QUI HOẠCH ĐỘNG TRÊN CÂY

Cây (Đồ thị vô hướng liên thông không có chu trình) là sự mở rộng tự nhiên của dãy số. Một dãy n phần tử có thể xem như là một trường hợp đặc biệt của cây nếu như coi mỗi phần tử như là một đỉnh, cạnh kề luôn nối từ một đỉnh đến đỉnh tiếp theo trên dãy số. Do vậy một cách tự nhiên có thể mở rộng các thuật toán trên mảng cho trường hợp cây. Trong chuyên đề này chúng ta sẽ mở rộng các thuật toán qui hoạch động trên mảng thành các thuật toán qui hoạch động trên cây.

I. Thứ tự DFS trên cây

Việc đầu tiên để có thể triển khai được các thuật toán qui hoạch động trên cây là sắp xếp lại các đỉnh của cây theo một dãy tuyến tính trong đó các đỉnh con, cháu,... của một đỉnh luôn đứng sau đỉnh đó còn các đỉnh tổ tiên luôn đứng phía trước (quá trình này còn được gọi là một phép sắp xếp topo trên cây). Có thể làm điều này bằng cách định hướng lại các cạnh của cây từ một đỉnh gốc (root) nào đó. Sử dụng BFS và DFS đều làm được điều này. Tuy nhiên trong thực tế thường sử dụng DFS vì khi đó tất cả các đỉnh con, cháu,... của một đỉnh làm thành một dãy liên tục trên thứ tự sắp xếp, do vậy, có thể sử dụng các cấu trúc áp dụng trên các đoạn liên tục để tăng tốc độ thuật toán (Segment Tree, Binary Indexed Tree...).

Tham khảo code dưới đây:

```
int Tp[maxn];              // Mảng thứ tự sắp xếp topo
int start[u];              // Mảng vị trí ban đầu (đối ngẫu với mảng Tp)
int stop[u];               // Mảng vị trí kết thúc DFS
int Time=0;                // Biến đếm

void DFS(int u,int p) {
    start[u]=++Time;
    Tp[Time]=u;
    depth[u]=depth[p]+1;
    parent[u]=p;
    for(int v: adj[u])
        if (v!=p) DFS(v,u);
    stop[u]=Time;
}
```

Một số kết quả đáng lưu ý khi thực hiện DFS trên cây từ một đỉnh gốc:

- Đỉnh u bất kỳ nằm tại vị trí $start[u]$ thỏa mãn yêu cầu: Tất cả các đỉnh con, cháu,... của u nằm ở phía sau còn tất cả các đỉnh tổ tiên của u nằm ở phía trước.
- Các đỉnh con, cháu,... của đỉnh u nằm từ vị trí $start[u]+1$ đến $stop[u]$
- Đỉnh u là tổ tiên của đỉnh v khi và chỉ khi $start[u] \le start[v] \le stop[v] \le stop[u]$

Hệ quả: Ta có hai qui trình qui hoạch động:

1) Tính công thức qui hoạch động của một đỉnh qua các đỉnh con của nó:

```
for(int i=n;i>=1;--i) {
    u=Tp[i];
    for(int v : adj[u]) if (parent[v]==u) {
            ....// Triển khai qui hoạch động tính từ u qua v
    }
}
```

2) Tính công thức qui hoạch động của một đỉnh qua đỉnh cha của nó

```
for(int i=1;i<=n;++i) {
    u=Tp[i]; w=parent[u];
    if (w) {
    .... // Triển khai công thức qui hoạch động tuwinhst tại u qua w
}
```

Ngoài ra, do tập hợp các đỉnh của cây con gốc u nằm từ vị trí $start[u]$ đến $stop[u]$ nên bài toán về sự thay đổi thông tin của toàn bộ một cây con được qui về bài toán thay đổi thông tin của một dãy liên tục. Do vậy có thể sử dụng các cấu trúc dữ liệu (ST, BIT,...) để làm điều này.

II. Một số bài toán qui hoạch động điển hình.

II.1 Bài toán 1: Cho một cây vô hướng n đỉnh (các đỉnh đánh số 1, 2, ..., n) và một đỉnh **root** là đỉnh gốc. Bằng cách định hướng lại các cạnh của cây từ root thì mỗi đỉnh sẽ là gốc của một cây con. Hãy Tìm độ dài đường đi dài nhất (tính bằng số cạnh) của đường đi đơn từ mỗi đỉnh đến các đỉnh con, cháu... của nó.

Đặt $f[u]$ là độ dài lớn nhất của đường đi đơn từ đỉnh u đến các đỉnh con, cháu,... ta có công thức qui hoạch động:

$$f[u] = \max\{f[v] + 1 : parent[v] = u\}$$

(Nếu u không có con thì $f[u] = 0$)

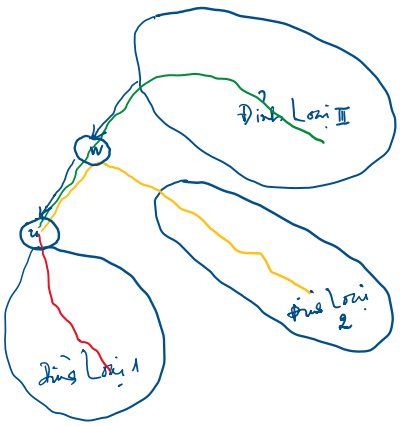
Tham khảo code:

```
for(i=n;i>=1;--i) {
    u=Tp[i]; f[u]=0;
    for(int v: adj[u])
        if (parent[v]==u) f[u]=max(f[u],f[v]+1);
    }
}
```

Cũng có thể kết hợp tính công thức qui hoạch động ngay trong quá trình DFS:

```
void DFS(int u,int p) {
    f[u]=0;
    for(int v: adj[u]) if (v!=p) {
        DFS(v,u);
        f[u]=max(f[u],f[v]+1);
    }
}
```

II.2 Bài toán 2: Tìm đường đi đơn dài nhất từ đỉnh u đến đỉnh khác trên cây
Ta chia các đường đi đơn từ đỉnh u thành 3 loại như hình vẽ dưới đây:



- Loại 1: Đường đi xuất phát từ u đến các đỉnh con, cháu,....trên hình vẽ là các đường màu đỏ.
- Loại 2: Đường đi xuất phát từ u qua đỉnh $w = parent[u]$ sau đó đến các đỉnh con cháu,... của w (không phải là u và con cháu của u). Trên hình vẽ là đường màu vàng
- Loại 3: Đường đi xuất phát từ u qua đỉnh $w = parent[u]$ và đi tiếp đến các đỉnh không phải con cháu của w. Trên hình vẽ là các đường màu xanh lá cây.

Đặt:

- $f[u]$ là đường dài nhất loại 1, $f2[u]$ là đường dài thứ nhì loại 1
- $g[u]$ là đường dài nhất loại 2 và 3

Ta có các công thức qui hoạch động như sau:

$f[u] = max\{f[v] + 1: parent[v] = u\}$
$f2[u]$ được tính bằng cách cập nhật $f[u]$
$g[u] = \max(L2, L3)$ trong đó $L2$ là độ dài lớn nhất các đường loại 2, $L3$ là độ dài lớn nhất các đường loại 3. Ta có các công thức để tính $L2$ và $L3$ như sau:

$$L2 = \begin{cases} 1 + f[w] & \text{nếu } f[w] > f[u] + 1 \\ 1 + f2[w] & \text{nếu } f[w] = f[u] + 1 \end{cases}$$

$L3 = 1 + g[w]$

Đường đi dài nhất từ đỉnh u đến các đỉnh khác được tính bằng công thức $\mathbf{max}(\boldsymbol{f}[\boldsymbol{u}], \boldsymbol{g}[\boldsymbol{u}])$.

Tham khảo code dưới đây:

```
// Tính f, f2
for(int i=n;i>=1;--i) {
    int u=Tp[i];
    f[u]=f2[u]=0;
    for(int v: adj[u]) if (parent[v]==u) {
        if (f[u]<f[v]+1) f2[u]=f[u], f[u]=f[v]+1;
        else if (f2[u]<f[v]+1) f2[u]=f[v]+1;
    }
}
// Tính g
for(int i=1;i<=n;++i) {
    int u=Tp[i];
    int w=parent[u];
    g[u]=0;
    if (w) g[u]=max(1+g[w], (f[w]==f[u]+1) ? 1+f2[w] : 1+f[w]);
}
```

II.3 Bài toán 3: Cho một cây n đỉnh (đánh số 1, 2, ..., n) đỉnh i được gán một trọng số a_i. Hãy tìm cây con (đồ thị con liên thông) có m đỉnh sao cho tổng trọng số các đỉnh giữ lại là lớn nhất.

Định hướng lại các cạnh của cây với gốc là 1. Khi đó mỗi đỉnh u là gốc một cây con. Đặt $s[u]$ là số đỉnh của cây con này. Ta có thể tính $s[u]$ như là bài toán qui hoạch động cơ bản:

```
for(int i=n;i>=1;--i) {
    int u=Tp[i];
    s[u]=1;
    for(int v: adj[u]) if (parent[v]==u) s[u]+=s[v];
}
```

Đặt $f[u,k]$ là tổng giá trị lớn nhất của cây con có k đỉnh với gốc là u. Gọi $v_1, v_2, \dots, v_p$ là các đỉnh con của u. Ta có công thức:

$$f[u,k] = a_u + \max\{f[v_1,x_1] + \cdots + f[v_p,x_p] : x_1 + \cdots x_p = k-1\}$$

Ta có một bài toán gần giống bài toán cổ điển "xếp vali". Chú ý rằng khi code, các đỉnh v_i xuất hiện dần.

Tham khảo code dưới đây:

```
vector<vector<int> > f, nho;

f.resize(n+1,vector<int>(n+1,-INF));
nho.resize(n+1,vector<int>(n+1,0));
for(int i=n;i>=1;--i) {
    int u=Tp[i];
    f[u][0]=0; f[u][1]=a[u];
    int sd=1;
    for(int v: adj[u]) if (parent[v]==u) {
        sd += s[v];
        for(int k=sd;k>=1;--k)
        for(int l=1;l<k;++l) {
            if (l>s[v]) break;
            if (f[u][k]<f[u][k-l]+f[v][l]) {
                f[u][k]=f[u][k-l]+f[v][l];
                nho[v][k]=l;
            }
        }
    }
}

// Tìm gốc của cây m đỉnh cực đại
int root=1;
for(int i=1;i<=n;++i) if (f[root][m]<f[i][m]) root=i;
// Tìm lại các cạnh của cây m đỉnh cực đại
Find_Tree(root,m);
```

Ở đây hàm Find_Tree được viết như sau:

```
vector<int> Tree;
Tree.resize(n+1,0);

void Find_Tree(int u,int i) {
    Tree[u]=1; // u là đỉnh được chọn của cây
    for(int v: adj[u]) if (parent[v]==u && nho[v][i]) {
         Find_Tree(v,nho[v][i]);
         i -= nho[v][i];
         if (!i) return;
    }
}
```

II.4 Bài toán 4: Cho một cây vô hướng n đỉnh (đánh số 1, 2, ..., n) được định hướng từ 1. Mỗi đỉnh được gán một giá trị. Khởi đầu giá trị của đỉnh u là a_u. Thực hiện m thao tác thuộc một trong hai loại:

- **1 u D:** Tăng giá trị các đỉnh thuộc cây con gốc u một lượng D
- **2 u** : Tìm giá trị lớn nhất của đỉnh trong cây con gốc u

Khi định hướng lại, tất cả các đỉnh thuộc cây con gốc u nằm từ vị trí $start[u]$ đến $stop[u]$. Do vậy dễ thấy cấu trúc sử dụng ở đây là một Segment Tree max với cập nhật lười tăng:

+) Thực hiện thao tác loại 1: **Tree.update(root, start[u], stop[u], D);**

+) Thực hiện thao tác loại 2: **cout << Tree.get(root,start[u], stop[u]);**