

**HỘI THẢO KHOA HỌC  
CÁC TRƯỜNG THPT CHUYÊN  
KHU VỰC DUYÊN HẢI VÀ ĐỒNG BẰNG BẮC BỘ  
NĂM 2020**

**Môn: Tin học**

**Bài toán: “Tổ tiên chung gần nhất”-LCA**

**Người viết: Nguyễn Như Thắng**

**Đơn vị công tác: Trường THPT Chuyên tỉnh Lào Cai**

**Tháng 9/2020**

## **MỤC LỤC**

Bảng chú thích một số tên, thuật ngữ viết tắt .....	4
1. Mở đầu .....	5
2. Một số khái niệm, kiến thức cơ bản.....	5
3. Dạng bài toán nào có thể cần đến LCA .....	6
4. Các phương pháp giải bài toán LCA .....	6
4.1. Duyệt tham lam.....	7
4.2. Chia căn .....	8
4.3. Kỹ thuật bảng thưa (Sparse table) .....	10
4.4. Dùng Euler tour .....	11
4.5. Xử lý kiểu Off-line (Tarjan's off-line LCA).....	12
5. Một số bài tập ví dụ .....	13
5.1. Bài 1: Bài tập cơ bản.....	14
5.1.1. Đề bài: LCA.....	14
5.1.2. Phân tích đề bài và đề xuất thuật toán .....	14
5.1.3. Test kèm theo.....	15
5.2. Bài 2: Tổ chức thi chạy Marathon .....	15
5.2.1. Đề bài: Marathon .....	15
5.2.2. Phân tích đề bài và đề xuất thuật toán .....	16
5.2.3. Test kèm theo.....	17
5.3. Bài 3: Du lịch thành phố (NAIPC 2016) .....	17
5.3.1. Đề bài: Tourist .....	17
5.3.2. Phân tích đề bài và đề xuất thuật toán .....	18
5.3.3. Test kèm theo.....	19
5.4. Bài 4: VOTREE (VNOI online 2015) .....	19
5.4.1. Đề bài.....	19
5.4.2. Phân tích đề bài và đề xuất thuật toán .....	20
5.4.3. Test kèm theo.....	22
5.5. Bài 5: Tăng lương (Chọn đội tuyển IOI CROATIAN 2010) .....	22
5.5.1. Đề bài POVISICE.....	22
5.5.2. Phân tích đề bài và đề xuất thuật toán .....	23
5.5.3. Test kèm theo.....	26
5.6. Bài 6: Nâng cấp mạng (VOI 2011).....	26
5.6.1. Đề bài: UPGRANET .....	26
5.6.2. Phân tích đề bài và đề xuất thuật toán .....	27
5.6.3. Test kèm theo.....	29
5.7. Bài 7: Đạo chơi đồng cỏ (PWALK – Spoj).....	29
5.7.1. Đề bài: PWALK .....	29
5.7.2. Phân tích đề bài và đề xuất thuật toán .....	30

5.7.3. Test kèm theo.....	31
5.8. Bài 8: TREEDGE .....	31
5.8.1. Đề bài TREEDGE .....	31
5.8.2. Phân tích đề bài và đề xuất thuật toán .....	32
5.8.3. Test kèm theo.....	34
5.9. Bài 9: Đường đi qua K cạnh .....	34
5.9.1. Đề bài: TREEQ.....	34
5.9.2. Phân tích đề bài và đề xuất thuật toán .....	34
5.9.3. Test kèm theo.....	36
5.10. Bài 10: Tom & Jerry .....	36
5.10.1. Đề bài.....	36
5.10.2. Phân tích đề bài và đề xuất thuật toán .....	37
5.10.3. Test kèm theo.....	40
5.11. Bài 11: Cập nhật thông tin trên cây 1 .....	40
5.11.1. Đề bài: Update tree .....	40
5.11.2. Phân tích đề bài và đề xuất thuật toán .....	40
5.11.3. Test kèm theo.....	42
5.12. Bài 12: Cập nhật thông tin trên cây 2 .....	42
5.12.1. Đề bài: Update tree2 .....	42
5.12.2. Phân tích đề bài và đề xuất thuật toán .....	43
5.12.3. Test kèm theo.....	46
5.13. Bài 13: Dạo chơi trên cây .....	46
5.13.1. Đề bài Walking.....	46
5.13.2. Phân tích đề bài và đề xuất thuật toán .....	47
5.13.3. Test kèm theo.....	49
5.14. Bài 14: Cây đôi gốc .....	49
5.14.1. Đề bài: ROOTLESS .....	49
5.14.2. Phân tích đề bài và đề xuất thuật toán .....	50
5.14.3. Test kèm theo.....	51
5.15. Bài 15: Cây đôi gốc 2 .....	51
5.15.1. Đề bài: Rootless2.....	51
5.15.2. Phân tích đề bài và đề xuất thuật toán .....	52
5.15.3. Test kèm theo.....	56
6. Một số bài tập tự luyện .....	56
7. Kết luận.....	56
8. Tài liệu tham khảo .....	57

**Bảng chú thích một số tên, thuật ngữ viết tắt**

<b>Thuật ngữ, tên viết tắt</b>	<b>Giải thích</b>
<i>adj</i>	Kề (adjacent)
<i>ancestor</i>	Tổ tiên
<i>Binary lifting</i>	Nâng nhị phân, nhảy nhị phân
<i>BIT</i>	Cây chỉ số nhị phân (Binary index tree)
<i>BFS</i>	Duyệt đồ thị theo chiều rộng
<i>Cây con gốc u</i>	Là bộ phận của cây ban đầu trong đó các đỉnh được gọi đến từ nó trong duyệt DFS(u)
<i>depth</i>	độ sâu
<i>DFS</i>	Duyệt đồ thị theo chiều sâu (Depth first search)
<i>dist</i>	Khoảng cách (distance)
<i>DSU</i>	Tập hợp không giao nhau (Disjoint set union)
<i>LCA</i>	Tổ tiên chung gần nhất (Lowest ancestor common)
<i>par</i>	Cha (parents)
<i>root</i>	Gốc của cây, là đỉnh được duyệt đầu tiên trên cây khi gọi DFS.
<i>Segment tree</i>	Cây phân đoạn (cây quản lý đoạn)
<i>Sparse table</i>	Bảng thưa

## 1. Mở đầu

Dạng bài về “Tổ tiên chung gần nhất” cũng khá phổ biến, đây đều là các dạng bài không dễ, đòi hỏi học sinh có tư duy khá, nhiều bài đòi hỏi học sinh sáng tạo mới vận dụng được.

Học sinh cần có một số kiến thức để đảm bảo học được chuyên đề này là: cơ bản về phương pháp Quy hoạch động trên cây; Đồ thị cơ bản, duyệt đồ thị; kĩ thuật bảng thưa (Sparse table), cấu trúc dữ liệu (Segment tree, BIT, DSU).

Trong quá trình dạy đội tuyển lớp HSG lớp 11, đội tuyển HSG Quốc gia. Từ các bài toán dạng này, cũng cho học sinh ôn lại các kiến thức liên quan khác để giải bài toán LCA như: cấu trúc dữ liệu sparse table, segment tree,...; ôn lại bài toán RMQ; kĩ thuật Heavy light decomposition; kĩ thuật duỗi cây thành mảng (Euler tour); duyệt đồ thị; Quy hoạch động trên cây;...

## 2. Một số khái niệm, kiến thức cơ bản

- Cây DFS: Quá trình duyệt đồ thị theo chiều sâu (DFS) bắt đầu từ đỉnh  $s$  cho ta một cây DFS gốc  $s$ .

- Quan hệ cha-con trên cây DFS: Khi duyệt DFS, nếu từ đỉnh  $u$  ta gọi hàm tới thăm đỉnh  $v$  thì đỉnh  $u$  là đỉnh cha đỉnh  $v$ . Ví dụ: Đỉnh 1 là cha của các đỉnh 2,3,4. Đỉnh 12,13 là con của đỉnh 10.

- Quan hệ tổ tiên-con cháu: Được định nghĩa đệ quy như sau:

- + Đỉnh  $u$  là tổ tiên của chính nó.
- + Đỉnh cha của đỉnh  $u$  là tổ tiên của đỉnh  $u$ .
- + Cha của tổ tiên của  $u$  cũng là tổ tiên của  $u$ .

Ta thấy, đỉnh  $u$  là tổ tiên của tất cả các đỉnh trong nhánh cây DFS gốc  $u$ .

Ví dụ:

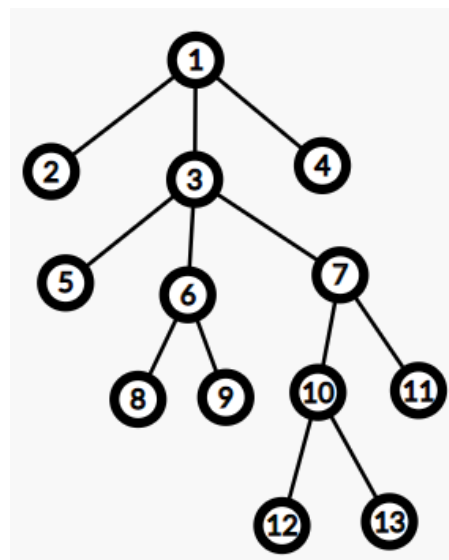
Đỉnh 7 là tổ tiên của các đỉnh 7,10,11,12,13 và cả chính nó.

Đỉnh 6 là tổ tiên của đỉnh 6,8,9.

- Độ sâu: Là khoảng cách đến đỉnh gốc (được tính bằng số đỉnh, hoặc số cạnh, không phải là trọng số của cạnh). Đỉnh gốc của cây DFS quy ước độ sâu là 1. Ví dụ: Đỉnh 3 có độ sâu là 2; đỉnh 12,13 có độ sâu là 5.

- Tổ tiên chung: Đỉnh  $w$  là tổ tiên của đỉnh  $u$  và  $v$  thì  $w$  được gọi là tổ tiên chung của  $u$  và  $v$ . Ví dụ: Đỉnh 3 là tổ tiên chung của đỉnh 12 và đỉnh 13.

- Định nghĩa tổ tiên chung gần nhất (LCA) trên wiki: Trong lý thuyết đồ thị và khoa học máy tính, LCA của 2 đỉnh  $u, v$  trên cây hoặc đồ thị có hướng không chu trình (DAG) gốc  $T$  là đỉnh  $w$  sâu nhất (hay đỉnh xa gốc nhất) mà nhận cả  $u, v$  làm con cháu, chúng ta coi một đỉnh cũng chính là tổ tiên của chính nó. Ví dụ: Đỉnh 3 là tổ tiên chung gần nhất của đỉnh 7 và 9. Đỉnh 10 là tổ tiên chung gần nhất của 12,13.



- Nhận xét thấy  $LCA(u, v)$  là đỉnh đầu tiên gặp nhau của đường đi từ  $u$  về gốc và đường đi từ  $v$  về gốc. Từ nhận xét này ta sẽ hình thành các cách giải bài toán LCA.

### 3. Dạng bài toán nào có thể cần đến LCA

***Trong chuyên đề này, tôi tập trung vào các dạng bài sau:***

- Các bài tập về tìm kiếm, cập nhật thông tin của các đỉnh nằm trên đường đi đơn từ  $u$  đến  $v$ . Ví dụ như tính khoảng cách giữa các cặp đỉnh trên cây (cây có trọng số, không có trọng số).

- Dạng bài LCA kết hợp cấu trúc dữ liệu như: Segment tree, Binay index tree, Disjoint set union,....

- Dạng bài LCA kết hợp với quy hoạch động trên cây, cây khung, cầu-khớp,...

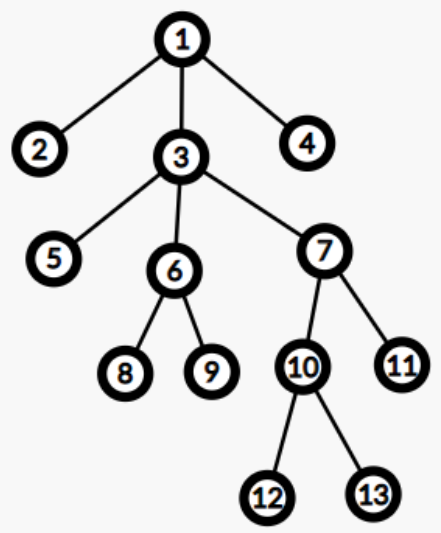
- Dạng bài LCA có đổi gốc.

- Đánh dấu, cộng dồn trên cây có áp dụng LCA.

Theo nhận định của tác giả thì các dạng bài khó sẽ là dạng bài phải biết kết hợp thêm các dạng bài khác về cây, kết hợp cấu trúc dữ liệu; các bài cần học sinh có sự sáng tạo mới phát hiện ra cần áp dụng dạng bài toán LCA như thế nào,...

### 4. Các phương pháp giải bài toán LCA

Để trình bày một số phương pháp giải bài toán LCA, tác giả đưa ra ví dụ cây có gốc là 1 như dữ liệu cho sau:

Dữ liệu vào	Kết quả ra	Giải thích
13	$lca(2,4)=1$	 <p>Cho đồ thị gồm 13 đỉnh, 12 cạnh, và 6 câu hỏi truy vấn tìm LCA.</p>
1 2	$lca(12,13)=10$	
1 3	$lca(2,5)=1$	
1 4	$lca(8,9)=6$	
3 5	$lca(12,11)=7$	
3 6	$lca(6,7)=3$	
3 7		
6 8		
6 9		
7 10		
7 11		
10 12		
10 13		
6		
2 4		
12 13		
2 5		
8 9		
12 11		
6 7		

#### 4.1. Duyệt tham lam

Nhận xét rằng để tìm tổ tiên chung gần nhất của 2 đỉnh  $u, v$  thì từ 2 đỉnh này, ta đi lên từng bước một về phía gốc cây. Đến vị trí gặp nhau đầu tiên thì đó chính là tổ tiên chung gần nhất.

##### Phương pháp:

Bước 1: Di chuyển đỉnh có độ sâu lớn hơn đến khi 2 đỉnh có cùng độ sâu.

Bước 2: Nếu 2 đỉnh chưa gặp nhau thì ta cùng di chuyển chúng đến khi gặp nhau thì lập tức dừng lại, đó chính là LCA của chúng.

Các làm này khá lâu nếu trong trường hợp cây DFS suy biến thành dạng đường thẳng.

Đánh giá độ phức tạp của thuật toán:

- Độ phức tạp tiền xử lí (duyet DFS):  $O(N)$ .

- Độ phức tạp của một truy vấn là:  $O(N)$ .

➔ Độ phức tạp thuật toán chung:  $O(N * Q)$ .

Chương trình tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
int n, q, par[100005], depth[100005];
vector<int> adj[100005];
//duyet DFS de xac dinh do xau va tim cha cua cac dinh
void dfs(int u, int p, int d) {
    depth[u] = d;
    par[u] = p;
    for(int v : adj[u]) {
        if(par[u] == v)
            continue;
        dfs(v, u, d + 1);
    }
}

int lca(int u, int v) { //Tim LCA theo kieu Brute force
    if(depth[u] < depth[v])
        swap(u, v);
    while(depth[u] > depth[v])
        u = par[u];
    while(u != v) {
        u = par[u];
        v = par[v];
    }
    return u;
}

int main() {
    cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
```

```
dfs(1, 0, 1);
cin>>q;
while(q--){
    int u,v;
    cin>>u>>v;
    cout<<"lca("<<u<<" "<<v<<"")="<<lca(u,v)<<"\n";
}
}
```

## 4.2. Chia căn

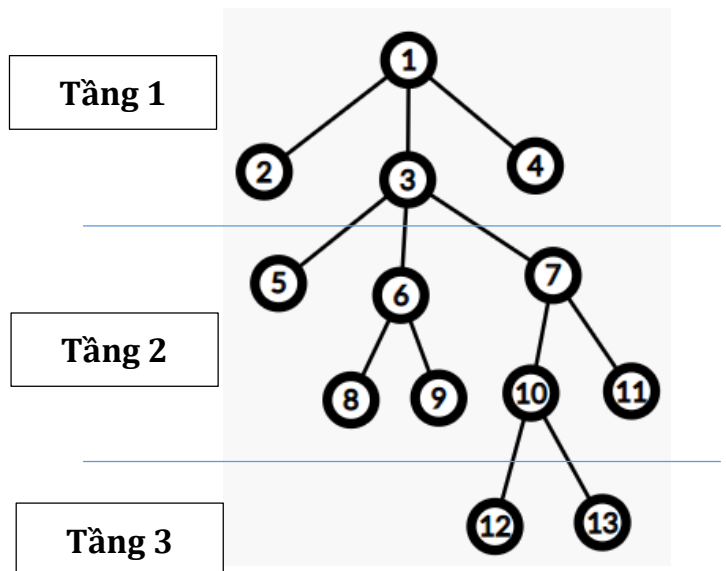
Nhận xét rằng cách làm Brute Force trên khá lâu vì mỗi lần ta chỉ đi lên được một đỉnh. Trong cách làm sau đây, ta tiền xử lí bằng cách chia cây có độ cao  $H$  thành các  $\sqrt{H}$  tầng. Đỉnh có độ sâu  $depth[u]$  thì được xếp vào tầng  $\frac{depth[u]+1}{\sqrt{H}}$ .

Có thể chọn  $H = N$ . Khi đó nếu  $u, v$  chưa cùng tầng thì ta nhảy theo tầng, nếu cùng tầng thì ta nhảy theo cha của nó để đến khi gặp nhau.

Đánh giá độ phức tạp của thuật toán:

- Độ phức tạp tiền xử lí:  $O(N * \sqrt{N})$ .
- Độ phức tạp của một truy vấn là:  $O(\sqrt{N})$ .
- ➔ Độ phức tạp thuật toán chung:  $O(N * \sqrt{N} + Q * \sqrt{N})$ .

Ví dụ:  $H=5$  và  $\lceil \sqrt{H} \rceil = 2$



## Chương trình tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
int n, q, H, s, par[100005], depth[100005], T[100005];
vector<int> adj[100005];
///duyet DFS de xac dinh do xau va tim cha cua cac dinh
void dfs0(int u, int p, int d) {
    depth[u] = d;
    H = max(H, d);
    par[u] = p;
    for(int v : adj[u]) {
        if(par[u] == v)
```



```
        continue;
        dfs0(v, u, d + 1);
    }
}

void dfs(int u, int p, int d) { ///tim to tien o tang tren cua u
    depth[u] = d;
    par[u] = p;
    if(depth[u] < s)
        T[u] = 1;
    else
        if((depth[u] + 1) % s)
            T[u] = T[par[u]];
        else
            T[u] = par[u];
    for(int v : adj[u]) {
        if(par[u] == v)
            continue;
        dfs(v, u, d + 1);
    }
}

int lca(int u, int v) { ///Tim LCA chia can
    while(T[u] != T[v])
        if(depth[u] > depth[v])
            u = T[u];
        else
            v = T[v];
    while(u != v) {
        if(depth[u] > depth[v])
            u = par[u];
        else
            v = par[v];
    }
    return u;
}

int main() {
    //freopen("LCA_sqrt.inp", "r", stdin);
    cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    dfs0(1, 0, 1);
    s = sqrt(H);
    dfs(1, 0, 1);
    cin >> q;
    while(q--) {
        int u, v;
        cin >> u >> v;
        cout << "lca(" << u << ", " << v << ")=" << lca(u, v) << "\n";
    }
}
```

### 4.3. Kỹ thuật bảng thưa (Sparse table)

Đây là kỹ thuật hay dùng nhất trong giải các bài toán về LCA.

Nhận xét rằng mọi số nguyên dương đều có thể biểu dưới dạng nhị phân. Nhận xét này khá quan trọng, từ đó ta có thể cải tiến cách làm của các thuật toán đã giới thiệu ở trên bằng cách nhảy lên các lũy thừa của 2 (binary lifting), từ đó việc tìm  $LCA(u, v)$  chỉ có độ phức tạp  $O(\log(N))$ .

Giả sử  $depth(u) - depth(v) = 5 = 5_{(10)} = 101_{(2)} = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$

Khi đó để  $u$  nhảy lên tổ tiên  $v$  trên nó 5 bậc thì ta nhảy đến cha trước đó  $2^2$  bậc, sau đó nhảy tiếp cha  $2^0$  bậc.

Ta sẽ sử dụng bảng thưa (Sparse table), cha thứ  $2^j$  của đỉnh  $i$  là  $T[i][j]$ .

Định nghĩa Sparse table theo công thức truy hồi như sau:

$$\begin{cases} T[u][0] = par[u] & \text{Cha cấp } 2^0 \\ T[u][i] = T[T[u][i-1]][i-1] & \text{Cha cấp } 2^i \\ & (\text{Vì } 2^i = 2^{i-1} + 2^{i-1}) \end{cases}$$

Đánh giá độ phức tạp của thuật toán:

- Độ phức tạp tiền xử lí (xây dựng bảng thưa) :  $O(N * \log(N))$

- Độ phức tạp của một truy vấn là:  $O(\log(N))$ .

➔ Độ phức tạp thuật toán chung:  $O(N * \log(N) + Q * \log(N))$ .

Chương trình tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
int n, q, depth[100005], T[100005][17];
vector<int> adj[100005];
void dfs(int u, int p) { ///tim to tien o tang tren cua u
    depth[u] = depth[p] + 1;
    T[u][0] = p;
    for(int i = 1; i < 17; i++)
        T[u][i] = T[T[u][i-1]][i-1];
    for(int v : adj[u]) {
        if(v == p)
            continue;
        dfs(v, u);
    }
}
int lca(int u, int v) { ///Tim LCA Sparse Table
    if(depth[u] < depth[v])
        swap(u, v);
    for(int i = 16; i >= 0; i--) ///nhay den cung do sau
        if(depth[T[u][i]] >= depth[v])
            u = T[u][i];
    if(u == v)
        return u;
    for(int i = 16; i >= 0; i--) ///nhay den LCA
        if(T[u][i] != T[v][i]) {
            u = T[u][i];
            v = T[v][i];
        }
    return T[u][0];
}
```

```

    }
    return T[u][0];
}
int main() {
    cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    dfs(1, 0);
    cin >> q;
    while(q--) {
        int u, v;
        cin >> u >> v;
        cout << "lca(" << u << ", " << v << ")=" << lca(u, v) << "\n";
    }
}

```

**Để ý thấy rằng dù là Brute Force, chia căn, hay sparse table đều dùng chung một cách làm đó là: đưa 2 đỉnh về cùng tầng đã, sau đó lại đưa tiếp chúng về đến LCA.**

#### 4.4. Dùng Euler tour

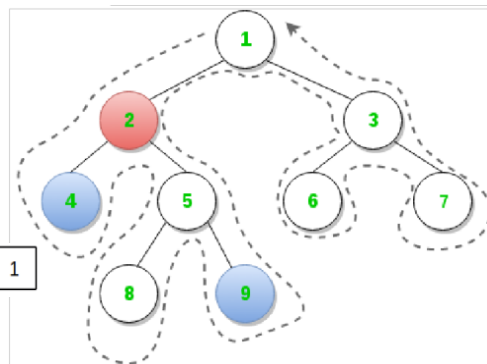
Quan sát đồ thị bên. Thứ tự các đỉnh được gọi đến theo đường nét đứt.

Với đồ thị trên ta có dãy các đỉnh được gọi đến như sau:

1	2	4	2	5	8	5	9	5	2	1	3	6	3	7	3	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Độ sâu tương ứng là:

1	2	4	2	3	4	3	4	5	2	1	2	3	2	3	2	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Nhận xét thấy rằng  $LCA(u, v)$  là đỉnh được gọi đến trong đoạn duyệt đến  $u, v$  mà có độ sâu nhỏ nhất.

Ví dụ:  $LCA(4, 9) = 2$  (đỉnh có độ sâu nhỏ nhất).

Kĩ thuật này có thể gọi nôm na là duỗi cây thành mảng. Đây là cách làm cũng rất hay, có nhiều ứng dụng. Kĩ thuật này có nhiều biến thể, nhưng chung quy nó đều kết hợp với DFS, trong khi DFS thì xác định một thứ tự duyệt đến, duyệt xong các đỉnh,...

Việc tìm đỉnh có độ sâu nhỏ nhất sau khi duỗi cây thành mảng là dạng bài toán Range Minimum Query (RMQ). Để giải quyết vấn đề này, chúng ta có thể sử dụng cấu trúc dữ liệu Sparse table hoặc Segment tree để giải quyết. Khi đó độ phức tạp thuật toán để trả lời  $Q$  truy vấn là:  $O(Q * \log(N))$ .

Vì đây là dạng quy bài toán LCA về RMQ, nên trong chuyên đề này không thể hiện lại chương trình mẫu. Bạn đọc có thể tự tham khảo tại: <http://vnoi.info/wiki/translate/topcoder/Range-Minimum-Query-and-Lowest-Common-Ancesor>

#### 4.5. Xử lý kiểu Off-line (Tarjan's off-line LCA)

Đây là thuật toán được Tarjan đề xuất năm 1979. Cách làm rất hay bằng cách kết hợp với cấu trúc dữ liệu DSU (Disjoint set union), độ phức tạp có thể coi gần như là hàm tuyến tính với kích thước cây ( $N$ ) và số câu truy vấn ( $Q$ ), thực tế nó cỡ hàm ngược của hàm Ackermann hoặc đánh giá tương đương  $\log(\log(N))$ .

Dựa theo có một số nhận xét sau:

- Khi duyệt một nhánh DFS gốc  $w$  thì các đỉnh con trong nhánh sẽ luôn thuộc thành phần DSU mà có tổ tiên là  $w$  khi duyệt xong nhánh DFS gốc  $w$ .

- Khi duyệt DFS, thì  $w = LCA(u, v)$  luôn được duyệt trước  $u, v$ . Giả sử khi duyệt xong một đỉnh  $u$  thì luôn kiểm tra xem đỉnh  $v$  đã duyệt chưa? Nếu duyệt rồi và  $v$  sẽ thuộc thành phần DSU có phần tử đại diện luôn là  $LCA(u, v)$ .

- Khi duyệt xong  $w$  thì ta sẽ trả lời xong tất cả các truy vấn  $LCA(u, v)$  với mọi  $u, v$  thuộc nhánh DFS gốc  $w$ . Do đó các truy vấn được trả lời theo thứ tự duyệt DFS chứ không phải thứ tự truy vấn cho ban đầu. Do vậy thuật toán này được ghi nhận là thuật toán kiểu Offline.

Giả sử  $u$  lại được duyệt xong trước  $v$ , vậy thì trong nhánh của cây DFS gốc  $w$  thì ta đã duyệt qua  $u$  và khi vừa duyệt xong  $v$  ta làm cách nào để in ra  $w$ ?. Để tìm nhanh ra  $w$ , ta luôn lưu lại các nhánh đã được duyệt trong nhánh DFS gốc  $w$  bằng cấu trúc DSU, trong đó tổ tiên của nhánh đó chính là  $w$ . Khi duyệt xong  $v$  thì ta cần in ra tổ tiên của tập mà chứa  $u$ .

```
cout<< ancestor[find_set(u)];
```

##### Chương trình tham khảo:

```
#include<bits/stdc++.h>
using namespace std;
vector<int> adj[100005], queries[100005];
int n, q, par[100005], rnk[100005], ancestor [100005];
bool visited[100005];
int find_set(int u) { ///cau truc DSU
    while(par[u] != u)
        u = par[u];
    return u;
}
void union_set(int x, int y) { ///cau truc DSU
    int xroot = find_set(x);
    int yroot = find_set(y);
    if(xroot == yroot)
        return;
    if(rnk[xroot] < rnk[yroot])
        par[xroot] = yroot;
    else
        if(rnk[xroot] > rnk[yroot])
            par[yroot] = xroot;
        else {
            par[xroot] = yroot;
            rnk[yroot]++;
        }
}
void dfs(int w) {
```

```

        visited[w] = true;
        ancestor[w] = w;
        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;
            }
        }
        ///tra loi cac truy van
        for(int u : queries[w]) {
            if(visited[u])
                cout << "lca(" << w << ", " << u
                    << ")= " << ancestor[find_set(u)] << "\n";
        }
    }
    int main() {
        //freopen("LCA_Tarjan.inp", "r", stdin);
        cin >> n;
        for(int i = 1; i < n; i++) {
            int u, v;
            cin >> u >> v;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        for(int i = 1; i <= n; i++)
            par[i] = i;
        cin >> q;
        while(q--) {
            int u, v;
            cin >> u >> v;
            queries[u].push_back(v);
            queries[v].push_back(u);
        }
        dfs(1);
    }
}

```

Để trả lời các truy vấn theo thứ tự dữ liệu cho ban đầu, chúng ta cần bổ sung thông tin vào mảng **queries** có thêm thông tin về thứ tự câu hỏi và in ra kết quả theo thứ tự đề bài yêu cầu.

Tóm lại, chuyên đề này giới thiệu 3 cách làm cơ bản một là greedy, quy về bài toán RMQ bằng Euler tour, xử lý Off-line theo kiểu của Tarjan. Ngoài ra kiểu Greedy có rất nhiều biến thể, chẳng hạn có thể kết hợp với kĩ thuật Heavy light decomposition,... ta không đi sâu ở đây.

## 5. Một số bài tập ví dụ

Trong các kĩ thuật giải bài toán LCA, ta tập trung chính vào kĩ thuật áp dụng bảng thưa (Sparse table), vì đây là kĩ thuật khá tốt, đơn giản trong cài đặt, đảm bảo tính hiệu quả. Các bài trong chuyên đề này tôi đều dùng Sparse table. Có một số bài tôi chọn cài đặt duyệt cây bằng BFS hoặc DFS. Ngay cả trong một số bài cần cập nhật thông tin ngược và xuôi trên cây (dạng quy hoạch động) chúng ta vẫn có thể không dùng DFS (ví dụ bài số 10- Tom và Jerry, để làm như vậy đương nhiên bạn cần phải lưu trữ thông tin đỉnh duyệt trước, sau của đỉnh bất kì  $u$ ).

## 5.1. Bài 1: Bài tập cơ bản

### 5.1.1. Đề bài: LCA

Cho một cây  $N$  đỉnh có gốc là 1. Có  $Q$  truy vấn tìm LCA của hai đỉnh  $x$  và  $y$ .  
Dữ liệu cho đảm bảo  $1 \leq N, Q \leq 10^5$  và  $1 \leq x, y \leq N$ .

**Dữ liệu vào:** Đọc vào từ tệp LCA.inp

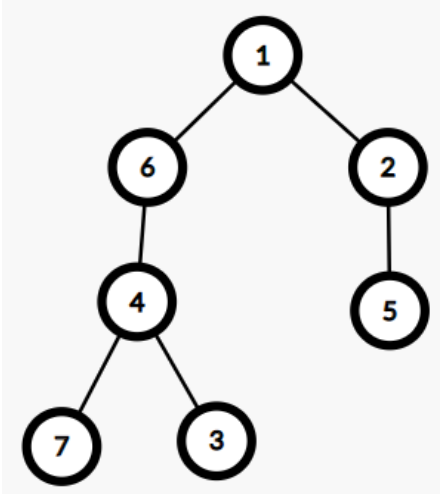
Dòng đầu tiên là số  $N$ . Sau đó  $N - 1$  cạnh của cây.

Số  $Q$ . Sau đó  $Q$  cặp số  $(x, y)$ .

**Kết quả ra:** Ghi kết quả ra tệp LCA.out

Với mỗi truy vấn, in ra kết quả cần tìm trên từng dòng.

**Ví dụ:**

LCA.inp	LCA.out	Hình minh họa
7 6 1 6 4 4 7 3 4 1 2 2 5 6 3 3 7 7 1 3 5 7 7 6 2 4	3 7 1 1 6 1	

### 5.1.2. Phân tích đề bài và đề xuất thuật toán

Đây là bài cơ bản, có thể giải bằng rất nhiều cách khác nhau đã nêu trên. Để đảm bảo AC thì cần chọn thuật toán có độ phức tạp  $O(Q \log N + N \log N)$ .

Do Sparse table được tạo ngay trong quá trình duyệt đồ thị, nên chúng ta có thể dùng DFS hoặc BFS đều được, tuy nhiên ta hay dùng DFS đệ quy để cho việc code đơn giản và ngắn gọn hơn.

Nhiều trường hợp số lần gọi đệ quy quá nhiều dẫn tới tràn bộ nhớ, chúng ta có thể cài đặt bằng DFS không đệ quy, hoặc BFS như sau:

```
void bfs() {
    queue<int> Q;
    Q.push(1);
    depth[1]=1;
    while(!Q.empty()) {
        int u=Q.front();
        Q.pop();
        if(!visited[u]) {
            visited[u]=1;
            for(int v:adj[u]) {
```

```

        if(v==up[u][0])
            continue;
        up[v][0]=u;
        depth[v]=depth[u]+1;
        for(int i=1; i<=16; i++) ///sparse table
            up[v][i]=up[up[v][i-1]][i-1];
        Q.push(v);
    }
}
}

```

### 5.1.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.2. Bài 2: Tổ chức thi chạy Marathon

### 5.2.1. Đề bài: Marathon

Thầy giáo dạy giáo dục thể chất tại trường chuyên XYZ đang cần tổ chức cuộc thi chạy cho học sinh, biết địa bàn thành phố là đồ thị vô hướng dạng cây gồm  $N$  đỉnh và  $N-1$  cạnh. Do cần giám sát, đảm bảo an toàn, giáo sư đã nhờ một chuyên gia khoa học máy tính thiết kế một camera để giám sát trên đoạn đường chạy. Chuyên gia đã đưa cho thầy giáo  $Q$  phương án. Mỗi phương án là bộ 3 số  $u, v, w$  trong đó  $u, v$  là điểm đầu, cuối của đoạn đường chạy,  $w$  là vị trí đặt camera. Bạn hãy giúp xem chuyên gia đã thực hiện đúng yêu cầu của thầy giáo đặt ra chưa.

**Dữ liệu vào:** Từ tệp Marathon.inp

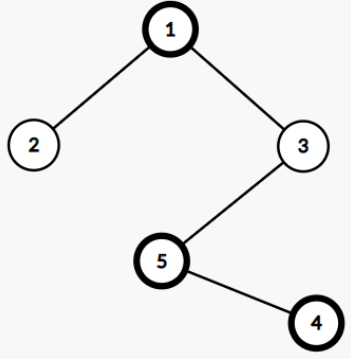
- Dòng đầu số đỉnh của đồ thị  $N$ , và số phương án chọn đường chạy  $Q$ .
- Các dòng tiếp theo thể hiện cạnh của đồ thị.
- Các dòng sau đó là  $Q$  phương án

**Kết quả ra:** Ghi ra tệp Marathon.out

Gồm  $Q$  dòng, nếu phương án đảm bảo việc lắp camera trên đường chạy thì in ra 1, ngược lại in ra 0.

**Ràng buộc:**  $1 \leq N, Q \leq 10^5$ .

**Ví dụ:**

Marathon.inp	Marathon.out	Giải thích
5 3 1 2 1 3 3 5 4 5 2 3 1 5 4 5 2 3 4	1 1 0	

### 5.2.2. Phân tích đề bài và đề xuất thuật toán

Đây là bài dễ về LCA. Bài toán cần kiểm tra xem  $w$  có nằm trên đường đi từ  $u$  đến  $v$  không. Có 3 cách để giải quyết vấn đề trên:

*Cách 1: Không áp dụng LCA.*

Mỗi truy vấn  $(u, v, w)$  ta duyệt DFS từ gốc  $u$  để xác định quan hệ cha con khi duyệt DFS. Sau đó đi ngược từ  $v$  tới gốc  $u$  nếu gặp  $w$  trên đó thì in ra 1, còn lại in ra 0. Độ phức tạp:  $O(N \cdot Q)$

*Cách 2: Áp dụng LCA dùng khoảng cách*

Nếu  $w$  nằm trên đường đi  $u \rightarrow v$  thì:

$$\text{dist}(u, v) = \text{dist}(v, w) + \text{dist}(u, w)$$

*Cách 3: Áp dụng LCA không dùng khoảng cách*

Nếu  $w$  nằm trên đường đi  $u \rightarrow v$  thì:

TH1: Nếu  $w = \text{LCA}(u, w)$  thì  $\text{LCA}(u, v) = \text{LCA}(w, v)$

TH2: Nếu  $w = \text{LCA}(v, w)$  thì  $\text{LCA}(u, v) = \text{LCA}(w, u)$

Độ phức tạp của cách có áp dụng LCA:  $O(N * \log(N))$ .

Code tham khảo dựa theo cách dùng khoảng cách, duyệt BFS.

```
#include<bits/stdc++.h>
using namespace std;
int n, q, depth[100005], up[100005][17], visited[100005];
vector<int> adj[100005];
void bfs() {
    queue<int> Q;
    Q.push(1);
    depth[1]=1;
    while(!Q.empty()) {
        int u=Q.front();
        Q.pop();
        if(!visited[u]) {
            visited[u]=1;
            for(int v:adj[u]) {
                if(v==up[u][0])
                    continue;
                up[v][0]=u;
                depth[v]=depth[u]+1;
                for(int i=1; i<=16; i++)
                    up[v][i]=up[up[v][i-1]][i-1];
                Q.push(v);
            }
        }
    }
}
int lca(int u, int v) { //Tim LCA Sparse upable
    if(depth[u] < depth[v])
        swap(u, v);
    for(int i = 16; i >= 0; i--) //nhay den cung do sau
        if(depth[up[u][i]] >= depth[v])
            u = up[u][i];
    if(u == v)
```



```
        return u;
    for(int i = 16; i >= 0; i--)///nhay den LCA
        if(up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    return up[u][0];
}
int dist(int u, int v) {
    return (depth[u]+depth[v]-2*depth[lca(u,v)]);
}
int main() {
    ios_base::sync_with_stdio(0);
    // freopen("marathon.inp", "r", stdin);
    cin >> n>>q;
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    bfs();
    while(q--) {
        int u, v,w;
        cin >> u >> v>>w;
        if(dist(u,v)==dist(u,w)+dist(v,w))
            cout<<"1\n";
        else
            cout<<"0\n";
    }
}
```

### 5.2.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.3. Bài 3: Du lịch thành phố (NAIPC 2016)

### 5.3.1. Đề bài: Tourist

Tại thành phố cây, có  $N$  điểm du lịch hấp dẫn được đánh số từ 1 đến  $N$ . Thành phố có  $N-1$  con đường 2 chiều để nối các điểm du lịch. Thị trưởng thành phố phát hiện ra là việc tổ chức các tour đi từ địa điểm  $u$  đến các địa điểm được đánh số là bội của nó sẽ rất thú vị, các tour như vậy thì du khách sẽ được thăm tất cả các địa điểm trên đường đi đơn giữa 2 địa điểm này. Hỏi với tất cả cách tổ chức tour như vậy thì tổng số địa điểm được thăm là bao nhiêu?

**Dữ liệu vào:** Từ tệp Tourist.inp

Dòng đầu tiên là số địa điểm du lịch  $N$  của thành phố

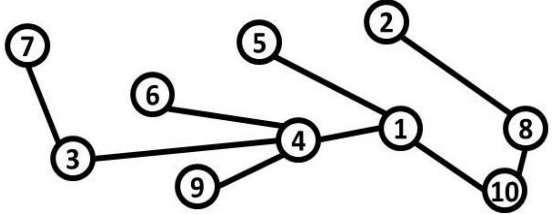
$N-1$  dòng tiếp theo thể hiện đường nối giữa các thành phố

**Kết quả ra:** Ghi ra tệp Tourist.out

Ghi tổng số địa điểm du lịch được thăm với tất cả các tour được xây dựng.

**Ràng buộc:**  $1 \leq N \leq 10^5$ .

**Ví dụ:**

Tourist.inp	Tourist.out	Giải thích
10 3 4 3 7 1 4 4 6 1 10 8 10 2 8 1 5 4 9	55	 <p>Chúng ta có tất cả các con đường và số địa điểm có thể thăm được như sau: 1→2=4 ; 1→3=3; 1→4=2; 1→5=2; 1→6=3; 1→7=4; 1→8=3; 1→9=3; 1→10=2; 2→4=5; 2→6=6; 2→8=2; 2→10=3; 3→6=3; 3→9=3 ; 4→8=4 ; 5→10=3.</p> <p>Do đó tổng số địa điểm du lịch được thăm sẽ là: 55.</p>

### 5.3.2. Phân tích đề bài và đề xuất thuật toán

Bài này liên quan đến việc tính tổng khoảng cách giữa các cặp  $(u, v)$  mà  $v$  là bội của  $u$ .

Duyệt tất cả các cặp tạo được tour du lịch với điểm cuối là bội của điểm đầu. Tính tổng (khoảng cách +1) vào kết quả.

$$ans = \sum_{u=1}^n dist(u, v) \text{ với mọi } v \text{ là bội của } u.$$

#### Chương trình tham khảo:

```
#include<bits/stdc++.h>
using namespace std;
int n, q, depth[100005], T[100005][17];
vector<int> adj[100005];
void dfs(int u, int p)
{
    depth[u] = depth[p] + 1;
    T[u][0] = p;
    for(int i = 1; i < 17; i++)    ///xay dung bang thua
        T[u][i] = T[T[u][i - 1]][i - 1];
    for(int v : adj[u])
    {
        if(v == p)
            continue;
        dfs(v, u);
    }
}
int lca(int u, int v)    ///Tim LCA Sparse Table
{
    if(depth[u] < depth[v])
        swap(u, v);
    for(int i = 16; i >= 0; i--)    ///nhay den cung do sau
        if(depth[T[u][i]] >= depth[v])
            u = T[u][i];
    if(u == v)
        return u;
}
```

```

        for(int i = 16; i >= 0; i--)///nhay den LCA
            if(T[u][i] != T[v][i])
            {
                u = T[u][i];
                v = T[v][i];
            }
        return T[u][0];
    }
    int dist(int u, int v)
    {
        return (depth[u]+depth[v]-2*depth[lca(u,v)]);
    }
    int main()
    {
        freopen("tourist.inp", "r", stdin);
        cin >> n;
        for(int i = 1; i < n; i++)
        {
            int u, v;
            cin >> u >> v;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        dfs(1, 0);
        long long ans = 0;
        for(int u = 1; u <= n; ++u)
        {
            for(int i = 2; u * i <= n; ++i)
            {
                int v=u*i;
                int c = lca(u, v);
                ans += dist(u,v)+1;
            }
        }
        cout << ans;
    }

```

### 5.3.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

Qua 3 bài ví dụ đầu giúp học sinh làm quen với các ứng dụng cơ bản nhất của bài toán LCA. Tiếp sau đây, tôi đưa ra dạng bài có kết hợp với cấu trúc dữ liệu.

## 5.4. Bài 4: VOTREE (VNOI online 2015)

### 5.4.1. Đề bài

Cho cây gồm  $N$  đỉnh ( $N \leq 70000$ ), có gốc là đỉnh 1. Bạn cần trả lời  $Q$  truy vấn, mỗi truy vấn gồm 2 số  $u, v$ . Bạn cần tìm đỉnh xa gốc nhất, mà là tổ tiên của tất cả các đỉnh  $u, u + 1, \dots, v$ .

**Dữ liệu vào:** Từ tệp VOTREE.inp

Dòng đầu ghi 2 số nguyên dương  $N$  và  $Q$  ( $1 \leq N, Q \leq 70000$ ).

$N - 1$  dòng tiếp theo, mỗi dòng chứa 2 số nguyên dương  $u$  và  $v$ , thể hiện có 1 cạnh nối giữa 2 đỉnh  $u$  và  $v$ . ( $u \neq v; 1 \leq u, v \leq N$ ).

$Q$  dòng tiếp theo, mỗi dòng gồm 2 số nguyên dương  $u$  và  $v$  ( $1 \leq u \leq v \leq N$ ), thể hiện 1 truy vấn.

**Kết quả ra:** Ghi ra tệp VOTREE.out

Với mỗi truy vấn, in ra 1 dòng duy nhất là đáp số của truy vấn.

**Ví dụ:**

VOTREE.inp	VOTREE.out	Hình vẽ
5 3 1 2 2 3 3 4 3 5 2 5 1 3 4 5	2 1 3	<pre> graph TD     1((1)) --- 2((2))     2 --- 3((3))     3 --- 4((4))     3 --- 5((5))                     </pre>

#### 5.4.2. Phân tích đề bài và đề xuất thuật toán

Việc tìm LCA của 2 đỉnh là bài cơ bản, tuy nhiên ở đây lại cần tìm LCA của các đỉnh từ  $u, u + 1, \dots, v$ . Nên nếu mỗi truy vấn ta tìm LCA đoạn  $[u, v]$  như sau:

$LCA(u, LCA(u + 1, LCA(u + 2, \dots)))$  thì sẽ không đảm bảo về mặt thời gian.

Nhận thấy xuất hiện truy vấn của một đoạn nên ta nghĩ đến sử dụng cấu trúc dữ liệu Segment tree để xử lí. Segment tree sẽ lưu thông tin về LCA của đoạn. Thời gian xây dựng trong  $O(N \cdot \log^2(N))$ . Mỗi truy vấn xử lí trong thời gian  $O(\log^2(N))$ .

Mỗi đỉnh id có 2 đỉnh con là  $2 \cdot id$  và  $2 \cdot id + 1$ . Thông tin cập nhật như sau:

$$ST[id] = LCA(ST[2 \cdot id], ST[2 \cdot id + 1])$$

Chương trình tham khảo:

```

#include<bits/stdc++.h>
#define maxn 100005
using namespace std;
int n,q,depth[maxn],up[maxn][20],ST[4*maxn];
vector<int> adj[maxn];
void dfs(int u, int p) {
    depth[u]=depth[p]+1;
    up[u][0]=p;
    for(int i=1; i<=17; i++)
        up[u][i]=up[up[u][i-1]][i-1];
    for(int v:adj[u]) {
        if(v==p)
            continue;
        dfs(v,u);
    }
}
int lca(int u, int v) {
    if(u<1)
        return v;
    if(v<1)
        return u;

```

```
    if(depth[u]<depth[v])
        swap(u,v);
    for(int i=16; i>=0; i--)
        if(depth[up[u][i]]>=depth[v])
            u=up[u][i];
    if(u==v)
        return u;
    for(int i=16; i>=0; i--)
        if(up[u][i]!=up[v][i])
            u=up[u][i],v=up[v][i];
    return up[u][0];
}
void update(int id, int L, int R, int vt, int val) {
    if(vt<L || R<vt)
        return;
    if(L==R) {
        ST[id]=val;
        return;
    }
    int mid=(L+R)/2;
    update(2*id,L,mid,vt,val);
    update(2*id+1,mid+1,R,vt,val);
    ST[id]=lca(ST[2*id],ST[2*id+1]);
}
int get(int id, int L, int R, int u, int v) {
    if(v<L || R<u)
        return 0;
    if(u<=L && R<=v)
        return ST[id];
    int mid=(L+R)/2;
    int x=get(2*id,L,mid,u,v);
    int y=get(2*id+1,mid+1,R,u,v);
    return lca(x,y);
}

int main() {
    ios_base::sync_with_stdio(0);
    cin>>n>>q;
    for(int i=1; i<n; i++) {
        int u,v;
        cin>>u>>v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    dfs(1,0);
    for(int i=1; i<=n; i++)
        update(1,1,n,i,i);
    for(int i=1; i<=q; i++) {
        int u,v;
        cin>>u>>v;
        if(u>v)
            swap(u,v);
        cout<<get(1,1,n,u,v)<<"\n";
    }
}
```

### 5.4.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.5. Bài 5: Tăng lương (Chọn đội tuyển IOI CROATIAN 2010)

### 5.5.1. Đề bài POVISICE

Mirko là ông chủ kiêu hãnh của một công ty phần mềm lớn. Ban đầu công ty chỉ có một mình Mirko. Công việc làm ăn phát đạt và công ty thuê  $n$  công nhân, lần lượt từng người, từng người một. Mirko được đánh số là 0. Các công nhân khác – đánh số từ 1 đến  $n$  theo trình tự thuê.

Mỗi người mới vào có một mức lương khởi điểm và chịu sự chỉ đạo của một ai đó đã có mặt trong công ty. Nếu lương công nhân cao hơn lương thủ trưởng trực tiếp của mình thì lương của người thủ trưởng đó được nâng lên bằng lương người dưới quyền mình. Quá trình điều chỉnh này được tiếp diễn cho đến khi đảm bảo được trong toàn công ty lương thủ trưởng không thấp hơn lương công nhân dưới quyền.

**Yêu cầu:** Với mỗi công nhân được tuyển chọn vào công ty hãy xác định số người phải điều chỉnh lương cho phù hợp với người mới được tuyển chọn.

**Dữ liệu vào:** Từ tệp POVISICE.INP:

- Dòng đầu tiên chứa số nguyên  $n$  ( $1 \leq n \leq 3 \cdot 10^5$ ),
- Dòng thứ 2 chứa một số nguyên – lương khởi điểm của Mirko,
- Dòng thứ  $i$  trong  $n$  dòng sau chứa 2 số nguyên  $S$  và  $B$  – lương khởi điểm và thủ trưởng của người công nhân thứ  $i$ .

Lương khởi điểm nằm trong phạm vi từ 1 đến  $10^9$ .

**Kết quả ra:** Ghi ra tệp POVISICE.OUT

$n$  số số nguyên, mỗi số trên một dòng, là kết quả tính được đối với mỗi người.

**Ràng buộc:**

- 30% số test tương ứng 30% số điểm có  $n \leq 5000$
- 30% số test tương ứng 30% số điểm có  $5000 < n \leq 50000$
- 40% số test tương ứng 40% số điểm có  $50000 < n \leq 300000$

**Ví dụ:**

POVISICE.INP	POVISICE.OUT	Hình vẽ
7	0	
5000	1	
4500 0	0	
6000 0	2	
4000 1	4	
5500 3	1	
7000 4	0	
6300 2		
6300 2		

### 5.5.2. Phân tích đề bài và đề xuất thuật toán

- Dựa theo dữ liệu đầu vào, ta hoàn toàn có thể xây dựng trước một cây. Mỗi nhân viên là một đỉnh, quan hệ cha con được thể hiện qua thứ tự khi duyệt cây bằng DFS từ gốc 0.

- Ban đầu lương của tất cả các nhân viên coi như bằng 0 (Tất cả các đỉnh được gán giá trị 0).

- Sau đó là các thao tác thêm nhân viên mới vào ta coi đó là **thao tác tăng lương** của nhân viên. Hỏi cần điều chỉnh mức lương của bao nhiêu quản lý cấp trên của nhân viên đó.

**Để trả lời N truy vấn ta có 2 cách như sau:**

Cách 1: Duyệt trâu

Xử lý các khá đơn giản như sau: Với mỗi truy vấn, ta đi ngược về gốc, nếu quản lý cấp trên nào của nó có mức lương thấp hơn thì ta cập nhật lại mức lương và tăng kết quả, đến khi quản lý cấp trên đã có mức lương không thấp hơn nhân viên thì dừng lại.

Độ phức tạp thuật toán:  $O(N^2)$ .

Cách 2: Sử dụng LCA kết hợp cấu trúc dữ liệu Segment tree.

Để phát hiện ra cần dùng LCA ở đâu trong bài này là không dễ dàng.

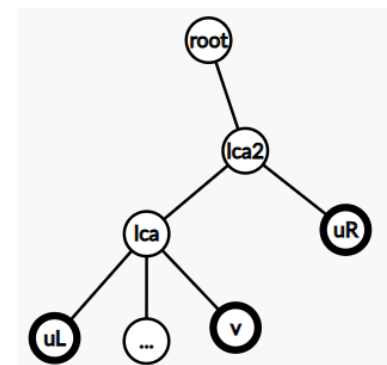
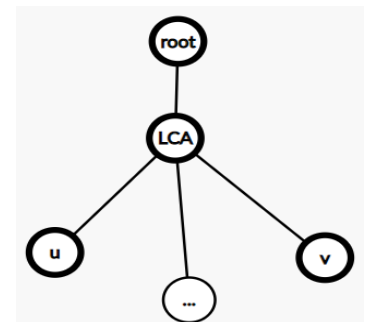
Quan sát thấy rằng, các quản lý cấp trên nếu bị thay đổi sẽ là đoạn liên tục các đỉnh tính từ vị trí nhân viên mới tăng lương ngược về gốc. Vấn đề là làm sao để xác định được đoạn cần điều chỉnh và tránh việc phải cập nhật lại đoạn phải điều chỉnh?

Thông thường, trong quá trình dạy HSG tôi thường định hướng cách giải quyết vấn đề như sau: khi gặp bài toán phức tạp, ta nên tìm cách chia nó ra thành các bài toán nhỏ hơn, hoặc tìm ra điểm đặc biệt của bài toán sau đó phán đoán, tìm ra điểm mấu chốt để giải bài

Với bài này, tình huống: nếu việc tăng lương diễn ra theo đúng thứ tự duyệt DFS thì sao? Khi đó tìm vị trí cần điều chỉnh như thế nào? Câu trả lời là ta chỉ quan tâm vị trí gần nhất  $u$  mà nó có mức tăng lớn hơn hoặc bằng vị trí  $v$  đang xét. Toàn bộ các vị trí từ  $u \rightarrow LCA(u, v) \rightarrow root$  đều đã được tăng khi thêm  $u$ . Các vị trí cần tăng khi thêm  $v$  chỉ là từ  $v \rightarrow LCA(u, v)$ .

Nếu  $u = LCA(u, v)$  thì hiển nhiên vẫn đúng.

Từ đó thấy rằng duyệt DFS không nhất thiết đã theo đúng thứ tự từ điển (là cách mà tăng lương diễn ra). Có thể đỉnh  $u$  ở trước hoặc sau  $v$  theo thứ tự từ điển. Gọi  $u_L, u_R$  là đỉnh bên trái, phải gần  $v$  nhất mà giá trị không nhỏ hơn tại  $v$ . Khi đó hiển nhiên số đỉnh bị tác động bởi quá trình tăng lương khi xét  $v$  là:



$$\min\{dist(LCA(u_L, v)), dist(LCA(u_R, v))\}$$

Để tìm ra  $u_L, u_R$  ta có thể nghĩ đến sử dụng Segment tree lưu giá trị max của đoạn khi cập nhật tăng lương.

Chương trình tham khảo:

```
#include <bits/stdc++.h>
#define N 333333
#define PB push_back
using namespace std;
int n, sal[N], in[N], h[N], p[N][20], dem;
vector<int> adj[N];
struct IT {
    int tree[N << 2], node[N << 2];
    void update(int l, int r, int id, int x, int val, int u) {
        if(l > x || r < x)
            return;
        if(l == r) {
            tree[id] = max(tree[id], val);
            node[id] = u;
            return;
        }
        int mid = (l + r) / 2;
        update(l, mid, id * 2, x, val, u);
        update(mid + 1, r, id * 2 + 1, x, val, u);
        tree[id] = max(tree[id * 2], tree[id * 2 + 1]);
    }
    int get_max(int l, int r, int id, int x, int y) {
        if(l > y || r < x)
            return 0;
        if(l >= x && r <= y)
            return tree[id];
        int mid = (l + r) / 2;
        int a = get_max(l, mid, id * 2, x, y);
        int b = get_max(mid + 1, r, id * 2 + 1, x, y);
        return max(a, b);
    }
    int get_left(int l, int r, int id, int x, int y, int u) {
        if(tree[id] < sal[u])
            return -1;
        if(l == r && tree[id] >= sal[u])
            return node[id];
        int mid = (l + r) / 2;
        if(y <= mid)
            return get_left(l, mid, id * 2, x, y, u);
        else {
            int res = get_left(mid + 1, r, id * 2 + 1, mid + 1, y, u);
            if(res != -1)
                return res;
            return get_left(l, mid, id * 2, x, mid, u);
        }
    }
    int get_right(int l, int r, int id, int x, int y, int u) {
        if(tree[id] < sal[u])
            return -1;
        if(l == r && tree[id] >= sal[u])
```



```
        return node[id];
    int mid = (l + r) / 2;
    if(x > mid)
        return get_right(mid + 1, r, id * 2 + 1, x, y, u);
    else {
        int res = get_right(l, mid, id * 2, x, mid, u);
        if(res != -1)
            return res;
        return get_right(mid + 1, r, id * 2 + 1, mid + 1, y, u);
    }
}
} t;
void nhap() {
    scanf("%d", &n);
    n++;
    scanf("%d", &sal[1]);
    for(int u = 2; u <= n; u++) {
        int v;
        scanf("%d %d", &sal[u], &v);
        v++;
        adj[v].PB(u);
    }
}
void DFS(int u) {
    in[u] = ++dem;
    for(int i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        h[v] = h[u] + 1;
        p[v][0] = u;
        for(int j = 1; j < 20; j++)
            p[v][j] = p[p[v][j - 1]][j - 1];
        DFS(v);
    }
}
void setup() {
    nhap();
    DFS(1);
}
int lca(int u, int v) {
    if(h[u] < h[v])
        swap(u, v);
    int diff = h[u] - h[v];
    for(int i = 19; i >= 0; i--)
        if(diff & (1 << i))
            u = p[u][i];
    if(u == v)
        return u;
    for(int i = 19; i >= 0; i--)
        if(p[u][i] != p[v][i]) {
            u = p[u][i];
            v = p[v][i];
        }
    return p[u][0];
}
void solve() {
    t.update(1, n, 1, 1, sal[1], 1);
}
```

```

for(int i = 2; i <= n; i++) {
    int l = t.get_left(1, n, 1, 1, in[i] - 1, i);
    int r = t.get_right(1, n, 1, in[i] + 1, n, i);
    int p = -1, q = -1;
    if(l != -1)
        p = lca(l, i);
    if(r != -1)
        q = lca(r, i);
    cout << p << " " << q << endl;
    if((h[p] < h[q] && q != -1) || p == -1)
        swap(p, q);
    if(l == -1 && r == -1)
        printf("%d\n", h[i]);
    else
        printf("%d\n", h[i] - h[p] - 1);
    t.update(1, n, 1, in[i], sal[i], i);
}
}
int main() {
    freopen("POVISICE.INP", "r", stdin);
    freopen("POVISICE.OUT", "w", stdout);
    setup();
    solve();
}

```

### 5.5.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.6. Bài 6: Nâng cấp mạng (VOI 2011)

### 5.6.1. Đề bài: UPGRANET

Một hệ thống gồm  $N$  máy tính đánh số từ 1 đến  $N$  được kết nối thành một mạng bởi  $M$  đoạn cáp mạng đánh số từ 1 đến  $M$ . Đoạn cáp mạng thứ  $i$  có thông lượng  $w_i$  kết nối hai máy  $u_i, v_i$  cho phép truyền dữ liệu theo cả hai chiều giữa hai máy này.

Một dãy các máy  $x_1, x_2, \dots, x_p$  trong đó giữa hai máy  $x_i$  và  $x_{i+1}$  ( $i=1,2,\dots,p-1$ ) có đoạn cáp nối được gọi là một đường truyền tin từ máy  $x_1$  tới máy  $x_p$ . Thông lượng của đường truyền tin được xác định như là thông lượng nhỏ nhất trong số các thông lượng của các đoạn cáp mạng trên đường truyền. Giả thiết là mạng được kết nối sao cho có đường truyền tin giữa hai máy bất kì và giữa hai máy có không quá một đoạn cáp mạng nối chúng.

Người ta muốn nâng cấp mạng bằng cách tăng thông lượng của một số đoạn cáp nối trong mạng. Để tăng thông lượng của mỗi đoạn cáp mạng thêm một lượng  $d$  ( $d > 0$ ) ta phải trả một chi phí đúng bằng  $d$ . Việc nâng cấp mạng phải đảm bảo là sau khi hoàn tất, thông lượng của mỗi đoạn cáp mạng  $i$  đều bằng thông lượng của đường truyền tin có thông lượng lớn nhất từ máy  $u_i$  tới máy  $v_i$ .

**Yêu cầu:** Tìm phương án nâng cấp các đoạn cáp mạng sao cho tổng chi phí nâng cấp là nhỏ nhất.

**Dữ liệu vào:** Từ tệp UPGRANET.inp

Dòng thứ nhất: Chứa hai số nguyên dương  $N, M$  ( $N, M \leq 10^5$ ).

Dòng thứ  $i$  trong số  $M$  dòng tiếp theo chứa ba số nguyên dương  $u_i, v_i, w_i$  ( $w_i \leq 10^6$ ),  $i = 1, 2 \dots M$ . Các số trên cùng một dòng được ghi cách nhau ít nhất một dấu cách.

**Kết quả ra:** Ghi ra tệp UPGRANET.inp

Ghi ra một số nguyên duy nhất là tổng chi phí nâng cấp thấp nhất.

Ví dụ:

UPGRANET.inp	UPGRANET.out	Giải thích
6 7 1 2 6 1 3 5 2 4 3 3 4 9 4 5 4 4 6 8 5 6 7	5	

### 5.6.2. Phân tích đề bài và đề xuất thuật toán

Nhận xét:

- Giữa 2 máy tính bất kì  $u, v$  chỉ có một đoạn cáp nối, nhưng có thể có nhiều đường đi giữa chúng.

- Đường đi có thông lượng lớn nhất giữa 2 máy tính  $u, v$  chính là đường đi trên cây khung lớn nhất.

- Với mỗi cạnh  $(u, v)$  không nằm trên khung lớn nhất, ta cần nâng cấp nó đảm bảo lớn hơn thông lượng của đường đi giữa chúng trên cây khung (hay là cạnh có trọng số nhỏ nhất trên đường đi từ  $u \rightarrow v$  trên cây khung lớn nhất).

Qua đó ta đã dễ dàng hình dung thuật toán để giải bài trên.

Có thể sử dụng nhiều kĩ thuật để giải bài này như: Heavy light decomposition; Tarjan offline LCA;... Nhưng kĩ thuật dễ dàng cài đặt nhất ta vẫn sử dụng bảng thưa. Bảng thưa lưu thông tin về đỉnh tổ tiên và cả cạnh có trọng số nhỏ nhất trên đường đi đến đỉnh tổ tiên của nó.

Độ phức tạp thuật toán là:  $O(N \cdot \log(N))$

Chương trình tham khảo:

```
#include <bits/stdc++.h>
#define maxn 1000005
#define S second
#define F first
#define maxc 1000000007
#define fort(i, a, b) for(int i = (a); i <= (b); i++)
#define ford(i, a, b) for(int i = (a); i >= (b); i--)
#define ll long long
using namespace std;
ll n, m, root[maxn], h[maxn], res;
pair<long long, long long> par[maxn][20];
```

```
bool dd[maxn];
vector<pair<ll, ll> > ke[maxn];
struct edge {
    ll u, v, w;
} ed[maxn];
bool cmp(edge p, edge q) {
    return p.w > q.w;
}
ll getroot(ll u) {
    if(root[u] == 0)
        return u;
    return root[u] = getroot(root[u]);
}
void MST() {
    sort(ed+1, ed+m+1, cmp);
    for(i, 1, m) {
        ll p = getroot(ed[i].u);
        ll q = getroot(ed[i].v);
        if(p == q)
            continue;
        root[p] = q;
        dd[i] = 1;
        ke[ed[i].u].push_back(make_pair(ed[i].v, ed[i].w));
        ke[ed[i].v].push_back(make_pair(ed[i].u, ed[i].w));
    }
}
void dfs(ll u, ll tr) {
    for(i, 0, int(ke[u].size()) - 1) {
        ll v = ke[u][i].F;
        if(v == tr)
            continue;
        h[v] = h[u] + 1;
        par[v][0] = make_pair(u, ke[u][i].S);
        for(j, 1, 18) {
            par[v][j].F = par[par[v][j-1].F][j-1].F;
            par[v][j].S = min(par[par[v][j-1].F][j-1].S, par[v][j-1].S);
        }
        dfs(v, u);
    }
}
pair<long long, long long> lca(ll u, ll v) {
    pair<long long, long long> p;
    p.S = 1ll* maxc * maxc;
    if(h[u] > h[v])
        swap(u, v);
    ll diff = h[v] - h[u];
    for(i, 18, 0)
        if((diff >> i) & 1) {
            p.S = min(p.S, par[v][i].S);
            v = par[v][i].F;
        }
    if(v == u)
        return make_pair(u, p.S);
    for(i, 18, 0)
        if(par[u][i].F != par[v][i].F) {
            p.S = min(p.S, min(par[v][i].S, par[u][i].S));
        }
}
```

```

        v = par[v][i].F;
        u = par[u][i].F;
    }
    return make_pair(par[u][0].F, min(p.S, min(par[u][0].S, par[v][0].S)));
}
void solve() {
    MST();
    h[1] = 1;
    dfs(1, 0);
    fort(i, 1, m)
    if(!dd[i]) {
        pair<long long, long long> l = lca(ed[i].u, ed[i].v);
        res += max(0ll, l.S - ed[i].w);
    }
    cout << res;
}
int main() {
    ios_base::sync_with_stdio(0);
    freopen("upgranet.inp", "r", stdin);
    freopen("upgranet.out", "w", stdout);
    cin >> n >> m;
    fort(i, 1, m)
    cin >> ed[i].u >> ed[i].v >> ed[i].w;
    solve();
}

```

### 5.6.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.7. Bài 7: Dạo chơi đồng cỏ (PWALK – Spoj)

### 5.7.1. Đề bài: PWALK

Có  $N$  con bò ( $1 \leq N \leq 10^5$ ), để thuận tiện ta đánh số từ  $1 \rightarrow N$ , đang ăn cỏ trên  $N$  đồng cỏ, để thuận tiện ta cũng đánh số các đồng cỏ từ  $1 \rightarrow N$ . Biết rằng con bò  $i$  đang ăn cỏ trên đồng cỏ  $i$ .

Một vài cặp đồng cỏ được nối với nhau bởi 1 trong  $N - 1$  con đường 2 chiều mà các con bò có thể đi qua. Con đường  $i$  nối 2 đồng cỏ  $A_i$  và  $B_i$  ( $1 \leq A_i, B_i \leq N$ ) và có độ dài  $L_i$  ( $1 \leq L_i \leq 10^4$ ).

Các con đường được thiết kế sao cho với 2 đồng cỏ bất kỳ đều có duy nhất 1 đường đi giữa chúng. Như vậy các con đường này đã hình thành 1 cấu trúc cây.

Các chú bò rất có tinh thần tập thể và muốn được thăm thường xuyên. Vì vậy lũ bò muốn bạn giúp chúng tính toán độ dài đường đi giữa  $Q$  ( $1 \leq Q \leq 1000$ ) cặp đồng cỏ (mỗi cặp được mô tả là 2 số nguyên  $u, v$  ( $1 \leq u, v \leq N$ )).

**Dữ liệu vào:** Nhập vào từ tệp PWALK.inp

Dòng đầu ghi 2 số nguyên cách nhau bởi dấu cách:  $N$  và  $Q$

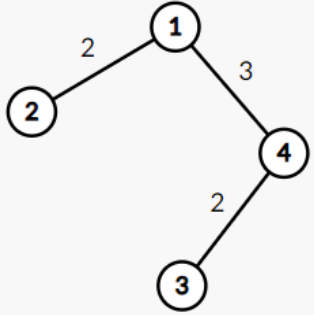
$N - 1$  dòng tiếp theo: Mỗi dòng chứa 3 số nguyên cách nhau bởi dấu cách:  $A_i, B_i$  và  $L_i$ , mô tả có đường đi trực tiếp giữa  $A_i, B_i$  và độ dài của nó là  $L_i$

$Q$  dòng tiếp theo: Mỗi dòng chứa 2 số nguyên  $(u, v)$  khác nhau yêu cầu tính toán độ dài 2 đồng cỏ mà lũ bò muốn đi thăm qua lại.

**Kết quả ra:** Ghi ra tệp PWALK.out

Ghi  $Q$  số là kết quả của từng yêu cầu theo thứ tự, mỗi số viết trên một dòng.

**Ví dụ:**

PWALK.inp	PWALK.out	Giải thích
4 2 2 1 2 4 3 2 1 4 3 1 2 3 2	2 7	 <p>Đường đi từ 1 → 2 độ dài 2. Đường đi từ 3 đến 2 độ dài 7.</p>

### 5.7.2. Phân tích đề bài và đề xuất thuật toán

Đây là bài toán cũng khá đơn giản, thực chất chỉ là truy vấn tính tổng độ dài trên đường đi giữa 2 đỉnh trên cây.

Giới hạn của bài toán được tác giả điều chỉnh tăng lên so với đề gốc (đề thi USACO 2008) khi đó việc giải các bài toán về đồ thị dạng cây chưa phổ biến trong các cuộc thi lập trình thi đấu.

Để giải quyết bài toán trên ta cần bổ sung thêm cập nhật thông tin khoảng cách đến gốc (tính theo độ dài đường đi) khi duyệt DFS (dạng bài toán quy hoạch động trên cây).

Độ phức tạp bài toán này là  $O(N * \log(N) + Q * \log(N))$

Chương trình tham khảo:

```
#include<bits/stdc++.h>
using namespace std;
struct edge {
    int u,w;
};
int n,q,up[100005][18],depth[100005],len[100005];
vector<edge> adj[100005];
void DFS(int u, int p, int w) {
    len[u]=len[p]+w; ///khoảng cách từ u đến gốc
    depth[u]=depth[p]+1; ///độ sâu
    up[u][0]=p; ///sparse table
    for(int i=1; i<17; i++)
        up[u][i]=up[up[u][i-1]][i-1];
    for(auto v:adj[u]) {
        if(v.u!=p)
            DFS(v.u,u,v.w);
    }
}
int lca(int u, int v) {
    if(depth[u]<depth[v])
        swap(u,v);
    for(int i=16; i>=0; i--) ///Binary lifting
```

```

        if (depth[up[u][i]] >= depth[v])
            u = up[u][i];
        if (u == v)
            return u;
        for (int i = 16; i >= 0; i--) // Binary lifting
            if (up[u][i] != up[v][i]) {
                u = up[u][i];
                v = up[v][i];
            }
        return up[u][0];
    }
    int main() {
        ios_base::sync_with_stdio(0);
        // freopen("PWALK.inp", "r", stdin);
        cin >> n >> q;
        for (int i = 1; i < n; i++) {
            int u, v, w;
            cin >> u >> v >> w;
            adj[u].push_back({v, w});
            adj[v].push_back({u, w});
        }
        DFS(1, 0, 0);
        while (q--) {
            int u, v;
            cin >> u >> v;
            cout << (len[u] + len[v] - 2 * len[lca(u, v)]) << "\n";
        }
    }
}

```

### 5.7.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

Mở rộng một chút bài 7, ta có bài toán sau:

## 5.8. Bài 8: TREEDGE

### 5.8.1. Đề bài TREEDGE

Cho một cây có trọng số gồm  $N$  đỉnh,  $N-1$  cạnh, đỉnh gốc là đỉnh 1. Có  $Q$  truy vấn, mỗi truy vấn cho dưới dạng  $(u, v, x)$  hỏi đường đi có trọng số lớn nhất giữa cặp  $(u, v)$  trên cây nếu được phép nối một đỉnh thuộc cây con gốc  $u$  với một đỉnh cây con gốc  $v$  (đảm bảo cây con gốc  $u$ , và cây con gốc  $v$  là khác nhau) bởi một cạnh có trọng số là  $x$  ( $x \geq 0$ ) bằng bao nhiêu?

**Dữ liệu vào:** Từ tệp TREEDGE.inp

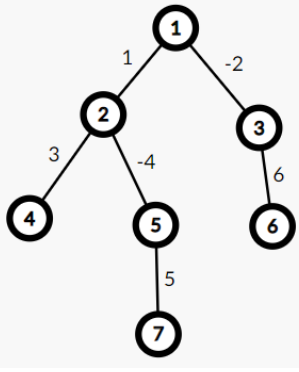
Dòng đầu ghi số  $N, Q$  ( $1 \leq N, Q \leq 2 * 10^5$ ) là số đỉnh của cây, số truy vấn.

$N - 1$  dòng tiếp ghi bộ số  $u, v, w$  ( $1 \leq u, v \leq N, |w| \leq 10^9$ ) mô tả các cạnh của cây, đỉnh đầu  $u$ , đỉnh cuối  $v$ , trọng số  $w$  của cạnh.

$Q$  dòng tiếp theo ghi bộ số  $u, v, x$  thể hiện truy vấn tìm trọng số đường đi từ  $u$  đến  $v$  lớn nhất khi được thêm một cạnh trọng số  $x$  ( $0 \leq x \leq 10^9$ ) nối một đỉnh thuộc cây con gốc  $u$  với 1 đỉnh thuộc cây con gốc  $v$ .

**Kết quả ra:** Ghi ra tệp TREEDGE.out

**Ví dụ:**

TREEDGE.inp	TREEDGE.out	Giải thích
7 3	10	<p>Với truy vấn đầu tiên (2,3,1): Tối ưu khi thêm cạnh nối giữa 4 và 6. Tạo ra đường đi <math>2 \rightarrow 4 \rightarrow 6 \rightarrow 3</math> với tổng trọng số: <math>3+1+6=10</math>.</p> 
1 2 1	7	
1 3 -2	5	
2 4 3		
2 5 -4		
5 7 5		
3 6 6		
2 3 1		
5 4 2		
5 6 0		

### 5.8.2. Phân tích đề bài và đề xuất thuật toán

Nhận xét:

Đồ thị dạng cây, nên nếu không nối thêm cạnh thì đường đi  $u \rightarrow v$  là duy nhất và ta tính trọng số giống bài 7.

$$\text{dist}(u, v) = \text{len}(u) + \text{len}(v) - 2 * \text{len}(\text{LCA}(u, v))$$

Với  $\text{len}(u)$  là khoảng cách đến  $u \rightarrow \text{root}$ .

Trong bài này được phép bổ sung thêm một cạnh có trọng số  $x$ , nên ta quan tâm thêm một đường đi nữa từ  $u$  đến con của nó sang con của  $v$  rồi đến  $v$ . Để tính nhanh tổng trọng số đường này, ta áp dụng dạng bài quy hoạch động trên cây (dạng cơ bản) để cập nhật đường đi lớn nhất từ một đỉnh  $u$  đến con của nó.

$$\text{dist2}(u, v) = \text{len2}(u) + \text{len2}(v) + x$$

Với  $\text{len2}(u)$  là đường đi lớn nhất từ  $u$  đến con của nó.

Kết quả của bài toán là:  $\max\{\text{dist}(u, v), \text{dist2}(u, v)\}$

Độ phức tạp thuật toán:

- Tiền xử lí để tính khoảng cách với LCA:  $O(N * \log(N))$
- Tiền xử lí để tính độ dài đường khi thêm cạnh:  $O(N)$ .
- Để tính đường đi thứ nhất:  $O(\log(N))$
- Để tính đường đi thứ hai:  $O(1)$ .

**Chương trình tham khảo:**

```
#include<bits/stdc++.h>
#define maxn 200005
#define int long long
using namespace std;
struct edge {
    int u,w;
};
int T,n,q,up[maxn][20],depth[maxn],len[maxn], len2[maxn];
vector<edge> adj[maxn];
void DFS(int u, int p, int w) {
    len[u]=len[p]+w;
```



```
depth[u]=depth[p]+1;
up[u][0]=p;
for(int i=1; i<=18; i++)
    up[u][i]=up[up[u][i-1]][i-1];
for(auto v:adj[u]) {
    if(v.u!=p)
        DFS(v.u,u,v.w);
}
}
void DFS2(int u, int p, int w) {
    len2[u]=0;
    for(auto v:adj[u]) {
        if(v.u!=p) {
            DFS2(v.u,u,v.w);
            len2[u]=max(len2[u],len2[v.u]+v.w);
        }
    }
}
int lca(int u, int v) {
    if(depth[u]<depth[v])
        swap(u,v);
    for(int i=18; i>=0; i--)
        if(depth[up[u][i]]>=depth[v])
            u=up[u][i];
    if(u==v)
        return u;
    for(int i=18; i>=0; i--)
        if(up[u][i]!=up[v][i]) {
            u=up[u][i];
            v=up[v][i];
        }
    return up[u][0];
}
main() {
    ios_base::sync_with_stdio(0);
    // freopen("TREEDGE.inp","r",stdin);
    cin>>n>>q;
    for(int i=1; i<=n; i++)
        adj[i].clear();
    for(int i=1; i<n; i++) {
        int u,v,w;
        cin>>u>>v>>w;
        adj[u].push_back({v,w});
        adj[v].push_back({u,w});
    }
    DFS(1,0,0);
    DFS2(1,0,0);
    while(q--) {
        int u,v,x;
        cin>>u>>v>>x;
        int res1=len[u]+len[v]-2*len[lca(u,v)];
        int res2=len2[u]+len2[v]+x;
        cout<<max(res1,res2)<<"\n";
    }
}
```

### 5.8.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.9. Bài 9: Đường đi qua K cạnh

### 5.9.1. Đề bài: TREEQ

Cho một cây gồm  $N$  đỉnh,  $N-1$  cạnh. Có  $Q$  truy vấn, mỗi truy vấn cho bởi bộ  $(x, y, a, b, k)$  hỏi rằng có cách nào đi từ  $a \rightarrow b$  qua đúng  $k$  cạnh nếu cây được nối thêm cạnh  $(x, y)$ , đường đi có thể qua các đỉnh, các cạnh nhiều lần.

**Dữ liệu vào:** Từ tệp TREEQ.inp

Dòng đầu tiên là ghi số  $N, Q$  ( $3 \leq N, Q \leq 10^5$ ) là số đỉnh, số truy vấn.

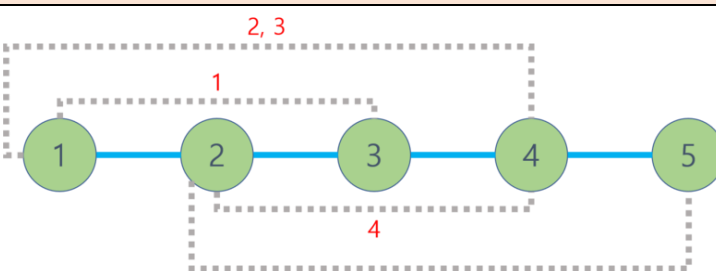
$N-1$  dòng tiếp theo thể hiện các cạnh của cây.

$Q$  dòng tiếp theo thể hiện các truy vấn mô tả như trên.

**Kết quả ra:** Ghi ra tệp TREEQ.out

Ghi ra  $Q$  dòng trả lời cho mỗi truy vấn, nếu tồn tại đường đi thỏa mãn thì ghi 1, ngược lại thì ghi 0.

Ví dụ:

TREEQ.inp	TREEQ.out	Giải thích
5	1	 <p>Truy vấn 1: Đi như sau: <math>1 \rightarrow 3 \rightarrow 2</math>  Truy vấn 2: Đi như sau: <math>1 \rightarrow 2 \rightarrow 3</math>  Truy vấn 4: Đi như sau:  <math>3 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 3</math></p>
1 2	1	
2 3	0	
3 4	1	
4 5	0	
5		
1 3 1 2 2		
1 4 1 3 2		
1 4 1 3 3		
4 2 3 3 9		
5 2 3 3 9		

### 5.9.2. Phân tích đề bài và đề xuất thuật toán

**Nhận xét 1:** Khi chưa thêm cạnh  $(x, y)$  thì giữa  $a, b$  luôn có đường đi qua  $k$  cạnh nếu

$$\begin{cases} dist(a, b) \leq k \text{ (đường đi trực tiếp } a \rightarrow b \text{ phải nhỏ hơn } k) \\ dist(a, b) \% 2 == k \% 2 \text{ (khoảng cách } a \rightarrow b \text{ cùng tính chẵn lẻ với } k) \end{cases}$$

Khi  $dist(a, b) \leq k$  và  $dist(a, b)$  cùng tính chẵn lẻ với  $k$  thì để đi từ  $a \rightarrow b$  qua đúng  $k$  cạnh thì ta chỉ cần lặp đi lặp lại một cạnh với số lần phù hợp thì sẽ đảm bảo yêu cầu của bài toán.

**Nhận xét 2:** Khi nối thêm cạnh  $(x, y)$  thì hiển nhiên ta có thêm cơ hội để thay đổi tính chẵn lẻ của đường đi  $a \rightarrow b$ . Khi đó ta quan tâm đến cách đi sau:

Cách 1: Đi từ  $a \rightarrow x$  mà không qua cạnh nối thêm, rồi đến  $y$  và từ  $y$  về  $b$  không qua cạnh nối thêm.

Cách 2: Đi từ  $a \rightarrow y$  mà không qua cạnh nối thêm, rồi đến  $x$ , và từ  $x$  về  $b$  không qua cạnh nối thêm.

Cách 3: Đi từ  $a \rightarrow b$  theo cách đi trên cây ban đầu.

Trong 3 cách đi, nếu có cách đi nào thỏa mãn ràng buộc trong nhận xét 1 thì bài toán có kết quả là 1, ngược lại thì là 0.

Bài toán này đương nhiên cần kết hợp cách giải bài toán LCA.

Độ phức tạp thuật toán:  $O(N * \log(N) + Q * \log(N))$

Chương trình tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
const int MAX_N = 100005;
const int LIM = 17;
const int INF = (int)1e9+7;
vector<int> adj[MAX_N];
int depth[MAX_N];
int up[MAX_N][LIM+1];
void DFS(int u, int p) {
    depth[u] = depth[p]+1;
    up[u][0] = p;
    for(int i = 1; i <= LIM; i++)
        up[u][i] = up[up[u][i-1]][i-1];
    for(int x : adj[u])
        if(x != p)
            DFS(x, u);
}
int lca(int a, int b) {
    if(depth[a] > depth[b])
        swap(a, b);
    for(int i = LIM; i >= 0; i--) {
        if(depth[up[b][i]] >= depth[a]) {
            b = up[b][i];
        }
    }
    if(a == b)
        return a;
    for(int i = LIM; i >= 0; i--) {
        if(up[a][i] != up[b][i]) {
            a = up[a][i];
            b = up[b][i];
        }
    }
    return up[a][0];
}
int dist(int u, int v) {
    return depth[u]+depth[v]-2*depth[lca(u,v)];
}
int main() {
    ios::sync_with_stdio(0);
    // freopen("TREEQ.inp", "r", stdin);
    int n, q;
```

```
cin >> n;
for(int i = 0; i < n - 1; i++) {
    int u, v;
    cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u);
}
DFS(1, 0);
cin >> q;
while(q--) {
    int x, y, a, b, k;
    cin >> x >> y >> a >> b >> k;
    int without=dist(a,b); ///chua them canh
    int with=min(dist(a,x)+dist(y,b),dist(a,y)+dist(x,b))+1;
    int ans = INF;
    if(without % 2 == k % 2)
        ans = without;
    if(with % 2 == k % 2)
        ans = min(ans, with);
    cout << (ans <= k ? "1" : "0") << '\n';
}
}
```

### 5.9.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.10. Bài 10: Tom & Jerry

### 5.10.1. Đề bài

Chuột Jerry khá nhanh nhẹn và thông minh, mỗi lần chơi đuổi bắt với mèo Tom thì ban đầu Jerry đều cố gắng thoát khỏi sự đuổi bắt của Tom lâu nhất có thể, trong trường hợp bị mèo Tom tóm được, Jerry bao giờ cũng nghĩ ra cách để troll mèo Tom. Lần này cũng vậy, Tom và Jerry chơi đuổi bắt trong một ngôi nhà có  $N$  căn phòng, mỗi căn phòng chỉ có một con đường duy nhất nối giữa chúng, từ hai căn phòng bất kì luôn chỉ có một hành lang giữa chúng (hay nói cách khác nó có dạng đồ thị cây  $N$  đỉnh,  $N-1$  cạnh, mỗi căn phòng là một đỉnh, hành lang là cạnh và có độ dài như nhau).

Mỗi một đơn vị thời gian thì Tom và Jerry có thể lựa chọn chạy từ phòng này sang phòng kia, hoặc có thể đứng im trong phòng, biết rằng chúng nhìn thấy nhau trong cả ngôi nhà. Khi Tom và Jerry cùng ở một phòng, thì Jerry lại phải nghĩ cách troll Tom để chạy thoát. Cho  $Q$  truy vấn, mỗi truy vấn cho vị trí của Tom và Jerry, hỏi cuộc đua đuổi được lâu nhất là bao nhiêu thì Jerry lại phải troll Tom.

**Dữ liệu vào:** Đọc vào từ tệp Tom.inp

Dòng đầu là số  $N, Q$  ( $1 \leq N, Q \leq 10^5$ ).


$N-1$  dòng tiếp theo, mỗi dòng chứa cặp  $(u, v)$  là có hành lang nối 2 phòng.

$Q$  dòng tiếp theo mô tả truy vấn, mỗi dòng chứa cặp  $(u, v)$  là vị trí của Tom và Jerry tương ứng.

**Kết quả ra:** Ghi ra tệp Tom.out

Ghi Q số là thời gian lâu nhất Tom và Jerry chạy dượt đuổi tương ứng với các truy vấn đã cho.

**Ví dụ:**

Tom.inp	Tom.out	Giải thích
3 2	2	 <p>Truy vấn 1: Tom ở 1, Jerry ở 2. Jerry sẽ chạy sang 3. Sau đó đứng tại đấy. Tom chạy sang 2, rồi sang 3 thì bắt được Jerry</p>
1 2	1	
2 3		
1 2		
2 3		
		Truy vấn 2: Jerry chỉ đứng yên và Tom chạy đến bắt.

### 5.10.2. Phân tích đề bài và đề xuất thuật toán

Nhận xét: Tại một thời điểm bất kì xét đường đi từ Tom (T) đến Jerry (J) thì thấy rằng tất cả các vị trí bên từ giữa đến J thì Jerry đều có thể chạy đến mà không bị bắt. Do đó để kéo dài trò chơi thì J nên chọn vị trí xa vị trí giữa nhất tính về nửa bên phải của mình (như hình).



Các cách chưa tối ưu (duyet trâu), tôi sẽ không đề cập ở đây.

Như nhận xét, ta thấy để giải quyết mỗi truy vấn, ta cần giải quyết 2 bài toán nhỏ:

- Bài toán 1: Tìm điểm ở giữa nhanh nhất (cần tính khoảng cách  $T \rightarrow J$ , cần xác định nhanh đỉnh nào ở giữa,...), đây liên quan đến LCA. Độ phức tạp  $O(\log(N))$ , nhưng tiền xử lí cho toàn bài trong  $O(N \cdot \log(N))$ .

Ta sẽ tính khoảng cách  $T \rightarrow J$

$$dist = depth[t] + depth[j] - depth[lca(t, j)] * 2$$

Khoảng cách từ  $T \rightarrow mid$ :  $distT = dist/2$

Khoảng cách từ  $J \rightarrow mid$ :  $distJ = (dist - 1)/2$

- Bài toán 2: Xét trong nửa bên phải đường đi phía J, cần tìm điểm xa điểm giữa nhất, đây là bài toán quy hoạch động trên cây. Độ phức tạp mỗi truy vấn  $O(1)$ , nhưng tiền xử lí cho toàn bài trong  $O(N)$ .

Ta sẽ dùng DFS để duyệt cây, cập nhật khoảng cách xa nhất từ dưới lên. Với mỗi đỉnh  $u$  cần có được thông tin đỉnh xa nhất trong nhánh DFS gốc  $u$ , và thông tin đỉnh xa nhất từ  $u$  ra ngoài nhánh DFS gốc  $u$ .

Trường hợp 1:  $depth[T] > depth[J]$

Kết quả sẽ là khoảng cách từ  $T \rightarrow mid$  cộng với khoảng cách xa nhất từ  $mid$  ra ngoài cây con DFS gốc  $mid$

$$res1 = distT + up[mid]$$

Trường hợp 2:  $depth[T] < depth[J]$

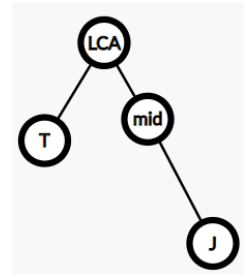
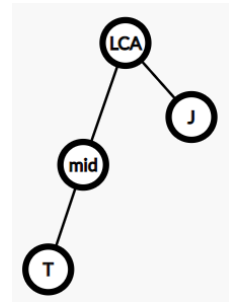
Kết quả sẽ là khoảng cách từ  $J \rightarrow mid$  cộng với khoảng cách xa nhất từ  $mid$  tới đỉnh trong cây con DFS gốc  $mid$

$$res2 = distJ + down[mid]$$

Độ phức tạp thuật toán:  $O(N \cdot \log(N) + Q \cdot \log(N))$

Chương trình tham khảo:

```
#include <bits/stdc++.h>
using namespace std;
#define L 20
#define N 100005
int n,q,depth[N],anc[N][L],downj[N],downl[N],down2[N],up[N];
vector<int> adj[N];
int ascend(int u, int d) { ///to tien thu d cua u
    for(int k = 0; k < L; k++) {
        if(d == 0)
            break;
        if(d & 1)
            u = anc[u][k];
        d >>= 1;
    }
    return u;
}
int lca(int u, int v) {
    if(depth[u]<depth[v])
        swap(u,v);
    for(int i=17; i>=0; i--) ///Binary lifting
        if(depth[anc[u][i]]>=depth[v])
            u=anc[u][i];
    if(u==v)
        return u;
    for(int i=17; i>=0; i--) ///Binary lifting
        if(anc[u][i]!=anc[v][i]) {
            u=anc[u][i];
            v=anc[v][i];
        }
    return anc[u][0];
}
void DFS(int u, int p) {
    depth[u]=depth[p]+1;
    anc[u][0]=p; ///sparse table
    for(int i=1; i<17; i++)
        anc[u][i]=anc[anc[u][i-1]][i-1];
}
```



```
        for(int v:adj[u]) {
            if(v==p)
                continue;
            DFS(v,u);
            int d=down1[v]+1;
            if(d>down1[u]) {
                down2[u]=down1[u];
                down1[u]=d;
                downj[u]=v;
            } else
                if(d>down2[u]) {
                    down2[u]=d;
                }
        }
    }
}

void DFS2(int u, int p) {
    for(int v:adj[u]) {
        if(v==p)
            continue;
        up[v]=up[u]+1;
        if(downj[u]==v)
            up[v]=max(up[v], down2[u]+1);
        else
            up[v]=max(up[v], down1[u]+1);
        DFS2(v,u);
    }
}

int answer_query(int t, int j) {
    if(t == j)
        return 0;
    int dist = depth[t] + depth[j] - depth[lca(t,j)] * 2;
    int distj = (dist - 1) >> 1;
    int distt = dist >> 1;
    return distt + (depth[t] > depth[j] ? up[ascend(t, distt)] :
down1[ascend(j, distj)]+ 1);
}

int main() {
    scanf("%d%d", &n, &q);
    for(int i = 0; i < n; i++) {
        adj[i].clear();
    }
    for(int i = 1; i < n; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        adj[a].push_back(b);
        adj[b].push_back(a);
    }
    DFS(1,0);
    DFS2(1,0);
    while(q--) {
        int t, j;
        scanf("%d%d", &t, &j);
        printf("%d\n", answer_query(t, j));
    }
}
```

### 5.10.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.11. Bài 11: Cập nhật thông tin trên cây 1

### 5.11.1. Đề bài: Update tree

Cho một cây có  $N$  đỉnh. Ban đầu đỉnh  $i$  có giá trị  $a_i$ . Có  $Q$  truy vấn, mỗi truy vấn tăng tất cả các đỉnh nằm trên đường đi từ  $u$  đến  $v$  một lượng là  $x$ . Hãy in ra giá trị của các đỉnh của cây sau khi xử lý hết truy vấn.

**Dữ liệu vào:** Đọc dữ liệu từ tệp UPDTREE.inp

Dòng đầu là số  $N$  ( $1 \leq N \leq 10^5$ ).

Dòng tiếp theo ghi  $N$  số  $a_1, a_2, \dots, a_N$  ( $1 \leq a_i \leq 10^4$ ) là giá trị ban đầu của các đỉnh.

$N-1$  dòng tiếp theo ghi các cặp  $(u, v)$  thể hiện cạnh của cây.

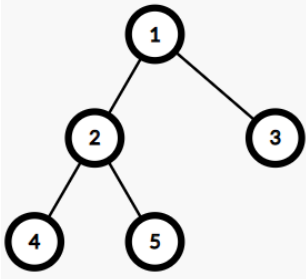
Dòng tiếp theo ghi số  $Q$  ( $1 \leq Q \leq 10^5$ ).

$Q$  dòng tiếp theo ghi bộ số  $(u, v, x)$  với  $1 \leq u, v, x \leq N$  thể hiện truy vấn.

**Kết quả ra:** Ghi ra tệp UPDTREE.out

Ghi ra một dòng  $N$  số là giá trị tại các đỉnh sau khi thực hiện  $Q$  truy vấn.

**Ví dụ:**

UPDTREE.inp	UPDTREE.out	Giải thích
5 0 0 0 0 0 1 2 1 3 2 4 2 5 2 4 5 2 3 5 1	1 3 1 2 3	 <p>Sau truy vấn 1: 0 2 0 2 2                      Sau truy vấn 2: 1 3 1 2 3</p>

### 5.11.2. Phân tích đề bài và đề xuất thuật toán

\*) Nhận xét:

Xét bài toán: Cho một mảng 1 chiều có  $N$  số  $a_1, a_2, \dots, a_N$  ban đầu đều bằng 0, có các truy vấn tăng đoạn  $[u, v]$  lên  $x$ . Khi đó mỗi truy vấn ta có thể gán  $a[u] = a[u] + x$  (tăng đầu đoạn) và gán  $a[v + 1] = a[v + 1] - x$ , thì sau  $Q$  truy vấn, ta thực hiện cộng dồn từ  $a_1 \rightarrow a_N$  thì ta được một mảng là dãy số cuối cùng.

\*) Tương tự, với cây thì ta thấy:

- Dữ liệu giá trị cho ban đầu của các đỉnh chỉ có tác dụng làm tăng, giảm độ khó của bài toán.

- Mỗi truy vấn tăng giá trị các đỉnh trên đường đi  $u$  đến  $v$  chắc chắn ta không thể tăng luôn tất cả các giá trị trên đường đi. Do đó việc lưu trữ thông tin để truy



vết đường đi là không hợp lí. Vì đường đi giữa các cặp  $(u, v)$  là duy nhất, nên mỗi truy vấn thay đổi giá trị các đỉnh ta cần phải lưu thông tin tại ít nhất ở 3 đỉnh là:  $u, v, LCA(u, v)$ .

- Thông tin có thể cập nhật từ đỉnh đầu đỉnh cuối về LCA kiểu cộng dồn từ các đỉnh con trong quá trình duyệt DFS.

Do vậy, mỗi truy vấn  $(u, v)$  ta sẽ tăng đỉnh giá trị tại đỉnh  $u, v$  thêm lượng  $x$ ; giảm giá trị tại đỉnh  $LCA(u, v)$  và cha của nó một lượng  $x$ .

Khi đó lúc duyệt DFS để cập nhật thông tin từ dưới lên ta sẽ giải quyết được các truy vấn (dạng quy hoạch động trên cây).

Độ phức tạp thuật toán:

- Tạo bảng thừa để Binary lifting:  $O(N \cdot \log(N))$ .
- Xử lí các truy vấn  $O(Q \cdot \log(N))$
- Cập nhật giá trị tăng thêm tại các đỉnh  $O(N)$ .

Chương trình tham khảo:

```
#include<bits/stdc++.h>
#define maxn 100005
#define int long long
using namespace std;
int n,q,a[maxn],visited[maxn],b[maxn],up[maxn][20],depth[maxn];
vector<int> adj[maxn];
void dfs(int u, int p) {
    depth[u]=depth[p]+1;
    up[u][0]=p;
    for(int i=1; i<=17; i++)
        up[u][i]=up[up[u][i-1]][i-1];
    for(int v:adj[u]) {
        if(v==up[u][0])
            continue;
        dfs(v,u);
    }
}
void dfs2(int u, int p) {
    for(int v:adj[u]) {
        if(v==p)
            continue;
        dfs2(v,u);
        a[u]=a[u]+a[v];
    }
}
int lca(int u, int v) {
    if(depth[u]<depth[v])
        swap(u,v);
    for(int i=16; i>=0; i--)
        if(depth[up[u][i]]>=depth[v])
            u=up[u][i];
    if(u==v)
        return u;
    for(int i=16; i>=0; i--)
        if(up[u][i]!=up[v][i])
            u=up[u][i],v=up[v][i];
}
```

```

        return up[u][0];
    }
    main() {
        ios_base::sync_with_stdio(0);
        // freopen("UPDTREE.inp", "r", stdin);
        cin >> n;
        for (int i = 1; i <= n; i++) {
            cin >> b[i];
        }
        for (int i = 1; i < n; i++) {
            int u, v;
            cin >> u >> v;
            adj[u].push_back(v);
            adj[v].push_back(u);
        }
        dfs(1, 0);
        cin >> q;
        while (q--) {
            int u, v, x;
            cin >> u >> v >> x;
            a[u] = a[u] + x, a[v] = a[v] + x;
            int l = lca(u, v);
            a[l] -= x, a[up[l][0]] -= x;
        }
        dfs2(1, 0);
        for (int i = 1; i <= n; i++)
            cout << a[i] + b[i] << " ";
    }
}

```

### 5.11.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtb?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtb?usp=sharing)

Bài này có thể kết hợp thêm yêu cầu trả lời tính tổng trên đường đi  $u \rightarrow v$ , bài toán khi đó không khó hơn mà chỉ là việc code dài hơn, nhưng kết hợp với việc tổng hợp thông tin trên đường đi từ cha xuống con trước khi trả lời truy vấn tính tổng.

## 5.12. Bài 12: Cập nhật thông tin trên cây 2

### 5.12.1. Đề bài: Update tree2

Nâng cấp bài UPDTREE, tương tự bài trên, ta có cây  $N$  đỉnh,  $Q$  truy vấn, mỗi truy vấn cho bởi bộ 4 số  $(A, B, C, D)$  tăng trọng số của các **cạnh** trên đường đi từ  $A$  đến  $B$  lên 1 đơn vị nhưng không tăng trọng số của các cạnh mà nằm trên trường đi từ  $C$  đến  $D$ .

Sau đó là  $P$  truy vấn tính tổng trọng số các cạnh trên đường đi từ  $E$  đến  $F$ .

**Dữ liệu vào:** Đọc vào từ tệp UPDTREE2.inp

Dòng đầu là số  $N, Q, P$  ( $1 \leq N, Q, P \leq 10^5$ ).

$N-1$  dòng tiếp theo ghi các cặp  $(u, v)$  thể hiện cạnh của cây.

$Q$  dòng tiếp theo ghi 4 số  $A, B, C, D$  với  $1 \leq A, B, C, D \leq N$  thể hiện truy vấn tăng trọng số cạnh

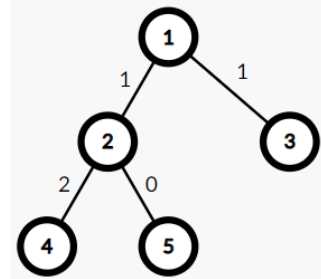
P dòng tiếp theo ghi 2 số  $E, F$  ( $1 \leq E, F \leq N$ ) thể hiện truy vấn tính tổng.

**Kết quả ra:** Ghi ra tệp UPDTREE2.out

Ghi ra một dòng gồm P số tương ứng P truy vấn tính tổng.

**Ví dụ:**

UPDTREE2.inp	UPDTREE2.out	Giải thích
5 2 2 1 2 2 4 2 5 1 3 1 4 2 3 3 4 2 5 4 5 4 3	2 4	Sau 2 truy vấn cập nhật, các cạnh có trọng số là: $(1,2)=1; (1,3)=1;$ $(2,4)=2; (2,5)=0$

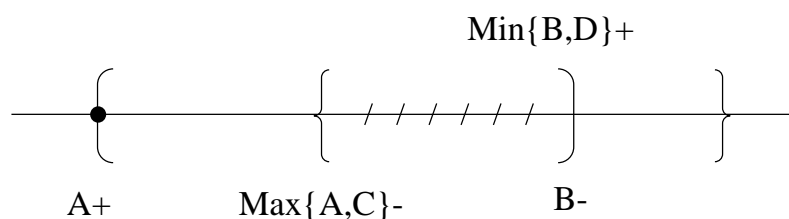


### 5.12.2. Phân tích đề bài và đề xuất thuật toán

Xét bài toán tương tự trên mảng 1 chiều truy vấn cập nhật cho bởi bộ 4 số  $(A, B, C, D)$ . Tăng mỗi vị trí trong đoạn  $[A, B]$  lên 1 nhưng không tăng trong đoạn  $[C, D]$ .

- Nếu 2 đoạn giao nhau thì mỗi truy vấn ta thực hiện ta tăng tại  $A$ , giảm tại  $\max(A, C)$ , tăng tại  $\min(B, D)$  và giảm tại  $B$ .

Ví dụ:  $A < C < B < D$



- Nếu  $B < C$  hoặc  $D < A$  thì 2 đoạn không giao nhau, thực hiện tương tự bài toán ví dụ của bài trước. Tăng tại vị trí  $A$ , giảm tại vị trí  $B+1$ . Hay khi đó thì

$$\max\{A, C\} > \min\{B, D\}$$

Kết hợp với cách làm của bài UPDTREE, ta có thể làm như sau, tăng đoạn  $[A, B]$  ban đầu và xem có những cạnh nào giao với  $[C, D]$  thì ta trừ đi. Để cập nhật trọng số các cạnh, ta vẫn lưu giá trị vào đỉnh, đỉnh thể hiện thông tin cần cập nhật lên cha của nó khi DFS.

Với việc tăng các cạnh từ  $A$  đến  $B$  ta thực hiện:

$$\begin{cases} value[A] ++ \\ value[B] ++ \\ value[lca(A, B)] -- \end{cases} = 2$$

Gọi  $anc1 = lca(A, B), anc2 = lca(C, D)$ . Ta cần loại bỏ thông tin tăng trọng số cạnh trên cạnh giao nhau của 4 cặp đoạn sau:

Cặp 1: [A,anc1] với [C,anc2]

Cặp 2: [A,anc1] với [D,anc2]

Cặp 3: [B,anc1] với [C,anc2]

Cặp 4: [B,anc1] với [D,anc2]

Xử lý 4 cặp là tương tự nhau. Ví dụ ta sẽ xử lý cặp 1 như sau:

- Nếu anc2 không nằm trên đường đi từ  $A \rightarrow anc1 \rightarrow root$  thì 2 đoạn đó không giao nhau. Dễ thấy, vì nếu ngược lại thì sẽ xuất hiện chu trình.

- Gọi  $x = lca(A, C)$  khi đó bài toán sẽ là tìm giao  $[A, anc1]$  với  $[x, anc2]$ . Khi này việc xử lý giống hết bài toán trên mảng 1 chiều đã xét ở trên.

- Độ phức tạp chung của thuật toán là:  $O(N \cdot \log(N) + Q \cdot \log(N))$

Chương trình tham khảo:

```
#include<bits/stdc++.h>
typedef unsigned long long ll;
using namespace std;
vector<int> adj[100005], tree[100005], depth, euler;
bool visited[100005];
int first[100005], last[100005];
ll value[100005];
int N, Q, P;
void dfs(int v, int level) {
    visited[v] = true;
    euler.push_back(v), depth.push_back(level);
    first[v] = (int)euler.size()-1;
    for(int u : adj[v]) {
        if(!visited[u]) {
            tree[v].push_back(u);
            dfs(u, level + 1);
            euler.push_back(v), depth.push_back(level);
        }
    }
    last[v] = (int)euler.size()-1;
}
vector<int> log_table;
vector<vector<pair<int, int>>> sT(18, vector<pair<int, int>>());
void build_sparse() {
    log_table.resize(depth.size()+1);
    for(int i = 2; i < log_table.size(); i++)
        log_table[i] = log_table[i/2] + 1;
    for(int i = 0; i < depth.size(); ++i)
        sT[0].push_back({depth[i], i});
    for(int i=1; i<18; i++) {
        sT[i].resize(depth.size());
        for(int j = 0; (j + (1<<i) <= sT[i].size()); ++j) {
            sT[i][j] = min(sT[i-1][j], sT[i-1][j+(1<<(i-1))]);
        }
    }
}
int lca(int v, int u) {
    if(v == u)
        return v;
```

```
int l = min(first[v], first[u]);
int r = max(first[v], first[u]);
int row = log_table[r-l];
return euler[min(sT[row][l], sT[row][r-(1<<row)])].second];
}
bool is_ancestor(int u, int v) {
    if(u == v)
        return true;
    return (first[u] < first[v] && last[v] < last[u]);
}
pair<int, int> intersection(int a, int anc1, int c, int anc2) {
    if(a != anc1 && c != anc2 && is_ancestor(anc2, a)) {
        int ac = lca(a, c);
        if(is_ancestor(anc1, anc2))
            swap(anc1, anc2);
        if(is_ancestor(anc1, ac))
            return {ac, anc1};
    }
    return {0, 0};
}
void update(int A, int B, int dx) {
    if(A > B)
        swap(A, B);
    value[A]+=dx,value[B]+=dx,value[lca(A, B)]-=2*dx;
}
void update_edges() {
    int A, B, C, D;
    for(int i=0; i<Q; i++) {
        cin>>A>>B>>C>>D;
        update(A, B, 1);
        /* handle intersection */
        int anc1 = lca(A, B);
        int anc2 = lca(C, D);
        auto path1 = intersection(A, anc1, C, anc2);
        auto path2 = intersection(A, anc1, D, anc2);
        auto path3 = intersection(B, anc1, C, anc2);
        auto path4 = intersection(B, anc1, D, anc2);
        update(path1.first, path1.second, -1);
        update(path2.first, path2.second, -1);
        update(path3.first, path3.second, -1);
        update(path4.first, path4.second, -1);
    }
}
ll diff(int v) {
    for(int u : tree[v])
        value[v] += diff(u);
    return value[v];
}
void build_partial(int v, ll prev) {
    value[v] += prev;
    for(int u : tree[v]) {
        build_partial(u, value[v]);
    }
}
void query_edges() {
    int E, F;
```

```

        for(int i=0; i<P; i++) {
            cin>>E>>F;
            ll sum = 0;
            sum+=value[E],sum+=value[F],sum -=2*value[lca(E,F)];
            printf("%llu\n", sum);
        }
    }
    int main() {
        ios_base::sync_with_stdio(0);
        // freopen("UPDTREE2.inp", "r", stdin);
        cin>>N>>Q>>P;
        for(int i=1; i<N; i++) {
            int u, v;
            cin>>u>>v;
            adj[u].push_back(v), adj[v].push_back(u);
        }
        dfs(1, 0);
        build_sparse();
        update_edges();
        diff(1);
        build_partial(1, 0);
        query_edges();
    }
}

```

### 5.12.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtb?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtb?usp=sharing)

## 5.13. Bài 13: Dạo chơi trên cây

### 5.13.1. Đề bài Walking

Cho một đồ thị dạng cây với  $N$  đỉnh và  $N-1$  cạnh. Hằng ngày có một chú khỉ đi từ một đỉnh  $a$  đến đỉnh  $b$ . Chú thấy rằng có một số đỉnh đã được thăm vào các ngày trước đó. Chú băn khoăn là trên đường đi ngày hôm nay từ  $a \rightarrow b$  có bao nhiêu đường đi của các ngày trước đó giao với ngày hôm nay, hai đường giao nhau là 2 đường có ít nhất một đỉnh chung.

**Dữ liệu vào:** Đọc vào từ tệp Walking.inp

Dòng đầu tiên ghi số  $N, Q$  ( $1 \leq N, Q \leq 10^5$ ) là số đỉnh và số ngày dạo chơi  
 $N-1$  dòng tiếp theo ghi cặp số  $(u, v)$  thể hiện cạnh trên cây

$Q$  dòng tiếp theo ghi cặp số  $(a, b)$  thể hiện lộ trình dạo chơi của các ngày.

**Kết quả ra:** Ghi ra tệp Walking.out

$Q$  dòng, trong đó dòng thứ  $i$  là số ngày có khác nhau đã từng đi qua một đỉnh trên lộ trình của ngày  $i$ .

**Ví dụ:**

Walking.inp	Walking.out	Giải thích
-------------	-------------	------------

5 4	0	Ngày 1: Chưa có con đường nào trước đó giao với đường 4→5.	
1 2	1		
1 3	2		
3 4	2	Ngày 2: Giao với ngày 1 tại đỉnh 4 3	
3 5		Ngày 3: Giao với ngày 1,2	
4 5		Ngày 4: Giao với ngày 2,3	
4 2			
1 3			
1 2			

### 5.13.2. Phân tích đề bài và đề xuất thuật toán

Để giải quyết bài toán, ta quan tâm đến 2 vấn đề sau:

Vấn đề 1: Làm sao để biết  $(u, v)$  có giao nhau với  $(u', v')$  của ngày trước đó?

Cách đơn giản nhất đó là kiểm tra tất cả các điểm trên  $(u, v)$  có điểm nào thuộc đường đi  $u' \rightarrow v'$  không. Cách này quá chậm, ta có cách khác như sau:

Gọi  $w = lca(u, v)$ ,  $w' = lca(u', v')$ .

- Nếu  $w'$  là tổ tiên của  $w$ , thì  $(u, v)$  giao với  $(u', v')$  khi  $u'$  hoặc  $v'$  nằm trong cây con gốc  $w$ .

- Nếu  $w'$  nằm trong cây con của  $w$ , thì đương nhiên  $u', v'$  cũng nằm trong cây con của  $w$ . Do đó, nếu  $(u, v)$  giao với  $(u', v')$  thì bắt buộc  $w'$  phải nằm trên đường  $u \rightarrow v$ .

Tuy nhiên, ta có tất cả cỡ  $Q^2$  cặp, mỗi cặp lại xử lý trong  $\log(N)$ , vậy độ phức tạp thuật toán sẽ là:  $O(Q^2 \cdot \log(N))$ .

Vấn đề 2: Làm sao để đếm nhanh được các cặp giao với  $(u, v)$ ?

Nhận xét thấy rằng, vấn đề có thể giải quyết với cấu trúc dữ liệu có khả năng đếm (ví dụ: BIT, Segment tree).

Ở đây ta sẽ dùng 2 cây BIT cho 2 trường hợp  $w'$  là tổ tiên của  $w$  hoặc  $w$  là tổ tiên của  $w'$ .

\*) Cây BIT1 sẽ giải quyết tình huống  $w'$  là tổ tiên của  $w$ . Mỗi khi thêm đường đi  $u' \rightarrow v'$  ta sẽ cập nhật thông tin giống bài UPDTREE2: Tăng thêm 1 tại  $u', v'$  và giảm 2 tại  $lca(u', v')$ .

Khi xét  $(u, v)$  thì kết quả được tăng thêm tổng trọng số các đỉnh thuộc cây con của  $w$ .

\*) Cây BIT2 sẽ giải quyết tình huống  $w'$  là con cháu của  $w$ . Khi đó bắt buộc  $w'$  phải nằm trên đường đi  $u \rightarrow v$ , hay nói cách khác cây BIT2 sẽ đếm số đỉnh mà là LCA của các  $(u, v)$  đi đã xuất hiện từ  $root \rightarrow a, root \rightarrow b$  rồi trừ đi  $root \rightarrow lca(a, b), root \rightarrow parent[lca(a, b)]$ .

Để sử dụng cây BIT quản lý thông tin, ta phải quản lý các đỉnh theo thứ tự duyệt đến và duyệt xong khi thực hiện DFS. Khi này LCA ta cũng quản lý theo thứ tự duyệt DFS luôn.

Độ phức tạp thuật toán:  $O(N \cdot \log(N)) + Q \cdot \log(N)$

### Chương trình tham khảo:

```
#include<bits/stdc++.h>
using namespace std;
const int N = 100005;
int n,q,cnt,BIT1[N],BIT2[N],depth[N], up[N][19], out[N], in[N];
vector<int> adj[N];
void dfs(int x, int p) {
    in[x] = ++cnt;
    up[cnt][0] = in[p];
    depth[cnt] = depth[in[p]]+1;;
    for(int i=1; i<=17; i++)
        up[cnt][i]=up[up[cnt][i-1]][i-1];
    for(int y : adj[x])
        if(y != p)
            dfs(y,x);
    out[in[x]] = cnt;
}
int lca(int u, int v) {
    if(depth[u] < depth[v])
        swap(u, v);
    for(int i = 17; i >= 0; i--)
        if(depth[up[u][i]] >= depth[v]) {
            u = up[u][i];
        }
    if(u == v)
        return u;
    for(int i = 17; i >= 0; i--)
        if(up[u][i] != up[v][i]) {
            u = up[u][i], v = up[v][i];
        }
    return up[u][0];
}
void update(int BIT[], int x, int val) {
    while(x<=n) {
        BIT[x] += val;
        x += (x & (-x));
    }
}
int get(int BIT[], int id) {
    int ret = 0;
    while(id>0) {
        ret += BIT[id];
        id &= (id-1);
    }
    return ret;
}
int get(int BIT[], int l, int r) {
    return get(BIT,r) - get(BIT,l-1);
}
int main() {
    scanf("%d%d",&n,&q);
    for(int i =1; i<n; i++) {
        int x,y;
        scanf("%d%d",&x,&y);
        adj[x].push_back(y);
    }
```



```

        adj[y].push_back(x);
    }
    dfs(1,0);
    while(q--) {
        int a,b;
        scanf("%d%d",&a,&b);
        a = in[a], b = in[b];
        int l = lca(a,b);
        int ans = get(BIT1,l,out[l]);
        update(BIT1, a, 1);
        update(BIT1, b, 1);
        update(BIT1, l, -2);
        ans+=get(BIT2,a)+get(BIT2,b);
        ans-=(get(BIT2,l)+(l==1?0:get(BIT2,up[l][0])));
        update(BIT2,l,1); ///cap nhat tang tai lca
        update(BIT2,out[l]+1,-1); ///giam thong tin tren nhanh khac
        printf("%d\n",ans);
    }
}

```

### 5.13.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.14. Bài 14: Cây đối gốc

Bài sau đây, tôi trình bày dạng bài tìm LCA trong trường hợp cây đối gốc, hay còn gọi là dynamic LCA.

### 5.14.1. Đề bài: ROOTLESS

Cho một đồ thị cây có  $N$  đỉnh  $N-1$  cạnh và  $Q$  truy vấn. Mỗi truy vấn được cho bởi bộ 3 số  $(u, v, r)$  yêu cầu hãy tìm  $LCA(u, v)$  khi chọn đỉnh  $r$  làm gốc.

**Dữ liệu vào:** Đọc từ tệp ROOTLESS.inp

Dòng 1: Số nguyên  $N, Q$  ( $1 \leq N, Q \leq 10^5$ ) là số đỉnh của cây, số truy vấn.

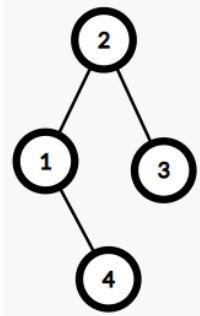
$N-1$  dòng tiếp theo ghi các cặp  $(u, v)$  thể hiện cạnh của cây.

$Q$  dòng tiếp theo, mỗi dòng ghi 3 số  $u, v, r$  ( $1 \leq u, v, r \leq N$ ).

**Kết quả ra:** Ghi ra tệp ROOTLESS.out

Đưa ra  $Q$  số là câu trả lời ứng với mỗi câu hỏi.

**Ví dụ:**

ROOTLESS.inp	ROOTLESS.out	Giải thích	
4 1 2 2 3 1 4 2 1 4 2 2 4 2	1 2	Khi chọn 1 làm gốc thì $lca(4,2)=1$ Khi chọn 2 làm gốc $lca(4,2)=2$	

### 5.14.2. Phân tích đề bài và đề xuất thuật toán

Trong trường hợp không có truy vấn đổi gốc thì nó chỉ là bài toán cơ bản. Tuy nhiên khi đổi gốc nếu tính lại Sparse table thì độ phức tạp thuật toán là  $O(N \cdot \log N \cdot Q)$ . Còn nếu duyệt DFS đơn thuần, không dùng Sparse table thì độ phức tạp thuật toán là  $O(N \cdot Q)$ .

#### Nhận xét:

Với truy vấn  $(u, v, r)$  thì  $\text{lca}(u, v)$  chỉ có thể là một trong 6 đỉnh :  $r, u, v, \text{lca}(u, v), \text{lca}(r, u), \text{lca}(v, r)$  trong đó  $\text{lca}(x, y)$  là tổ tiên chung gần nhất của 2 đỉnh  $x, y$  khi vẫn chọn 1 làm gốc.

Gọi  $x = \text{lca}(u, v)$  khi chọn  $r$  là gốc thì tổng khoảng cách  $x \rightarrow u, x \rightarrow v, x \rightarrow r$  sẽ là nhỏ nhất.

Dựa theo nhận xét trên thì ta chỉ cần điều chỉnh lại chương trình ban đầu một chút thì vẫn giải quyết các truy vấn trong  $O(\log(N))$ .

Độ phức tạp thuật toán vẫn là  $O(N \cdot \log(N) + Q \cdot \log(N))$

Chương trình tham khảo:

```
#include<bits/stdc++.h>
#define maxn 100005
using namespace std;
vector <int> adj[maxn];
int up[maxn][20], depth[maxn], n, q;
void dfs(int u) {
    for(int i = 1; i < 20; i++)
        up[u][i] = up[up[u][i - 1]][i - 1];
    for(int v:adj[u]) {
        if(!depth[v]) {
            up[v][0] = u;
            depth[v] = depth[u] + 1;
            dfs(v);
        }
    }
}
int lca(int u, int v) {
    if(depth[u] < depth[v])
        swap(u, v);
    for(int i = 17; i >= 0; i--)
        if(depth[up[u][i]] >= depth[v]) {
            u = up[u][i];
        }
    if(u == v)
        return u;
    for(int i = 17; i >= 0; i--)
        if(up[u][i] != up[v][i]) {
            u = up[u][i];
            v = up[v][i];
        }
    return up[u][0];
}
int dist(int u, int v) {
    int x = lca(u, v);
```

```
int res = depth[u] + depth[v] - 2 * depth[x];
return res;
}
int main() {
    cin >> n;
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    depth[1] = 1;
    dfs(1);
    cin >> q;
    pair<int, int> p[6];
    while(q--) {
        int u, v, r;
        cin >> u >> v >> r;
        p[0].second = u;
        p[1].second = v;
        p[2].second = r;
        p[3].second = lca(r, u);
        p[4].second = lca(r, v);
        p[5].second = lca(u, v);
        for(int i = 0; i < 6; i++) {
            int x = p[i].second;
            p[i].first = dist(x, r) + dist(x, u) + dist(x, v);
        }
        sort(p, p + 6);
        cout << p[0].second << endl;
    }
}
```

### 5.14.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 5.15. Bài 15: Cây đối gốc 2

### 5.15.1. Đề bài: Rootless2

Đây là phiên bản nâng cấp của bài ROOTLESS, đề bài như sau:

Cho một đồ thị cây có  $N$  đỉnh  $N-1$  cạnh và  $Q$  truy vấn. Gốc cây ban đầu là đỉnh 1. Truy vấn có 3 loại sau:

- Loại 1: cho bởi dạng  $(1, u)$  chọn đỉnh  $u$  làm gốc của cây.
- Loại 2: Cho bởi dạng  $(2, u, v, x)$  cộng tất cả các đỉnh trong cây con nhỏ nhất mà chứa  $u, v$  lên  $x$ . Lưu ý là cây con với đỉnh gốc đang có, chứ chưa chắc là đỉnh gốc 1.
- Loại 3: Cho bởi dạng  $(3, u)$  yêu cầu tính tổng giá trị tại các đỉnh trong cây con đỉnh  $u$ .

**Dữ liệu vào:** Đọc vào từ tệp Rootless2.inp

Dòng đầu tiên số  $N, Q$  ( $1 \leq N, Q \leq 10^5$ ) là số đỉnh và số truy vấn.

Dòng tiếp theo ghi  $N$  số  $a_1, a_2, \dots, a_N$  (với  $|a_i| \leq 10^5$ ) là giá trị ban đầu tại các đỉnh.

$N-1$  dòng tiếp theo ghi cặp số  $(u, v)$  thể hiện cạnh của cây.

$Q$  dòng tiếp theo mô tả các truy vấn trong 3 loại đã nêu ở trên.

**Kết quả ra:** Ghi ra tệp Rootless2.out

Ghi lần lượt các kết quả tương ứng với truy vấn 3 xuất hiện.

**Ví dụ:**

Rootless2.inp	Rootless2.out	Giải thích
4 6	18	Truy vấn 1: (3,1) tính tổng các đỉnh trong cây con đỉnh 1. KQ: $4+3+5+6=18$
4 3 5 6	21	Truy vấn 2: Đưa đỉnh 3 thành gốc.
1 2		Truy vấn 3: (2,2,4,3). Tăng các đỉnh trong cây con chứa đỉnh 2,4 lên 3. Khi đó trọng số các đỉnh lần lượt là: 7,6,8,9
2 3		Truy vấn 4: Chọn 1 làm gốc
3 4		Truy vấn 5: Giảm các đỉnh cây con chứa 2,4 đi 3. Khi đó trọng số là: 7,3,5,6
3 1		Truy vấn 6: Tổng trọng số cây con đỉnh 1 khi này là: $7+3+5+6=21$
1 3		
2 2 4 3		
1 1		
2 2 4 -3		
3 1		

### 5.15.2. Phân tích đề bài và đề xuất thuật toán

Với truy vấn loại 1 thì giá trị tại các đỉnh không bị thay đổi.

Với truy vấn loại 2 thì giá trị các đỉnh trong cây con có gốc là  $LCA(u, v)$  được tăng một lượng là  $x$ .

Với truy vấn loại 3 thì ta tính tổng các đỉnh con của gốc  $u$ .

Để giải quyết bài này, ta sẽ trải cây thành mảng, để thực hiện truy vấn loại 2, với cập nhật cây con đỉnh  $u$  có kích thước  $s$  thì ta cập nhật đoạn từ  $u$  đến  $u + s - 1$ . Điều này có thể dễ dàng thực hiện bằng cấu trúc Segment tree có Lazy propagation.

Mọi thứ sẽ trở nên phức tạp khi gốc  $r$  thay đổi, vậy làm sao để giảm độ phức tạp khi đó, chắc chắn ta sẽ không thể cập nhật lại thông tin toàn bộ cây. Đương nhiên ta vẫn sẽ giữ nguyên kết quả của việc tính toán ban đầu với gốc 1.

**Ta cần phải giải quyết 2 vấn đề sau:**

**Vấn đề 1:** Làm thế nào để xác định được  $lca(u, v)$  khi gốc là  $r$ ?

Nếu cả  $u, v$  vẫn thuộc cây con gốc  $r$  (trong cây gốc 1 ban đầu) thì  $lca(u, v)$  không thay đổi so với ban đầu.

Nếu chỉ có một trong hai  $u$  hoặc  $v$  thuộc cây con gốc  $r$  thì  $lca(u, v)$  khi gốc là  $r$  sẽ chính là  $r$ .

Nếu cả  $u, v$  đều không thộc cây con gốc  $r$ , thì khi đó ta đi tìm  $p = LCA(u, r)$  và  $q = LCA(v, r)$  trong cây ban đầu, đỉnh nào có độ sâu lớn hơn thì sẽ chính là  $LCA(u, v)$  khi chọn gốc là  $r$ .

Kết luận:  $lca(u, v)$  khi chọn gốc  $r$  là đỉnh sâu nhất trong các đỉnh sau:  $LCA(u, v), LCA(u, r), LCA(v, r)$ . Với cách tìm  $lca(u, v)$  như thế này ta cũng có thể điều chỉnh lại thuật toán bài ROOTLESS.

**Vấn đề 2:** Cập nhật lại thông tin như thế nào khi đã tìm được  $w = LCA(u, v)$  khi chọn  $r$  làm gốc?

- Nếu  $w = r$  thì cập nhật toàn bộ cây.
- Nếu  $w$  là con của  $r$  thì chỉ cập nhật cây con gốc  $w$ .
- Nếu  $w$  là tổ tiên của  $r$  thì cập nhật toàn bộ cây, sau đó trừ đi nhánh con của  $w$  mà chứa  $r$ .

Nên nhớ, bài này cần trải cây thành mảng và sử dụng kĩ thuật Lazy propagation trên cấu trúc dữ liệu Segment tree.

**Việc tính tổng trong cây con gốc  $w$  khi chọn gốc của cây là  $r$  xử lí tương tự như thao tác cập nhật.**

Độ phức tạp thuật toán:

- Truy vấn đổi gốc:  $O(1)$
- Truy vấn cập nhật:  $O(\log(N))$
- Truy vấn tính tổng:  $O(\log(N))$

Chương trình tham khảo:

```
#include<bits/stdc++.h>
#define N 200000
using namespace std;
int g[N],a[N*2][2],v[N],deep[N],fa[N][20],l[N],r[N],to[N];
long long f[N*3],sig[N*3];
int n,q;
void ins(int x,int y) {
    static int sum=1;
    a[++sum][0]=y,a[sum][1]=g[x],g[x]=sum;
}
void dfs(int x) {
    static int sum=0;
    to[l[x]=++sum]=x;
    deep[x]++;
    for(int i=0; fa[fa[x][i]][i]; i++)
        fa[x][i+1]=fa[fa[x][i]][i];
    for(int i=g[x]; i; i=a[i][1])
        if(a[i][0]!=fa[x][0]) {
            deep[a[i][0]]=deep[x];
            fa[a[i][0]][0]=x;
            dfs(a[i][0]);
        }
    r[x]=sum;
}
void build(int l,int r,int s) {
```

```
        if(l==r) {
            f[s]=v[to[l]];
            return;
        }
        build(l,(l+r)/2,s+s);
        build((l+r)/2+1,r,s+s+1);
        f[s]=f[s+s]+f[s+s+1];
    }
    void down(int l,int r,int s) {
        if(sig[s]) {
            f[s]+=(r-l+1)*sig[s];
            if(l!=r)
                sig[s+s]+=sig[s],sig[s+s+1]+=sig[s];
            sig[s]=0;
        }
    }
    void ins(int l,int r,int s,int ll,int rr,int x) {
        down(l,r,s);
        if(r<ll||rr<l)
            return;
        if(ll<=l&&r<=rr) {
            sig[s]=x;
            down(l,r,s);
            return;
        }
        ins(l,(l+r)/2,s+s,ll,rr,x),ins((l+r)/2+1,r,s+s+1,ll,rr,x);
        f[s]=f[s+s]+f[s+s+1];
    }
    long long get(int l,int r,int s,int ll,int rr) {
        down(l,r,s);
        if(r<ll||rr<l)
            return 0;
        if(ll<=l&&r<=rr)
            return f[s];
        return get(l,(l+r)/2,s+s,ll,rr)+get((l+r)/2+1,r,s+s+1,ll,rr);
    }
    int getlca(int x,int y) {
        int i=19;
        if(deep[x]<deep[y])
            swap(x,y);
        while(deep[x]!=deep[y]) {
            while(deep[fa[x][i]]<deep[y])
                i--;
            x=fa[x][i];
        }
        i=19;
        while(x!=y) {
            while(fa[x][i]==fa[y][i]&&i)
                i--;
            x=fa[x][i],y=fa[y][i];
        }
        return x;
    }
    int up(int x,int y) {
        int i=19;
        while(deep[x]!=y) {
```

```
        while(deep[fa[x][i]]<y)
            i--;
        x=fa[x][i];
    }
    return x;
}
int main() {
    scanf("%d %d",&n,&q);
    for(int i=1; i<=n; i++)
        scanf("%d",&v[i]);
    for(int i=1; i<n; i++) {
        int x,y;
        scanf("%d %d",&x,&y);
        ins(x,y);
        ins(y,x);
    }
    dfs(1);
    build(1,n,1);
    int root=1;
    while(q--) {
        int sig,u,v,x;
        scanf("%d %d",&sig,&v);
        if(sig==1) {
            root=v;
        } else
            if(sig==2) {
                scanf("%d %d",&u,&x);
                int LCA1=getlca(u,root);
                int LCA2=getlca(v,root);
                int LCA3=getlca(u,v);
                if(deep[LCA1]<deep[LCA2])
                    swap(u,v),swap(LCA1,LCA2);
                if(LCA1==root&&deep[LCA3]<=deep[root])
                    ins(1,n,1,1,n,x);
                else
                    if(LCA1==LCA2&&LCA1!=LCA3)
                        ins(1,n,1,l[LCA3],r[LCA3],x);
                    else {
                        int LCA4=up(root,deep[LCA1]+1);
                        ins(1,n,1,1,n,x);
                        ins(1,n,1,l[LCA4],r[LCA4],-x);
                    }
            } else {
                int LCA1=getlca(v,root);
                if(root==v)
                    printf("%I64d\n",get(1,n,1,1,n));
                else
                    if(LCA1!=v)
                        printf("%I64d\n",get(1,n,1,l[v],r[v]));
                    else {
                        int LCA2=up(root,deep[v]+1);
                        printf("%I64d\n",get(1,n,1,1,n)-
get(1,n,1,l[LCA2],r[LCA2]));
                    }
            }
    }
}
```

}

### 5.15.3. Test kèm theo

[https://drive.google.com/drive/folders/1KXbl\\_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing](https://drive.google.com/drive/folders/1KXbl_1eGLcxtUAe3BovVJS7cERBZTtbp?usp=sharing)

## 6. Một số bài tập tự luyện

<https://codeforces.com/contest/1062/problem/E>  
<https://codeforces.com/contest/1328/problem/E>  
<https://www.spoj.com/problems/DISQUERY/>  
<http://vnoi.info/problems/show/LUBENICA/>  
<https://www.spoj.com/problems/QTREE/>  
<http://vnoi.info/problems/show/QTREE3/>  
<https://vn.spoj.com/problems/FSELECT/>  
<https://codeforces.com/contest/208/problem/E>  
<https://codeforces.com/contest/191/problem/C>  
<https://codeforces.com/contest/519/problem/E>  
<https://codeforces.com/contest/587/problem/C>  
<https://codeforces.com/contest/609/problem/E>  
<https://codeforces.com/contest/176/problem/E>  
<https://www.hackerrank.com/contests/101hack26/challenges/sherlock-and-queries-on-the-graph>

## 7. Kết luận

Bài toán LCA cũng là dạng bài toán xuất hiện trên đồ thị dạng cây, hoặc các đồ thị có thể quy về dạng đồ thị dạng cây.

Đồ thị dạng cây cũng là đồ thị đặc biệt, ta có thể dễ dàng biến cây thành mảng bằng Euler tour. Do đó, ta cũng có lớp bài toán trên cây áp dụng với các cấu trúc dữ liệu, thuật toán như đối với mảng.

Bài toán LCA có thể kết hợp với các dạng bài về cấu trúc dữ liệu khác (Disjoint set union, Segment tree, Binary index tree,...), các dạng bài toán khác về cây (quy hoạch động, cây khung nhỏ nhất, cầu và khớp,...).

Dạng bài về LCA tôi thường áp dụng dạy cho học sinh đầu lớp 11. Do kinh nghiệm chưa nhiều, nên cách nhìn nhận vấn đề còn chưa toàn diện và chưa bao quát được hết các bài toán có sử dụng LCA. Cách trình bày, cách thể hiện thuật toán có chỗ chưa thể hiện sự sáng tạo, vậy nên tôi rất mong nhận được chia sẻ, đóng góp của bạn bè đồng nghiệp để có cái nhìn đa chiều và hoàn thiện hơn về dạng bài toán LCA.

Tôi xin chân thành cảm ơn!



## **8. Tài liệu tham khảo**

- [1]. Hồ Sĩ Đàm (2016), Tài liệu giáo khoa chuyên Tin học quyển 1, Nhà xuất bản giáo dục Việt Nam.
- [2]. <https://vnoi.info/wiki/Home>
- [3]. <https://codeforces.com>
- [4]. <https://www.codechef.com>
- [5]. <https://vn.spoj.com>