

Quy hoạch động trên cây

Quy hoạch động trên cây

Người viết:

- Phạm Công Minh, THPT chuyên Khoa học Tự Nhiên, ĐHQGHN.

Reviewer:

- Lê Minh Hoàng, Đại học Khoa học Tự nhiên, ĐHQG-HCM.
- Nguyễn Minh Hiền, Trường Đại học Công nghệ, ĐHQGHN.
- Nguyễn Minh Nhật, THPT chuyên Khoa học Tự Nhiên, ĐHQGHN.

Giới thiệu

Bài viết này sẽ đề cập một số bài toán quy hoạch động trên cây điển hình. Nếu chưa nắm vững về quy hoạch động, các bạn có thể đọc thêm về quy hoạch động ở [VNOI - Quy hoạch động](#) ☑.

Định nghĩa

Trước khi đi vào các bài toán, ta có một số các định nghĩa như sau:

- $w(u, v)$ là trọng số của cạnh nối đỉnh u và đỉnh v
- $d(i, j)$ là độ dài đường đi **ngắn nhất** từ đỉnh i đến đỉnh j .
- Trong một cây T có đỉnh gốc là R , đỉnh v nằm trong cây con đỉnh u nếu đường đi ngắn nhất từ R đến v đi qua u .
- $\text{subtree}(V)$ là tập hợp tất cả các đỉnh nằm trong cây con V , trong đó có cả V .
- $\text{sz}(V)$ là số đỉnh nằm trong cây con V , hay $\text{sz}(V) = |\text{subtree}(V)|$.

Bài toán 1

Đề bài

Cho một cây T gồm N đỉnh ($1 \leq N \leq 2 \times 10^5$), đỉnh thứ i có một đồng xu giá trị C_i ($|C_i| \leq 10^9$). Bài toán yêu cầu chọn một tập con các đỉnh sao cho tập con không tồn tại 2 đỉnh được nối trực tiếp bởi một cạnh và tổng các đồng xu trên các đỉnh được chọn là lớn nhất.

Lời giải

Bài toán này khá giống với một bài toán quy hoạch động trên mảng 1 chiều: cho một mảng A_1, A_2, \dots, A_n , chọn một tập con các phần tử sao cho không có 2 phần tử nằm cạnh nhau đều được chọn và tổng các phần tử là lớn nhất.

Đối với bài toán trên mảng 1 chiều, ta định nghĩa trạng thái quy hoạch động $dp(i)$ là đáp án của bài toán nếu xét i phần tử đầu A_1, A_2, \dots, A_i .

Ta có 2 trường hợp là chọn A_i hoặc không chọn A_i . Vì vậy ta suy ra được công thức truy hồi $dp(i) = \max(dp(i-1), dp(i-2) + A_i)$

Đối với bài toán trên cây, thay vì xét một tiền tố, trạng thái quy hoạch động thường là đáp án nếu xét một cây con nào đó. Để định nghĩa cây con, cây cần có một gốc nào đó. Ta đặt gốc của cây là đỉnh 1 và định nghĩa $dp(V)$ là đáp án của bài toán nếu chỉ xét cây con của đỉnh V , như vậy, đáp án cuối cùng của bài toán là $dp(1)$.

Tương tự như bài toán trên mảng 1 chiều, tại mỗi đỉnh V ta cần quyết định có chọn V hay không. Nếu không chọn đỉnh V , ta có thể chọn các đỉnh con tùy ý. Nếu có chọn đỉnh V , ta không được chọn đỉnh con nhưng vẫn có thể chọn các đỉnh cháu của V

Gọi v_1, v_2, \dots, v_n là con của V . Như vậy ta có công thức truy hồi như sau:

$$dp(V) = \max \left(\sum_{i=1}^n dp(v_i), C_V + \sum_{i=1}^n \left(\sum_{j \in \text{subtree}(v_i) \setminus \{v_i\}} dp(j) \right) \right)$$

Để chuyển trạng thái gọn gàng hơn, ta định nghĩa 2 trạng thái quy hoạch động mới:

- $dp_1(V)$ là đáp án bài toán nếu chỉ xét các đỉnh trong cây con của V và đỉnh V không được chọn.
- $dp_2(V)$ là đáp án bài toán nếu chỉ xét các đỉnh trong cây con của V và đỉnh V được chọn.

Đối với $dp_1(V)$, đỉnh V không được chọn nên ta có thể chọn các con của V , do đó $dp_1(V) = \sum_{i=1}^n \max(dp_1(v_i), dp_2(v_i))$

Đối với $dp_2(V)$, đỉnh V được chọn nên ta không được chọn các con của V , do đó $dp_2(V) = C_V + \sum_{i=1}^n dp_1(v_i)$

Đáp án của bài toán sẽ là $\max(dp_1(1), dp_2(1))$.

Cài đặt

Do trạng thái quy hoạch động chỉ dựa vào các đỉnh con, ta chỉ cần duyệt và tính trong quá trình DFS:

```
1  const int MAX_N = 2e5 + 5;
2
3  int N;
4
5  int C[MAX_N]; //
6  long long dp1[MAX_N], dp2[MAX_N]; // Khai báo cây và mảng lưu tr
7  vector<int> adj[MAX_N]; //
8
9  void calc(int V, int pV) { // hàm tính dp1[V] và dp2[V]
10     dp1[V] = 0;
11     dp2[V] = C[V];
12     for(int v_i: adj[V]) {
13         if(v_i == pV) continue;
14         calc(v_i, V); // DFS xuống và tính các đỉnh con
15         dp1[V] += max(dp1[v_i], dp2[v_i]);
16         dp2[V] += dp1[v_i];
17     }
18 }
19
20 long long solve(){ // hàm tính và trả về đáp án
21     calc(1, 0);
22     return max(dp1[1], dp2[1]);
23 }
```

Độ phức tạp

- Ta thực hiện DFS trên cây một lần, do đó độ phức tạp thời gian là $O(N)$.
- Độ phức tạp không gian: $O(N)$

Bài toán 2 - Kỹ thuật chuyển gốc

Đề bài

Cho một cây T gồm N đỉnh ($1 \leq N \leq 2 \times 10^5$), các cạnh có trọng số nguyên dương không quá 10^9 . Tính $\sum_{i=1}^n d(k, i)$ với mọi k từ 1 đến n . Nói cách khác, với mỗi đỉnh, tìm tổng khoảng cách từ đỉnh đó đến mọi đỉnh từ 1 đến n .

Lời giải

Ta xét bài toán nếu chỉ cần tính với $k = 1$. Ta đặt gốc của cây là 1 và định nghĩa trạng thái quy hoạch động: $\text{dp}(V) = \sum_{j \in \text{subtree}(V)} d(V, j)$. Nói cách khác, $\text{dp}(V)$ lưu tổng khoảng cách từ đỉnh V đến mọi đỉnh trong cây con của chính nó.

Để xây dựng công thức truy hồi, ta cần xét đóng góp của từng đỉnh con của V vào $\text{dp}(V)$.

Xét đỉnh con v_i của V . Ta nhận thấy mọi đường đi từ V đến một đỉnh bất kỳ trong cây con của v_i đều đi qua cạnh (V, v_i) . Do đó, với một đỉnh j nằm trong cây con của v_i , ta có:

$$d(V, j) = w(V, v_i) + d(v_i, j)$$

Như vậy, tổng khoảng cách từ V đến mọi đỉnh trong cây con của v_i là:

$$\begin{aligned} \sum_{j \in \text{subtree}(v_i)} d(V, j) &= \sum_{j \in \text{subtree}(v_i)} (w(V, v_i) + d(v_i, j)) \\ &= \sum_{j \in \text{subtree}(v_i)} w(V, v_i) + \sum_{j \in \text{subtree}(v_i)} d(v_i, j) \\ &= \text{sz}(v_i) \times w(V, v_i) + \text{dp}(v_i) \end{aligned}$$

Từ đây, ta có công thức chuyển đổi trạng thái quy hoạch động:

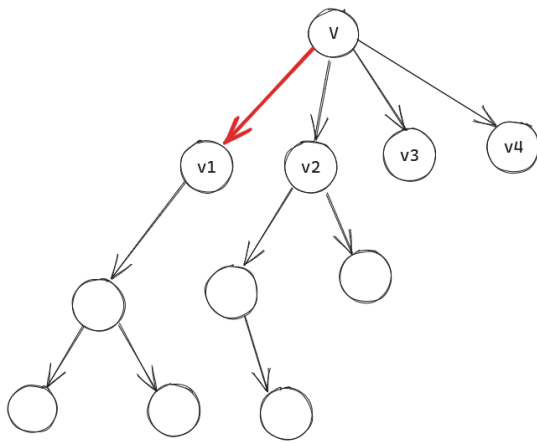
$$\text{dp}(V) = \sum_{i=1}^n (\text{sz}(v_i) \times w(V, v_i) + \text{dp}(v_i))$$

Việc tính $\text{sz}(V)$ là một bài toán quy hoạch động trên cây cơ bản.

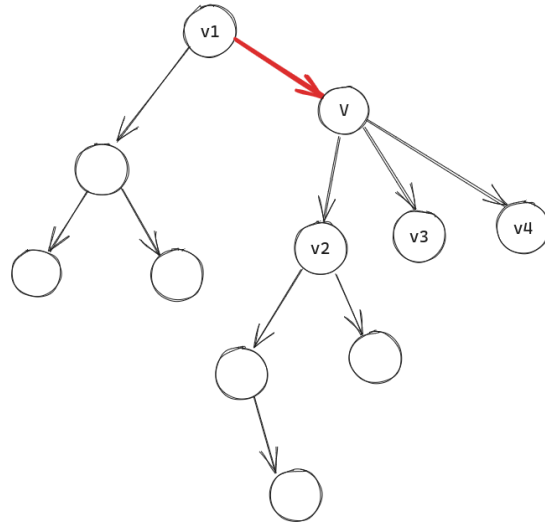
Quay lại bài toán ban đầu, do công thức này chỉ tính được đáp án cho gốc đã chọn, nếu ta duyệt và chọn từng đỉnh làm gốc rồi tính lại, ta giải được bài toán với độ phức tạp $O(n^2)$. Ở đây ta sẽ sử dụng kĩ thuật **quy hoạch động chuyển gốc**, cụ thể như sau:

Giả sử gốc hiện tại đang là V , ta cần tìm cách chuyển gốc thành đỉnh con v_i trong $O(1)$. Ta có một số nhận xét như sau:

- Ngoài đỉnh V và v_i , tất cả các đỉnh khác đều có cây con không thay đổi, do đó trạng thái quy hoạch động của chúng cũng không thay đổi.
- Đối với đỉnh V , cây con của V sẽ bỏ đi cây con v_i . Khi thành gốc mới, cây con v_i sẽ bao gồm thêm cây con mới của V .



Cây trước khi chuyển gốc



Cây sau khi chuyển gốc

Ta gọi dp_{old} và dp_{new} lần lượt là trạng thái quy hoạch động trước và sau khi đổi gốc, sz_{old} và sz_{new} lần lượt là số đỉnh trong cây con trước và sau khi đổi gốc. Dựa vào các nhận xét trên, ta có:

- ▶ Đối với các đỉnh j không phải V và v_i , ta có $dp_{new}(j) = dp_{old}(j)$ và $sz_{new}(j) = sz_{old}(j)$.
- ▶ Ta tính được $dp_{new}(V) = dp_{old}(V) - (sz_{old}(v_i) \times w(V, v_i) + dp_{old}(v_i))$ và $sz_{new}(V) = sz_{old}(V) - sz_{old}(v_i)$.
- ▶ Ta tính được $dp_{new}(v_i) = dp_{old}(v_i) + (sz_{new}(V) \times w(V, v_i) + dp_{new}(V))$ và $sz_{new}(v_i) = N$.

Như vậy, ta có thể chuyển gốc sang một đỉnh con của gốc hiện tại trong $O(1)$. Nếu ta chuyển gốc theo thứ tự duyệt DFS, ta giải được bài toán trong độ phức tạp $O(N)$.

Cài đặt

```
const int MAX_N = 2e5 + 5;

int N;

long long dp[MAX_N], sz[MAX_N], ans[MAX_N];
vector<pair<int, int>> adj[MAX_N]; // mảng kề lưu pair (chỉ số, t

void pre_calc(int V, int pV) { // hàm tính sz và dp ban đầu
    dp[V] = 0;
    sz[V] = 1;
    for(pair<int, int> edge: adj[V]) {
        int v_i = edge.first;
        int w = edge.second;
        if(v_i == pV) continue;
```

```

15     pre_calc(v_i, V);
16     dp[V] += sz[v_i]*w + dp[v_i];
17     sz[V] += sz[v_i];
18 }
19 }
20
21 void calc(int V, int pV) { // hàm chuyển gốc theo thứ tự DFS trong
22
23     ans[V] = dp[V]; // đáp án tại đỉnh V là dp[V] khi V là gốc
24
25     for(pair<int, int> edge: adj[V]) {
26         int v_i = edge.first;
27         int w = edge.second;
28         if(v_i == pV) continue;
29         long long old_dp_v_i = dp[v_i]; //
30         long long old_dp_V = dp[V]; // Khai báo các giá trị cũ
31         long long old_sz_v_i = sz[v_i]; //
32         long long old_sz_V = sz[V]; //
33         dp[V] = old_dp_V - (old_sz_v_i*w + old_dp_v_i); //
34         sz[V] = old_sz_V - old_sz_v_i; // Chuyển
35         dp[v_i] = old_dp_v_i + (sz[V]*w + dp[V]); //
36         sz[v_i] = N; //
37
38         calc(v_i, V); // tiếp tục chuyển gốc trong cây con v_i
39
40         dp[V] = old_dp_V; //
41         sz[V] = old_sz_V; // Chuyển gốc từ v_i thành V
42         dp[v_i] = old_dp_v_i; //
43         sz[v_i] = old_sz_v_i; //
44     }
45 }
46
47 void solve() { // hàm tính đáp án
48
49     pre_calc(1, 0); // tính dp và sz với gốc là đỉnh 1
50
51     calc(1, 0); // chuyển gốc và tính đáp án bắt đầu từ đỉnh 1
52
53 }

```

Độ phức tạp

- Độ phức tạp thời gian: $O(N)$
- Độ phức tạp thời gian: $O(N)$

Bài toán 3 - Kỹ thuật knapsack trên cây

Đề bài

Cho cây T gồm N đỉnh có gốc là đỉnh 1 ($1 \leq N \leq 5000$), đỉnh thứ i có giá trị là C_i và một chỉ số K_i ($|C_i| \leq 10^9, 1 \leq K_i \leq N$). Chọn một tập con các đỉnh sao cho trong cây con



Lời giải

Bài toán này khá tương đồng với bài toán quy hoạch động cơ bản knapsack.

Trạng thái quy hoạch động ở bài này $dp(V, k)$ là tổng giá trị lớn nhất nếu chọn k đỉnh trong cây con V thỏa mãn điều kiện đề bài. Nếu không có cách nào chọn k đỉnh mà vẫn thỏa mãn các điều kiện, ta định nghĩa $dp(V, k) = -\infty$. Như vậy đáp án của bài toán là $\max_{k=0}^n dp(1, k)$

Để tính được giá trị này, ta cần thêm một trạng thái quy hoạch động phụ. Gọi v_1, v_2, \dots, v_n là con của V và $f_V(i, k)$ là tổng giá trị lớn nhất nếu chọn k đỉnh trong các cây con của i đỉnh con đầu tiên thỏa mãn điều kiện đề bài. Nói cách khác, $f_V(i, k)$ là trạng thái quy hoạch động cho tập các đỉnh thuộc $subtree(v_1), subtree(v_2), \dots, subtree(v_i)$.

Theo định nghĩa này, ta có $f_V(0, 0) = 0$ và $f_V(0, k) = -\infty \forall k > 0$. Ta tính được $f_V(i, k)$ dựa theo $f_V(i-1, k)$:

$$f_V(i, k) = \max_{j=0}^k (f_V(i-1, j) + dp(v_i, k-j))$$

Để tính $dp(V, k)$, ta có 2 lựa chọn là lấy hoặc không lấy đỉnh V . Kết hợp điều kiện đề bài, ta có:

$$dp(V, k) = \begin{cases} -\infty & \text{nếu } k > K_i \\ \max(f_V(n, k), f_V(n, k-1) + C_V) & \text{nếu } k \leq K_i \end{cases}$$

Độ phức tạp để tính $f_V(i, k)$ là $O(k) = O(N)$ do giá trị của k không quá N , và độ phức tạp để tính $dp(V, k)$ là $O(1)$. Có $O(N^2)$ trạng thái $f_V(i, k)$ và N^2 trạng thái $dp(V, k)$ cần tính nên độ phức tạp tổng là $O(N \times N^2 + N^2) = O(N^3)$.

Cài đặt 1

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

const int MAX_N = 5e3+5;
const long long INF = -1e18;

int N;
```

```

9
10 int C[MAX_N], K[MAX_N], sz[MAX_N];
11 long long dp[MAX_N][MAX_N];
12 long long fV[MAX_N][MAX_N];
13 vector<int> child[MAX_N]; // child[V] chứa các con của V
14
15 void calc(int V){
16     int n = child[V].size();
17
18     for(int v_i: child[V]) {
19         calc(v_i);
20     }
21
22     for(int i = 0; i <= n; i++) fill(fV[i], fV[i] + N + 1, -INF);
23     fV[0][0] = 0;
24
25     for(int i = 1; i <= n; i++){
26         int v_i = child[V][i - 1];
27         for(int k = 0; k <= N; k++){
28             for(int j = 0; j <= k; j++){
29                 fV[i][k] = max(fV[i][k], fV[i-1][j] + dp[v_i][k -
30             }
31         }
32     }
33
34     for(int k = 0; k <= N; k++){
35         if(k > K[V]) dp[V][k] = -INF;
36         else {
37             if(k > 0) dp[V][k] = max(fV[n][k], fV[n][k-1] + C[V]);
38             else dp[V][k] = fV[n][k];
39         }
40     }
41 }
42
43 long long solve() {
44     calc(1);
45     return *max_element(dp[1], dp[1] + N + 1);
46 }

```

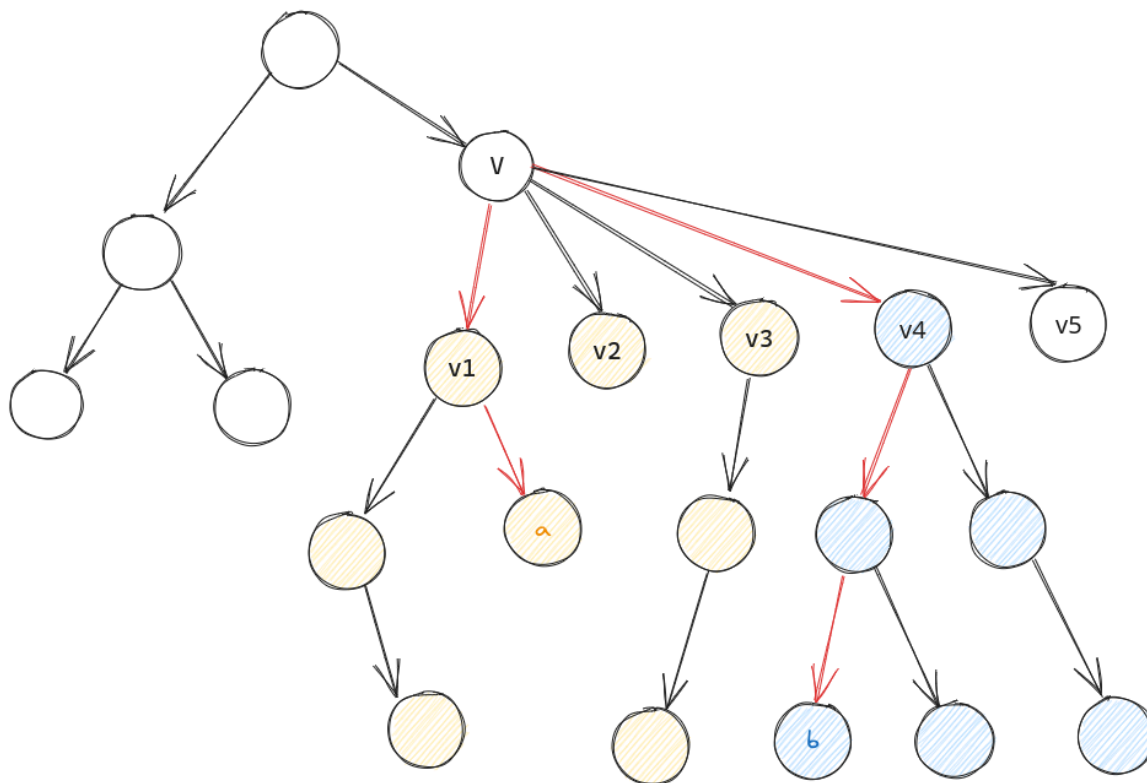
Tối ưu

Tuy nhiên ta có thể làm tốt hơn như thế. Nhận thấy rằng khi tính $f_V(i, k)$ dựa vào $f_V(i-1, k)$ và $dp(v_i, k)$, ta có thể duyệt qua mọi cặp $f_V(i-1, a)$ và $dp(v_i, b)$ rồi cập nhật vào $f_V(i, a+b)$. Ta cần duyệt mọi cặp a, b thỏa mãn $a \leq \sum_{j=1}^{i-1} sz(v_j)$ và $b \leq sz(v_i)$.

Tổng số cặp cần duyệt là $(\sum_{j=1}^{i-1} sz(v_j)) \times sz(v_i)$. Ta biểu diễn tổng này thành $\sum_{1 \leq j < i} sz(v_j) \times sz(v_i)$.

Số cặp duyệt này tương đương với số cặp đỉnh a, b sao cho a nằm trong cây con v_i và b nằm trong cây con v_j thỏa mãn $j < i$.

Do 2 đỉnh a và b nằm ở 2 cây con khác nhau của đỉnh V , V sẽ là tổ tiên chung thấp nhất của a và b . Mỗi cặp đỉnh chỉ có duy nhất một tổ tiên chung thấp nhất, do đó với cách duyệt này mỗi cặp đỉnh chỉ được duyệt tối đa 1 lần. Có $\frac{N \times (N-1)}{2}$ cặp đỉnh khác nhau, do đó độ phức tạp của cách duyệt này là $O(N^2)$!



Tổ tiên chung của a và b là V

Cài đặt 2

```
const int MAX_N = 5e3+5;
const long long INF = 1e18;

int N;

int C[MAX_N], K[MAX_N], sz[MAX_N];
long long dp[MAX_N][MAX_N];
long long fV[MAX_N][MAX_N];
vector<int> child[MAX_N]; // child[V] chứa các con của V
```

```

12 void calc(int V){
13     int n = child[V].size();
14
15     for(int v_i: child[V]) {
16         calc(v_i);
17     }
18
19     for(int i = 0; i <= n; i++) fill(fV[i], fV[i] + N + 1, -INF);
20     fV[0][0] = 0;
21
22     for(int i = 1; i <= n; i++){
23         int v_i = child[V][i - 1];
24         for(int a = 0; a <= sz[V]; a++){           // sz[V] lưu tổng
25             for(int b = 0; b <= sz[v_i]; b++){
26                 fV[i][a+b] = max(fV[i][a+b], fV[i-1][a] + dp[v_i][
27                     ]
28             }
29             sz[V] += sz[v_i];
30         }
31
32         for(int k = 0; k <= N; k++){
33             if(k > K[V]) dp[V][k] = -INF;
34             else {
35                 if(k > 0) dp[V][k] = max(fV[n][k], fV[n][k-1] + C[V]);
36                 else dp[V][k] = fV[n][k];
37             }
38         }
39         sz[V]++;
40     }
41
42     long long solve() {
43         calc(1);
44         return *max_element(dp[1], dp[1] + N + 1);
45     }







```

Độ phức tạp

- Độ phức tạp thời gian: $O(N^2)$
- Độ phức tạp không gian: $O(N^2)$

Tuy nhiên, ta có thể phân tích độ phức tạp thời gian kĩ hơn. Giả sử bài toán trên có điều kiện ($1 \leq K_i \leq M$), khi đó việc cập nhật thêm một cây con v vào knapsack có độ phức tạp $O(\min(M, sz(v)))$ và độ phức tạp tổng sẽ là $O(N \times \min(N, M))$. Phần chứng minh sẽ là bài tập dành cho bạn đọc

Bài tập tham khảo

- [Codeforces - Fake plastic tree](#) 
- [Codeforces - Hanging Hearts](#) 
- [HackerRank - Tree Pruning](#) 
- [Codeforces - Appleman and Tree](#) 
- [Codeforces - Berland Federalization](#) 
- [Codeforces - Multiset of Strings](#) 

Được cung cấp bởi [Wiki.js](#)