

CHUYÊN ĐỀ HEAVY LIGHT DECOMPOSITION

I-Đặt vấn đề

Heavy Light Decomposition (HLD) là một cấu trúc dữ liệu được xây dựng trên cây cho phép giải quyết một lớp các bài toán truy vấn khi có sự thay đổi các giá trị cho trên các đỉnh hoặc các cạnh của cây. Chính xác hơn Heavy light Decomposition là một cách phân rã cây.

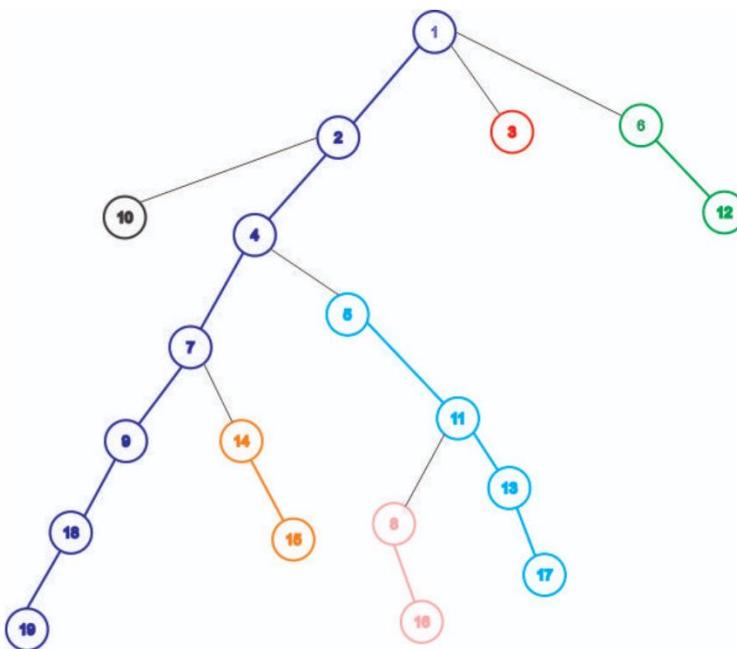
Giả sử cho một cây có trọng số $G=(V,E)$. Bằng cách thực hiện DFS trên cây từ gốc r ta định hướng lại cây. Khi đó mọi đỉnh $v \in V$ đều là gốc của một cây con. Đặt:

- $Pd[v]$ - đỉnh cha của đỉnh v
- $s[v]$ - số lượng đỉnh có trên cây con gốc v

Khi đó ta định nghĩa:

- Một cạnh (u, v) với $Pd[v] = u$ (v là đỉnh con của đỉnh u) được gọi là cạnh nặng (heavy) nếu như số đỉnh lượng đỉnh của cây con gốc v nhiều hơn một nửa số con, cháu... của đỉnh u : $s[v] > \frac{1}{2}(s[u] - 1) \leftrightarrow 2s[v] \geq s[u]$. Để thấy với mỗi đỉnh u thì chỉ có nhiều nhất một cạnh nặng nối từ nó đến con của nó
- Cạnh không phải là cạnh nặng thì được gọi là cạnh nhẹ (light)

Hình dưới đây cho một mô tả trực quan về cạnh nặng và cạnh nhẹ (các cạnh nhẹ có màu đen):



Nếu không xét các cạnh nhẹ thì tập hợp các cạnh nặng tạo thành một rừng các cây cạnh nặng có dạng đường thẳng (ta gọi tắt là các đoạn cạnh nặng).

Bây giờ chúng ta tạo một cây mới bằng cách coi mỗi đoạn cạnh nặng là một đỉnh, các cạnh của cây mới là cạnh nhẹ. Cây này được gọi là Heavy light Decomposition (HLD). Nếu như số đỉnh của cây ban đầu là n thì độ sâu của HLD không vượt quá $\log n$ vì mỗi khi đi xuống theo một cạnh nhẹ thì số đỉnh bị giảm đi tối thiểu là một nửa. Do vậy mọi truy vấn tiến hành trên cây mới sẽ có thời gian không vượt quá $O(\log n)$

Về nguyên tắc mọi truy vấn về đường đi trong cây ban đầu sẽ được chuyển thành truy vấn đường đi trên HLD và truy vấn trên các đỉnh của HLD. Vì các đỉnh của HLD có cấu trúc đường thẳng nên các truy vấn trên đỉnh có thể sử dụng Interval Tree (IT), Binary Indexed Tree (BIT) hay cây nhị phân đầy đủ (BST). Và chi phí thời gian cho mỗi truy vấn sẽ là $O(\log^2 n)$.

II-Cài đặt phân rã HLD

Việc cài đặt HLD thực hiện qua 2 giai đoạn:

Giai đoạn 1: Thực hiện duyệt cây theo chiều sâu (DFS) để định hướng lại cây (xây dựng quan hệ cha-con) và tính số lượng đỉnh trong mỗi cây con.

Giai đoạn 2: Thực hiện duyệt chiều sâu lần nữa trên cây. Tuy nhiên trong lần duyệt chiều sâu mới này thì đỉnh con của một đỉnh có số lượng đỉnh trong cây con của nó lớn nhất sẽ được ưu tiên duyệt trước.

Đặt:

$$x_1, x_2, \dots, x_n$$

là thứ tự thăm của lần duyệt chiều sâu thứ hai. Do việc ưu tiên duyệt các con có số lượng đỉnh trong cây con lớn nhất trước nên dễ thấy rằng các "đường nặng" lập thành dãy liên tiếp trong thứ tự nói trên. Đây cũng chính là tính chất quan trọng để có thể sử dụng các cấu trúc như IT (interval tree), BIT (binary indexed tree) thực hiện thay đổi dữ liệu trên các cạnh và các đỉnh của cây.

Ta cụ thể hóa các mô tả trên như sau:

Input:

```
int n; // Số đỉnh của cây
vector<int> g[maxn]; // Cây được biểu diễn bằng mảng vector
```

Output:

```
int Pd[maxn]; // Đỉnh cha
int pos[maxn]; // Vị trí các đỉnh khi DFS lần thứ hai (HLD)
int head[maxn]; // head[u] - Đầu "đường nặng" chứa u
int d_HLD[maxn]; // Mảng độ sâu trên HLD
```

Các biến nhập

```
int cl[maxn], s[maxn], id, smax[maxn];
```

1) DFS lần 1 để tính s[...], smax[...] và đưa cạnh nặng về đầu

```
void DFS(int u) {
    cl[u]=1;
    s[u]=1;
    smax[u]=0;
    int sz=g[u].size(), imax=0;
    for(int i=0;i<sz;++i) {
        int v=g[u][i];
        if (cl[v]==0) {
            Pd[v]=u;
            DFS(v);
            s[u] += s[v];
            if (smax[v]>imax) imax=smax[v];
        }
    }
    smax[u]=imax;
}
```

```

        //Ghi nhận vị trí của đỉnh con v có s[v] Lớn nhất
        if (s[v]>smax[u]) {
            smax[u]=s[v];
            imax=i;
        }
    }
    // Đưa đỉnh con v có s[v] Lớn nhất lên vị trí đầu tiên
    swap(g[u][0],g[u][imax]);
}

```

2) DFS lần thứ 2 để tính mảng pos, head

```

void HLD(int u) {
    pos[u]=++id; // Đỉnh u có vị trí id trên thứ tự DFS
    for(auto &v : g[u])
        if (Pd[v]==u) {
            // Nếu (u,v) là cạnh nặng thì điểm đầu đường nặng chứa v chính
            // Là điểm đầu của đường nặng chứa u. Ngoài ra trên cây HLD thì
            // u và v chập vào nhau do vậy u, v cùng độ sâu. Trường hợp (u,v)
            // Là cạnh nhẹ thì đỉnh v bắt đầu vị trí của một đường nặng mới

            if (2*s[v]>=s[u]) head[v]=head[u], d_HLD[v]=d_HLD[u];
            else head[v]=v, d_HLD[v]=d_HLD[u]+1;
            HLD(v); // Gọi đệ qui
        }
    }
}

```

Ghi nhớ: Sau khi thực hiện 2 quá trình duyệt chiều sâu nói trên chúng ta có được 3 mảng quan trọng:

- **pos[u]:** Là vị trí của đỉnh u sau khi DFS lần 2
- **head[u]:** Đỉnh đầu của đường nặng chứa đỉnh u
- **d_HLD[u]:** Độ sâu của đường nặng chứa u trên cây HLD

III-Ứng dụng cơ bản của HLD

1) Tính tổ tiên chung gần nhất (LCA)

Việc tìm tổ tiên chung gần nhất (LCA) của hai đỉnh được thực hiện theo phương pháp cổ điển trên cây HLD (cây mà mỗi đường nặng là một đỉnh). Có thể mô tả phương pháp này như sau:

- +B1) : Di chuyển "đỉnh" có độ sâu lớn hơn theo đỉnh cha nó cho đến khi hai "đỉnh" có cùng độ sâu. Chú ý rằng "đỉnh" ở đây là một đường nặng.
- +B2) : Chừng nào hai "đỉnh" (đường nặng) khác nhau thì di chuyển đồng thời về cha của nó. Quá trình này kết thúc khi đi đến một "đỉnh" chung.
- +B3) : Lúc này có hai đỉnh nằm trên một "đỉnh" (đường nặng) chung. Do vậy đỉnh nào có vị trí (pos) nhỏ hơn đỉnh đó sẽ là LCA cần tìm:

```

int LCA(int u,int v) {
    while (d_HLD[u]>d_HLD[v]) u=Pd[head[u]];

```

```

while (d_HLD[v] > d_HLD[u]) v=Pd[head[v]];
while (head[u] != head[v]) {
    u=Pd[head[u]];
    v=Pd[head[v]];
}
if (pos[u] < pos[v]) return u;
else return v;
}

```

Vì chiều sâu của cây HLD là $O(\log n)$ nên độ phức tạp của thuật toán trên là $O(\log n)$. Lưu ý rằng trong mỗi bước **u=Pd[head[u]]** ta thực hiện 2 công đoạn:

- Đưa đỉnh u về vị trí đầu tiên trên đường nặng chứa u: **u=head[u]**
- Nhảy qua "cạnh nhẹ" để đưa u về "đường nặng" độ sâu thấp hơn: **u=Pd[u]**

2) Cập nhật tăng các cạnh trên đường đi từ u đến v

Bài toán: Giả sử mỗi cạnh của cây ban đầu có một trọng số. Hãy tăng trọng số của tất cả các cạnh trên đường đi đơn từ đỉnh u đến đỉnh v một lượng **Delta**.

Nhận xét rằng đường đi từ u đến v được chia thành hai phần từ u đến LCA(u,v) và từ LCA(u,v) đến v. Trong thuật toán tìm LCA ta di chuyển đồng thời u và v đến LCA(u,v). Tư tưởng chính là trong quá trình di chuyển như vậy đồng thời ta ghi nhận cập nhật trọng số các cạnh trên hành trình di chuyển. Có hai lưu ý:

- Khi di chuyển từ u đến đầu đường nặng: Thực tế ta di chuyển đỉnh từ vị trí pos[u] đến vị trí pos[head[u]] và di chuyển dọc theo các cạnh nặng liên tiếp của một đường nặng. Các cạnh này nằm liên tiếp từ vị trí pos[head[u]]+1 đến vị trí pos[u]. Có thể sử dụng IT (interval tree) với cập nhật lười (lazy update) để làm điều này.
- Khi di chuyển qua một cạnh nhẹ, đơn giản chỉ việc thực hiện tăng trên cạnh này

Tối đa chúng ta đi qua $O(\log n)$ cạnh nhẹ, ngoài ra mỗi bước di chuyển $u=Pd[head[u]]$ thực hiện một lần lazy update trên IT. Do đó độ phức tạp thuật toán là $O(\log^2 n)$

```

int E_pd[maxn];           // E_pd[u] - độ dài cạnh (u,Pd[u])

void IncEdges(int u,int v,int Delta) {
    while (d_HLD[u]>d_HLD[v]) {
        // Từ u đến head[u]: Lazy update trên IT
        update(1,1,n,pos[head[u]]+1,pos[u],Delta);
        u=head[u];

        // Nhảy qua cạnh nhẹ -ghi nhận trên mảng E_pd[...]
        E_pd[u] += Delta;
        u=Pd[u];
    }
    while (d_HLD[v]>d_HLD[u]) {
        // Từ v đến head[v]: Lazy update trên IT
        update(1,1,n,pos[head[v]]+1,head[v],Delta);
        v=head[v];
    }
}

```

```

// Nhảy qua cạnh nhẹ -ghi nhận trên mảng E_pd[...]
E_pd[v] += Delta;
v=Pd[v];
}

// Di chuyển u, v đồng thời
while (head[u]!=head[v]) {
    update(1,1,n, pos[head[u]]+1, pos[u],Delta);
    u=head[u];
    E_pd[u] += Delta;
    u=Pd[u];
    update(1,1,n, pos[head[v]]+1, head[v],Delta);
    v=head[v];
    E_pd[v] += Delta;
    v=Pd[v];
}

if (pos[u]<pos[v]) {
    update(1,1,n, pos[u]+1, pos[v],Delta); // u là LCA
} else {
    update(1,1,n, pos[v]+1, pos[u],Delta); // v là LCA
}
}

```

3) Lấy cạnh lớn nhất trên đường đi từ u đến v

```

int GetEdges(int u,int v) {
    int kq=-INF;
    while (d_HLD[u]>d_HLD[v]) {
        kq=max(kq,get(1,1,n, pos[head[u]]+1, pos[u]));
        u=head[u];
        kq=max(kq,E_pd[u]);
        u=Pd[u];
    }
    while (d_HLD[v]>d_HLD[u]) {
        kq=max(kq,get(1,1,n, pos[head[v]]+1, pos[v]));
        v=head[v];
        kq=max(kq,E_Pd[v]);
        v=Pd[v];
    }
    while (head[u]!=head[v]) {
        kq=max(kq,get(1,1,n, pos[head[u]]+1, pos[u]));
        u=head[u];
        kq=max(kq,E_pd[u]);
        u=Pd[u];
        kq=max(kq,get(1,1,n, pos[head[v]]+1, pos[v]));
        v=head[v];
        kq=max(kq,E_Pd[v]);
        v=Pd[v];
    }
}

```

```

        kq=max(kq,E_Pd[v]);
        v=Pd[v];
    }
    if (pos[u]<pos[v]) {
        kq=max(kq,get(1,1,n,pos[u]+1,pos[v]));
    } else {
        kq=max(kq,get(1,1,n,pos[v]+1,pos[u]));
    }
    return kq;
}

```

IV- Một số bài tập áp dụng

Bài 1: Trồng cỏ [GPLANT.*]

(Bài tập trên USACO)

Link tests:

<https://drive.google.com/file/d/10IFNLtPgXDpkDw-WDJNICNsFjv3orbrU/view?usp=sharing>

Nông dân John (FJ) có N cánh đồng cỏ cắn cỗi được nối với nhau bởi N-1 đường đi hai chiều sao cho giữa hai cánh đồng cỏ chỉ có duy nhất một đường đi đến nhau. Bessie, một cô bò yêu thời gian thường phàn nàn về việc không có cỏ trên con đường nối giữa hai cánh đồng cỏ (để có thể tranh thủ vừa đi vừa ăn). FK rất yêu quý Bessie nên ông ta quyết định trồng cỏ trên những con đường. Ông ta sẽ thực hiện việc trồng cỏ theo một qui trình gồm có M ($1 \leq M \leq 10^5$) bước. Mỗi bước sẽ có một trong hai sự kiện sau xảy ra:

- FJ chọn ra hai cánh đồng cỏ và trồng thêm trên mỗi con đường nối hai cánh đồng này một nhúm cỏ
- Bessie sẽ hỏi có bao nhiêu nhúm cỏ trên đường đi nối giữa hai đồng cỏ nào đó và FJ phải trả lời cô câu hỏi này.

FJ không giới trong việc tính toán. Hãy giúp ông ta trả lời những câu hỏi của Bessie

Input:

- Dòng đầu tiên ghi hai số nguyên N, M ($2 \leq N \leq 10^5$, $1 \leq M \leq 10^5$)
- N-1 dòng tiếp theo, mỗi dòng ghi hai số nguyên thể hiện một đường nối
- M dòng tiếp theo mô tả các sự kiện xảy ra lần lượt. Đầu tiên là ký tự P hoặc Q thể hiện việc FJ trồng cỏ hay trả lời câu hỏi. Tiếp theo là hai số nguyên a_i , b_i thể hiện công việc cần làm của FJ hoặc câu hỏi cần trả lời

Output: Mỗi dòng ghi một câu trả lời cho câu hỏi theo trình tự xuất hiện trong file dữ liệu

Example:

input	output
4 6	2
1 4	1
2 4	2
3 4	
P 2 3	
P 1 3	
Q 3 4	
P 1 4	
Q 2 4	
Q 1 4	

Thuật toán:

Sử dụng phân rã HLD để tạo ra thứ tự DFS trên HLD (mảng pos). Khi đó việc cập nhật và tính toán có thể sử dụng cấu trúc IT (interval tree) với cập nhật lười (lazy update).

Chương trình tham khảo:

```
#include <bits/stdc++.h>
#define z 100005
using namespace std;
int n,m;
int cl[z],s[z],smax[z];
vector < int > g[z];
int pre[z];
int pos[z],id=0;
int head[z],d_HLD[z];
int it[5*z],dt[5*z];
int E_pre[z];
void DFS(int u)
{
    cl[u]=1;
    s[u]=1;
    smax[u]=0;
    int imax=0;
    int sz=g[u].size();
    for(int i=0; i<sz; ++i)
    {
        int v=g[u][i];
        if(cl[v]==0)
        {
            pre[v]=u;
            DFS(v);
            s[u]+=s[v];
            if(s[v]>smax[u])
            {
                smax[u]=s[v];
                imax=i;
            }
        }
    }
    swap(g[u][0],g[u][imax]);
}
void HLD(int u)
{
    pos[u] = ++id;
    for (auto &v : g[u])
        if (pre[v] == u)
        {
            if (2*s[v] >= s[u])
            {
                head[v] = head[u];
                d_HLD[v] = d_HLD[u];
            }
            else
            {
                head[v] = v;
                d_HLD[v] = d_HLD[u] + 1;
            }
            HLD(v);
        }
}
void update(int r, int k, int l,int u, int v, int delta)
{
    if(u>l||v<k)
        return;
    if(u<=k&&l<=v)
    {
        dt[r]+=delta;
        return;
    }
}
```

```

int g=(k+1)/2;
dt[2*r]+=dt[r];
dt[2*r+1]+=dt[r];
dt[r]=0;
update(2*r,k,g,u,v,delta);
update(2*r+1,g+1,l,u,v,delta);
it[r]=it[2*r]+dt[2*r]*(g-k+1)+it[2*r+1]+dt[2*r+1]*(l-g);
}
int get(int r,int k,int l,int u,int v)
{
    if(u>l||v<k)
        return 0;
    if(u<=k&&l<=v)
        return it[r]+dt[r]*(l-k+1);
    int g=(k+1)/2;
    dt[2*r]+=dt[r];
    dt[2*r+1]+=dt[r];
    dt[r]=0;
    int tl=get(2*r,k,g,u,v);
    int tr=get(2*r+1,g+1,l,u,v);
    it[r]=it[2*r]+dt[2*r]*(g-k+1)+it[2*r+1]+dt[2*r+1]*(l-g);
    return tl+tr;
}
void plant(int u, int v,int delta)
{
    while(d_HLD[u]>d_HLD[v])
    {
        update(1,1,n,pos[head[u]]+1,pos[u],delta);
        u=head[u];
        E_pre[u]+=delta;
        u=pre[u];
    }
    while(d_HLD[v]>d_HLD[u])
    {
        update(1,1,n,pos[head[v]]+1,pos[v],delta);
        v=head[v];
        E_pre[v]+=delta;
        v=pre[v];
    }
    while(head[u]!=head[v])
    {
        update(1,1,n,pos[head[u]]+1,pos[u],delta);
        u=head[u];
        E_pre[u]+=delta;
        u=pre[u];
        update(1,1,n,pos[head[v]]+1,pos[v],delta);
        v=head[v];
        E_pre[v]+=delta;
        v=pre[v];
    }
    if(pos[u]<pos[v])
        update(1,1,n,pos[u]+1,pos[v],delta);
    else
        update(1,1,n,pos[v]+1,pos[u],delta);
}
int dem(int u, int v )
{
    int kq=0;
    while(d_HLD[u]>d_HLD[v])
    {
        kq+=get(1,1,n,pos[head[u]]+1,pos[u]);
        u=head[u];
        kq+=E_pre[u];
        u=pre[u];
    }
}

```

```

while(d_HLD[v]>d_HLD[u])
{
    kq+=get(1,1,n, pos[head[v]]+1, pos[v]);
    v=head[v];
    kq+=E_pre[v];
    v=pre[v];
}
while(head[u]!=head[v])
{
    kq+=get(1,1,n, pos[head[u]]+1, pos[u]);
    u=head[u];
    kq+=E_pre[u];
    u=pre[u];
    kq+=get(1,1,n, pos[head[v]]+1, pos[v]);
    v=head[v];
    kq+=E_pre[v];
    v=pre[v];
}
if(pos[u]<pos[v])
    kq+=get(1,1,n, pos[u]+1, pos[v]);
else
    kq+=get(1,1,n, pos[v]+1, pos[u]);
return kq;
}
int main ()
{
    freopen("GPLANT.inp", "r", stdin);
    freopen("GPLANT.out", "w", stdout);
    scanf("%d %d \n", &n, &m);
    for(int i=1; i<n; ++i)
    {
        int u,v;
        scanf("%d %d \n", &u, &v);
        g[u].push_back(v);
        g[v].push_back(u);
    }
    DFS(1);
    memset(cl ,0 , sizeof cl);
    HLD(1);
    for(int i=1; i<=m; ++i)
    {
        char c;
        scanf("%c ", &c);
        if(c=='P')
        {
            int u,v;
            scanf("%d %d \n", &u, &v);
            plant( u, v, 1 );
        }
        else
        {
            int u,v;
            scanf("%d %d \n", &u, &v);
            printf("%d \n", dem( u, v));
        }
    }
}

```

Bài 2: Tính khoảng cách [DISNODE.*]

Link tests:

<https://drive.google.com/file/d/1AwOhRRQsiINrV-XAP89WoZh9wYZumCQC/view?usp=sharing>

Cho một cây có n đỉnh có trọng số, các đỉnh đánh số từ 1 đến n . Trọng số của một đường đi được tính bằng tổng trọng số các đỉnh trên đường đi này.

Hãy viết chương trình thực hiện các truy vấn thuộc 2 loại sau:

- **1 i c:** Đỉnh thứ i sẽ được mang trọng số mới là c
- **2 u v:** Tìm độ dài đường đi ngắn nhất nối đỉnh u với đỉnh v

Input:

- Dòng đầu tiên ghi số nguyên dương n
- $n - 1$ dòng tiếp theo, mỗi dòng ghi 3 số u, v, w thể hiện có một cạnh nối đỉnh u với đỉnh v và có trọng số là w
- Dòng tiếp theo ghi số nguyên dương Q là số lượng truy vấn
- Q dòng cuối, mỗi dòng thể hiện một truy vấn thực hiện theo thứ tự

Output: Với mỗi truy vấn loại 2 in ra giá trị tìm được trên một dòng

Ràng buộc:

- $1 \leq i \leq n$
- $1 \leq u, v \leq n; u \neq v$
- $1 \leq w, c \leq 10^4$

Example:

input	output
5	8
1 2 2	6
2 3 4	
4 2 3	
5 4 1	
3	
2 5 3	
1 3 1	
2 5 3	

Ghi chú:

- Subtask 1: $1 \leq n, Q \leq 10^3$
- Subtask 2: $1 \leq n, Q \leq 10^5$

Thuật toán:

Nhận xét rằng trọng số của các đỉnh đều là số không âm nên đường đi ngắn nhất giữ hai đỉnh chính là đường đi đơn trên cây. Sử dụng kỹ thuật HLD kết hợp với IT tương tự như bài 1. Chú ý rằng việc cập nhật là cập nhật trên đỉnh chứ không phải trên cạnh.

Chương trình tham khảo:

```
#include <iostream>
#include <cstdio>
#include <stdio.h>
#include <math.h>
#include <algorithm>
#include <vector>

using namespace std;

const int maxn = 100005;

typedef pair<int, int> II;
typedef pair<II, int> III;

int N, M, A, B, id = 0, nT = 0, root;
char ch; bool color[maxn];
int Depth_HLD[maxn], Depth_DFS[maxn];
int Prev[maxn], Child[maxn], LightPrev[maxn];
int Stop[maxn], Head[maxn], Pos[maxn];
int it[maxn * 5], dt[maxn * 5], deg[maxn];
```

```

vector<II> g[maxn];
III Edge[maxn];

int ReadInt()
{
    char ch;
    do ch = getchar();
    while (ch != '+' && ch != '-' && (ch < '0' || ch > '9'));
    int sign = (ch == '-') ? -1 : 1;
    int res = (ch >= '0' && ch <= '9') ? ch - '0' : 0;
    ch = getchar();
    while (ch >= '0' && ch <= '9') {
        res = res * 10 + ch - '0';
        ch = getchar();
    }
    return res * sign;
}

void WriteInt(int x)
{
    if (x < 0) putchar('-'), x = -x;
    if (x > 9) WriteInt(x / 10);
    putchar(x % 10 + '0');
}

void Input()
{
    N = ReadInt();
    for (int i = 1, u, v, w; i < N; ++i) {
        u = ReadInt(); v = ReadInt(); w = ReadInt();
        Edge[i] = III(II(u, v), w);
        g[u].push_back(II(v, w)); ++deg[u];
        g[v].push_back(II(u, w)); ++deg[v];
    }
    M = ReadInt();
}

void DFS(int u)
{
    color[u] = true; Child[u] = 1;
    int maxChild = 0, imax = -1;
    for (int i = 0; i < deg[u]; ++i) {
        int v = g[u][i].first;
        if (!color[v]) {
            Prev[v] = u; Depth_DFS[v] = Depth_DFS[u] + 1;
            DFS(v); Child[u] += Child[v];
            if (maxChild < Child[v]) maxChild = Child[v], imax = i;
        }
    }
    if (imax >= 0) swap(g[u][0], g[u][imax]);
}

void HLD(int u)
{
    Pos[u] = ++id;
    for (int i = 0; i < deg[u]; ++i) {
        int v = g[u][i].first;
        if (Prev[v] == u) {
            Head[v] = (i == 0 && 2 * Child[v] >= Child[u]) ? Head[u] : v;
            if (Head[v] == v) Depth_HLD[v] = Depth_HLD[u] + 1;
            else Depth_HLD[v] = Depth_HLD[Head[v]];
            HLD(v);
        }
    }
    Stop[u] = id;
}

```

```

}

void Update(int r, int k, int l, int u, int v, int val)
{
    if (v < k || u > l) return ;
    if (u <= k && l <= v) {dt[r] += val; return ;}
    dt[2 * r] += dt[r];
    dt[2 * r + 1] += dt[r];
    dt[r] = 0;
    int g = (k + l) / 2;
    Update(2 * r, k, g, u, v, val);
    Update(2 * r + 1, g + 1, l, u, v, val);
    it[r] = it[2 * r] + (g - k + 1) * dt[2 * r] + it[2 * r + 1] + dt[2 * r + 1] * (l - g);
}
}

int Get(int r, int k, int l, int u, int v)
{
    if (v < k || u > l) return 0;
    if (u <= k && l <= v) return it[r] + dt[r] * (l - k + 1);
    dt[2 * r] += dt[r];
    dt[2 * r + 1] += dt[r];
    dt[r] = 0;
    int g = (k + l) / 2;
    int tL = Get(2 * r, k, g, u, v);
    int tR = Get(2 * r + 1, g + 1, l, u, v);
    it[r] = it[2 * r] + (g - k + 1) * dt[2 * r] + it[2 * r + 1] + (l - g) * dt[2 * r + 1];
    return tL + tR;
}

void IncEdge(int u, int v, int val)
{
    while (Depth_HLD[u] > Depth_HLD[v]) {
        Update(1, 1, N, Pos[Head[u]], Pos[u] - 1, val);
        LightPrev[Head[u]] += val;
        u = Prev[Head[u]];
    }

    while (Depth_HLD[v] > Depth_HLD[u]) {
        Update(1, 1, N, Pos[Head[v]], Pos[v] - 1, val);
        LightPrev[Head[v]] += val;
        v = Prev[Head[v]];
    }

    while (Head[u] != Head[v]) {
        Update(1, 1, N, Pos[Head[u]], Pos[u] - 1, val);
        LightPrev[Head[u]] += val;
        u = Prev[Head[u]];

        Update(1, 1, N, Pos[Head[v]], Pos[v] - 1, val);
        LightPrev[Head[v]] += val;
        v = Prev[Head[v]];
    }

    if (Depth_DFS[u] > Depth_DFS[v])
        Update(1, 1, N, Pos[v], Pos[u] - 1, val);
    else Update(1, 1, N, Pos[u], Pos[v] - 1, val);
}

int AmountGrass(int u, int v)
{
    int res = 0;
    while (Depth_HLD[u] > Depth_HLD[v]) {
        res += Get(1, 1, N, Pos[Head[u]], Pos[u] - 1);

```

```

        res += LightPrev[Head[u]];
        u = Prev[Head[u]];
    }

    while (Depth_HLD[v] > Depth_HLD[u]) {
        res += Get(1, 1, N, Pos[Head[v]], Pos[v] - 1);
        res += LightPrev[Head[v]];
        v = Prev[Head[v]];
    }

    while (Head[u] != Head[v]) {
        res += Get(1, 1, N, Pos[Head[u]], Pos[u] - 1);
        res += LightPrev[Head[u]];
        u = Prev[Head[u]];

        res += Get(1, 1, N, Pos[Head[v]], Pos[v] - 1);
        res += LightPrev[Head[v]];
        v = Prev[Head[v]];
    }

    if (Depth_DFS[u] > Depth_DFS[v])
        res += Get(1, 1, N, Pos[v], Pos[u] - 1);
    else res += Get(1, 1, N, Pos[u], Pos[v] - 1);

    return res;
}

void Solve()
{
    DFS(1);
    HLD(1);

    for (int i = 1; i < N; ++i) {
        int u = Edge[i].first.first, v = Edge[i].first.second, t = Edge[i].second;
        if (Prev[u] == v) swap(u, v);
        IncEdge(u, v, t);
    }

    int u, v, i, c, t, type;
    while (M--) {
        type = ReadInt();
        if (type == 1) {
            i = ReadInt(); c = ReadInt();
            u = Edge[i].first.first;
            v = Edge[i].first.second;
            t = -AmountGrass(u, v) + c;
            IncEdge(u, v, t);
        }
        else {
            u = ReadInt(); v = ReadInt();
            WriteInt(AmountGrass(u, v)), putchar('\n');
        }
    }
}

void Output()
{

}

int main()
{
    #define TASK "DISNODE"
    #ifdef TASK
    freopen(TASK".INP", "r", stdin);

```

```

freopen(TASK".OUT", "w", stdout);
#else
freopen("INPUT.INP", "r", stdin);
#endif // TASK
Input();
Solve();
Output();
}

```

Bài 3: Xây cầu đường bộ [BRBUILD.*]

Link tests:

https://drive.google.com/file/d/1laxfUJbe5PDJVxVeqnHU_p66ZgKWAOPq/view?usp=sharing

Quốc đảo Byteotia gồm n thành phố (đánh số từ 1 đến n) được nối với nhau bởi các tuyến đường thủy, thỏa mãn giữa hai thành phố bất kỳ có đúng một cách đi lại giữa chúng (trực tiếp hoặc qua một số thành phố trung gian). Tuy nhiên do sự bất tiện mà đường thủy mang lại, chính quyền ở quốc đảo đã lên lộ trình thanh thế toàn bộ các tuyến đường thủy bằng các cầu đường bộ.

Công ty vận tải ABC có trụ sở đặt tại thành phố số hiệu 1 cần vận chuyển m chuyến hàng xen kẽ với lịch thay thế các tuyến đường thủy và họ rất quan tâm xem mỗi lượt vận chuyển đó cần bao nhiêu tuyến đường thủy (vì thực sự chi phí cho nó là rất tốn kém).

Yêu cầu: Hãy giúp công ty ABC xác định số tuyến đường thủy phải qua trong mỗi lượt vận chuyển.

Input:

- Dòng đầu tiên chứa số nguyên n
- $n - 1$ dòng tiếp theo, mỗi dòng chứa một cặp số nguyên (u, v) thể hiện có một tuyến đường thủy nối hai thành phố có số hiệu u và v ($u < v$)
- Dòng tiếp theo chứa số nguyên m
- $m + n - 1$ dòng tiếp theo mỗi dòng có hai dạng (xuất hiện theo trình tự thời gian):
 - $A u v$: Chỉ việc thay thế tuyến đường thủy nối hai thành phố số hiệu u và v bằng cầu đường bộ. ($u < v$ và có $n - 1$ dòng loại này)
 - $W v$: Chỉ một lượt vận chuyển hàng từ thành phố số hiệu 1 đến thành phố có số hiệu v (có m dòng loại này)

(Các số trên cùng một dòng được ghi cách nhau ít nhất một dấu trống)

Output: Ghi ra trên m dòng, mỗi dòng là một số nguyên là số tuyến đường thủy phải đi qua tương ứng với mỗi lượt chuyển hàng của công ty ABC (theo thứ tự thời gian)

Example:

input	output
5	2
1 2	1
1 3	0
1 4	1
4 5	
4	
W 5	
A 1 4	
W 5	
A 4 5	
W 5	
W 2	
A 1 2	
A 1 3	

Thuật toán: Xây dựng đồ thị với mỗi hòn đảo là một đỉnh, mỗi tuyến đường thủy là một cạnh. Chúng ta có mô hình một cây. Gán trọng số các cạnh là 1 nếu là đường thủy và là 0 nếu là

đường bộ. Khi đó mỗi truy vấn đếm số đường thủy là truy vấn tính độ dài đường đi đơn từ đỉnh 1 đến đỉnh cần xét.

Sử dụng phân rã HLD kết hợp với IT tổng ta giải quyết được bài toán này. Chú ý IT được xây dựng trên các phần tử cơ bản là các cạnh.

Chương trình tham khảo:

```
#include <iostream>
#include <stdio.h>
#include <cstdio>
#include <math.h>
#include <algorithm>
#include <vector>

using namespace std;

const int maxn = 1000005;

struct Tnode {
    int idmin, idmax;
    int L, R;
    int val, delta;
} Tree[2 * maxn];

int N, M, id = 0, nT = 0; bool color[maxn];
int Start[maxn], Stop[maxn];
int Depth[maxn], Prev[maxn];
vector<int> g[maxn];

int ReadInt()
{
    char ch;
    do ch = getchar();
    while (ch != '-' && ch != '+' && (ch < '0' || ch > '9'));
    int sign = (ch == '-') ? -1 : 1;
    int res = (ch >= '0' && ch <= '9') ? ch - '0' : 0;
    ch = getchar();
    while (ch >= '0' && ch <= '9') {
        res = res * 10 + ch - '0';
        ch = getchar();
    }
    return res * sign;
}

void WriteInt(int x)
{
    if (x < 0) putchar('-'), x = -x;
    if (x > 9) WriteInt(x / 10);
    putchar(x % 10 + '0');
}

void Input()
{
    N = ReadInt();
    for (int i = 1, u, v; i < N; ++i) {
        u = ReadInt(); v = ReadInt();
        g[u].push_back(v);
        g[v].push_back(u);
    }
}
```

```

        g[u].push_back(v);
        g[v].push_back(u);
    }

    M = ReadInt() + N - 1;
}

void DFS(int u)
{
    color[u] = true;
    Start[u] = ++id;
    for (int v : g[u]) {
        if (!color[v]) {
            Prev[v] = u;
            Depth[v] = Depth[u] + 1;
            DFS(v);
        }
    }
    Stop[u] = id;
}

int AddNode()
{
    ++nT; return nT;
}

void InitTree(int r)
{
    int k = Tree[r].idmin, l = Tree[r].idmax;
    if (k == l) {
        Tree[r].val = 0; return;
    }
    int g = (k + l) / 2;
    int rLeft = AddNode(), rRight = AddNode();
    Tree[r].L = rLeft, Tree[r].R = rRight;
    Tree[rLeft].idmin = k; Tree[rRight].idmin = g + 1;
    Tree[rLeft].idmax = g; Tree[rRight].idmax = l;
    InitTree(rLeft); InitTree(rRight);
    Tree[r].val = Tree[rLeft].val + Tree[rRight].val;
}

void DownTree(int r)
{
    int rL = Tree[r].L, rR = Tree[r].R;
    Tree[rL].delta += Tree[r].delta;
    Tree[rR].delta += Tree[r].delta;
    Tree[r].delta = 0;
}

void UpTree(int r)
{
    int rL = Tree[r].L, rR = Tree[r].R;
    int Left = Tree[rL].val + Tree[rL].delta * (Tree[rL].idmax - Tree[rL].idmin + 1);
    int Right = Tree[rR].val + Tree[rR].delta * (Tree[rR].idmax - Tree[rR].idmin + 1);
    Tree[r].val = Left + Right;
}

void Update(int r, int u, int v, int val)

```

```

{
    int k = Tree[r].idmin, l = Tree[r].idmax;
    if (v < k || l < u) return;
    if (u <= k && l <= v) { Tree[r].delta += val; return; }
    DownTree(r);
    Update(Tree[r].L, u, v, val);
    Update(Tree[r].R, u, v, val);
    UpTree(r);
}

int Get(int r, int u)
{
    int k = Tree[r].idmin, l = Tree[r].idmax;
    if (u < k || l < u) return 0;
    if (u <= k && l <= u) return Tree[r].val + Tree[r].delta * (l - k + 1);
    DownTree(r);
    int t1 = Get(Tree[r].L, u);
    int t2 = Get(Tree[r].R, u);
    UpTree(r);
    return t1 + t2;
}

void Solve()
{
    DFS(1);
    int root = AddNode();
    Tree[root].idmin = 1; Tree[root].idmax = N;
    InitTree(root);
    for (int i = 1; i <= N; ++i)
        Update(root, Start[i], Start[i], Depth[i]);
    char ch; int u, v;
    while (M--) {
        ch = getchar();
        if (ch == 'W') {
            u = ReadInt();
            WriteInt(Get(root, Start[u])), putchar('\n');
        }
        else {
            u = ReadInt(); v = ReadInt();
            if (Prev[v] == u) swap(u, v);
            Update(root, Start[u], Stop[u], -1);
        }
    }
}

void Output()
{

}

int main()
{
    #define TASK "BRBUILD"
    #ifdef TASK
    freopen(TASK".INP", "r", stdin);
    freopen(TASK".OUT", "w", stdout);
    #else

```

```

freopen("INPUT.INP", "r", stdin);
#endif // TASK
Input();
Solve();
Output();
}

```

Bài 4: Giấy mơ băng [PENGUIN.*]

(Bài toán từ COCI)

Link tests:

https://drive.google.com/file/d/1nhaDhWdYV9W8_3NIqizpaAIQqpovId--/view?usp=sharing

Trước đây Mirko đã sáng lập một đại lý du lịch tên là "Giấc mơ băng". Đại lý đã mua N hòn đảo phủ băng gần Nam Cực và bây giờ đang tổ chức các cuộc thám hiểm. Đặc biệt nổi tiếng ở đây là loài chim cánh cụt hoàng đế, có thể được tìm thấy với số lượng lớn trên các hòn đảo.

Đại lý của Mirko's đã trở nên cực kỳ thành công; cực kỳ lớn đến nỗi nó không còn hiệu quả cho việc sử dụng thuyền để thám hiểm. Đại lý sẽ phải xây các cây cầu giữa các hòn đảo và vận chuyển hành khách bằng xe buýt. Mirko muốn giới thiệu một chương trình máy tính để quản lý việc xây cầu sao cho có ít sai sót nhất.

Các hòn đảo được đánh số từ 1 đến N. Ban đầu không có hai hòn đảo nào được nối bằng cầu. Số lượng chim cánh cụt ban đầu trên mỗi đảo cũng được cho biết. Số lượng này có thể thay đổi sau này, nhưng luôn luôn nằm trong khoảng từ 0 đến 1000.

Chương trình của bạn phải thực hiện 3 loại lệnh sau:

1. "bridge A B" – một đề nghị yêu cầu xây cầu giữa hai đảo A và B (A và B phải khác nhau). Để giới hạn chi phí, chương trình của bạn chỉ được chấp nhận đề nghị này nếu như chưa tồn tại một cách đi từ một trong 2 đảo đến đảo còn lại mà sử dụng các cây cầu đã xây trước đó. Nếu đề nghị được chấp nhận, chương trình phải in ra "yes", và sau đó cây cầu được xây. Nếu đề nghị bị từ chối, chương trình phải in ra "no".
2. "penguins A X" – các chú chim cánh cụt trên đảo A đã được đếm lại và bây giờ có X con. Đây chỉ là một lệnh có tính chất thông báo và chương trình không cần trả lời.
3. "excursion A B" – một nhóm khách du lịch muốn tổ chức một cuộc thám hiểm từ đảo A đến đảo B. Nếu có thể tổ chức được (có thể đi từ A đến B qua các cầu đã xây), chương trình phải in ra tổng số lượng chim cánh cụt mà nhóm khách du lịch sẽ thấy trên hành trình (kể cả ở đảo A và B). Ngược lại, chương trình phải in ra "impossible".

Input:

- Dòng đầu tiên chứa số nguyên N ($1 \leq N \leq 30\,000$), số lượng đảo.
- Dòng thứ hai chứa N số nguyên nằm trong khoảng từ 0 đến 1000, số lượng chim cánh cụt ban đầu ở mỗi đảo.
- Dòng thứ ba chứa số nguyên Q ($1 \leq Q \leq 300\,000$), số lượng câu lệnh.
- Q lệnh theo sau đó, mỗi lệnh trên 1 dòng.

Output: Trả lời cho các câu lệnh "bridge" và "excursion", mỗi câu trên 1 dòng.

Example:

input	output
5	4
4 2 4 5 6	impossible
10	yes
excursion 1 1	6
excursion 1 2	yes
bridge 1 2	yes
excursion 1 2	15
bridge 3 4	yes
bridge 3 5	15
excursion 4 5	16

bridge 1 3	
excursion 2 4	
excursion 2 5	

Thuật toán:

Xây dựng đồ thị với mỗi hòn đảo là một đỉnh. Khởi đầu không có các cạnh nào.

- +) B1: Xử lý các truy vấn dạng **bridge...** : Dùng kỹ thuật Dijoint set nhập các thành phần liên thông. Nếu lệnh brige... tương ứng nối hai thành phần liên thông ta có câu trả lời 'yes' và thêm một cạnh vào đồ thị với trọng số là số hiệu của lệnh bridge.. tương ứng. Kết thúc việc xử lý bridge... ta có được một "rừng" các cây.
- +) B2: Thêm các cạnh giả (với trọng số Q+1) vào để nhập các rừng thành một cây.
- +) B3: Thực hiện phân rã HLD ta có được thứ tự DFS trên HLD (mảng pos, head)
- +) B4: Xây dựng cấu trúc IT tổng với mỗi lá là một đỉnh của đồ thị. Khi đó truy vấn pengui... là cập nhật lại giá trị của lá, còn truy vấn excursion... là tính tổng trên đường đi đơn giữa hai đỉnh (chú ý làm lại B1 và việc tính tổng chỉ thực hiện khi hai đỉnh thuộc cùng một thành phần liên thông)

Chương trình tham khảo:

```
#include <iostream>
#include <stdio.h>
#include <cstdio>
#include <math.h>
#include <algorithm>
#include <vector>
#include <string.h>

using namespace std;

const int maxn = 300005;
const int oo = 2e9;

int N, Q = 0, id = 0, nT = 0, root;
char type[maxn], s[20]; bool color[maxn], Mark[maxn];
int Depth_DFS[maxn], Depth_HLD[maxn], deg[maxn];
int Child[maxn], Prev[maxn], LT[maxn], Pos[maxn];
int A[maxn], B[maxn], Amount_Penguins[maxn], Head[maxn];
int it[maxn * 4], d[maxn * 4];
vector<int> g[maxn];

int ReadInt()
{
    char c;
    for( c = getchar() ; ( c < '0' || c > '9') && c != '-' ; c =getchar());
    int ans;
    ans = (c >= '0' && c <= '9') ? c - '0' : 0;
    for( c = getchar() ; c >= '0' && c <= '9' ; c =getchar()) ans = ans * 10 + c - '0';
;
    return ans;
}

void WriteInt(int x)
{
    if (x < 0) putchar('-'), x = -x;
    if (x > 9) WriteInt(x / 10);
```

```

    putchar(x % 10 + '0');
}

int FindRoot(int u)
{
    if (LT[u] == u) return LT[u];
    LT[u] = FindRoot(LT[u]);
    return LT[u];
}

void Input()
{
    N = ReadInt();
    for (int i = 1; i <= N; ++i) Amount_Penguins[i] = ReadInt(), LT[i] = i;
    Q = ReadInt();
    for (int i = 1; i <= Q; ++i) {
        scanf("%s", s); A[i] = ReadInt(); B[i] = ReadInt();
        type[i] = s[0];
        if (type[i] == 'b') {
            int u = A[i], v = B[i];
            u = FindRoot(u), v = FindRoot(v);
            if (u != v) {
                LT[u] = LT[v]; Mark[i] = true;
                g[A[i]].push_back(B[i]); ++deg[A[i]];
                g[B[i]].push_back(A[i]); ++deg[B[i]];
            }
        }
        else if (type[i] == 'e') {
            if (FindRoot(A[i]) != FindRoot(B[i])) Mark[i] = true;
        }
    }
}

void DFS(int u)
{
    color[u] = true; Child[u] = 1;
    int maxChild = 0, imax = -1;
    for (int i = 0; i < deg[u]; ++i) {
        int v = g[u][i];
        if (!color[v]) {
            Prev[v] = u; Depth_DFS[v] = Depth_DFS[u] + 1;
            DFS(v); Child[u] += Child[v];
            if (maxChild < Child[v]) maxChild = Child[v], imax = i;
        }
    }
    if (imax >= 0) swap(g[u][0], g[u][imax]);
}

void HLD(int u)
{
    Pos[u] = ++id;
    for (int i = 0; i < deg[u]; ++i) {
        int v = g[u][i];
        if (Prev[v] == u) {
            Head[v] = (i == 0 && 2 * Child[v] >= Child[u]) ? Head[u] : v;
            if (Head[v] == v) Depth_HLD[v] = Depth_HLD[u] + 1;
            else Depth_HLD[v] = Depth_HLD[Head[v]];
        }
    }
}

```

```

        HLD(v);
    }
}
}

void Update(int r, int k, int l, int u, int v, int val)
{
    if (l < u || v < k) return ;
    if (u <= k && l <= v) {
        it[r] = val; d[r] = val; return;
    }
    if (d[r]) {
        d[r << 1] = d[r];
        d[r << 1 | 1] = d[r];
        it[r << 1] = d[r];
        it[r << 1 | 1] = d[r];
        d[r] = 0;
    }
    int g = (k + 1) / 2;
    Update(r << 1, k, g, u, v, val);
    Update(r << 1 | 1, g + 1, l, u, v, val);
    it[r] = it[r << 1] + it[r << 1 | 1];
}

int Get(int r, int k, int l, int u, int v)
{
    if (l < u || v < k) return 0;
    if (u <= k && l <= v) return it[r];
    if (d[r]) {
        d[r << 1] = d[r];
        d[r << 1 | 1] = d[r];
        it[r << 1] = d[r];
        it[r << 1 | 1] = d[r];
        d[r] = 0;
    }
    int g = (k + 1) / 2;
    return Get(r << 1, k, g, u, v) + Get(r << 1 | 1, g + 1, l, u, v);
}

int Amount_Penguins_on(int u, int v)
{
    int ans = 0;
    if (Depth_HLD[u] < Depth_HLD[v]) swap(u, v);
    while(Depth_HLD[u] > Depth_HLD[v]) {
        ans += Get(1, 1, N, Pos[Head[u]], Pos[u]);
        u = Prev[Head[u]];
    }

    while (Head[u] != Head[v]) {
        ans += Get(1, 1, N, Pos[Head[u]], Pos[u]);
        ans += Get(1, 1, N, Pos[Head[v]], Pos[v]);
        u = Prev[Head[u]], v = Prev[Head[v]];
    }

    if (Depth_DFS[u] < Depth_DFS[v]) swap(u, v);
    ans += Get(1, 1, N, Pos[v], Pos[u]);
    return ans;
}

```

```

}

void Solve()
{
    for (int i = 1; i <= N; ++i) if (!color[i]) DFS(i), HLD(i);

    for (int i = 1; i <= N; ++i)
        Update(1, 1, N, Pos[i], Pos[i], Amount_Penguins[i]);

    for (int i = 1; i <= Q; ++i) {
        if (type[i] == 'b') {
            if (Mark[i]) puts("yes"); else puts("no");
        }
        else if (type[i] == 'p') Update(1, 1, N, Pos[A[i]], Pos[A[i]], B[i]);
        else {
            if (Mark[i]) puts("impossible");
            else WriteInt(Amount_Penguins_on(A[i], B[i])), putchar('\n');
        }
    }
}

void Output()
{

}

int main()
{
    #define TASK "PENGUIN"
    #ifdef TASK
    freopen(TASK".INP", "r", stdin);
    freopen(TASK".OUT", "w", stdout);
    #else
    freopen("INPUT.INP", "r", stdin);
    #endif // TASK
    Input();
    Solve();
    Output();
}

```

Bài 5. Công viên Rồng [DRAGONPARK.*]

(Bài từ USACO, có sửa đổi background)

Link tests:

https://drive.google.com/file/d/16ycHkfjPKHdsiZSoZ_0omKbEchq0jKK/view?usp=sharing

Công viên Rồng (Dragon Park) ở thành phố Hạ Long là một trong những điểm vui chơi nổi tiếng ở miền Bắc. Tại đây khách du lịch có thể tham gia nhiều trò chơi khác nhau, trong đó có nhiều trò chơi mạo hiểm chỉ có duy nhất trong cả nước. Mỗi trò chơi như vậy được tổ chức tại một địa điểm trong công viên.

Có tổng cộng N địa điểm vui chơi khác nhau ($2 \leq N \leq 10^5$), đánh số 1, 2, ..., N , được kết nối với nhau bởi $N - 1$ con đường hai chiều sao cho bằng những con đường này thì từ một điểm vui chơi bất kỳ luôn có thể đi đến được điểm vui chơi khác. Điểm vui chơi i được đánh giá mức độ hấp dẫn bằng một số nguyên e_i ($i = 1 \div N$) tuy nhiên giá trị này có thể thay đổi trong ngày tùy theo từng thời điểm.

Một du khách đi từ điểm vui chơi i đến điểm vui chơi j sẽ trải nghiệm tất cả các trò chơi trên tuyến đường đơn từ i đến j (tuyến đường đơn là tuyến đường mà mỗi điểm vui chơi đi qua không quá 1 lần) và thật kỳ lạ, cảm giác hấp dẫn của du khách sẽ bằng tổng XOR của mức độ hấp dẫn của tất cả các điểm vui chơi mà anh (cô) ta đi qua.

Viết chương trình xác định cảm giác hấp dẫn của du khách trên mỗi hành trình của anh (cô) ta?.

Dữ liệu: Vào từ file văn bản DRAGONPARK.INP:

- Dòng đầu ghi hai số nguyên dương N - số điểm vui chơi và Q ($1 \leq Q \leq 10^5$) - số truy vấn cần thực hiện.
- Dòng thứ hai ghi N số nguyên e_1, e_2, \dots, e_N ($0 \leq e_i \leq 10^9$) là mức độ hấp dẫn ban đầu của các điểm vui chơi.
- N dòng tiếp theo, mỗi dòng chứa hai số nguyên a, b thể hiện có một con đường hai chiều nối điểm vui chơi a với điểm vui chơi b ($1 \leq a, b \leq N; a \neq b$)
- Cuối cùng là Q dòng, mỗi dòng mô tả một truy vấn thực hiện lần lượt thuộc một trong hai loại:
 - **1 i v:** Điểm vui chơi i sẽ có độ hấp dẫn mới là v
 - **2 i j:** Xác định cảm giác hấp dẫn của du khách trên hành trình từ điểm vui chơi i đến điểm vui chơi j .

Kết quả: Ghi ra file văn bản DRAGONPARK.OUT:

Với các truy vấn **2 i j** in ra kết quả tìm được trên một dòng (theo thứ tự trong file dữ liệu)

Ví dụ:

DRAGONPARK . INP	DRAGONPARK . OUT
5 5	21
1 2 4 8 16	20
1 2	4
1 3	20
3 4	
3 5	
2 1 5	
1 1 16	
2 3 5	
2 1 5	
2 1 3	

Thuật toán:

Xây dựng đồ thi với mỗi điểm vui chơi là một đỉnh còn các con đường hai chiều là một cạnh ta có được mô hình một cây với mỗi đỉnh được gán một trọng số.

Xây dựng IT quản lý các đoạn với giá trị $it[r]$ là tổng XOR của tất cả các lá trong cây. Ta có công thức

$$it[r] = it[2*r] \text{ XOR } it[2*r+1]$$

Ngoài ra do $a \text{ XOR } 0 = a$ nên việc quản lý IT này hoàn toàn tương tự như quản lý tổng.

Do vậy bài toán trở thành lập phân rã HLD kết hợp với IT nói trên để xử lý XOR (tương tự như xử lý phép cộng) với chú ý rằng các lá của IT là các đỉnh của đồ thị

Chương trình tham khảo:

```
#include <bits/stdc++.h>
#define maxn 100001

using namespace std;

int n, e[maxn];
vector<int> g[maxn];
int Pd[maxn], pos[maxn], head[maxn], d_HLD[maxn];
```

```

int it[5*maxn];

int cl[maxn], s[maxn];
void DFS(int u) {
    cl[u]=1; s[u]=1;
    int sz=g[u].size();
    int imax=0, smax=0;
    for(int i=0;i<sz;++i) {
        int v=g[u][i];
        if (cl[v]==0) {
            Pd[v]=u;
            DFS(v);
            s[u] += s[v];
            if (smax<s[v]) {
                smax=s[v];
                imax=i;
            }
        }
    }
    swap(g[u][imax],g[u][0]);
}

int id;
void HLD(int u) {
    pos[u]=++id;
    for(auto &v : g[u]) if (Pd[v]==u) {
        if (2*s[v]>=s[u]) head[v]=head[u], d_HLD[v]=d_HLD[u];
        else head[v]=v, d_HLD[v]=d_HLD[u]+1;
        HLD(v);
    }
}

void update(int r,int k,int l,int u,int val) {
    if (u<k || u>l) return;
    if (u<=k && l<=u) {it[r]=val; return;}
    int g=(k+1)/2;
    update(2*r,k,g,u,val);
    update(2*r+1,g+1,l,u,val);
    it[r]=it[2*r] ^ it[2*r+1];
}

int get(int r,int k,int l,int u,int v) {
    if (v<k || u>l) return 0;
    if (u<=k && l<=v) return it[r];
    int g=(k+1)/2;
    int tL=get(2*r,k,g,u,v);
    int tR=get(2*r+1,g+1,l,u,v);
    return tL ^ tR;
}

void Xuly1() {
    int i, v; scanf("%d %d", &i, &v);
    int u=pos[i];
    update(1,1,n,u,v);
}

int calc(int u,int v) {
    int res=0;
    while (d_HLD[u]>d_HLD[v]) {
        res = res ^ get(1,1,n,pos[head[u]],pos[u]);
        u=Pd[head[u]];
    }
    while (d_HLD[v]>d_HLD[u]) {
        res = res ^ get(1,1,n,pos[head[v]],pos[v]);
        v=Pd[head[v]];
    }
}

```

```

    }
    while (head[u]!=head[v]) {
        res = res ^ get(1,1,n,pos[head[u]],pos[u]);
        res = res ^ get(1,1,n,pos[head[v]],pos[v]);
        u=Pd[head[u]]; v=Pd[head[v]];
    }
    if (pos[u]<pos[v]) res = res ^ get(1,1,n,pos[u],pos[v]);
    else res = res ^ get(1,1,n,pos[v],pos[u]);
    return res;
}

void Xuly2() {
    int u, v; scanf("%d %d", &u, &v);
    printf("%d\n", calc(u,v));
}

int main() {
    freopen("DRAGONPARK.inp","r",stdin);
    freopen("DRAGONPARK.out","w",stdout);
    int Q;
    scanf("%d %d", &n, &Q);
    for(int i=1;i<=n;++i) scanf("%d", &e[i]);
    for(int i=1;i<n;++i) {
        int u, v; scanf("%d %d", &u, &v);
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for(int i=1;i<=n;++i) cl[i]=0;
    DFS(1);
    id=0; HLD(1);
    for(int i=1;i<=n;++i) {
        int u=pos[i];
        update(1,1,n,u,e[i]);
    }
    while (Q--) {
        int loai; scanf("%d", &loai);
        if (loai==1) Xuly1();
        else Xuly2();
    }
}

```

V-Kết luận

Chuyên đề này được viết lại dựa trên các bài giảng của tôi khi dạy cho học sinh về cấu trúc cây và các bài toán trên cấu trúc cây. Khi giảng dạy, điều đặc biệt quan trọng là cách thức xây dựng thứ tự DFS trên cây HLD. Đây là điểm mấu chốt để có thể áp dụng các cấu trúc dữ liệu đặc biệt (IT, BIT...).

Về hệ thống bài tập, trong phạm vi của chuyên đề tôi chỉ đưa ra các bài toán điển hình với hy vọng rằng khi học sinh code được các bài tập này, các em có thể dễ dàng sử dụng HLD cho các bài tập tương tự.

Tất nhiên chuyên đề này còn có nhiều sai sót. Rất mong các thầy cô cho ý kiến đánh giá để có thể hoàn thiện tốt hơn!