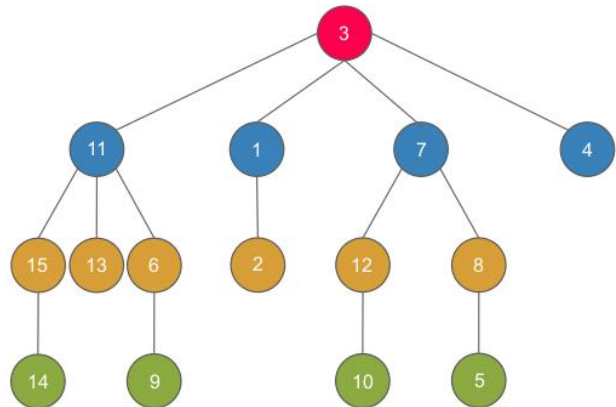
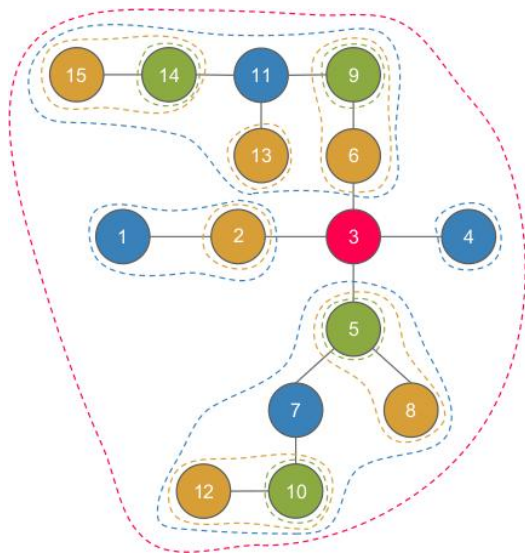


CENTROID DISCOMPOSITION VÀ BÀI TẬP ỨNG DỤNG



I. Mở đầu

Cây là một dạng đặc biệt của đồ thị, các bài toán liên quan trên đồ thị dạng cây giúp chúng ta khám phá được nhiều thuật toán hay để xử lý. Trong quá trình giảng dạy đội tuyển quốc gia các năm, chúng tôi đã học hỏi thêm được một kĩ thuật hay có thể ứng dụng vào nhiều bài toán phức tạp để tìm ra một lời giải đơn giản hơn. Kĩ thuật đó thường được biết đến với cái tên Centroid Decomposition.

Kĩ thuật này chưa có nhiều người biết đến cũng như xuất hiện khá ít trong các kì thi lập trình trong nước hiện nay (đa phần mọi người ưa chuộng Heavy Light Decomposition hơn vì ý tưởng dễ nghĩ tới) nên thường bỏ qua Centroid Decomposition. Tuy nhiên mỗi phương pháp đều có những ưu nhược điểm nhất định nên ta cần phải linh hoạt chuyển đổi các kĩ thuật nhằm giúp bài toán trở nên đơn giản, dễ triển khai hơn.

Vì vậy chúng tôi quyết định chọn chuyên đề **Centroid Decomposition** để mọi người cùng tham khảo, nghiên cứu với nhau cũng như triển khai rộng rãi thuật toán giúp mọi người có những hướng suy nghĩ đa dạng, tìm ra những lời giải hay hơn.

Những hiểu biết, kiến thức của tôi có thể chưa chính xác, nếu có sai sót xin hãy góp ý để chuyên đề ngày càng hoàn thiện hơn.

II. Một số khái niệm cơ bản

Đây là một kĩ thuật lạ, không quen thuộc với nhiều người và có thể mất thời gian để hiểu thấu đáo. Để có thể tiếp cận chuyên đề một cách suôn sẻ nhất, ta cần phải nắm một số khái niệm, thuật toán cơ bản sau:

1. Đồ thị dạng cây

Giả sử $T = (V, E)$ là đồ thị vô hướng với n đỉnh. Khi đó các mệnh đề sau là tương đương (Daisy Chain Theorem):

- T là cây
- T không chứa chu trình đơn và có $n - 1$ cạnh
- T liên thông và mỗi cạnh của nó đều là cầu

- Giữa hai đỉnh bất kì của T đều tồn tại đúng một đường đi đơn
- T không chứa chu trình đơn nhưng nếu thêm một cạnh vào đồ thị ta thu được một chu trình đơn
- T liên thông và có $n - 1$ cạnh

Các chứng minh định lý trên có thể tham khảo thêm tại “Tài liệu giáo khoa chuyên Tin Quyển 1”

2 Các thuật toán cơ bản trên đồ thị

- Thuật toán tìm kiếm, duyệt đồ thị (DFS, BFS)
- Thuật toán tìm tổ tiên chung gần nhất (LCA):

Có thể tham khảo thêm về LCA tại đây: <https://vnoi.info/wiki/algo/data-structures/lca.md>

3. Các cấu trúc dữ liệu hỗ trợ (Vector, Set, Map, Fenwick Tree, Segment Tree,..)

- STL C++ điển hình là Vector, Set, Map:
https://drive.google.com/file/d/1iqlQ1TmgGy_CKwZ0_9KPfu_ZHsnrT3Tu/view
- Cây chỉ số nhị phân (Binary Index Tree): <https://vnoi.info/wiki/algo/data-structures/fenwick.md>
- Cây phân đoạn (Segment Tree): <https://vnoi.info/wiki/algo/data-structures/segment-tree-extend.md>

III. Nội dung

1. Bài toán tô màu

TREE1

Cho một cây gồm N đỉnh, các đỉnh được đánh số từ 1 tới N . Đỉnh đầu tiên của cây được tô màu đỏ và các đỉnh còn lại được tô màu xanh. Có hai loại truy vấn trong số M truy vấn sau:

- 1 V : Tô đỉnh V thành màu đỏ
- 2 V : Biết được một đỉnh V , tính khoảng cách giữa V và đỉnh có màu đỏ gần V nhất (có thể chính nó)

INPUT

- Dòng đầu tiên chứa hai số nguyên N, M ($N \leq 2 * 10^5, M \leq 10^5$)
- Trong $N - 1$ dòng tiếp theo, mỗi dòng chứa hai số nguyên a_i, b_i ($1 \leq a_i, b_i \leq N, a_i \neq b_i$) mô tả một cạnh của cây
- Trong M dòng tiếp theo, mỗi dòng gồm hai số nguyên t_i, v_i ($1 \leq t_i \leq 2, 1 \leq v_i \leq N$). Nếu $t_i = 1$, ta sẽ thực hiện thao tác tô đỉnh v thành màu đỏ.
Nếu $t_i = 2$, ta phải in ra khoảng cách giữa V và đỉnh có màu đỏ gần V nhất
- Dữ liệu đảm bảo đồ thị được cho dưới dạng cây.

OUTPUT

- In ra các câu trả lời ứng với những truy vấn loại 2

SUBTASKS:

- Subtask 1: $N, M \leq 1000$
- Subtask 2: Tất cả các truy vấn loại 1 được thực hiện trước truy vấn loại 2
- Subtask 3: Không có giới hạn gì thêm

INPUT	OUTPUT
5 4	0
1 2	3
2 3	2
2 4	
4 5	
2 1	
2 5	
1 2	
2 5	

2. Các hướng giải quyết cơ bản

Thông thường, chúng ta sẽ nghĩ đến các hướng tiếp cận cơ bản để giải các subtask trước rồi từ đó làm tiền đề suy nghĩ, cải tiến thành một lời giải hoàn chỉnh.

2.1 Thuật toán duyệt:

- Với mỗi truy vấn loại 1, ta cập nhật đỉnh đó ở trạng thái màu đỏ. $O(1)$
- Với một truy vấn loại 2, xây dựng một thuật toán duyệt DFS/BFS đi từ đỉnh đó lên đến mọi đỉnh của cây và cập nhật lấy min khoảng cách khi gặp các đỉnh màu đỏ. $O(N)$

Tuy nhiên trong trường hợp xấu nhất có thể phải trả lời rất nhiều truy vấn $O(M)$

Nhận xét: Độ phức tạp $O(N.M)$

2.2 Thuật toán Quy hoạch động

- Với trường hợp đặc biệt như ở Subtask 2, ta có thể áp dụng thuật toán Quy hoạch động để khởi tạo sẵn bảng kết quả trong $O(N)$ và trả lời truy vấn loại 2 trong $O(1)$
- Tô hết các đỉnh thành màu đỏ ở truy vấn 1
- Gọi $D[u]$ là khoảng cách gần nhất từ u tới đỉnh màu đỏ nằm trong các đỉnh con của nó (bao gồm chính nó)
- $F[u]$ là khoảng cách gần nhất giữa u với đỉnh màu đỏ nằm trên toàn bộ cây
- Ta thực hiện hai lần duyệt DFS, một lần để tính các giá trị $D[u]$ và một lần sau đó tính các giá trị $F[u]$

$$\text{Khởi tạo: } D[u] = F[u] = n - 1$$

$$\begin{cases} D[u] = 1 \text{ nếu } u \text{ là đỉnh được tô màu đỏ} \\ D[u] = \min(D[u], D[v] + 1) \text{ với } v \text{ là con liền kề của } u \end{cases}$$

$$F[u] = \min(D[u], F[p] + 1) \text{ với } p \text{ là cha của } u$$

Nhận xét: Độ phức tạp $O(N + M)$

Tuy nhiên các thuật trên vẫn chưa thể giải quyết được vấn đề hiện tại và cũng khó để nhìn ra được một cải tiến nào đó. Những lúc như vậy ta thường nghĩ tới các Cấu trúc dữ liệu hay những kĩ thuật phân rã bài toán như Heavy Light Decomposition, Segment Tree

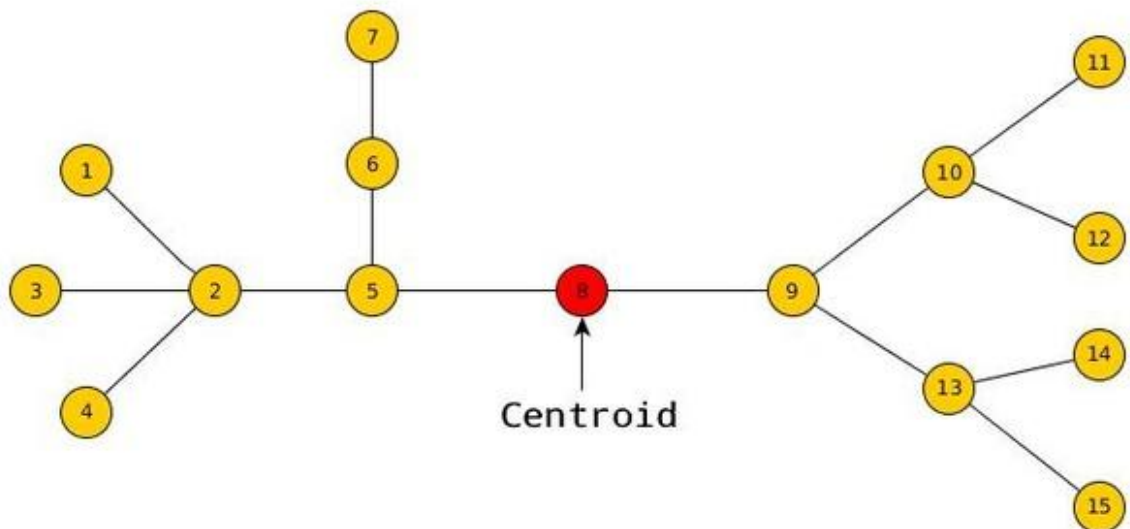
hay SQRT Decomposition,.. Có rất nhiều cách để giải quyết bài toán, tôi sẽ đề xuất một phương pháp theo chuyên đề hôm nay: Centroid Decomposition.

3. Centroid Decomposition

3.1 Các đỉnh “Centroid”

Theo định lý (Jordan, 1869) Centroid là một đỉnh thỏa mãn các điều kiện:

- Khi bị xóa đi, cây hiện tại có N đỉnh sẽ bị phân thành hai cây mới mà mỗi cây đó có không quá $\left\lfloor \frac{N}{2} \right\rfloor$ đỉnh so với cây ban đầu.
- Luôn có ít nhất một đỉnh như vậy ở bất kì cây nào đi nữa.



Hình minh họa 1

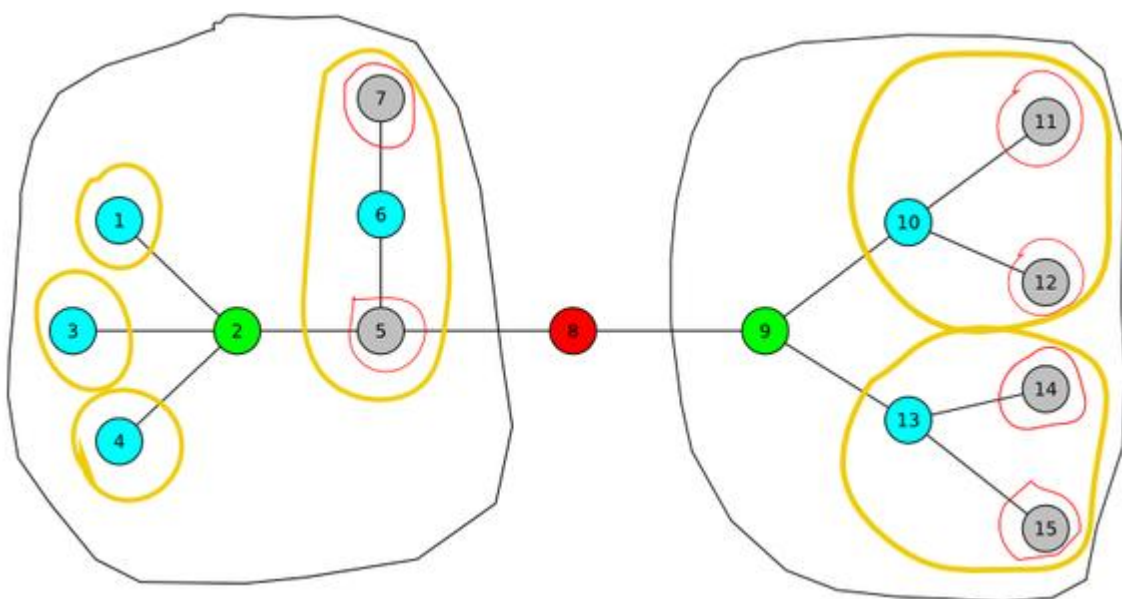
Chứng minh định lý và thuật toán tìm centroid:

- Ta bắt đầu bằng một đỉnh gốc R nào đó. Nếu R là một centroid thì kết thúc, nếu không, ta luôn có một cây con của R với số lượng đỉnh lớn hơn $\left\lfloor \frac{N}{2} \right\rfloor$. Đi xuống đỉnh gốc của cây con đó, gọi nó là S , bây giờ hoặc S là centroid hoặc ta tiếp tục đi xuống cây con của S có số đỉnh lớn hơn $\left\lfloor \frac{N}{2} \right\rfloor$.
- Cứ tiếp tục quá trình này cho tới khi tìm ra đỉnh thỏa mãn, và vì không quay lại các đỉnh đã thăm cũng như những thành phần chứa chúng luôn ít hơn $\left\lfloor \frac{N}{2} \right\rfloor$ đỉnh, ta

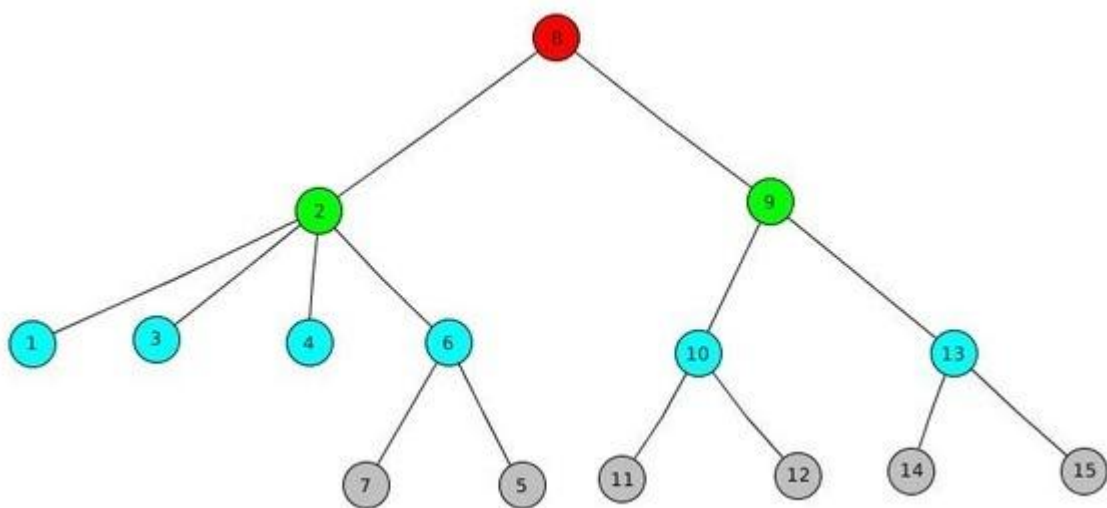
chia được cây thành nhiều phần khác nhau mà vẫn đảm bảo có cây luôn có số lượng đỉnh tối thiểu là $\left\lfloor \frac{N}{2} \right\rfloor + 1$ so với cây trước đó cũng như khi ta đi xuống các cây con, số lượng đỉnh của nó sẽ giảm ít nhất 1. Vậy nên khi quá trình kết thúc ta nhất định sẽ tìm được một centroid, thao tác này có độ phức tạp $O(N)$ trong trường hợp xấu nhất. Và đây cũng chính là thuật toán ta tìm một centroid trong chuyên đề này.

3.2 Cây Centroid

Ta sẽ xây dựng mô hình đệ quy tạo ra một cây centroid từ cây ban đầu như sau. Lấy ra một centroid ban đầu, cây sẽ phân rã thành nhiều cây khác nhau mà mỗi cây sẽ không có quá $\left\lfloor \frac{N}{2} \right\rfloor$ đỉnh so với cây gốc. Dùng centroid này làm gốc của cây centroid, gọi đệ quy phân rã các cây mới tạo ra, lấy ra những centroid của cây đó làm đỉnh con của centroid trước vừa tìm được. Tiếp tục thực hiện thuật toán đệ quy cho tới khi phân rã hết mọi cây thành những đỉnh centroid khác nhau.



Hình minh họa 2



Khi thuật toán kết thúc ta sẽ thu được cây centroid như sau:

Hình minh họa 3

3.3 Tính chất

Cây Centroid có những tính chất cơ bản sau đây:

- Cây chứa hết tất cả những đỉnh của cây cũ.

- Cây có độ cao lớn nhất là $\log(n)$. Dễ thấy cứ mỗi giai đoạn phân ra, số lượng đỉnh của mỗi cây mới không vượt quá một nửa của cây ban đầu vậy nên độ cao của cây có thể tính bằng câu hỏi đơn giản: “Có thể chia một số N cho 2 bao nhiêu lần thì nó sẽ nhỏ hơn hoặc bằng 1“. Hiển nhiên câu trả lời là $\log_2 N$.
- Với hai đỉnh A, B bất kì từ cây ban đầu, đường đi giữa chúng trên cây gốc đều sẽ trung gian qua một đỉnh C (với C là LCA của A và B ở cây centroid).
- Lưu ý: cây centroid chỉ là “giả”, khoảng cách và đường đi giữa hai đỉnh trên này khác với khoảng cách thực của nó ở cây gốc

3.4 Khai triển mô hình cây centroid

Các định nghĩa và nội dung lý thuyết tuy có vẻ phức tạp nhưng khi triển khai thì lại khá đơn giản. Ta sẽ khai triển code các thành phần đã chia như ở trên:

3.4.1. Định nghĩa các biến, dữ liệu:

- Tsize: Số lượng đỉnh của cây hiện tại chuẩn bị phân rã
- Subsize[u]: Số lượng đỉnh của cây con gốc u
- Par[u]: Cha của u trên cây centroid

```
int Subsize[N], Par[N], Tsize = 0;
```

3.4.2. Tính lại số lượng đỉnh của các cây con mới được tạo sau khi phân rã

```
void ReSubsize(int u, int p)
{
    Tsize++;
    Subsize[u] = 1;
    for (int v : A[u]) if (v != p && !Par[v]) {
        ReSubsize(v, u);
        Subsize[u] += Subsize[v];
    }
}
```

3.4.3. Tìm centroid

```

int GetCentroid(int u, int p)
{
    for (int v : A[u]) if (v != p && !Par[v] && Subsize[v] > Tsize/2)
        return GetCentroid(v, u);
    return u;
}

```

3.4.4. Phân rã cây

```

void Decompose(int u, int p)
{
    Tsize = 0;
    ReSubsize(u, 0);
    int Centroid = GetCentroid(u, 0);
    if (p == 0) Par[Centroid] = Centroid;
    else Par[Centroid] = p;
    for (int v : A[Centroid]) if (!Par[v]) Decompose(v, Centroid);
}

```

Ở ngoài chương trình ta khởi tạo cây centroid bằng lệnh:

```
Decompose(1, 0);
```

4. Áp dụng vào bài toán

Để thuận tiện cho việc triển khai thuật toán, chúng ta cần phải xây dựng rõ ràng các phần sau:

4.1. LCA và công thức tính khoảng cách

- Thuật toán LCA rất quan trọng trong các bài toán dạng cây, có nhiều cách để xây dựng LCA như đã đề cập ở phần trước. Dưới đây là một cách để xây dựng LCA với độ phức tạp khởi tạo là $O(N \log N)$ và độ phức tạp thao tác lấy LCA là $O(1)$.

```

typedef pair <int, int> ii;
#define X first
#define Y second

```

```

int Time, Depth[N], Start[2 * N];
ii RMQ[20][2 * N];

void DFS(int u, int p)
{
    Depth[u] = Depth[p] + 1;
    Start[u] = ++Time;
    RMQ[0][Time] = {Depth[u], u};
    for (int v : A[u]) if (v != p) {
        DFS(v, u);
        RMQ[0][++Time] = {Depth[u], u};
    }
}

void BuildRMQ()
{
    int limit = log2(Time);
    for (int k = 1; k <= limit; k++) {
        int tmp = (1 << (k - 1));
        for (int i = 1; i <= Time; i++)
            RMQ[k][i] = min(RMQ[k - 1][i], RMQ[k - 1][i + tmp]);
    }
}

int GetLCA(int u, int v)
{
    int Left = Start[u];
    int Right = Start[v];
    if (Left > Right) swap(Left, Right);
    int k = log2(Right - Left + 1);
    return min(RMQ[k][Left], RMQ[k][Right - (1 << k) + 1]).Y;
}

```

- Ta sẽ không đi sâu vào thuật toán LCA nữa mà sẽ tìm công thức tính khoảng cách giữa hai đỉnh bất kì trên cây gốc. Nhận thấy đường đi từ $u \rightarrow v$ trên cây luôn có thể chia thành hai phần: từ $u \rightarrow lca(u, v)$ và từ $lca(u, v) \rightarrow v$.
- Ta có công thức:

$$dist(u, v) = dist(u, lca(u, v)) + dist(lca(u, v), v)$$

$$dist(u, v) = (depth[u] - depth[lca(u, v)]) + (depth[v] - depth[lca(u, v)])$$

$$dist(u, v) = depth[u] + depth[v] - 2 * depth[lca(u, v)]$$

```

int Dist(int u, int v)
{
    return Depth[u] + Depth[v] - 2 * Depth[GetLCA(u, v)];
}

```

- Thao tác chuẩn bị LCA, cây centroid:

```

void Prepare()
{
    DFS(1, 0);
    BuildRMQ();
    Decompose(1, 0);
}

```

4.2. Thao tác cập nhật và truy vấn kết quả:

Ta gọi $best[u]$ là khoảng cách đỉnh màu đỏ gần nhất trong cây con của u ở centroid. Lưu ý, khoảng cách này được tính dựa trên cây gốc, không phải là khoảng cách giữa hai đỉnh trên centroid.

Nhắc lại hai tính chất quan trọng :

- Với hai đỉnh A, B bất kì từ cây ban đầu, đường đi giữa chúng trên cây gốc đều sẽ trung gian qua một đỉnh C (với C là LCA của A và B ở cây centroid).

- Độ cao của cây là $\log(N)$

Từ hai tính chất trên chúng ta xây dựng được mô hình cập nhật và lấy kết quả bằng cách chỉ xét qua những điểm trung gian là những nút cha của chúng. Thật vậy, với đỉnh v bất kì ở cây centroid, ta luôn có hai trường hợp:

- Đỉnh v nằm trong cây con của u , như vậy đường đi giữa u và v là đường đi trực tiếp và phải qua u .
- Đỉnh v nằm ngoài cây con của u , như vậy đường đi giữa u và v phải đi qua một cha chung nào đó của chúng ở cây centroid

Ngoài ra số nút cha ở trên của nó chỉ không quá $\log(N)$ nút, do đó luôn đảm bảo chỉ cần cập nhật và lấy kết quả từ những đỉnh cha ở trên là đã có thể lưu đầy đủ thông tin và bao quát, xử lý được hết các truy vấn trong thời gian $O(\log)$

4.2.1. Thao tác cập nhật

Với thao tác cập nhật tô màu đỏ vào đỉnh u , ta duyệt các cha trên của u và cập nhật khoảng cách của những đỉnh cha đó với đỉnh màu đỏ ($best[p]$).

```
void Update(int u)
{
    best[u] = 0; /// Tô u màu đỏ
    int Cur = u; /// Cur là đỉnh ban đầu của u ở cây centroid
    while (u != Par[u]) { /// Nếu u chưa là đỉnh gốc của cây centroid
        u = Par[u]; /// Đi lên cha u
        best[u] = min(best[u], Dist(u, Cur)); /// Update
    }
}
```

Nhận xét: Độ phức tạp $O(\log)$

4.2.2. Truy vấn kết quả

Thao tác truy vấn kết quả cũng tương tự:

- Nếu các đỉnh màu đỏ nằm trong cây con của u ở cây centroid, hiển nhiên $best[u]$ đã được cập nhật trước đó.
- Nếu các đỉnh màu đỏ nằm ngoài cây con của u ở cây centroid, vậy thì chắc chắn đỉnh này đã cập nhật giá trị cho những cha chung ở trên. Như vậy ta chỉ cần đi lên các cha p của đỉnh u và sử dụng lại các $best[p]$ đã được cập nhật trước đó
- Đường đi từ $A \rightarrow B$ trung gian qua C là LCA của A và B ở cây centroid và $dist(B, C) \geq best(C)$ nên ta có công thức:

$best[A] = \min(best[A], best[C] + dist(A, C))$ với C là các đỉnh cha trên của A

```
int Getans(int u)
{
    int Cur = u; /// Cur là đỉnh ban đầu của u
    while (u != Par[u]) { /// Nếu u chưa là đỉnh gốc của cây centroid
        u = Par[u]; /// Đi lên cha u
        best[Cur] = min(best[Cur], Dist(Cur, u) + best[u]);
    }
    return best[Cur];
}
```

4.3. Chương trình minh họa

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

```

typedef pair <int, int> ii;
#define X first
#define Y second

const int Inf = 1e9;
const int N = 2e5 + 5;

int n, m, Time, Depth[N], Start[2 * N], best[N];
int Subsize[N], Par[N], Tsize = 0;
ii RMQ[20][2 * N];
vector <int> A[N];

void Input()
{
    cin >> n >> m;
    for (int i = 1; i < n; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        A[u].push_back(v);
        A[v].push_back(u);
    }
}

void DFS(int u, int p)
{
    Depth[u] = Depth[p] + 1;
    Start[u] = ++Time;
    RMQ[0][Time] = {Depth[u], u};
    for (int v : A[u]) if (v != p) {
        DFS(v, u);
        RMQ[0][++Time] = {Depth[u], u};
    }
}

void BuildRMQ()
{
    int limit = log2(Time);
    for (int k = 1; k ≤ limit; k++) {
        int tmp = (1 << (k - 1));
        for (int i = 1; i ≤ Time; i++)
            RMQ[k][i] = min(RMQ[k - 1][i], RMQ[k - 1][i + tmp]);
    }
}

```

```

    }
}

int GetLCA(int u, int v)
{
    int Left = Start[u];
    int Right = Start[v];
    if (Left > Right) swap(Left, Right);
    int k = log2(Right - Left + 1);
    return min(RMQ[k][Left], RMQ[k][Right - (1 << k) + 1]).Y;
}

int Dist(int u, int v)
{
    return Depth[u] + Depth[v] - 2 * Depth[GetLCA(u, v)];
}

void ReSubsize(int u, int p)
{
    Tsize++;
    Subsize[u] = 1;
    for (int v : A[u]) if (v != p && !Par[v]) {
        ReSubsize(v, u);
        Subsize[u] += Subsize[v];
    }
}

int GetCentroid(int u, int p)
{
    for (int v : A[u]) if (v != p && !Par[v] && Subsize[v] > Tsize/2)
        return GetCentroid(v, u);
    return u;
}

void Decompose(int u, int p)
{
    Tsize = 0;
    ReSubsize(u, 0);
    int Centroid = GetCentroid(u, 0);
    if (p == 0) Par[Centroid] = Centroid;
    else Par[Centroid] = p;
}

```

```

    for (int v : A[Centroid]) if (!Par[v]) Decompose(v, Centroid);
}

void Prepare()
{
    DFS(1, 0);
    BuildRMQ();
    Decompose(1, 0);
}

void Update(int u)
{
    best[u] = 0; // Tô u màu đỏ
    int Cur = u; // Cur là đỉnh ban đầu của u ở cây centroid
    while (u != Par[u]) { // Nếu u chưa là đỉnh gốc của cây centroid
        u = Par[u]; // Đi lên cha u
        best[u] = min(best[u], Dist(u, Cur)); // Update
    }
}

int Getans(int u)
{
    int Cur = u; // Cur là đỉnh ban đầu của u
    while (u != Par[u]) { // Nếu u chưa là đỉnh gốc của cây centroid
        u = Par[u]; // Đi lên cha u
        best[Cur] = min(best[Cur], Dist(Cur, u) + best[u]);
    }
    return best[Cur];
}

void Solve()
{
    for (int i = 1; i ≤ n; i++) best[i] = Inf; // Khởi tạo best[i]
    Update(1); // Đỉnh 1 đã được tô màu đỏ
    while (m--) {
        int type, u;
        scanf("%d%d", &type, &u);
        if (type == 1) Update(u);
        else printf("%d\n", Getans(u));
    }
}

```



```
#define task "test"
int main()
{
    if (fopen(task ".inp", "r")){
        freopen(task ".inp", "r", stdin);
        freopen(task ".out", "w", stdout);
    }
    Input();
    Prepare();

    Solve();
}
```

Nhận xét:

- Thao tác khởi tạo để tính LCA có độ phức tạp là $O(N \log N)$, thao tác tìm LCA mất $O(1)$
- Mỗi truy vấn cập nhật và xử lý kết quả có độ phức tạp $O(\log N)$ (do thao tác tìm LCA chỉ $O(1)$)
- Độ phức tạp bài toán: $O(N \log N + M \log N)$

Link nộp thử bài tập tại đây: <https://codeforces.com/contest/342/problem/E>

IV. Các bài tập ứng dụng

1. Bài toán tô màu 2

1.1. Đề bài

TREE2

Cho một cây gồm N đỉnh, các đỉnh được đánh số từ 1 tới N . Mỗi đỉnh sẽ được tô bởi một trong hai màu đỏ và xanh. Ban đầu tất cả các đỉnh đều được tô màu xanh.

Có hai loại truy vấn trong số M truy vấn sau:

- 0 V : Thay đổi màu hiện tại của đỉnh V (từ đỏ thành xanh và từ xanh thành đỏ)
- 1 V : Biết được một đỉnh V , tính khoảng cách giữa V và đỉnh có màu đỏ gần V nhất (có thể chính nó). Nếu không có đỉnh màu đỏ nào thì in -1

INPUT

- Dòng đầu tiên chứa số nguyên N , ($N \leq 10^5$)
- Trong $N - 1$ dòng tiếp theo, mỗi dòng chứa hai số nguyên a_i, b_i ($1 \leq a_i, b_i \leq N$, $a_i \neq b_i$) mô tả một cạnh của cây
- Dòng tiếp theo chứa số nguyên M ($M \leq 10^5$)
- Trong M dòng tiếp theo, mỗi dòng gồm hai số nguyên t_i, v_i ($0 \leq t_i \leq 1$, $1 \leq v_i \leq N$). Nếu $t_i = 0$, ta sẽ thực hiện thao tác đổi màu cho đỉnh V (đỏ thành xanh và xanh thành đỏ). Nếu $t_i = 1$, ta phải in ra khoảng cách giữa V và đỉnh có màu đỏ gần V nhất
- Dữ liệu đảm bảo đồ thị được cho dưới dạng cây.

OUTPUT

- In ra các câu trả lời ứng với những truy vấn loại 2

INPUT	OUTPUT
10	2
1 2	2
1 3	2
2 4	3

1 5	0
1 6	
4 7	
7 8	
5 9	
1 10	
10	
0 6	
0 6	
0 6	
1 3	
0 1	
0 1	
1 3	
1 10	
1 4	
1 6	

1.2. Phân tích thuật toán

- Bài này tương tự với bài TREE1 tuy nhiên có thêm thao tác đổi từ màu đỏ về lại màu xanh nên ta không thể dùng mảng `best[u]` như trên để lưu một giá trị nhất định được mà phải quản lý một tập các giá trị khoảng cách rồi lấy giá trị nhỏ nhất ở tập đó.
- Ta nghĩ ngay đến STL C++, ở bài này ta có thể chọn Multiset để biểu diễn tập hợp trên.
- Gọi multiset `<int> best[u]`: lưu tập hợp các giá trị khoảng cách sẽ được cập nhật ở `u` trên cây centroid

- Quy ước: Color[u] = 0 ứng với đỉnh u màu xanh và 1 ứng với đỉnh u mang màu đỏ.

1.2.1. Thao tác cập nhật

- Khi cập nhật đỉnh Cur đổi màu xanh sang đỏ, ta thêm các giá trị Dist(Cur, p) vào các best[p].
- Khi cập nhật đỉnh Cur đổi màu đỏ về màu xanh, ta chỉ việc xóa các giá trị Dist(Cur,p) ra khỏi các best[p].

```
void Update(int u)
{
    Color[u] ^= 1; // Đổi màu cho u
    // Cập nhật best[u]
    if (Color[u] == 0) best[u].erase(best[u].find(0));
    else best[u].insert(0);

    int Cur = u; // Cur là đỉnh ban đầu của u ở cây centroid
    while (u != Par[u]) { // Nếu u chưa là đỉnh gốc của cây centroid
        u = Par[u]; // Đi lên cha u
        // Update
        if (Color[Cur] == 0) best[u].erase(best[u].find(Dist(Cur, u)));
        else best[u].insert(Dist(Cur, u));
    }
}
```

1.2.2. Truy vấn kết quả

- Tương tự như TREE1 ta sử dụng các giá trị best[p] của cha p để tính toán.
- Nếu best[p] rỗng thì ta bỏ qua (không có đỉnh màu đỏ nào để sử dụng)
- Nếu best[p] có phần tử ta sử dụng giá trị nhỏ nhất trong best[p] để cập nhật

$$ret = \min(ret, *best[p].begin() + Dist(Cur, p))$$

```
int Getans(int u)
{
    int ret = Inf;
    int Cur = u; // Cur là đỉnh ban đầu của u
    if (best[u].size()) ret = min(ret, *best[u].begin());
    while (u != Par[u]) { // Nếu u chưa là đỉnh gốc của cây centroid
        u = Par[u]; // Đi lên cha u
        // Lấy giá trị từ cha
        if (best[u].size())
            ret = min(ret, *best[u].begin() + Dist(Cur, u));
    }
    if (ret == Inf) return -1;
    else return ret;
}
```

1.3. Chương trình minh họa

```
#include <bits/stdc++.h>
using namespace std;
typedef pair <int, int> ii;
#define X first
#define Y second

const int Inf = 1e9;
const int N = 1e5 + 5;

int n, m, Time, Depth[N], Start[2 * N], Color[N];
int Subsize[N], Par[N], Tsize = 0;
ii RMQ[20][2 * N];
vector <int> A[N];
multiset <int> best[N];

void Input()
{
    cin >> n;
    for (int i = 1; i < n; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        A[u].push_back(v);
        A[v].push_back(u);
    }
    cin >> m;
}

void DFS(int u, int p)
{
    Depth[u] = Depth[p] + 1;
    Start[u] = ++Time;
    RMQ[0][Time] = {Depth[u], u};
    for (int v : A[u]) if (v != p) {
        DFS(v, u);
        RMQ[0][++Time] = {Depth[u], u};
    }
}

void BuildRMQ()
```

```

{
    int limit = log2(Time);
    for (int k = 1; k ≤ limit; k++) {
        int tmp = (1 << (k - 1));
        for (int i = 1; i ≤ Time; i++)
            RMQ[k][i] = min(RMQ[k - 1][i], RMQ[k - 1][i + tmp]);
    }
}

int GetLCA(int u, int v)
{
    int Left = Start[u];
    int Right = Start[v];
    if (Left > Right) swap(Left, Right);
    int k = log2(Right - Left + 1);
    return min(RMQ[k][Left], RMQ[k][Right - (1 << k) + 1]).Y;
}

int Dist(int u, int v)
{
    return Depth[u] + Depth[v] - 2 * Depth[GetLCA(u, v)];
}

void ReSubsize(int u, int p)
{
    Tsize++;
    Subsize[u] = 1;
    for (int v : A[u]) if (v != p && !Par[v]) {
        ReSubsize(v, u);
        Subsize[u] += Subsize[v];
    }
}

int GetCentroid(int u, int p)
{
    for (int v : A[u]) if (v != p && !Par[v] && Subsize[v] > Tsize/2)
        return GetCentroid(v, u);
    return u;
}

void Decompose(int u, int p)

```

```

{
    Tsize = 0;
    ReSubsize(u, 0);
    int Centroid = GetCentroid(u, 0);
    if (p == 0) Par[Centroid] = Centroid;
    else Par[Centroid] = p;
    for (int v : A[Centroid]) if (!Par[v]) Decompose(v, Centroid);
}

void Prepare()
{
    DFS(1, 0);
    BuildRMQ();
    Decompose(1, 0);
}

void Update(int u)
{
    Color[u] ^= 1; /// Đổi màu cho u
    /// Cập nhật best[u]
    if (Color[u] == 0) best[u].erase(best[u].find(0));
    else best[u].insert(0);

    int Cur = u; /// Cur là đỉnh ban đầu của u ở cây centroid
    while (u != Par[u]) { /// Nếu u chưa là đỉnh gốc của cây centroid
        u = Par[u]; /// Đi lên cha u
        /// Update
        if (Color[Cur] == 0) best[u].erase(best[u].find(Dist(Cur, u)));
        else best[u].insert(Dist(Cur, u));
    }
}

int Getans(int u)
{
    int ret = Inf;
    int Cur = u; /// Cur là đỉnh ban đầu của u
    if (best[u].size()) ret = min(ret, *best[u].begin());
    while (u != Par[u]) { /// Nếu u chưa là đỉnh gốc của cây centroid
        u = Par[u]; /// Đi lên cha u
        /// Lấy giá trị từ cha
        if (best[u].size())

```

```

        ret = min(ret, *best[u].begin() + Dist(Cur, u));
    }
    if (ret == Inf) return -1;
    else return ret;
}

void Solve()
{
    for (int i = 1; i ≤ n; i++) Color[i] = 0; /// Tất cả màu xanh
    while (m--) {
        int type, u;
        scanf("%d%d", &type, &u);
        if (type == 0) Update(u);
        else printf("%d\n", Getans(u));
    }
}

#define task "test"
int main()
{
    if (fopen(task ".inp", "r")){
        freopen(task ".inp", "r", stdin);
        freopen(task ".out", "w", stdout);
    }
    Input();
    Prepare();
    Solve();
}

```

Nhận xét:

- Thao tác khởi tạo để tính LCA có độ phức tạp là $O(N \log N)$, thao tác tìm LCA mất $O(1)$
- Thao tác sử dụng multiset mất thêm $O(\log N)$ nhưng thực tế giá trị này không lớn
- Vậy mỗi truy vấn cập nhật có độ phức tạp trung bình $O(\log^2 N)$
- Độ phức tạp bài toán: $O(N \log N + M \log^2 N)$

- Qua đây ta dần nắm rõ hơn cách kết hợp các Cấu trúc dữ liệu với Centroid Decomposition.

Link nộp thử bài tập tại đây: <https://www.spoj.com/problems/QTREE5/>

2. Bài toán xếp hạng

2.1. Đề bài:

TREE3

Fox Ciel trở thành một chỉ huy của Tree Land. Giống như cái tên Tree Land đã nói, vùng đất này có N thành phố được đánh số từ 1 tới N và nối với nhau bởi $N-1$ con đường vô hướng, với bất kỳ hai thành phố nào thì luôn tồn tại một đường đi giữa chúng.

Fox Ciel cần chỉ định ra một sĩ quan cho mỗi thành phố. Mỗi sĩ quan có một cấp bậc - một chữ cái từ 'A' đến 'Z'. Vì vậy, sẽ có 26 cấp bậc khác nhau, và 'A' là trên cùng, 'Z' là cuối cùng, luôn có đủ sĩ quan cho từng cấp bậc riêng.

Nhưng có một quy tắc đặc biệt phải tuân theo: nếu x và y là hai thành phố khác nhau và các sĩ quan của hai thành phố đó có cùng cấp bậc, thì con đường đơn giữa x và y phải đi qua một thành phố z có một sĩ quan có cấp bậc cao hơn hai thành phố đó. Để đảm bảo quy tắc rằng giữa các sĩ quan cùng cấp bậc sẽ được giám sát bởi sĩ quan cấp cao hơn.

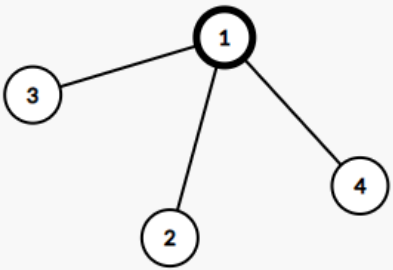
Hãy giúp Ciel lập một kế hoạch phân chia cấp bậc sĩ quan hợp lệ giữa các thành phố, nếu không thể, in ra “Impossible!”

INPUT

- Dòng đầu tiên chứa số nguyên N ($2 \leq N \leq 10^5$)
- Trong $N-1$ dòng tiếp theo, mỗi dòng chứa hai số nguyên a_i, b_i ($1 \leq a_i, b_i \leq N, a_i \neq b_i$) mô tả một đường đi trực tiếp giữa hai thành phố
- Dữ liệu đảm bảo đồ thị được cho dưới dạng cây.

OUTPUT

- Nếu tồn tại cách phân chia hợp lệ, in ra N kí tự. Kí tự i cho biết cấp bậc sĩ quan của thành phố i
- Nếu không in ra “Impossible!”

INPUT	OUTPUT
4 1 2 1 3 1 4	 A B B B

2.2. Phân tích thuật toán

Đây là bài toán áp dụng điển hình của cây centroid. Nhiệm vụ của chúng ta chỉ là dựng lên cây centroid và ứng với mỗi nút lấy cấp bậc của nó bằng chính độ cao của của đỉnh đó trên cây centroid. Giải thích:

- Những đỉnh cùng cấp bậc thì đường đi của chúng phải qua đỉnh có cấp bậc cao hơn, đây cũng là tính chất quan trọng của cây centroid và những đỉnh cấp bậc cao hơn đó chính là các tổ tiên chung của nó (LCA)
- Chỉ được sử dụng 26 màu, cây centroid có độ cao không vượt qua $\log(N)$ xấp xỉ bằng 20. Vậy tính chất của cây centroid thỏa mãn luôn đưa ra được lời giải cho bài toán.

Như vậy thuật toán bài này chỉ đơn giản là dựng cây centroid sau đó tính độ cao của mỗi nút trên cây centroid.

```

int Getrank(int u)
{
    if (u == Par[u]) return Ranked[u] = 0;
    else if (Ranked[u] != -1) return Ranked[u];
    return Ranked[u] = Getrank(Par[u]) + 1;
}

void Solve()
{
    for (int i = 1; i <= n; i++) Ranked[i] = -1;
    for (int i = 1; i <= n; i++)
        cout << (char) (Getrank(i) + 'A') << " ";
}

```

2.3. Chương trình minh họa

```

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

typedef pair <int, int> ii;
#define X first
#define Y second

const int Inf = 1e9;
const int N = 1e5 + 5;

int n, m, Ranked[N];
int Subsize[N], Par[N], Tsize = 0;
vector <int> A[N];

void Input()
{
    cin >> n;
    for (int i = 1; i < n; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        A[u].push_back(v);
        A[v].push_back(u);
    }
}

void ReSubsize(int u, int p)

```

```

{
    Tsize++;
    Subsize[u] = 1;
    for (int v : A[u]) if (v != p && !Par[v]) {
        ReSubsize(v, u);
        Subsize[u] += Subsize[v];
    }
}

int GetCentroid(int u, int p)
{
    for (int v : A[u]) if (v != p && !Par[v] && Subsize[v] > Tsize/2)
        return GetCentroid(v, u);
    return u;
}

void Decompose(int u, int p)
{
    Tsize = 0;
    ReSubsize(u, 0);
    int Centroid = GetCentroid(u, 0);
    if (p == 0) Par[Centroid] = Centroid;
    else Par[Centroid] = p;
    for (int v : A[Centroid]) if (!Par[v]) Decompose(v, Centroid);
}

void Prepare()
{
    Decompose(1, 0);
}

int Getrank(int u)
{
    if (u == Par[u]) return Ranked[u] = 0;
    else if (Ranked[u] != -1) return Ranked[u];
    return Ranked[u] = Getrank(Par[u]) + 1;
}

void Solve()
{
    for (int i = 1; i ≤ n; i++) Ranked[i] = -1;
}

```

```

    for (int i = 1; i ≤ n; i++)
        cout << (char) (Getrank(i) + 'A') << " ";
}

#define task "test"
int main()
{
    if (fopen(task ".inp", "r")){
        freopen(task ".inp", "r", stdin);
        freopen(task ".out", "w", stdout);
    }
    Input();
    Prepare();
    Solve();
}

```

Nhận xét:

- Cây centroid rất hữu dụng trong những bài toán cần xây dựng lên một cấu hình nào đó nhờ vào những tính chất của nó, đặc biệt là độ cao của cây luôn đáp ứng nhỏ hơn $\log(N)$ giúp ta tìm được các lời giải đẹp, gọn gàng mà đảm bảo sử dụng đủ không gian và thời gian cho phép.
- Độ phức tạp: $O(N \log N)$

Link nộp thử bài tập tại đây: <https://codeforces.com/contest/321/problem/C>

3. Đường đi độ dài K

3.1. Đề bài

Race (IOI 2011)

Cho đồ thị vô hướng liên thông có trọng số gồm N đỉnh và $N - 1$ cạnh được đánh số từ 0 tới $N - 1$. Xác định đường đi mà đi qua ít cạnh nhất, mỗi đỉnh chỉ qua một lần sao cho độ dài của đường đi đúng bằng K.

INPUT

Dòng đầu chứa 2 số nguyên N ($1 \leq N \leq 200\,000$), K ($1 \leq K \leq 1\,000\,000$).

$N - 1$ dòng tiếp theo mỗi dòng chứa 3 số u_i, v_i, c_i biểu diễn cạnh (u_i, v_i) có độ dài là c_i .

OUTPUT

Ghi trên một dòng là kết quả của bài toán. Nếu không có đường đi nào thỏa mãn ghi ra -1.

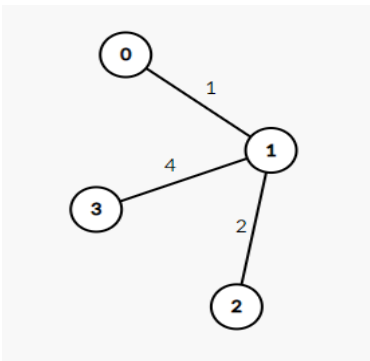
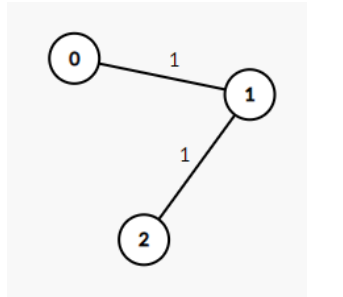
SUBTASK

Subtask 1 (9 điểm): $1 \leq N \leq 100, 1 \leq K \leq 100$. Đỉnh i sẽ nối đến đỉnh $i + 1$.

Subtask 2 (12 điểm): $1 \leq N \leq 1000, 1 \leq K \leq 1000000$.

Subtask 3 (22 điểm): $1 \leq N \leq 200\,000, 1 \leq K \leq 100$.

Subtask 4 (57 điểm): Không có giới hạn gì thêm.

INPUT	OUTPUT
4 3 0 1 1 1 2 2 1 3 4	2 
3 3 0 1 1 1 2 1	-1 

3.2. Phân tích thuật toán

3.2.1. Thuật toán duyệt 1

- Duyệt với mọi đỉnh u , ta DFS đi thăm hết tất cả các đỉnh v còn lại và kiểm tra xem đường đi từ u tới v có bằng K không, nếu có thì cập nhật kết quả với số cạnh đã đi được. Độ phức tạp: $O(N^2)$ đủ để qua Subtask 1 và 2.
- Một cải tiến tốt hơn là chỉ đi tiếp nếu như tổng độ dài đường đi còn lại chưa vượt quá K . Với Subtask 3 $K \leq 100$ nên độ dài đường đi của mỗi đỉnh u sẽ không vượt quá 100 cạnh. Độ phức tạp: $O(N.K)$

3.2.2. Thuật toán tốt hơn

- Gọi Root là đỉnh của cây hiện tại, một đường đi từ u qua v với độ dài K khi qua đỉnh Root sẽ có thể chia thành hai đường đi: từ $u \rightarrow$ Root và từ Root $\rightarrow v$.

- Như vậy ta gọi $D[i]$ = số cạnh ít nhất để đi từ một đỉnh con tới Root mà độ dài đúng bằng i ở cây ban đầu.
- Ta DFS để tính các $D[i]$ ở các nhánh con và chỉ việc lấy $ans = \min(ans, D[i] \text{ ở nhánh } x + D[K - i] \text{ ở nhánh } y)$.
- Tiếp tục đi xuống các đỉnh S là gốc cây con của Root và lại thực hiện thuật toán như trên, ưu điểm của thuật toán này là khi duyệt S ta không phải đi lại các đỉnh nằm ở nhánh cây con khác của Root, giúp cho thuật toán chạy hiệu quả hơn rất nhiều.
- Độ phức tạp mỗi lần chạy thuật toán với một đỉnh S bất kì bằng với số nút con của cây gốc S và ta phải chạy thuật toán trên với N đỉnh. Trong trường hợp xấu nhất là cây suy biến: độ phức tạp $O(N^2)$
- Tuy nhiên so với thuật toán duyệt 1 ở trên, thì thuật toán này vẫn cho ta một hướng đi mới là làm thế nào để tối ưu “Độ phức tạp mỗi lần chạy thuật toán với một đỉnh S bất kì bằng với số nút con của cây gốc S ”

3.2.3. Kết hợp Centroid Decomposition

- Đầu tiên ta phải hiểu và cài đặt thuật toán trên để có thể tiếp tục cải tiến bằng kĩ thuật này.
- Khi ta phân rã cây thành các Centroids, các nút này sẽ quản lý các tập con gồm $N, \frac{N}{2}, \frac{N}{4}, \dots, \frac{N}{2^h}, 1$ đỉnh trên cây “thực”. Như vậy áp dụng thuật toán vừa nêu trên với thứ tự duyệt đỉnh lần lượt là các đỉnh của cây Centroid sẽ giúp cho độ phức tạp thuật toán trở thành:

$$T(n) = 2 * T\left(\frac{n}{2}\right) + O(n)$$

Hay chúng ta còn biết đến $F(N) = N + \frac{N}{2} + \frac{N}{4} + \dots + \frac{N}{2^h} + 1 \cong N \log N$ đủ để giải quyết bài toán.

- Quan trọng: Thứ tự duyệt đỉnh là thứ tự ở cây centroid, thuật toán DFS tính khoảng cách là thuật toán tính và xử lý ở cây “thực”

- Lưu ý cần xử lý khéo léo để tránh trường hợp $D[i]$ và $D[K - i]$ ở cùng một nhánh con bằng cách tính $ans = \min(ans, \text{số cạnh hiện tại} + D[K - i])$ rồi mới cập nhật $D[i]$ sau. và tránh những $D[K - i]$ dư thừa từ những lần tính ở các đỉnh trước đó bằng cách lưu thêm đỉnh gốc hiện tại của giá trị $D[i]$ đang xét (xem code để tham khảo thêm)
- Ta sẽ lưu: $\text{pair} \langle \text{int}, \text{int} \rangle D[i] = \{\text{đỉnh gốc hiện tại}, \text{số cạnh min đã đi qua}\}$ thỏa độ dài đường đi từ đỉnh nào đó tới gốc hiện tại bằng đúng i
- Vector Subtree lưu $\text{child} = \{\text{số cạnh đã đi qua hiện tại}, \text{khoảng cách hiện tại so với gốc đang xét}\}$

3.3. Chương trình minh họa

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

typedef pair <int, int> ii;
#define X first
#define Y second

const int Inf = 1e9;
const int N = 2e5 + 5;
const int M = 1e6 + 5;

int n, k, visited[N], res = M;
int Subsize[N], Par[N], Tsize = 0, Root;
ii D[M];
vector <int> Cen_adj[N];
vector <ii> A[N], Subtree;

void Input()
{
    cin >> n >> k;
    for (int i = 1; i < n; i++) {
        int u, v, z;
        scanf("%d%d%d", &u, &v, &z);
        u++; v++;
        A[u].push_back({v, z});
        A[v].push_back({u, z});
    }
}
```

```

    }
}

void ReSubsize(int u, int p)
{
    Tsize++;
    Subsize[u] = 1;
    for (ii v : A[u]) if (v.X != p && !Par[v.X]) {
        ReSubsize(v.X, u);
        Subsize[u] += Subsize[v.X];
    }
}

int GetCentroid(int u, int p)
{
    for (ii v : A[u]) if (v.X != p && !Par[v.X] && Subsize[v.X] > Tsize/2)
        return GetCentroid(v.X, u);
    return u;
}

void Decompose(int u, int p)
{
    Tsize = 0;
    ReSubsize(u, 0);
    int Centroid = GetCentroid(u, 0);
    if (p == 0) {
        Par[Centroid] = Centroid;
        Root = Centroid;
    }
    else {
        Par[Centroid] = p;
        Cen_adj[p].push_back(Centroid);
    }
    for (ii v : A[Centroid]) if (!Par[v.X]) Decompose(v.X, Centroid);
}

void Prepare()
{
    Decompose(1, 0);
}

```

```

void DFS(int u, int par, int cnt, int dist)
{
    if (u != par && dist ≤ k) Subtree.push_back(ii(cnt, dist));
    for (ii tmp: A[u]) {
        int v = tmp.X, w = tmp.Y;
        if (v != par && !visited[v]) {
            DFS(v, u, cnt + 1, min(k + 1, dist + w));
            if (u != par) continue;

            for (ii child: Subtree) if (D[k - child.Y].X == u)
                res = min(res, D[k - child.Y].Y + child.X);

            while (Subtree.size()) {
                ii child = Subtree.back(); Subtree.pop_back();
                if (D[child.Y].X != u) D[child.Y] = {u, child.X};
                else D[child.Y].Y = min(D[child.Y].Y, child.X);
            }
        }
    }
}

void Solve()
{
    memset(visited, false, sizeof(visited));
    deque<int> DQ;
    DQ.push_back(Root);
    /// Tính theo thứ tự độ cao trên xuống ở cây Centroid
    while (DQ.size()) {
        int u = DQ.front(); DQ.pop_front();
        D[0] = {u, 0};
        DFS(u, u, 0, 0);
        visited[u] = true;
        for (int v: Cen_adj[u]) DQ.push_back(v);
    }
    if (res == M) cout << -1;
    else cout << res << endl;
}

#define task "test"
int main()
{

```

```
if (fopen(task ".inp", "r")){
    freopen(task ".inp", "r", stdin);
    freopen(task ".out", "w", stdout);
}
Input();
Prepare();
Solve();
}
```

Nhận xét:

- Độ phức tạp thuật toán: $O(N \log N)$
- Đây là một bài toán thú vị có thể giải quyết bằng Centroid Decomposition với một lời giải đẹp nhờ vào tính chất “chia nửa cây cù” của Centroid.
- Bài toán có thể mở rộng ra với $K < 10^9$ bằng cách kết hợp thêm Map hay Hashmap để lưu các giá trị ở mảng D.

Link nộp thử bài tập tại đây: <https://vnoi.info/problems/show/LQDRACE/>

Link bộ test và lời giải chính thức: <https://ioinformatics.org/page/ioi-2011/37>

4. Khoảng cách trên cây

4.1. Đề bài

TREE4

Cho đồ thị vô hướng liên thông gồm N đỉnh và $N - 1$ cạnh được đánh số từ 1 tới N . Xác định xem có bao nhiêu đường đi khác nhau giữa các đỉnh, mỗi đỉnh chỉ qua đúng một lần sao cho độ dài của đường đi đúng bằng K . Đường đi từ u sang v và từ v sang u chỉ tính là một đường, hai đường đi gọi là khác nhau nếu chúng khác nhau ở đỉnh xuất phát hoặc đỉnh kết thúc.

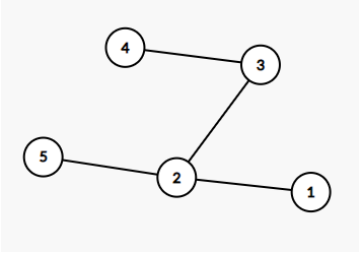
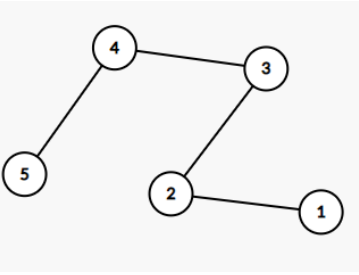
INPUT

Dòng đầu chứa 2 số nguyên N ($1 \leq N \leq 200\,000$), K ($1 \leq K \leq 1\,000\,000$).

$N - 1$ dòng tiếp theo mỗi dòng chứa 2 số u_i, v_i biểu diễn cạnh (u_i, v_i) .

OUTPUT

Ghi trên một dòng là kết quả của bài toán. Nếu không có đường đi nào thỏa mãn ghi ra - 1.

INPUT	OUTPUT
5 2 1 2 2 3 3 4 2 5	 4
5 3 1 2 2 3 3 4 4 5	 2

4.2. Phân tích thuật toán

- Bài này có hướng nghĩ tương tự như với bài trên chỉ khác là tính số đường đi có thể tạo ra độ dài đúng bằng K. Nhắc lại: thứ tự duyệt đỉnh là thứ tự ở cây centroid, thuật toán DFS tính khoảng cách là thuật toán tính và xử lý ở cây “thực”
- Gọi Root là đỉnh của cây centroid hiện tại, một đường đi từ u qua v với độ dài K khi qua đỉnh Root sẽ có thể chia thành hai đường đi: từ u -> Root và từ Root -> v.
- Gọi $D[i]$ = số đỉnh ở cây con có đường đi tới Root mà độ dài đúng bằng i trên cây “thực”
- DFS ở cây “thực” để tính các $D[i]$ ở các nhánh con và tính $ans = ans + (D[i] \text{ ở nhánh } x) * (D[K - i] \text{ ở nhánh } y)$.

- Lưu ý cần xử lý khéo léo để tránh trường hợp $D[i]$ và $D[K - i]$ ở cùng một nhánh con bằng cách tính $ans = ans + (D[K - i])$ ở các nhánh y) rồi mới cập nhật $D[i]$ sau. và tránh những $D[K - i]$ dư thừa từ những lần tính ở các đỉnh trước đó bằng cách lưu thêm đỉnh gốc hiện tại của giá trị $D[i]$ đang xét (xem code để tham khảo thêm)
- Tiếp tục đi xuống các đỉnh S là gốc cây con của Root trên cây centroid và lại thực hiện thuật toán như trên theo thứ tự độ cao của cây centroid
- Ta sẽ lưu: $\text{pair} \langle \text{int}, \text{int} \rangle D[i] = \{\text{đỉnh gốc hiện tại}, \text{số lượng đỉnh}\}$ thỏa độ dài đường đi của chúng tới gốc hiện tại bằng đúng i
- Vector Subtree lưu $w =$ khoảng cách hiện tại so với gốc đang xét

4.3. Chương trình minh họa

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

typedef pair <long long, long long> ii;
#define X first
#define Y second

const int Inf = 1e9;
const int N = 2e5 + 5;
const int M = 1e6 + 5;

int n, k, visited[N];
int Subsize[N], Par[N], Tsize = 0, Root;
ii D[M];
long long res = 0;
vector <int> Cen_adj[N], A[N];
vector <long long> Subtree;

void Input()
{
    cin >> n >> k;
    for (int i = 1; i < n; i++) {
        int u, v, z;
        scanf("%d%d", &u, &v);
```

```

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

void ReSubsize(int u, int p)
{
    Tsize++;
    Subsize[u] = 1;
    for (int v : A[u]) if (v != p && !Par[v]) {
        ReSubsize(v, u);
        Subsize[u] += Subsize[v];
    }
}

int GetCentroid(int u, int p)
{
    for (int v : A[u]) if (v != p && !Par[v] && Subsize[v] > Tsize/2)
        return GetCentroid(v, u);
    return u;
}

void Decompose(int u, int p)
{
    Tsize = 0;
    ReSubsize(u, 0);
    int Centroid = GetCentroid(u, 0);
    if (p == 0) {
        Par[Centroid] = Centroid;
        Root = Centroid;
    }
    else {
        Par[Centroid] = p;
        Cen_adj[p].push_back(Centroid);
    }
    for (int v : A[Centroid]) if (!Par[v]) Decompose(v, Centroid);
}

void Prepare()
{
    Decompose(1, 0);
}

```

```

}

void DFS(int u, int par, int dist)
{
    if (u != par && dist ≤ k) Subtree.push_back(dist);
    for (int v : A[u]) if (v != par && !visited[v]) {
        DFS(v, u, min(k + 1, dist + 1));
        if (u != par) continue;

        for (int w: Subtree) if (D[k - w].X == u)
            res += (long long) D[k - w].Y;
        while (Subtree.size()) {
            int w = Subtree.back(); Subtree.pop_back();
            if (D[w].X != u) D[w] = {u, 1};
            else D[w].Y++;
        }
    }
}

void Solve()
{
    memset(visited, false, sizeof(visited));
    deque<int> DQ;
    DQ.push_back(Root);
    /// Tính theo thứ tự độ cao trên xuống ở cây Centroid
    while (DQ.size()) {
        int u = DQ.front(); DQ.pop_front();
        D[0] = {u, 1};
        DFS(u, u, 0);
        visited[u] = true;
        for (int v: Cen_adj[u]) DQ.push_back(v);
    }
    cout << res << endl;
}

#define task "test"
int main()
{
    if (fopen(task ".inp", "r")){
        freopen(task ".inp", "r", stdin);
        freopen(task ".out", "w", stdout);
    }
}

```



```
}  
Input();  
Prepare();  
Solve();  
}
```

Nhận xét:

- Độ phức tạp thuật toán: $O(N \log N)$
- Centroid Decomposition còn có thể biến những bài toán đếm, tính tổng, xor trên đường đi giữa các đỉnh trên cây vốn là những bài khó, phức tạp trở nên dễ dàng hơn bao giờ hết.
- Bài toán này có thể mở rộng ra với $K \leq 10^9$ bằng cách kết hợp thêm Map hay Hashmap để lưu các giá trị ở mảng D.

Link nộp thử bài tập tại đây: <https://codeforces.com/contest/161/problem/D>

5. Đếm số đường đi không quá L

5.1. Đề bài:

TREE5

Cho đồ thị vô hướng liên thông có trọng số gồm N đỉnh và N - 1 cạnh được đánh số từ 1 tới N. Gọi khoảng cách giữa hai đỉnh u và v trên cây là $d(u,v)$.

Có M truy vấn trên cây. Mỗi truy vấn sẽ đưa ra hai số nguyên v, L và bạn cần phải trả lời câu hỏi: Có bao nhiêu đỉnh u thỏa mãn $d(u, v) \leq L$ (có thể bao gồm chính nó)

INPUT

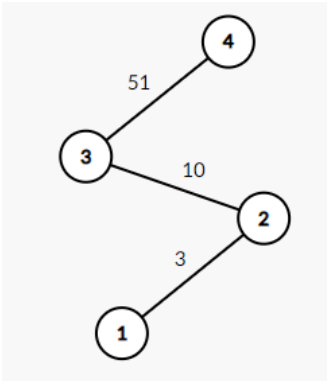
Dòng đầu chứa 2 số nguyên N ($1 \leq N, M \leq 10^5$)

N - 1 dòng tiếp theo mỗi dòng chứa 3 số u_i, v_i, c_i biểu diễn cạnh (u_i, v_i) với trọng số $c_i \leq 10^9$

Trong M dòng kế tiếp, mỗi dòng chứa hai số nguyên v và L ($1 \leq L \leq 10^{14}$)

OUTPUT

Ghi trên M dòng là kết quả của từng truy vấn.

INPUT	OUTPUT
4 6 1 2 3 2 3 10 3 4 51 1 2 1 10 3 30 3 51 2 60 2 61	1 2 3 4 3 4 

5.2. Phân tích thuật toán

- Bài này có một cách xử lý offline là dồn các truy vấn của một đỉnh để giải quyết một lần cũng bằng cách xây dựng centroid và tiến hành tìm kiếm nhị phân trên đây. Nhưng ta vẫn có thuật toán khác cũng sử dụng Centroid Decomposition để xử lý online như sau.
- Đầu tiên chúng ta xây dựng và triển khai LCA, tìm centroids và dựng cây cây centroid. Sau đó ta sẽ tiến hành xây dựng hai vector có ý nghĩa:
- Vector `<long long> ans[u]`: Lưu các khoảng cách $d(x, u)$ theo thứ tự tăng dần với x là con u trong cây centroid.
- Vector `<long long> cntPa[u]`: Lưu các khoảng cách $d(x, Par[u])$ theo thứ tự tăng dần với x là con u trong cây centroid .

```
void Update()
{
    for (int i = 1; i <= n; i++) {
        /// ans[u] Lưu các khoảng cách d(x, u) với x là con u
        int u = i;
        while (true) {
            ans[u].push_back(Dist(u, i));
            if (u == Par[u]) break;
            u = Par[u];
        }
    }
    for (int i = 1; i <= n; i++) {
        /// cntPa[u] Lưu các khoảng cách d(x, Par[u]) với x là con u
        int u = i;
        while (u != Par[u]) {
            cntPa[u].push_back(Dist(Par[u], i));
            u = Par[u];
        }
    }
    for (int i = 1; i <= n; i++) {
        sort(ans[i].begin(), ans[i].end());
        sort(cntPa[i].begin(), cntPa[i].end());
    }
}
```

- Hai vector này có thể tiền xử lý trước như sau:

Để đếm xem có bao nhiêu đỉnh x thỏa $d(u, x) \leq L$ ta xét hai trường hợp:

- Nếu x là con của u tại cây centroid, ta đơn giản chỉ cần sử dụng thuật toán tìm

```
ret = upper_bound(ans[u].begin(), ans[u].end(), L)
      - ans[u].begin();
```

kiểm nhị phân ở $ans[u]$ như định nghĩa mô tả ở trên để đếm xem có bao nhiêu x như vậy.

- Nếu x nằm ngoài cây con của u tại cây centroid, rõ ràng đường đi giữa chúng phải đi qua các cha chung nào đó ở trên u . Như vậy chỉ với Pa là một cha trên của u ta cần đếm xem có bao nhiêu đỉnh x trong $ans[Pa]$ thỏa mãn $d(u, Pa) + ans[Pa] \leq L$.

```
long long d = L - Dist(Cur, Par[u]);
int Add = upper_bound(ans[Par[u]].begin(), ans[Par[u]].end(), d)
          - ans[Par[u]].begin();
```

- Lưu ý rằng trong $ans[Pa]$ lại có chứa những đỉnh con x' của u nên việc cộng trực tiếp như vậy sẽ không đúng vì thao tác này chỉ đúng với những đỉnh nằm ngoài cây con gốc u ở cây centroid. Vậy nên ta phải trừ đi những giá trị x' là con của u thì kết quả mới chính xác, điều này cũng dễ dàng thực hiện được nhờ vector $cntPa[u]$ đã đề cập ở trên.

```
int Sub = upper_bound(cntPa[u].begin(), cntPa[u].end(), d)
          - cntPa[u].begin();
ret += Add - Sub;
```

Hàm tính kết quả các truy vấn sẽ được khai triển như sau:

```
int Getans(int u, long long L)
{
    int ret = 0, Cur = u;
    ret = upper_bound(ans[u].begin(), ans[u].end(), L)
          - ans[u].begin();
    while (u != Par[u]) {
        long long d = L - Dist(Cur, Par[u]);
        int Add = upper_bound(ans[Par[u]].begin(), ans[Par[u]].end(), d)
                  - ans[Par[u]].begin();
        int Sub = upper_bound(cntPa[u].begin(), cntPa[u].end(), d)
                  - cntPa[u].begin();
        ret += Add - Sub;
        u = Par[u];
    }
    return ret;
}
```

5.3. Chương trình minh họa

```

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

typedef pair <int, long long> ii;
#define X first
#define Y second

const int Inf = 1e9;
const int N = 2e5 + 5;
const int M = 1e6 + 5;

int n, m, Time, Depth[N], Start[2 * N];
int Subsize[N], Par[N], Tsize = 0, Root;
ii RMQ[20][2 * N];
long long Dis[N];
vector <long long> ans[N], cntPa[N];
vector <ii> A[N];

void Input()
{
    cin >> n >> m;
    for (int i = 1; i < n; i++) {
        int u, v, z;
        scanf("%d%d%d", &u, &v, &z);
        A[u].push_back({v, z});
        A[v].push_back({u, z});
    }
}

void DFS(int u, int p, long long c)
{
    Depth[u] = Depth[p] + 1;
    Dis[u] = Dis[p] + c;
    Start[u] = ++Time;
    RMQ[0][Time] = {Depth[u], u};
    for (ii v : A[u]) if (v.X != p) {
        DFS(v.X, u, v.Y);
        RMQ[0][++Time] = {Depth[u], u};
    }
}

```

```

void BuildRMQ()
{
    int limit = log2(Time);
    for (int k = 1; k ≤ limit; k++) {
        int tmp = (1 << (k - 1));
        for (int i = 1; i ≤ Time; i++)
            RMQ[k][i] = min(RMQ[k - 1][i], RMQ[k - 1][i + tmp]);
    }
}

int GetLCA(int u, int v)
{
    int Left = Start[u];
    int Right = Start[v];
    if (Left > Right) swap(Left, Right);
    int k = log2(Right - Left + 1);
    return min(RMQ[k][Left], RMQ[k][Right - (1 << k) + 1]).Y;
}

long long Dist(int u, int v)
{
    return Dis[u] + Dis[v] - 2 * Dis[GetLCA(u, v)];
}

void ReSubsize(int u, int p)
{
    Tsize++;
    Subsize[u] = 1;
    for (ii v : A[u]) if (v.X != p && !Par[v.X]) {
        ReSubsize(v.X, u);
        Subsize[u] += Subsize[v.X];
    }
}

int GetCentroid(int u, int p)
{
    for (ii v : A[u]) if (v.X != p && !Par[v.X] && Subsize[v.X] > Tsize/2)
        return GetCentroid(v.X, u);
    return u;
}

```

```

void Decompose(int u, int p)
{
    Tsize = 0;
    ReSubsize(u, 0);
    int Centroid = GetCentroid(u, 0);
    if (p == 0) Par[Centroid] = Centroid;
    else Par[Centroid] = p;
    for (ii v : A[Centroid]) if (!Par[v.X]) Decompose(v.X, Centroid);
}

void Prepare()
{
    DFS(1, 0, 0);
    BuildRMQ();
    Decompose(1, 0);
}

void Update()
{
    for (int i = 1; i ≤ n; i++) {
        // ans[u] Lưu các khoảng cách d(x, u) với x là con u
        int u = i;
        while (true) {
            ans[u].push_back(Dist(u, i));
            if (u == Par[u]) break;
            u = Par[u];
        }
    }
    for (int i = 1; i ≤ n; i++) {
        // cntPa[u] Lưu các khoảng cách d(x, Par[u]) với x là con u
        int u = i;
        while (u != Par[u]) {
            cntPa[u].push_back(Dist(Par[u], i));
            u = Par[u];
        }
    }
    for (int i = 1; i ≤ n; i++) {
        sort(ans[i].begin(), ans[i].end());
        sort(cntPa[i].begin(), cntPa[i].end());
    }
}

```

```

int Getans(int u, long long L)
{
    int ret = 0, Cur = u;
    ret = upper_bound(ans[u].begin(), ans[u].end(), L)
        - ans[u].begin();
    while (u != Par[u]) {
        long long d = L - Dist(Cur, Par[u]);
        int Add = upper_bound(ans[Par[u]].begin(), ans[Par[u]].end(), d)
            - ans[Par[u]].begin();
        int Sub = upper_bound(cntPa[u].begin(), cntPa[u].end(), d)
            - cntPa[u].begin();
        ret += Add - Sub;
        u = Par[u];
    }
    return ret;
}

void Solve()
{
    Update();
    while (m--) {
        int u; scanf("%d", &u);
        long long L; scanf("%lld", &L);
        printf("%d\n", Getans(u, L));
    }
}

#define task "test"
int main()
{
    if (fopen(task ".inp", "r")){
        freopen(task ".inp", "r", stdin);
        freopen(task ".out", "w", stdout);
    }
    Input();
    Prepare();
    Solve();
}

```


Nhận xét:

- Các thao tác LCA, Decompose và Update tính trước hai vector chỉ trong thời gian $O(N \log N)$. Với mỗi thao tác tính truy vấn ta phải đi lên các cha và sử dụng tìm kiếm nhị phân nên độ phức tạp trung bình là $O(\log^2 N)$. Độ phức tạp thuật toán: $O(N \log N + M \log^2 N)$
- Việc linh hoạt chuyển đổi các cấu trúc dữ liệu cũng như các thuật toán hỗ trợ để kết hợp với Centroid Decomposition khá quan trọng và sẽ ảnh hưởng tới việc lời giải đưa ra có đẹp và gọn hay không. Cũng thật may, việc cập nhật và xử lý truy vấn trên cây centroid khá đơn giản và dễ cài đặt.

Link nộp thử bài tập tại đây: <https://codeforces.com/gym/100570/problem/F>

V. Bài tập luyện tập

- <https://www.spoj.com/problems/QTREE4/>
- <https://codeforces.com/contest/914/problem/E> (Centroid với các thao tác bitmask)
- <https://www.hackerrank.com/challenges/bst-maintenance> (Centroid và cây BST)
- <https://www.codechef.com/problems/GERALD2>
- <https://www.codechef.com/problems/TESTERS>
- <https://www.codechef.com/problems/BTREE>
- <https://codeforces.com/gym/100633/problem/D> (Tô màu nhiều đỉnh)
- <https://www.codechef.com/problems/PRIMEDST>
- <https://www.codechef.com/FEB19A/problems/TRDST/>
- <https://codeforces.com/problemset/problem/1303/G> (Sum of Prefix Sum)
- <https://codeforces.com/problemset/problem/990/G> (GCD Counting)
- <https://vnoi.info/problems/list/?tag=119&page=1>
- <https://codeforces.com/contest/809/problem/E>

VI. Nhận xét, đánh giá

Đây là một chuyên đề khá khó và phức tạp tuy nhiên nếu áp dụng được vào các bài toán thì sẽ cho ra những lời giải rất độc đáo và ngắn gọn. Thông thường những dạng toán cập nhật màu của đỉnh và tìm đường đi ngắn nhất hay đếm số lượng đường đi, tính các tổng, xor độ dài trên cây cũng như xây dựng các cấu hình, mô hình chia để trị đều có thể áp dụng Centroid Decomposition mà độ phức tạp trung bình vẫn luôn là $O(N \log N)$

Centroid Decomposition có thể được sử dụng dưới nhiều hình thức, ta thường tiếp cận với mô hình “bottom – up”, bắt đầu từ những đỉnh con dưới để cập nhật, truy vấn dần lên các đỉnh cha ở cây centroid. Ngoài ra còn có thể áp dụng mô hình “top – down” ở một vài bài toán, chẳng hạn như chia để trị để xử lý dần các cây con.

Qua chuyên đề này, hi vọng các em học sinh cũng như các thầy cô có thể tiếp cận rộng rãi hơn tới phương pháp này và tìm ra những ứng dụng khác mà tôi còn thiếu sót, cũng như hỗ trợ cho những kì thi lập trình khu vực, quốc gia, quốc tế.

Do thời gian còn hạn chế và kiến thức chưa đủ sâu rộng nên chuyên đề có thể có nhiều thiếu sót. Rất mong được các thầy cô cùng tôi tìm hiểu thêm và góp ý để chuyên đề ngày càng trở nên hoàn thiện, đảm bảo cho các em học sinh tiếp cận được kiến thức rõ ràng và dễ hiểu nhất có thể.

Để đọc đề tiếng việt, đường dẫn vào làm các bài tập đã phân tích ở trên:

<https://cqhh.contest.codeforces.com>

Các quý thầy cô và các em học sinh có thể tự tạo hoặc sử dụng tài khoản Codeforces có sẵn để đăng nhập và submit kiểm thử bài tập.

Hoặc vào tài khoản chung sau:

Username: qhh123

Password: qhh123

VII. Các nguồn tài liệu tham khảo

- <https://threadsiiithyderabad.quora.com/Centroid-Decomposition-of-a-Tree>
- Tài liệu giáo khóa chuyên tin quyển 1
- <https://www.youtube.com/watch?v=-DmMLQCmz94>

MỤC LỤC

I. Mở đầu	1
II. Một số khái niệm cơ bản cần nắm	1
1 Đồ thị dạng cây	1
2 Các thuật toán cơ bản trên đồ thị	2
3. Các cấu trúc dữ liệu hỗ trợ (Vector, Set, Map, Fenwick Tree, Segment Tree,..)2	
III. Nội dung	2
1. Bài toán tô màu.....	2
2. Các hướng giải quyết cơ bản	4
3. Centroid Decomposition	5
4. Áp dụng vào bài toán	9
IV. Các bài tập ứng dụng	17
1. Bài toán tô màu 2.....	17
2. Bài toán xếp hạng	24
3. Đường đi độ dài K	29
4. Khoảng cách trên cây.....	35
5. Đếm số đường đi không quá L.....	41
VI. Nhận xét, đánh giá	48
VII. Các nguồn tài liệu tham khảo	49