với nhau, xác định các phần tử có trong cùng một tập hợp hay không.

DSU còn có các tên gọi khác như Disjoint-Set Data Structure, Union-Find.

2. Các phép toán cơ bản

Ban đầu mỗi phần tử của dữ liệu thuộc một tập riêng. Các phép xử lý cơ sở trên cấu trúc là:

- **Make_set(v):** Tạo ra một tập hợp mới để chứa phần tử mới v.
- **Union(a, b):** Hợp nhất hai tập hợp (tập hợp chứa phần tử a và tập hợp chứa phần tử b).
- **Find_set(v):** Trả về phần tử đại diện của tập hợp mà chứa phần tử v. Phần tử đại diện này được lựa chọn cho mỗi tập hợp và phần tử này có thể thay đổi và được chọn lại sau phép toán Union_set. Phần tử đại diện này được sử dụng để kiểm tra hai phần tử có cùng một tập hợp hay không.
- **Xét bài toán**

  Cho đồ thị gồm n đỉnh được đánh số : $0,1,\dots,n-1$ và không có cạnh nào.Lần lượt thêm m cạnh vô hướng vào đồ thị. Cho biết số miền liên thông sau mỗi bước thêm cạnh

Thuật toán

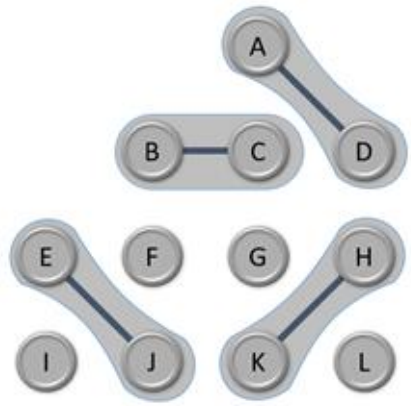
- Khởi tạo: n tập hợp, mỗi tập gồm 1 đỉnh, số miền liên thông $cc = n =$ số tập
- Thêm cạnh (u, v):

  Trường hợp 1: Nếu u, v thuộc cùng 1 tập → bỏ qua

Trường hợp 2: Nếu u, v thuộc hai tập khác nhau → hợp nhất tập chứa u và tập chứa $v; --cc$.

Giả mã

```
for (∀u ∈ V) MakeSet(u);
cc = n;
for (∀(u, v) ∈ E thêm vào)
{
   r = FindSet(u); s = FindSet(v);
   if (r ≠ s)
   {
      Union(r, s);
      --cc;
   }
   Output ← cc;
}
```

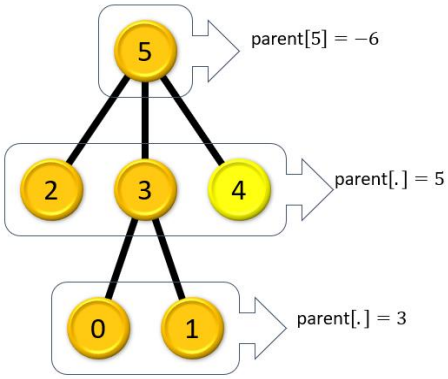


DSU được cài đặt hiệu quả với cấu trúc cây

Ban đầu chúng ta có rừng các tập rời nhau

- Đỉnh: $0 \ldots n-1$
- Tập hợp đỉnh ≡ cây
    - Mỗi nút chứa một đỉnh
    - Nút ≡ đỉnh chứa trong nút
- Biểu diễn cây: sử dụng mảng $parent[0 \ldots n-1]$ trong đó
    - v không phải gốc: $parent[v] =$ nút cha của v
    - v là gốc: $parent[v] = -$Số nút trong cây gốc v
    - Khởi tạo: $parent[0 \ldots n-1] = -1$

Ví dụ



$\{0,1,2,3,4,5\}$

```
//Dựng cây ứng với tập {u}
void MakeSet(int u)
{
   parent[u] = -1;
}
void solve()
{
for (∀u ∈ V) MakeSet(u);
cc = n;
for (∀(u, v) ∈ E thêm vào)
{
   r = FindSet(u); s = FindSet(v);
   if (r ≠ s)
   {
      Union(r, s);
      --cc;
   }
   Output ← cc;
}
}
```

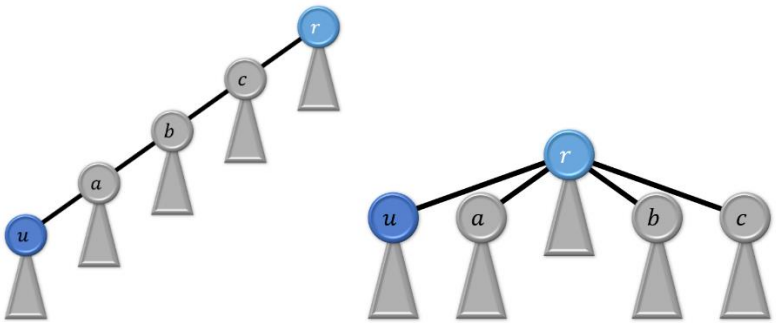
a) Viết hàm FindSet

Sử dụng phương pháp nén đường

FindSet(u): Tìm gốc cây chứa u

Cách làm: Đi từ u lên gốc, kết quả trả về gốc. Dọc đường đi, đặt các nút đi qua làm con trực tiếp của gốc

```
int FindSet(int u)
{
   return parent[u] < 0 ?  u : parent[u] = FindSet(parent[u]);
}
```



ĐPT: phương pháp nén đường mang lại hiệu quả trung bình O(logn) cho mỗi truy vấn.

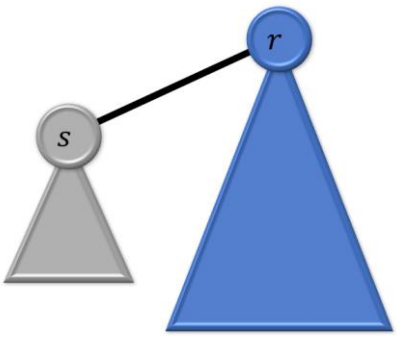
b) Viết hàm Union

Unio(r,s): Hợp nhất cây gốc r và cây gốc s

Sử dụng phương pháp Hợp theo hạng(Union – by – Rank)

Cách làm: lấy gốc cây có lượng đỉnh bé hơn làm con của gốc cây có số lượng đỉnh to hơn

```
void Union(int r, int s)
{
   if (parent[s] < parent[r])
      swap(r, s);
   parent [r] += parent[s];
   parent[s] = r;
}
```



Thời gian thực hiện hai phép xử lý trên:

- Việc viết hai hàm FindSet và Union như trên sẽ làm cho độ phức tạp xử lý mỗi truy vấn trở thành một hằng. Chứng minh điều này khá phức tạp và đã nêu trong nhiều tài liệu khác nhau, ví dụ Tarjan năm 1975, Kurt Mehlhorn và Peter Sanders năm 2008.
- Người ta đã chứng minh được rằng độ phức tạp của mỗi truy vấn khi viết các phép xử lý trên là O(α(n)) trong đó α(n) là hàm nghịch đảo Akkerman có độ tăng rất chậm, đến mức trong phạm vi $n<=10^{600}$ α(n) có giá trị không quá 4. Vì vậy có thể nói hệ thống các tập không giao nhau hoạt động với độ phức tạp gần như một hằng.

II. Bài tập vận dụng

Bài 1. https://vn.spoj.com/problems/IOIBIN/

Bin – Các thùng nước

Có N thùng nước được đánh số từ 1 đến N, giữa 2 thùng bất kỳ đều có một ống nối có một van có thể khóa hoặc mở. Ở trạng thái ban đầu tất cả các van đều đóng.

Bạn được cho một số yêu cầu, trong đó mỗi yêu cầu có 2 dạng:

Dạng X Y 1 có ý nghĩa là bạn cần mở van nối giữa 2 thùng X và Y.

Dạng X Y 2 có ý nghĩa là bạn cần cho biết với trạng thái các van đang mở / khóa như hiện tại thì 2 thùng X và Y có thuộc cùng một nhóm bình thông nhau hay không? Hai thùng được coi là thuộc cùng một nhóm bình thông nhau nếu nước từ bình này có thể chảy đến được bình kia qua một số ống có van đang mở.

Input: BIN.INP. Dòng đầu tiên ghi một số nguyên dương P là số yêu cầu. Trong P dòng tiếp theo, mỗi dòng ghi ba số nguyên dương X, Y, Z với ý nghĩa có yêu cầu loại Z với 2 thùng X và Y.

Output: BIN.OUT. Với mỗi yêu cầu dạng X Y 2 (với Z = 2) bạn cần ghi ra số 0 hoặc 1 trên 1 dòng tùy thuộc 2 thùng X và Y không thuộc hoặc thuộc cùng một nhóm bình. Giới hạn: 1 ≤ N ≤ 10000, 1 ≤ P ≤ 50000. Thời gian: 1 s/test. Bộ nhớ: 1 MB

BIN.INP	BIN.OUT
9	0
1 2 2	0
1 2 1	1
3 7 2	0
2 3 1	1
1 3 2	0
2 4 2	
1 4 1	
3 4 2	
1 7 2	

Ví dụ:

Gợi ý. Đây là bài toán tìm vùng liên thông. Chương trình sau dùng phương pháp hợp nhất cây.

```
#include <bits/stdc++.h>
#define fop(i,a,b) for(int i = a; i <= b; i++)
#define fom(i,a,b) for(int i = a; i >= b; i--)
#define inp() freopen("BIN.inp","r",stdin)
#define out() freopen("BIN.out","w",stdout)
using namespace std;
typedef long long ll;
int P,X,Y,Z,r1,r2;
int parent[100005];
int find_root(int u)
{
       return parent[u] < 0 ? u : parent[u] =
find_root(parent[u]);
}
void union_(int r, int s)
{
    if (parent[s] < parent[r])
        swap(r, s);
    parent[r] += parent[s];
    parent[s] = r;
 }
int main()
{
    inp();
    out();
    cin >> P;
    memset(parent,-1,sizeof(parent));
    while(P)
    {
        cin >> X >> Y >> Z;
        if(Z==1)
        {
            r1 = find_root(X);
            r2 = find_root(Y);
```

```
            if(r1 != r2) union_(r1,r2);
        }
        else
        {
            r1 = find_root(X);
            r2 = find_root(Y);
            if(r1==r2) cout << 1<<'\n';
            else cout << 0 << '\n';
        }
        P--;
    }
    return 0;
}
```

Cảm nhận: Bài này có thể làm bằng nhiều cách như duyệt DFS để liệt kê các TPLT nhưng nếu áp dụng thuật toán hợp nhất cây mới hiệu quả và giải quyết bài toán một cách triệt để.

Bài 2 - Thay thế ký tự

Cho hai xâu ký tự *s* và *t* đều có *n* ký tự là các chữ cái tiếng Anh in thường. Người ta muốn thay thế các ký tự trong hai xâu để chúng giống hệt nhau. Với một phép biến đổi, ta có thể thay đổi một số chữ cái trên 2 xâu. Bạn hãy tính toán số phép biến đổi tối thiểu để hoàn thành việc này.

Chính xác là, Bạn sử dụng các phép biến đổi dạng *R*(*c1, c2*) (trong đó *c1* và *c2* là các chữ cái). Bạn có thể thực hiện một phép biến đổi nào đó với số lần tùy ý để biến đổi một chữ cái *c1* thành một chữ cái *c2* và ngược lại trên cả hai hai xâu *s* và *t*. Bạn cần tìm số phép biến đổi tối thiểu để cho *s* và *t* giống hệt nhau. Thêm nữa, bạn cần in ra chi tiết về các phép biến đổi đó. Xem ví dụ để rõ hơn.

Dữ liệu

- Dòng đầu chứa số nguyên *n* ($1 \leq n \leq 10^5$) là độ dài các xâu ký tự.
- Dòng thứ hai chứa *n* chữ cái tiếng Anh in thường, mô tả xâu *s*.
- Dòng thứ ba chứa *n* chữ cái tiếng Anh in thường, mô tả xâu *t*.

Kết quả

- Dòng đầu in ra số nguyên *k* là tổng số phép biến đổi tối thiểu cần thực hiện

- Trong *k* dòng tiếp theo, mỗi dòng in ra một cặp ký tự *c1, c2* cách nhau một dấu cách để mô tả về một phép biến đổi. Các cặp này có thể in theo trật tự bất kỳ. Chú ý, các phép biến đổi có thể không phải duy nhất.

Ví dụ

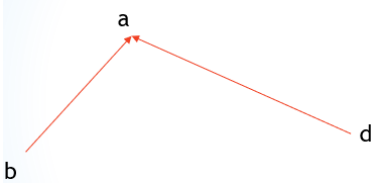
Replace.inp	Replace.out	Replace.inp	Replace.out
3 abb dad	2 a d b a	8 drpepper cocacola	7 l e e d d c c p p o o r r a

Thuật toán:

- Với mỗi cặp kí tự của 2 xâu ta kiểm tra xem có cách biến đổi nào đưa về giống nhau không tức là kiểm tra xem có đường đi từ a đến b ko. Nếu coi mỗi phép thay thế là 1 cạnh trên đồ thị. Mỗi kí tự là 1 đỉnh.
- Ví dụ

Input1	Output1
3 abb dad	2 a d b a

- *Ta* có thể dùng 2 phép biến đổi: (*'a', 'd'*) và (*'b', 'a'*). Như vậy các chữ cái đầu sẽ trùng khớp khi ta sẽ thay thế chữ *'a'* bằng *'d'*. Các chữ cái thứ hai sẽ trùng khớp khi ta thay *'b'* bằng *'a'*. Các ký tự thứ ba sẽ trùng khớp khi ta thay thế *'b'* bằng *'a'* và *'a'* bằng *'d'*.
- Đọc kí tự đầu tiên của mỗi xâu: a và d, thuộc 2 cây khác nhau, hợp nhất cây chứa 2 đỉnh, **res++**
- Đọc kí tự thứ hai của mỗi xâu: b và a, thuộc 2 cây khác nhau, hợp nhất cây chứa 2 đỉnh. **ress++**
- Đọc kí tự thứ hai của mỗi xâu: b và d, thuộc cùng một cây,bỏ qua.



- Xuất:res
- Xuất: **v và parent[v]**

Code

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

```
using namespace std;
const int MAXN = 256;
string s, t;
int par[MAXN];
int F(int x){
  return par[x]==x ? x : F(par[x]);
}
int main(){
     ios_base::sync_with_stdio(0);            cin.tie();
cout.tie();
    freopen("replace.inp", "r", stdin);
    freopen("replace.out", "w", stdout);
  int n;
  cin >> n >> s >> t;
  for(int i = 0; i < MAXN; ++i) par[i]=i;
  int res = 0;
  for(int i = 0; i < n; ++i){
       int x = F(s[i]), y = F(t[i]);
       if(x != y){
            par[x] = y;
            ++res;
       }
  }
  cout << res << endl;
  for(int i = 0; i < MAXN; ++i){
       if(par[i] != i){
            cout <<(char) i <<' '<<(char)par[i] << endl;
       }
  }
  return 0;
}
```

Cảm nhận: Mô hình hoá bài toán trên đồ thị bài toán đưa về kiểm tra mỗi cặp kí tự của 2 xâu ta kiểm tra xem có đường đi từ a đến b ko. Ta sẽ hợp nhất dần các cạnh (thực hiện phép biến đổi $R(c1, c2)$ để đồ thị liên thông).

Bài 3. **Moocast (Nguồn: USACO 2016)**

Nông dân John có N con bò (1≤N≤1000) muốn tổ chức một hệ thống "moo-cast" khẩn cấp để phát các thông điệp quan trọng giữa chúng.

Thay vì la ó với nhau trong khoảng cách dài, những con bò quyết định trang bị cho mình bộ đàm, mỗi con một bộ đàm. Mỗi bộ đàm này đều có bán kính truyền giới hạn, nhưng các con bò có thể chuyển tiếp thông điệp cho nhau theo một con đường bao gồm một số bước nhảy, vì vậy không nhất thiết mỗi con bò đều có thể truyền trực tiếp cho mọi con bò khác.

Những con bò cần quyết định số tiền sẽ chi cho bộ đàm của chúng. Nếu chúng chi X đô la, mỗi con bò sẽ nhận được một bộ đàm có khả năng truyền đi một khoảng cách là $\sqrt{X}$. Tức là, khoảng cách bình phương giữa hai con bò phải nhiều nhất là X để chúng có thể giao tiếp.

Vui lòng giúp những con bò xác định giá trị nguyên tối thiểu của X sao cho cuối cùng một chương trình phát sóng từ bất kỳ con bò nào cũng có thể đến được với mọi con bò khác.

DLV: (tệp **moocast.inp):**

- Dòng đầu tiên của đầu vào chứa N.
- N dòng tiếp theo, mỗi dòng chứa tọa độ x và y của một con bò duy nhất. Cả hai là các số nguyên trong phạm vi 0… 25.000.

DLR: (tệp **moocast.out**):

- Viết một dòng kết quả duy nhất chứa số nguyên X cho biết số tiền tối thiểu mà những con bò phải chi cho bộ đàm.

Ví dụ

moocast.inp	**moocast.out**
4 1 3 5 4 7 2 6 1	17

Thuật toán:

- Trong bài toán này, chúng ta có N con bò nằm rải rác trên mặt phẳng và muốn tính khoảng cách D tối thiểu sao cho tất cả các con bò có thể giao tiếp với nhau (thông qua một số con bò trung gian), vì hai con bò chỉ có thể giao tiếp trực tiếp nếu khoảng cách giữa chúng nhỏ hơn hoặc bằng D.
- Chúng ta có thể mô hình bài toán này như một bài toán đồ thị trong đó mỗi con bò là một đỉnh nối với mọi con bò khác với một cạnh với trọng lượng

là khoảng cách giữa hai con bò. Do đó, chúng ta muốn tính trọng số cạnh tối thiểu sao cho đồ thị được liên thông.

- Chúng ta có thể giải quyết vấn đề này bằng cách sử dụng cấu trúc dữ liệu union-find. Một cấu trúc dữ liệu hỗ trợ hai hoạt động:
- Find(x) - trả về một định danh duy nhất cho đỉnh x Find (x) = Find (y) khi và chỉ khi x và y nằm trong cùng một thành phần liên thông.
- Merge (x, y) - nối x và y với một cạnh.
- Với N đỉnh ban đầu đều bị ngắt kết nối giữa các cặp, chúng ta có thể lặp lại các cạnh theo thứ tự không giảm theo trọng số và thêm một cạnh giữa hai đỉnh nếu chúng chưa nằm trong cùng một TPLT. Sau khi chúng ta thêm N - 1 cạnh, đồ thị đã được kết nối và cạnh cuối cùng cho chúng ta câu trả lời.

Code

```
#include <bits/stdc++.h>
#define ll long long
#define DD double
#define LD long double
#define PII pair<int, int>
#define F first
#define S second
#define MP make_pair
#define VI vector<int>
#define PB push_back
#define Hm(I) (1<<(I))
#define Gr(I) ((I)&(-(I)))
#define getbit(I, X) (((X)>>(I))&1)
#define MS(A, X) memset(A, X, sizeof(A))
#define ffu(I, CUOI) for(int I=1; I<=CUOI; I++)
#define ffd(I, DAU) for(int I=DAU; I>=1; I--)
#define fou(I, DAU, CUOI) for(int I=DAU; I<=CUOI; I++)
#define fod(I, DAU, CUOI) for(int I=DAU; I>=CUOI; I--)
#define _bp(X) ((X)*(X))
#define _kc2(X, Y, XX, YY) (_bp((X)-(XX))+_bp((Y)-(YY)))
using namespace std;
const int NMX=4320000, oo=0x3C3C3C3C, Bs=1000000007;
int n, m=0, parent[NMX], sz[NMX], nt=0, kq;
```

```
PII a[NMX];
pair<PII, int> e[NMX];
bool cmp(pair<PII, int> x, pair<PII, int> y)
{
    return x.S<y.S;
}
int DVy(int x)
{
    return x==parent[x] ? x : parent[x]=DVy(parent[x]);
}
void TNg(int x, int y)
{
    x=DVy(x);
    y=DVy(y);
    if (x!=y)
    {
        if (sz[x]<sz[y]) swap(x, y);
        parent[y]=x;
        sz[x]+=sz[y];
    }
}
int main()
{
    freopen("moocast.INP", "r", stdin);
    freopen("moocast.OUT", "w", stdout);
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> n;
    ffu(i, n) cin >> a[i].F >> a[i].S;
    ffu(i,  n-1)  fou(j,  i+1,  n)  e[++m]=MP(MP(i,  j),
_kc2(a[i].F, a[i].S, a[j].F, a[j].S));
    sort(e+1, e+1+m, cmp);
    ffu(i, n)
    {
        parent[i]=i;
```

```
            sz[i]=1;
        }
        ffu(i, m)
        {
            if (nt==n-1) break;
            if (DVy(e[i].F.F)==DVy(e[i].F.S)) continue;
            nt++;
            TNg(e[i].F.F, e[i].F.S);
            kq=e[i].S;
        }
        cout << kq;
        return 0;
    }
```

Cảm nhận: Bài này áp dụng đúng cách làm của DSU. Đầu tiên các cạnh của đồ thị chưa được kết nối. Sau đó kết nạp dần các cạnh, khi kết nạp đủ n-1 cạnh đồ thị liên thông và cạnh cuối cùng chính là kết quả của bài toán. Chương trình bài này ban đầu gán parent[i]=i; (i=1->n).

Bài 4. GALAKSIJA (Nguồn COCI 2015)

Cách đây rất lâu trong một thiên hà xa có N hành tinh. Ngoài ra còn có N - 1 con đường kết nối tất cả các hành tinh (trực tiếp hoặc gián tiếp). Nói cách khác, mạng lưới các hành tinh và những con đường tạo thành một cây. Ngoài ra, mỗi con đường dẫn liệt kê với một số nguyên biểu thị sự tò mò của con đường.

Một cặp hành tinh A, B thật nhàm chán nếu những điều sau đây là:

• A và B là các hành tinh khác nhau

• Đường đi giữa hành tinh A và B có thể sử dụng một hoặc nhiều đường

• XOR nhị phân của sự tò mò của tất cả các con đường đó bằng 0

Than ôi, thời thế đã thay đổi và một hoàng đế độc ác đang cai trị thiên hà. Ông ta quyết định sử dụng Thần lực để phá hủy tất cả các con đường nối các hành tinh theo một thứ tự nhất định.

Bạn hãy xác định số lượng cặp hành tinh nhàm chán trước khi hoàng đế bắt đầu sự hủy diệt và sau khi mỗi lần phá hủy.

DLV: **GALAKSIJA**.INP

- Dòng đầu tiên của đầu vào chứa số nguyên N (1 <= N <= 100 000).
- Mỗi dòng trong số N - 1 dòng sau chứa ba số nguyên Ai, Bi, Zi (1 <= Ai; Bi <= N, 0 <=Zi <= 1 000 000 000) biểu thị rằng hành tinh Ai và Bi được kết nối trực tiếp với một con đường tò mò Zi.
- Dòng sau chứa hoán vị của N - 1 số nguyên đầu tiên biểu thị thứ tự trong đó hoàng đế đang phá hủy các con đường. Nếu phần tử của hoán vị là j, thì hoàng đế đã phá hủy con đường giữa hai hành tinh Aj và Bj trong cùng một bước.

DLR: **GALAKSIJA**.OUT

- Đầu ra phải chứa N dòng, dòng thứ k chứa số cặp hành tinh buồn chán A, B từ nhiệm vụ sau khi hoàng đế phá hủy đúng k - 1 con đường.

Ràng buộc:

- 20% tổng số điểm, nó sẽ giữ N <= 1 000.
- 30% tổng số điểm, sự tò mò của mọi con đường sẽ bằng 0

Ví dụ

GALAKSIJA.INP	**GALAKSIJA**.OUT
2 **1 2 0** **1**	**1** **0**
3 **1 2 4** **2 3 4** **1 2**	**1** **0** **0**
4 **1 2 0** **2 3 0** **2 4 0** **3 1 2**	**6** **3** **1** **0**

- Giải thích ví dụ đầu tiên: Trước khi bị hủy diệt, con đường giữa hành tinh 1 và 2 rất nhàm chán. Sau hủy diệt, con đường giữa chúng không tồn tại nữa.
- Giải thích ví dụ thứ hai: Trước khi bị hủy diệt, một cặp hành tinh (1, 3) là nhàm chán. Con đường giữa 1 và 3 không còn có thể xảy ra sau lần phá hủy đầu tiên và sau lần hủy diệt thứ hai, và không có cặp nào trong số các cặp còn lại của hành tinh là nhàm chán.
- Giải thích ví dụ thứ ba: Lưu ý rằng trong ví dụ này, mỗi cặp hành tinh có một đường đi có thể giữa chúng là nhàm chán bởi vì tất cả các con đường có sự tò mò 0.

Thuật toán

- Chúng ta chọn một nút tùy ý **r** làm gốc của một số cây. Ký hiệu P_x là tổng XOR của tất cả các điểm tò mò trên đường dẫn từ nút r đến nút x. Dễ dàng nhận thấy rằng một cặp hành tinh x, y là nhàm chán nếu và chỉ nếu $\mathbf{P_x}$ **XOR** $\mathbf{P_y} = 0 \Leftrightarrow \mathbf{P_x} = \mathbf{P_y}$.
- Ta có thể "đảo ngược" dữ liệu đầu vào, thay vì phá hủy đường dẫn, chúng ta có thể thêm chúng.
- Để kết nối các thành phần, chúng tôi sẽ sử dụng thuật toán DSU, đối với mỗi thành phần, ta nhớ gốc, kích thước của nó và ngoài ra, một map ghi nhớ số lần P_x xuất hiện trong thành phần.
- Khi chúng ta cần kết nối hai thành phần, thành phần mới được tạo sẽ có gốc là gốc của thành phần lớn hơn. Bây giờ chúng ta phải duyệt toàn bộ thành phần nhỏ hơn (tức là sử dụng thuật toán DFS) và sửa các giá trị trong các giá trị lớn hơn map của thành phần.
- Độ phức tạp thời gian của thuật toán này là O (Nlg^2N)

Code

```
#include <cstdio>
#include <cassert>
#include <map>
#include <algorithm>
#include <vector>
using namespace std;
```

```
const int MAXN = 100005;
typedef pair <int, int> pii;
typedef long long llint;
llint curr;
llint ans[MAXN];
int n;
int a[MAXN], b[MAXN], z[MAXN];
int p[MAXN];
int path[MAXN];
int dad[MAXN];
int sz[MAXN];
map <int, vector<int>> M[MAXN];
void join (int a, int b, int c) {
  if (sz[dad[a]] < sz[dad[b]]) swap(a, b);
  int da = dad[a];
  int db = dad[b];
  for (auto it: M[db])
    for (auto x: it.second)
      curr += M[da][path[a] ^ c ^ path[b] ^ path[x]].size();
  int old = path[b];
  for (auto it: M[db]) {
    for (auto x: it.second) {
      path[x] = path[x] ^ old ^ c ^ path[a];
      dad[x] = da;
      M[da][path[x]].push_back(x);
    }
  }
  sz[da] += sz[db];
  M[db].clear();
  sz[db] = 0;
}
int main (void) {
  scanf("%d", &n);
  for (int i = 0; i < n-1; ++i) {
    scanf("%d%d%d", &a[i], &b[i], &z[i]);
    --a[i]; --b[i];
```

```
  }
  for (int i = 0; i < n-1; ++i) scanf("%d", &p[i]);
  for (int i = 0; i < n-1; ++i) --p[i];
  for (int i = 0; i < n; ++i) {
    M[i][0].push_back(i);
    dad[i] = i;
    sz[i] = 1;
  }
  for (int i = n-2; i >= 0; --i) {
    join(a[p[i]], b[p[i]], z[p[i]]);
    ans[i] = curr;
  }
  ans[n-1] = 0;
  for (int i = 0; i < n; ++i)
    printf("%lld\n", ans[i]);
  return 0;
}
```

Cảm nhận: Các bài toán áp dụng thuật toán DSU hầu như là ta phải tư duy bài toán theo chiều ngược lại. Giả sử các cặp đỉnh chưa được kết nối với nhau sau đó ta áp dụng DSU để gộp các thành phần vào. Đây là dạng bài tương đối khó với học sinh.

Bài 5. Sjekira (Nguồn COCI 2020)

Mirko cảm thấy mệt mỏi với công việc hàng ngày của mình vì vậy anh quyết định sống một cuộc sống đơn giản và chuyển đến một vùng nông thôn. Tuy nhiên, mùa đông ở ngôi làng xa xôi anh ấy mới chuyển đến rất khắc nghiệt, vì vậy anh ta quyết định tự mình đi chặt củi.

Hôm nay, anh ấy sẽ chặt chiếc cây đầu tiên của mình. Trước khi cắt, anh ta dán nhãn các bộ phận của thân cây đủ nhỏ để vừa với lò sưởi và đo độ cứng của chúng. Mirko là một lập trình viên, vì vậy anh ấy nhận thấy rằng các bộ phận và mối liên hệ giữa chúng tạo thành biểu đồ cây.

Thiệt hại trên chiếc rìu của anh ta do cắt một kết nối trên thân cây bằng **tổng của độ cứng tối đa trong hai thành phần được kết nối được tạo thành bằng cách cắt kết nối**. Mirko chỉ có một chiếc rìu, vì vậy anh ấy muốn tổng sát thương càng

nhỏ càng tốt. Anh ấy muốn biết Tổng thiệt hại tối thiểu đối với chiếc rìu, nếu anh ta cắt toàn bộ thân cây thành các bộ phận nhỏ vừa với lò sưởi.

DLV: **Sjekira.inp**

- Dòng đầu tiên chứa số nguyên n, số phần. Các phần được gắn nhãn bởi các số nguyên từ 1 đến n.
- Dòng thứ hai chứa n số nguyên ti ($1 \le ti \le 10^9$). Số ti là độ cứng của phần có nhãn i.
- Mỗi dòng trong số n - 1 dòng sau chứa hai số nguyên x và y ($1 \le x, y \le n$) - nhãn của các bộ phận là kết nối trực tiếp.

DLR: **Sjekira.out**

- Tạo ra tổng thiệt hại tối thiểu sau n - 1 lần cắt.

Sjekira.inp	Sjekira.out
3 1 2 3 1 2 2 3	8
4 2 2 3 2 1 3 3 2 4 3	15

Ràng buộc:

- Sub1: 5% test với 1<=n<=10
- Sub2: 5% test với thành phần i và i+1 được nối trực tiếp
- Sub3: 30% test n<=1000
- Sub4: 60% test với n<=10^5

- **Giải thích ví dụ 1:** Có hai cách để cắt thân cây này. Đầu tiên anh ta có thể cắt kết nối (1, 2), gây ra 1 + 3 = 4 thiệt hại, và sau đó cắt kết nối (2, 3), gây ra thiệt hại 2 + 3 = 5. Tổng thiệt hại là 9 trong này trường hợp. Nếu không, trước tiên anh ta có thể cắt (2, 3), và sau đó (1, 2). Tổng thiệt hại trong trường hợp đó (2 + 3) + (1 + 2) = 8.

Thuật toán:

Sub1: $n \le 10$, có thể thử tất cả các hoán vị của việc cắt kết nối và xác định từng giá hoán vị bằng cách sử dụng dfs hoặc cấu trúc DSU.

Nhận xét: nó sẽ luôn là tối ưu để cắt các kết nối xung quanh nút độ cứng lớn nhất trong cây đầu tiên. Chứng minh: trên đường đi giữa nút cứng nhất có giá trị M và nút cứng thứ hai của giá trị m, chúng ta phải cắt một số kết nối tại một số điểm và

trả m + M. Nếu kết nối gần M hơn, việc cắt giảm chi phí hiệu quả hơn vì sẽ có nhiều nút hơn trong cây con có giá trị lớn nhất là m ≤ M.

Vì vậy, sẽ luôn có một giải pháp tìm nút có độ cứng tối đa trong cây, cắt đứt tất cả các kết nối xung quanh nó, ta lại tìm nút cực đại trong các cây mới được hình thành, và đi xuống một cách đệ quy vào chúng. Bài toán đưa về tìm nút có độ cứng lớn nhất một cách hiệu quả.

Sub2: sử dụng cây phân đoạn (Segment Tree), cho phép bạn nhanh chóng tìm kiếm giá trị tối đa tại một khoảng (hoặc cây con) theo thời gian độ phức tạp O (n log n).

Sub3: (n ≤ 1000), chỉ cần tìm kiếm giá trị lớn nhất bằng cách sử dụng dfs là đủ, với độ phức tạp về thời gian O(n^2).

Sub4: . Giải pháp tương đương với

$$\sum_{i=1}^{n} t_i - \max_{1 \le i \le n} t_i + \sum_{i=1}^{n-1} \max(t_{x_i}, t_{y_i}).$$

Chứng minh: phụ thuộc vào n. Trường hợp cơ sở n = 1 là đúng. Khi chúng ta cắt kết nối a - bj xung quanh mức tối đa đỉnh a, ta cộng thêm $t_a + t_{aj}$, trong đó aj là đỉnh lớn nhất trong cây con của bj. Bây giờ thêm vào công thức cho các cây con kết thúc bước quy nạp.

Code

```
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;
const int MAXN = 100005;
typedef pair<int,int> ii;
int n;
int t[MAXN];
vector<ii> e;
int uf[MAXN], mv[MAXN];
bool cmp(const ii& a, const ii& b) {
    int x = max(t[a.first], t[a.second]);
```

```
    int y = max(t[b.first], t[b.second]);
    return x<y;
}
int fnd(int x){
    if(x==uf[x]) return x;
    else return uf[x]=fnd(uf[x]);
}
long long sol = 0;
void un(int x, int y){
    x = fnd(x);
    y = fnd(y);
    if(x != y){
        sol += mv[x] + mv[y];
        mv[y] = max(mv[y], mv[x]);
        uf[x] = y;
    }
}
int main(){
    scanf("%d", &n);
    for(int i=0;i<n;i++){
        scanf("%d", &t[i]);
        uf[i] = i;
        mv[i] = t[i];
    }
    for(int i=1;i<n;i++){
        int u,v;
        scanf("%d%d", &u, &v);
        e.push_back(ii(u-1, v-1));
    }
    sort(e.begin(), e.end(), cmp);
    for(auto i:e){
        un(i.first, i.second);
    }
    printf("%lld\n", sol);
}
void dfs(int w)
```

```
{
 visited[w] = true;
 ancestor[w] = w;//Đặt w vào một tập không giao nhau riêng
 for(int u : adj[w])
    {
      if(!visited[u])
       {
          dfs(u);
          union_set(w, u); ///hợp các đỉnh trong nhánh w
          ancestor[find_set(u)] = w;//vì trong quá trình
hợp nhất đại diện có thể bị  thay đổi
       }
    }
 ///tra loi cac truy van
 for(int u : queries[w])//Duyệt mọi truy vấn (w,u)
  {
 if(visited[u])
 cout << "lca(" << w << "," << u
 << ")= " << ancestor[find_set(u)] << "\n";
 }
}
```

Một số bài tập khác

Bài 6. Chiến binh (Nguồn Thầy Hiếu)

Một phú ông giàu có tại một vùng nọ, một hôm cảm thấy chán nản vì ko có gì để chơi, liền nghĩ ra một trò vô cùng hấp dẫn. Phú ông thuê n chiến binh đánh số họ từ 1…n. Sau đó phú ông cho các chiến binh chiến đấu với nhau trong m cuộc chiến, trong mỗi cuộc chiến phú ông sẽ chọn ra các chiến binh có chỉ số từ l đến r và cho họ chiến đấu với nhau, người thua sẽ bị loại ra. Chiến binh cuối cùng sót lại sau cuộc chiến sẽ là chiến binh thắng tất cả các chiến binh trong cuộc chiến đó, đương nhiên một chiến binh đã bị loại trong một cuộc chiến thì không thể tham gia các cuộc chiến sau đó nữa.

Cho n chiến binh và m cuộc chiến, hãy xác định chiến binh thứ i thua chiến binh nào.

Input

• Dòng đầu tiên chứa n và m (2≤n≤3×10^5, 1≤m≤3×10^5)

• m dòng tiếp theo mỗi dòng chứa li, ri và xi mô tả cuộc chiến thứ i, các chiến binh được chọn có chỉ số từ li đến ri và chiến binh xi là chiến binh thắng cuộc trong cuộc chiến đó.

Knight.inp	Knight.out
8 4 3 5 4 3 7 6 2 8 8 1 8 1	0 8 4 6 4 8 6 1

Output

- Gồm một dòng duy nhất chứa n số xi với xi là chiến binh thắng chiến binh i, xi = 0 nếu chiến binh i chưa chết sau m cuộc chiến.

Ví dụ

Thuật toán

Gọi next[u] là người bên phải còn sống gần nhất. Ban đầu next[u]=u.

Viết hàm get_next(u): phần tử đại diện của tập chứa u.

Code

```
while (true)  //L<=R
{
int u=get_next(l);
if (u>r) break;
if (u<k)
{
res[u]=k;
nextt[u]=k;
}
else if (u>k)
{
res[u]=k;
nextt[u]=des;
}
l=u+1;
}
} //Xuất mảng res
```

```
# include <bits/stdc++.h>
# define N 30000005
using namespace std;
int n,m;
int nextt[N],res[N];
int get_next(int x)
{
if (x==nextt[x]) return x;
else return nextt[x]=get_next(nextt[x]);
}
int main()
{
cin>>n>>m;
for (int i=1;i<=n+1;i++) {nextt[i]=i;res[i]=0;}
for (int i=1;i<=m;i++)
{
int l,r,k;
cin>>l>>r>>k;
int des=get_next(r+1);
while (true)
{
int u=get_next(l);
if (u>r) break;
if (u<k)
{
res[u]=k;
nextt[u]=k;
}
else if (u>k)
{
res[u]=k;
nextt[u]=des;
}
```

```
l=u+1;
}
}
for (int i=1;i<=n;i++) cout<<res[i]<<" ";
}
```

Bài 7 **CFARM - Closing the Farm (Nguồn USACO2016)**

Nông dân John và những con bò của anh ta dự định rời thị trấn để đi nghỉ dài ngày, vì vậy FJ muốn tạm thời đóng cửa trang trại của mình để tiết kiệm tiền trong thời gian chờ đợi.

Trang trại bao gồm N chuồng trại kết nối với M đường dẫn hai chiều giữa một số cặp chuồng ($1 \leq N, M \leq 3000$). Để đóng cửa trang trại, FJ có kế hoạch đóng cửa từng chuồng một. Khi một chuồng trại đóng cửa, tất cả các lối đi liền kề với chuồng trại đó cũng đóng lại và không thể sử dụng được nữa.

FJ quan tâm đến việc biết từng thời điểm (ban đầu và sau mỗi lần đóng cửa) liệu trang trại của anh ấy có liên thông hay không - nghĩa là có thể đi từ bất kỳ chuồng đang mở nào đến bất kỳ chuồng đang mở khác dọc theo một chuỗi đường thích hợp. Vì trang trại của FJ ban đầu ở trong tình trạng hư hỏng, nó thậm chí có thể không được kết nối hoàn toàn.

INPUT: Dữ liệu vào từ file **CFARM.INP**

- Dòng đầu tiên chứa N và M.
- Tiếp theo M mỗi dòng mô tả một con đường về cặp chuồng mà nó kết nối (các chuồng được đánh số thuận tiện 1 … N).
- N dòng cuối là một hoán vị của 1 … N mô tả thứ tự đóng cửa các chuồng trại.

OUTPUT: Ghi ra file **CFARM.OUT** gồm N dòng, mỗi dòng chứa "YES" hoặc "NO". Dòng đầu tiên cho biết trang trại ban đầu đã được kết nối đầy đủ hay chưa và dòng i + 1 cho biết liệu trang trại đã được liên thông sau khi nông trại thứ i đóng cửa.

VÍ DỤ

CFARM.INP	CFARM.OUT
4 3 1 2 2 3 3 4 3 4 1 2	YES NO YES YES

Thuật toán

- Ta mô tả bài toán dưới dạng đồ thị: chuồng: đỉnh (có n đỉnh), con đường: cạnh (có m cạnh). Ban đầu trang trại được kết nối đầy đủ nếu tất cả các đỉnh thuộc thành phần liên thông
- Giải pháp đơn giản nhất là làm theo đúng quá trình. Sau khi đóng mỗi chuồng, làm lại ds kề đồ thị và đếm số TPLT. Cụ thể ta duyệt DFS bắt đầu từ chuồng mở và kết thúc khi đã thăm hết chuồng thì đồ thị liên thông. Việc làm lại đồ thị và tìm kiếm mất O(m+n). Vì vậy tổng số lần đóng chuồng ta ĐPT:O(n^2+mn)
- Chú ý nếu (u, v) là một cạnh trong đồ thị ban đầu, thì u và v vẫn được kết nối cho đến khi u hoặc v bị loại bỏ. Do đó, chúng ta muốn một cấu trúc dữ liệu có thể theo dõi thành phần được kết nối mà một đỉnh nằm trong đó và cũng hỗ trợ hoạt động ngắt kết nối hai đỉnh. Ta có một cấu trúc dữ liệu được gọi là disjoint-set (DSU) hỗ trợ hiệu quả hai hoạt động - theo dõi thành phần được kết nối mà một đỉnh nằm trong đó và kết nối hai đỉnh.
- Nếu chúng ta muốn sử dụng DSU, chúng ta cần phải kết nối các đỉnh với nhau, vì vậy hãy tưởng tượng quá trình đang diễn ra ngược lại. Chúng ta bắt đầu với một trang trại trống và giới thiệu lại từng chuồng một, thêm

đường từ chuồng mới đến các chuồng hiện có nếu chúng là các cạnh trong đồ thị ban đầu. Đối với mỗi con đường mà chúng ta thêm vào, hãy sử dụng thao tác tìm DSU để kiểm tra xem các chuồng trại ở các điểm cuối có nằm trong các thành phần được kết nối khác nhau hay không. Nếu vậy, hãy sử dụng thao tác hợp nhất DSU để nối hai thành phần được kết nối. Điều này cho chúng ta ĐPT O(mlogn)

- Ta lưu ý: giả sử ta có (u,v) là một cạnh. Nếu đỉnh u có truy vấn thực hiện sau v thì ta cho u nối vào v. Tức là đỉnh v có đỉnh kề là u nhưng u không có đỉnh kề là v.

  Ví dụ: Order[1]=3, Order[2]=4, Order[3]=1, Order[4]=2,

  Place[3]=1, Place[4]=2, Place[1]= 3, Place[2]=4

  cạnh (1,2) ta có:adj[1]=2, do place[1]<place[2]

  cạnh (2,3) ta có:adj[3]=2, do place[3]<place[2]

  cạnh (3,4) ta có:adj[3]=4, do place[3]<place[4]

  Đọc truy vấn các đỉnh theo thứ tự ngược

Code

```
#include <bits/stdc++.h>
#define ffu(I, CUOI) for(int I=1; I<=CUOI; I++)
#define ffd(I, DAU) for(int I=DAU; I>=1; I--)
#define fou(I, DAU, CUOI) for(int I=DAU; I<=CUOI; I++)
using namespace std;
const int NMX=432000;
int n, m, parent[NMX], sz[NMX], u[NMX], v[NMX], a[NMX],
b[NMX], slt=0;
vector<vector<int> > g;
bool kq[NMX];
void KDg(int x)
{
    parent[x]=x;
    sz[x]=1;
}
int DVy(int x)
```

```
{
    if (x==parent[x]) return x;
    return parent[x]=DVy(parent[x]);
}
void TNg(int x, int y)
{
    x=DVy(x);
    y=DVy(y);
    if (x!=y)
    {
        if (sz[x]<sz[y]) swap(x, y);
        parent[y]=x;
        sz[x]+=sz[y];
        slt--;
    }
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    freopen("CFARM.INP","r",stdin);
    freopen("CFARM.OUT","w",stdout);
    cin >> n >> m;
    ffu(i, n) KDg(i);
    ffu(i, m) cin >> u[i] >> v[i];
    ffu(i, n)
    {
        cin >> a[i];
        b[a[i]]=i;
    }
    g.resize(n+1);
    ffu(i, m) if (b[u[i]]>b[v[i]]) g[v[i]].push_back(u[i]);
    else g[u[i]].push_back(v[i]);
    ffd(i, n)
    {
```

```
            int u=a[i];
            slt++;
            for (int j=0; j<g[u].size(); j++)
            {
                int v=g[u][j];
                TNg(u, v);
            }
            kq[i]=(slt<=1);
        }
        ffu(i, n) if (kq[i]) cout << "YES\n";
        else cout << "NO\n";
        return 0;
}
```

Bài 8. **MOOTUBE (Nguồn USACO 2018)**

Trong thời gian rảnh rỗi, Farmer John đã tạo ra một dịch vụ chia sẻ video mới, anh đặt tên là MooTube. Trên MooTube, những con bò của Farmer John có thể ghi lại, chia sẻ và khám phá nhiều video thú vị. Bò của anh ấy đã đăng N video ($1 \leq N \leq 100000$), được đánh số thuận tiện 1 … N. Tuy nhiên, FJ không thể tìm ra cách giúp những con bò của mình tìm thấy những video mới mà chúng có thể thích.
FJ muốn tạo danh sách "video đề xuất" cho mọi video trên MooTube. Bằng cách này, những con bò sẽ được giới thiệu những video phù hợp nhất với những video mà chúng đã xem.

FJ đặt ra một chỉ số về "mức độ liên quan", xác định mức độ liên quan của hai video với nhau. Anh ấy chọn N- 1 các cặp video và tính toán mức độ liên quan theo từng cặp theo cách thủ công. Sau đó, FJ hình dung các video của mình như một mạng lưới, nơi mỗi video là một nút và N- 1 các cặp video mà anh ấy coi là được kết nối theo cách thủ công. Một cách thuận tiện, FJ đã chọn N- 1 ghép nối để có thể truy cập bất kỳ video nào từ bất kỳ video nào khác theo đường dẫn kết nối theo đúng một cách. FJ quyết định rằng mức độ liên quan của bất kỳ cặp video nào phải được xác định là mức độ liên quan tối thiểu của bất kỳ kết nối nào dọc theo đường đi này.

Farmer John muốn chọn một giá trị K để bên cạnh bất kỳ video nào trên MooTube, tất cả các video khác có mức độ liên quan ít nhất K video đó sẽ được đề xuất. Tuy nhiên, FJ lo lắng rằng quá nhiều video sẽ được gợi ý cho những con bò của anh ấy, điều này có thể khiến chúng mất tập trung vào việc sản xuất sữa! Do đó, anh ấy muốn cẩn thận đặt một giá trị thích hợp của K. Farmer John muốn bạn giúp trả lời một số câu hỏi về các video được đề xuất cho một số giá trị nhất định của K.

INPUT: Dữ liệu vào từ file **MOOTUBE.INP** gồm

- Dòng đầu tiên chứa **N** và **Q (1 ≤ Q ≤ 100000).**
- Tiếp theo **N- 1** dòng mô tả một cặp video mà FJ so sánh thủ công. Mỗi dòng bao gồm ba số nguyên p_i, q_i và r_i **(1 ≤** p_i**,** q_i **≤ N, 1 ≤**r_i**≤** 10^9**),** cho biết rằng video p_i và q_i được kết nối với mức độ liên quan r_i.

- Tiếp theo **Q** dòng, mô tả có Q các câu hỏi. Mỗi câu hỏi chứa hai số nguyên k_i và v_i **(1 ≤**k_i**≤** 10^9 **, 1 ≤**v_i**≤ N),** với câu hỏi thứ I, có bao nhiêu video sẽ được gợi ý cho người xem video v_i nếu **K=**k_i**.**

OUTPUT: Với mỗi câu hỏi trong, ghi ra tệp **MOOTUBE.OUT** câu trả lời có bao nhiêu video liên quan đến video v_i.

MOOTUBE.INP	MOOTUBE.OUT
4 3 1 2 3 2 3 2 2 4 4 1 2 4 1 3 1	3 0 2

GIẢI THÍCH

Farmer John nhận thấy rằng video một và hai có mức liên quan ba, video hai và ba có mức liên quan hai và video hai và bốn có mức liên quan bốn. Dựa trên điều này, video một và ba có mức liên quan tối thiểu (3, 2) = 2, video một và bốn có mức liên quan tối thiểu (3, 4) = 3 và video ba và bốn có mức liên quan tối thiểu (2, 4) = 2.

Farmer John muốn biết có bao nhiêu video sẽ được đề xuất từ video hai nếu K= 1, từ video một nếu K= 3 và từ video một nếu K= 4. Chúng tôi thấy điều đó với K= 1, video 1, 3 và 4 sẽ được đề xuất trên video hai. Với K= 4, không có video nào sẽ được đề xuất từ video một. Với K= 3. Tuy nhiên, video 2 và 4 sẽ được gợi ý cho video một.

Thuật toán

- Chúng ta có một cây có trọng số vô hướng. Giả sử f (v, w) là trọng số nhỏ nhất trên tất cả các cạnh trên đường đi từ v đến w. Chúng tôi muốn trả lời một số truy vấn cho một đỉnh v và k đã cho có dạng - có bao nhiêu đỉnh w tồn tại trong đó f (v, w) ≥k? Để trả lời truy vấn này cho một đỉnh nhất định, chúng ta có thể bắt đầu bằng cách thực hiện một BFS từ v. Chúng ta không bao giờ đi qua một cạnh có trọng lượng cạnh nhỏ hơn k, vì vậy chúng ta bỏ qua các cạnh đó. Sau đó chúng ta có thể đếm xem chúng ta đã đến thăm bao nhiêu đỉnh khác**.**
- Chúng ta có thể đọc toàn bộ đồ thị và các truy vấn, sau đó sắp xếp các truy vấn theo thứ tự giảm dần của ngưỡng liên quan. Việc sắp xếp chúng theo thứ tự giảm dần giúp chúng ta như thế nào? Nếu chúng ta bắt đầu với một đồ thị trống và các truy vấn xử lý, chúng ta có thể sử dụng tất cả các cạnh từ truy vấn trước đó và thêm vào bất kỳ cạnh mới nào hiện có ít nhất lớn bằng ngưỡng mà chúng ta đang truy vấn. Tuy nhiên, lưu ý rằng bây giờ chúng ta chỉ đơn giản là đếm số đỉnh trong một thành phần được kết nối. Chúng ta có thể sử dụng cấu trúc dữ liệu union-find để duy trì kích thước của mọi thành phần được kết nối và hợp nhất hai thành phần bất cứ khi nào một cạnh trở nên hợp lệ.
- TT: ĐT đọc vào danh sách cạnh
- -Sắp xếp các truy vấn giảm dần theo mức độ liên quan
- Sắp xếp cạnh theo mức độ liên quan
- Duyệt các truy vấn theo thứ tự trên: Mỗi truy vấn duyệt các cạnh có mức độ liên quan>=mức độ liên quan của truy vấn đó. Hợp nhất cây chứa 2 đỉnh của cạnh. Xuất ra số đỉnh của vùng liên thông chứa đỉnh của truy vấn cần tìm

Code

```
# include<bits/stdc++.h>
using namespace std;
int n,q;
int root[1000005],sz[1000005];
int res[1000005];
struct data {
```

```
   int u,v;
   long long c;
}p[1000005];
struct dataz{
    long long k;
    int x,i;
}ord[1000005];
bool cmp(data x, data y)
{
    return x.c>y.c;
}
bool cmpz (dataz x, dataz y)
{
    return x.k>y.k;
}
int find_set(int x)
{
    if (x==root[x]) return x;
    else return root[x]=find_set(root[x]);
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    freopen("MOOTUE.INP","r",stdin);
    freopen("MOOTUE.OUT","w",stdout);
    cin>>n>>q;
    for (int i=1;i<n;i++)
    {
        cin>>p[i].u>>p[i].v>>p[i].c;
    }
    sort(p+1,p+n,cmp);
    for (int i=1;i<=q;i++)
    {
        cin>>ord[i].k>>ord[i].x;
```

```
        ord[i].i=i;
    }
    sort(ord+1,ord+q+1,cmpz);
    for (int i=1;i<=n;i++)
    {
        root[i]=i;
        res[i]=0;
        sz[i]=1;
    }
    int idx=1;
    for (int i=1;i<=q;i++)
    {
        while (idx<n&&p[idx].c>=ord[i].k)
        {
            int a=find_set(p[idx].u);
            int b=find_set(p[idx].v);
            idx++;
            if (a==b) continue;
            if (sz[a]<sz[b]) swap(a,b);
            root[b]=a;
            sz[a]+=sz[b];
        }
        res[ord[i].i]=sz[find_set(ord[i].x)]-1;
    }
    for (int i=1;i<=q;i++) cout<<res[i]<<endl;
}
```

Bài 9 Favorite Colors (Nguồn USACO 2020)

Mỗi con bò trong N con bò của Nông dân John ($1 \leq N \leq 2*10^5$) có một màu yêu thích. Những con bò được dán nhãn 1 … N và mỗi màu có thể được biểu thị bằng một số nguyên trong phạm vi 1…N.

Có M cặp bò (a,b) mô tả cho con bò b ngưỡng mộ con bò a ($1 \leq M \leq 2*10^5$). Có thể là a = b, trong trường hợp đó con bò tự ngưỡng mộ chính mình. Đối với một màu c bất kì, nếu có 2 con bò x và y cùng ngưỡng mộ con bò a có màu sắc yêu thích c, thì con bò x và y cũng yêu thích màu c.

Với thông tin này, hãy xác định việc gán màu sắc ưa thích của các con bò sao cho số lượng màu sắc ưa thích riêng biệt giữa tất cả các con bò là tối đa. Vì có nhiều cách gán đáp ứng yêu cầu này, hãy xuất ra đáp án nhỏ nhất về mặt từ vựng (nghĩa là bạn nên gán màu cho các con bò theo thứ tự từ 1 .. N).

INPUT: Dữ liệu vào từ file **FCOLOR.INP**

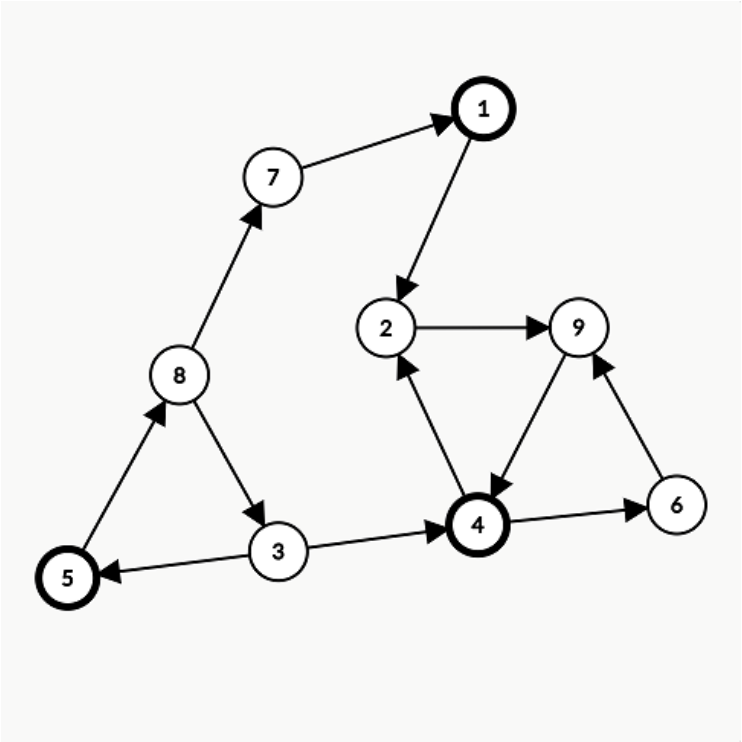
- Dòng đầu tiên chứa N và M.
- Tiếp theo M dòng, mỗi dòng chứa hai số nguyên được phân tách bằng dấu cách a và b ($1 \leq a, b \leq N$), biểu thị con bò b ngưỡng mộ con bò a. Cùng một cặp có thể xuất hiện nhiều lần trong đầu vào.

OUTPUT:

Với mỗi con bò thứ i trong các con bò từ 1 …N, ghi ra file **FCOLOR.OUT** màu yêu thích của con bò thứ i tương ứng.

FCOLOR.INP	FCOLOR.OUT
9 12 1 2 4 2 5 8 4 6 6 9 2 9 8 7 8 3 7 1 9 4 3 5 3 4	1 2 3 1 1 2 3 2 3

Trong hình dưới đây, các vòng tròn có đường viền in đậm đại diện cho những con bò có màu yêu thích 1.



Sol

- Nếu cả hai con bò b và c ngưỡng mộ con bò a thì cả hai con bò b và c phải có cùng màu sắc. Từ nay, chúng ta có thể coi cả b và c như một con bò; thay đổi tất cả các lần xuất hiện của c thành b và hợp nhất danh sách liền kề của c thành danh sách của b.
- Lặp lại quá trình đó khi mà vẫn có ít nhất hai con bò khác biệt chiêm ngưỡng cùng một con. Khi chúng ta đạt được cấu hình trong đó một con bò được ngưỡng mộ bởi nhiều nhất một con bò, quá trình này sẽ kết thúc; chúng ta có thể chỉ định mỗi con bò một màu riêng biệt.
- Nếu chúng ta luôn hợp nhất danh sách kề nhỏ hơn của hai con bò thành danh sách lớn hơn thì giải pháp của chúng ta sẽ chạy trong thời gian O ((N + M) logN).

Code

```
#include <bits/stdc++.h>
using namespace std;

void setIO(string s) {
      ios_base::sync_with_stdio(0); cin.tie(0);
      freopen((s+".in").c_str(),"r",stdin);
      freopen((s+".out").c_str(),"w",stdout);
}
```

```
const int MX = 2e5+5;

int N,M;

int par[MX],cnt[MX];
vector<int> adj[MX], rpar[MX];
queue<int> q;

void merge(int a, int b) {
	a = par[a], b = par[b];
	if (rpar[a].size() < rpar[b].size()) swap(a,b);
	for (int t: rpar[b]) { par[t] = a;
rpar[a].push_back(t); }
	adj[a].insert(end(adj[a]),begin(adj[b]),end(adj[b]))
;
	adj[b].clear();
	if (adj[a].size() > 1) q.push(a);
}

int main() {
	setIO("fcolor");
	cin >> N >> M;
	for (int i = 0; i < M; ++i) {
		int a,b; cin >> a >> b;
		adj[a].push_back(b);
	}
	for (int i = 1; i <= N; ++i) {
		par[i] = i; rpar[i].push_back(i);
		if (adj[i].size() > 1) q.push(i);
	}
	while (q.size()) {
		int x = q.front(); if (adj[x].size() <= 1) {
q.pop(); continue; }
		int a = adj[x].back(); adj[x].pop_back();
		if (par[a] == par[adj[x].back()]) continue;
		merge(a,adj[x].back());
	}
	int co = 0;
	for (int i = 1; i <= N; ++i) {
		if (!cnt[par[i]]) cnt[par[i]] = ++co;
		cout << cnt[par[i]] << "\n";
	}
}
```

Bài 10. Suma(Nguồn COCI 2014)

Mirko sống trong một khu rừng lớn đầy mê hoặc, nơi cây cối rất cao và phát triển rất nhanh. Khu rừng đó có thể được đại diện bởi ma trận N · N trong đó mỗi ô chứa một cây.

Mirko rất thích những cái cây trong khu rừng đầy mê hoặc. Anh đã dành nhiều năm quan sát chúng và đo từng cây đã phát triển bao nhiêu mét trong một năm. Cây cối mọc liên tục. Nói cách khác, nếu cây cao 5 mét trong một năm, nó sẽ phát triển 2,5 mét trong nửa năm.

Hôm qua, anh tự hỏi kích thước lớn nhất của nhóm cây được **kết nối** với nhau mà tất cả đều có chiều cao bằng nhau sẽ là bao nhiêu nếu cây cối tiếp tục phát triển với cùng tốc độ mà chúng đang phát triển tại thời điểm đó. Mirko nhanh chóng đo chiều cao hiện tại của tất cả các cây trong rừng và yêu cầu bạn trả lời câu hỏi của anh ấy.

Hai cây **kề nhau** nếu các ô của chúng trong ma trận có chung một cạnh.

Hai cây được **kết nối** với nhau nếu có một dãy cây liền kề dẫn từ cây thứ nhất đến cây thứ hai.

Một nhóm cây được **kết nối** nếu mọi cặp cây trong nhóm được **kết nối**

DLV: file **SUMA.INP**

- Dòng đầu tiên của đầu vào chứa số nguyên N (1 <= N <= 700).
- Sau dòng đầu tiên, N dòng tiếp theo, mỗi dòng chứa N số nguyên.
- Dòng thứ i chứa các số nguyên hij (1 <= hij <= 10^6), chiều cao ban đầu của cây trong hàng thứ i và cột thứ j, được cho trong mét.
- Sau đó, thêm N dòng tiếp theo với N số nguyên.
- Dòng thứ i chứa các số nguyên vij (1 <= vij <= 10^6), tốc độ phát triển của cây ở hàng thứ i và cột thứ j, đã cho tính bằng mét.

Suma.inp	Suma.out
2 3 1 3 3 2 5 2 5	3

DLR: SUMA.OUT

- Dòng đầu tiên và duy nhất của đầu ra phải chứa số yêu cầu từ nhiệm vụ.

ĐIỂM

30% tổng số điểm với 1 <= N <=70

Ví dụ

Giải thích: sau 8 tháng (hai phần ba của năm), các cây ở (0, 0), (0, 1) và (1, 0) chiều cao sẽ là 13/ 3 mét

Sol

Kiểm tra hai ô liền kề. Các cây trong các ô đó có thể bằng chiều cao trong nhiều nhất một thời điểm. Khoảnh khắc khi hai cây liền kề các ô có cùng độ cao sẽ được gọi là sự kiện.

Số sự kiện nhỏ hơn 4N. Đối với mỗi sự kiện, chúng ta có thể sử dụng DFS (hoặc BFS) để duyệt từ ô nơi cây có cùng chiều cao và đếm cây trong một thành phần, hãy cẩn thận trong một thời điểm nhất định chúng ta làm không lặp lại trên một ô nhiều hơn một lần.

Thuật toán này có độ phức tạp là O (N^2) vì chúng tôi sẽ lặp lại mọi ô tối đa 4 lần (một lần cho mỗi sự kiện bao gồm ô đó). Một thành phần cây được kết nối có thể phát triển cùng nhau (nếu độ cao ban đầu và tốc độ phát triển của chúng bằng nhau). Nếu chúng ta áp dụng thuật toán được mô tả trước, chúng ta có độ phức tạp thời gian là O (N^4). Đạt 30% tổng số điểm.

Ý tưởng tiếp theo là nén thành phần của các cây mọc cùng nhau thành một nút và xây dựng đồ thị trong đó hai nút được kết nối nếu các thành phần tương ứng của chúng nằm liền kề trong ma trận. Cách tiếp cận này không thay đổi độ phức tạp vì có thể tồn tại một thành phần có nhiều cạnh (độ lớn N^2) và DFS cho mọi sự kiện, chúng tôi sẽ lặp lại trên tất cả các cạnh của nó. Cách tiếp cận này, đưa ra một số tối ưu hóa bổ sung là đủ cho 50% tổng điểm.

Giải pháp cho 100% tổng số điểm dựa trên giải pháp trên, nhưng ta sẽ không sử dụng BFS hoặc DFS mà sử dụng cấu trúc DSU để xử lý sự kiện.

DSU sẽ nhớ cha của mỗi nút và gốc của mỗi tập hợp được kết nối sẽ ghi nhớ kích thước của tập hợp tương ứng của nó.

Đầu tiên, chúng ta nén tất cả các cây được kết nối phát triển như nhau thành một nút duy nhất, sau đó tính toán tất cả các sự kiện cho hai nút liền kề. Các sự kiện

được xử lý để chúng ta kết nối các nút nếu thời điểm đó chúng có độ cao bằng nhau. Trong quá trình xử lý sự kiện, chúng ta nhớ tất cả những thay đổi mà chúng ta đã thực hiện để được có thể khôi phục cấu trúc về trạng thái trước khi quá trình xử lý sự kiện bắt đầu.

Sau khi xử lý sự kiện, chúng ta khôi phục cấu trúc union-find về tình trạng ban đầu. Giải pháp này có độ phức tạp O (N^2 lg N).

Code

```
#include <algorithm>
#include <cassert>
#include <cstring>
#include <iostream>
using namespace std;
#define LOG(x) cerr << #x << " = " << (x) << "\n"
typedef long long llint;
typedef pair<int,int> pii;
const int MAXN = 710;
const int dx[] = { -1, 0, 0, 1 };
const int dy[] = { 0, -1, 1, 0 };
struct event { int y1, x1, y2, x2, k, l; };
struct change {
  pii B, old_dad, A;
  int old_size;
};
bool cmp(const event &A, const event &B) {
  return (llint)A.l * B.k < (llint)B.l * A.k;
}
int n;
int l[MAXN][MAXN];
int k[MAXN][MAXN];
vector<event> events;
struct union_find {
  pii dad[MAXN][MAXN];
  int sz[MAXN][MAXN];
  vector<change> changes;
  union_find() {
```

```
    for (int i = 0; i < MAXN; ++i)
      for (int j = 0; j < MAXN; ++j)
        dad[i][j] = {-1, -1}, sz[i][j] = 1;
  }
  pii find_dad(int y, int x, bool permanent=true) {
    if (dad[y][x].first == -1) return {y, x};
    pii      new_dad      =      find_dad(dad[y][x].first,
dad[y][x].second);
    if (permanent == false)
      changes.push_back({pii(y, x), dad[y][x], {-1, -1}, -
1});
    return dad[y][x] = new_dad;
  }
  int merge_sets(int y1, int x1, int y2, int x2, bool
permanent) {
    pii A = find_dad(y1, x1, permanent);
    pii B = find_dad(y2, x2, permanent);
    if (A == B) return sz[A.first][A.second];
    if (sz[A.first][A.second] < sz[B.first][B.second])
      swap(A, B);
    if (permanent == false)
      changes.push_back({B,   dad[B.first][B.second],   A,
sz[A.first][A.second]});
    sz[A.first][A.second] += sz[B.first][B.second];
    dad[B.first][B.second] = A;
    return sz[A.first][A.second];
  }

  void restore() {
    reverse(changes.begin(), changes.end());
    for (const change &c : changes) {
      dad[c.B.first][c.B.second] = c.old_dad;
      if (c.A.first != -1)
        sz[c.A.first][c.A.second] = c.old_size;
    }
    changes.clear();
```

```
  }
} U;
void process_neighbours(int y1, int x1, int y2, int x2) {
  if (y2 < 0 || y2 >= n || x2 < 0 || x2 >= n) return;
  int k_diff = k[y2][x2] - k[y1][x1];
  int l_diff = l[y1][x1] - l[y2][x2];
  if (k_diff == 0) {
    if (l_diff == 0) U.merge_sets(y1, x1, y2, x2, true);
    return;
  }
  if (k_diff < 0) {
    l_diff = -l_diff;
    k_diff = -k_diff;
  }
  if (l_diff < 0) return;
  events.push_back({y1, x1, y2, x2, k_diff, l_diff});
}

int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      scanf("%d", l[i]+j);
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      scanf("%d", k[i]+j);
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
      for (int d = 0; d < 2; ++d)
        process_neighbours(i, j, i+dy[d], j+dx[d]);
  int ans = 0;
  for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j) {
      U.find_dad(i, j);
      ans = max(ans, U.sz[i][j]);
    }
```

```
  sort(events.begin(), events.end(), cmp);
  for (int i = 0; i < (int)events.size(); ++i) {
    const event &e = events[i];
    ans = max(ans, U.merge_sets(e.y1, e.x1, e.y2, e.x2,
false));
    if (i + 1 == (int)events.size() || cmp(e, events[i+1]))
      U.restore();
  }
  printf("%d\n", ans);
  return 0;
}
```

TÀI LIỆU THAM KHẢO

1. Tập huấn GV 2019, Thầy Lê Minh Hoàng
2. Bài tập tin học chọn lọc, Thầy Trần Đỗ Hùng
3. Bài tập trên các trang USACO, COCI…
4. Một số tài liệu của các đồng nghiệp.

MỤC LỤC