HEAVY LIGHT DESCOMPOSITION

A. Cài đặt phân rã HLD

Input:

```
int n;                      // Số đỉnh của cây
vector<vector<int> > adj;   // Cây được biểu diễn bằng mảng vector
```

Output:

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

Các biến nháp

```
int s[maxn], id=0;
```

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

```
void DFS (int u, int dad) {
    s[u] = 1;
    int smax = 0, imax = 0;
    int sz = adj[u].size();
    for (int i = 0; i < sz; ++i) {
        int v = adj[u][i];
        if (v != dad) {
            Prev[v] = u;
            DFS (v, u);
            s[u] += s[v];
            if (s[v] > smax)
                smax = s[v], imax = i;
        }
    }
    // Đưa con lớn nhất về đầu danh sách kề
    swap (adj[u][0], adj[u][imax]);
}
```

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

```
void HLD (int u, int dad) {
    pos[u] = ++id;
    for (int v : adj[u])
        if (v != dad) {
            // Nếu (u,v) là cạnh nặng thì điểm đầu đường nặng chứa v chính
            // là điểm đầu của đường nặng chứa u.Ngoài ra trên cây HLD thì
            // u và v chập vào nhau do vậy u, v cùng độ sâu. Trường hợp (u,v)
            // là cạnh nhẹ thì đỉnh v bắt đầu vị trí của một đường nặng mới
            if (2 * s[v] >= s[u])
                head[v] = head[u], d_HLD[v] = d_HLD[u];
            else
```

```
                head[v] = v;
            d_HLD[v] = d_HLD[u] + 1;
            HLD (v, u);
        }
}
```

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

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

B. Tìm tổ tiên chung gần nhất (LCA)nhờ phân rã HLD

Việc tìm tổ tiên chung gần nhất (LCA) của hai đỉnh được thực hiện theo phương pháp cổ điển trên cây HLD (cây mà mỗi đường nặng là một đỉnh). Có thể mô tả phương pháp này như sau:
+B1) : Di chuyển "đỉnh" có độ sâu lớn hơn theo đỉnh cha nó cho đến khi hai "đỉnh" có cùng độ sâu. Chú ý rằng "đỉnh" ở đây là một đường nặng.
+B2) : Chừng nào hai "đỉnh" (đường nặng) khác nhau thì di chuyển đồng thời về cha của nó. Quá trình này kết thúc khi đi đến một "đỉnh" chung.
+B3) : Lúc này có hai đỉnh nằm trên một "đỉnh" (đường nặng) chung. Do vậy đỉnh nào có vị tri (pos) nhỏ hơn đỉnh đó sẽ là LCA cần tìm:

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

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

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

C. IT Max trên HLD

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

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

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

```
void IncEdges(int u,int v,int Delta) {
    while (d_HLD[u]>d_HLD[v]) {
        // Từ u đến head[u]
        Tree.Update(1,1,n,pos[head[u]]+1,pos[u],Delta);
        u=head[u];

        // Nhảy qua cạnh nhẹ
        Tree.Update(1,1,n,pos[u],pos[u],Delta);
        u=Prev[u];
    }
    while (d_HLD[v]>d_HLD[u]) {
        // Từ v đến head[v]
        Tree.Update(1,1,n,pos[head[v]]+1,head[v],Delta);
        v=head[v];

        // Nhảy qua cạnh nhẹ
        Tree.Update(1,1,n,pos[v],pos[v],Delta);
        v=Prev[v];
    }

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

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

```
    }
}
```

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

```
int GetEdges(int u,int v) {
    int kq=-INF;
    while (d_HLD[u]>d_HLD[v]) {
        kq=max(kq,Tree.Get(1,1,n,pos[head[u]]+1,pos[u]));
        u=head[u];
        kq=max(kq,Tree.Get(1,1,n,pos[u],pos[u]));
        u=Prev[u];
    }
    while (d_HLD[v]>d_HLD[u]) {
        kq=max(kq,Tree.Get(1,1,n,pos[head[v]]+1,pos[v]));
        v=head[v];
        kq=max(kq,Tree.Get(1,1,n,pos[v],pos[v]));
        v=Prev[v];
    }
    while (head[u]!=head[v]) {
        kq=max(kq,Tree.Get(1,1,n,pos[head[u]]+1,pos[u]));
        u=head[u];
        kq=max(kq,Tree.Get(1,1,n,pos[u],pos[u]));
        u=Prev[u];
        kq=max(kq,Tree.Get(1,1,n,pos[head[v]]+1,pos[v]));
        v=head[v];
        kq=max(kq,Tree.Get(1,1,n,pos[v],pos[v]));
        v=Prev[v];
    }
    if (pos[u]<pos[v]) {
        kq=max(kq,Tree.Get(1,1,n,pos[u]+1,pos[v]);
    } else {
        kq=max(kq,Tree.Get(1,1,n,pos[v]+1,pos[u]);
    }
    return kq;
}
```