BÀI TOÁN Lowest Common Ancestor (LCA)
(Kỹ thuật sử dụng bảng thưa - Sparse Table)

I. Một số thông tin bổ sung trong quá trình DFS trên cây:

- **int topo[maxn]:** Thứ tự topo DFS
- **int Prev[maxn]:** Số hiệu đỉnh cha
- **int depth[maxn]**: Độ sâu của gốc được định nghĩa depth[root]= 0. Nếu Prev[v]=u thì depth[v]=depth[u]+1.
- **int time_in[maxn]**: Thời điểm bắt đầu DFS
- **int time_out[maxn]:** Thời điểm kết thúc DFS

Đoạn code dưới đây để tính tất cả các thông tin trên

```
void DFS(int u,int dad) {
    time_in[u]=++id;
    topo[id]=u;
    for(int v: adj[u]) if (v!=dad) {
        depth[v]=depth[u]+1;
        Prev[v]=u;
        DFS(v, u);
    }
    time_out[u]=id;
}
```

II. Tổ tiên.

Định nghĩa:

- Với mỗi đỉnh u, Prev[u] được định nghĩa là tổ tiên đời thứ 1 của u (cha của u).
- Nếu u là tổ tiên đời thứ k của v thì Prev[u] là tổ tiên đời thứ k+1 của v

Với mỗi đỉnh v, tất cả các tổ tiên của nó là các đỉnh nằm trên đường đi đơn từ v về đỉnh gốc (không tính đỉnh v).

Đỉnh u là tổ tiên của đỉnh v khi và chỉ khi:

time_in[u]<time_in[v] && time_out[v]≤time_out[u]

Khi đó nó là tổ tiên đời thứ **depth[v]-depth[u]** của v

Như vậy ta có thể kiểm tra u là tổ tiên của v bằng hàm:

```
bool is_ancestor(int u,int v) {
    return time_in[u]<time_in[v] && time_out[v]≤time_out[u];
}
```

Bảng thưa (sparse table) lưu các tổ tiên:

Đặt f[u,k] là tổ tiên đời thứ 2^k của u. Ta có công thức qui hoạch động:

- f[u,0]=Prev[u]
- f[u,k]=f[f[u,k-1],k-1]

Đoạn code dưới đây để xây dựng mảng f

```
vector<vector<int> > f(n+1,vector<int>(int(log2(n))+2,0));
for(int u=1;u<=n;++u) f[u][0]=Prev[u];
int m=log2(n)+1;
for(int k=1;k<=m;++k) {
    for(int u=1;u<=n;++u)
```

```
    f[u][k]=f[f[u][k-1]][k-1];
}
```

Tìm tổ tiên đời thứ x của đỉnh u (x≤depth[u])

```
int ancestor(int u,int x) {
    while (x>0) {
        int k=log2(x);
        u=f[u][k];
        x-=(1<<k);
    }
    return u;
}
```

Tổ tiên chung gần nhất:

- Đỉnh w được gọi là tổ tiên chung của u và v nếu như nó vừa là tổ tiên của u, vừa là tổ tiên của v.
- Tổ tiên chung gần nhất được định nghĩa là tổ tiên chung có độ sâu lớn nhất.

Ta có thể tìm tổ tiên chung gần nhất bằng bảng thưa lưu các tổ tiên:

```
int LCA(int u,int v) {
    while (depth[v]>depth[u]) {
        int k=log2(depth[v]-depth[u]);
        v=f[v][k];
    }
    while (depth[u]>depth[v]) {
        int k=log2(depth[u]-depth[v]);
        u=f[u][k];
    }
    if (u!=v) {
        int k=int(log2(depth[u]))+1;
        while (k--) if (f[u][k]!=f[v][k]) {
            u=f[u][k];
            v=f[v][k];
        }
        u=f[u][0]; v=f[v][0];
    }
    return u;
}
```

III. Các bài toán thường gặp

Bài toán 1: Cho một cây vô hướng có trọng số ở cạnh. Hãy trả lời m truy vấn dạng (u,v) với ý nghĩa là tìm tổng trọng số của đường đi đơn từ đỉnh u đến đỉnh v.
Đặt d[u] là khoảng cach từ đỉnh gốc đến đỉnh u (ta có thể tính mảng này bằng cách sử dụng BFS hoặc DFS) Khi đó:

Distance(u,v) = d[u] + d[v] - 2*d[LCA(u,v)]

Bài toán 2: Cho một cây vô hướng có trọng số ở đỉnh. Hãy trả lời các truy vấn dạng (u,v) với ý nghĩa tính tổng trọng số các đỉnh trên đường đi đơn từ đỉnh u đến đỉnh v
Đặt d[u] là tổng trọng số các đỉnh từ đỉnh gốc đến đỉnh u (mảng d có thể thực hiện bằng DFS). Đặt w=LCA(u,v), w1=Prev[w]. Ta có:

Sum(u,v) = (d[u] - d[w1]) + (d[v] - d[w])

Bài toán 3: Cho một cây vô hướng có trọng số ở cạnh. Hãy trả lời m truy vấn dạng (u,v) với ý nghĩa là tìm giá trị lớn nhất của cạnh trên đường đi đơn từ u đến v.

Đặt Gmax[u,k] là trọng số lớn nhất của cạnh nằm trên đường đi từ u đến f[u,k]. Ta có:

- Gmax[u,0]=Ts(u,Prev[u])
- Gmax[u,k]=max(Gmax[u,k-1],Gmax[f[u][k-1],k-1])

Đoạn code dưới đây để chuẩn bị mảng Gmax

```
vector<vector<int> > Gmax(n+1,vector<int>(int(log2(n))+2,-INF));
for(int u=1;u<=n;++u) Gmax[u][0]=Ts(u,Prev[u]);
int m=log2(n)+1;
for(int k=1;k<=m;++k) {
    for(int u=1;u<=n;++u)
    Gmax[u][k]=max(Gmax[u][k-1],Gmax[f[u][k-1]][k-1]);
}
```

Kết hợp với thuật toán tìm LCA ta có hàm tìm giá trị lớn nhất của cạnh nằm trên đường đi đơn từ u đến v như sau:

```
int CalcMax(int u,int v) {
    int res=-INF;
    while (depth[v]>depth[u]) {
        int k=log2(depth[v]-depth[u]);
        res=max(res,Gmax[v][k]); v=f[v][k];
    }
    while (depth[u]>depth[v]) {
        int k=log2(depth[u]-depth[v]);
        res=max(res,Gmax[u][k]); u=f[u][k];
    }
    if (u!=v) {
        int k=log2(depth[u])+1;
        while (k--) if (f[u][k]!=f[v][k]) {
            res=max(res,Gmax[u][k]); u=f[u][k];
            res=max(res,Gmax[v][k]); v=f[v][k];
        }
        res=max(res,Gmax[u][0]); u=f[u][0];
        res=max(res,Gmax(v][0]); v=f[v][0];
    }
    return res;
}
```