

Thuật toán phân tách trọng tâm - Centroid decomposition

Thuật toán phân tách trọng tâm - Centroid decomposition

Tác giả:

- Cao Thanh Hậu - Trường đại học Khoa học Tự Nhiên - ĐHQG-HCM

Reviewer:

- Lê Minh Hoàng - Đại học Khoa học Tự nhiên - ĐHQG-HCM
- Hồ Ngọc Vĩnh Phát - Đại học Khoa học Tự nhiên - ĐHQG-HCM
- Ngô Nhật Quang - Trường THPT chuyên Khoa học Tự Nhiên - ĐHQGHN

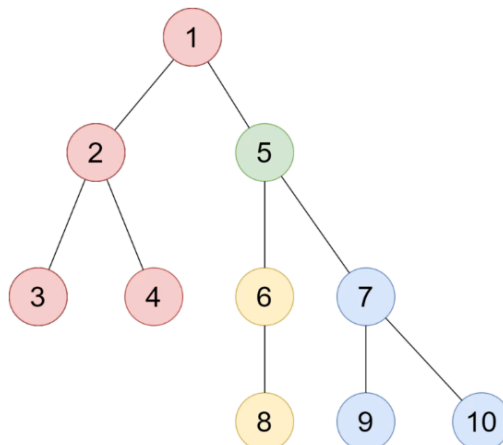
Giới thiệu

Thuật toán phân tách trọng tâm có thể hiểu là thuật toán "chia để trị" trên cây. Thuật toán này hoạt động bằng cách liên tục chia nhỏ cây và xử lý trên mỗi cây được chia.

Trọng tâm của cây

Định nghĩa

Trọng tâm của cây - centroid - là một đỉnh trên cây mà khi bỏ nó ra khỏi cây, mỗi thành phần liên thông còn lại có số đỉnh không quá một nửa số đỉnh của cây ban đầu.



Cây trong hình trên có trọng tâm là đỉnh 5.

Trọng tâm của cây có thể không độc nhất

Trong một số trường hợp, cây có thể có 2 trọng tâm, và khi đó 2 trọng tâm của cây sẽ kề nhau. Tuy nhiên, điều này không làm ảnh hưởng đến các thuật toán được nêu trong bài.

Tìm trọng tâm của cây

Từ định nghĩa, ta có ý tưởng cơ bản để tìm trọng tâm của cây như sau: giả sử trọng tâm của cây chắc chắn thuộc cây con gốc u , với mọi đỉnh v là con trực tiếp của u , nếu cây con gốc v có nhiều hơn $n/2$ đỉnh thì trọng tâm của cây chắc chắn thuộc cây con gốc v . Nếu không tìm được đỉnh v nào thỏa mãn thì u chính là trọng tâm của cây. Trong đó, n là số đỉnh của cây.

Mở rộng ý tưởng, cần xây dựng hàm $findCentroid(u)$ với ý nghĩa: $findCentroid(u)$ được gọi chỉ khi trọng tâm cây chắc chắn thuộc cây con gốc u , và giá trị trả về của hàm luôn là trọng tâm của cây. Bên trong hàm này thực hiện tìm v như ý tưởng cơ bản. Nếu tìm được v , trả về $findCentroid(v)$, nếu không tìm được thì trả về u .

Sau đây là code ví dụ, lưu ý trước khi tìm trọng tâm, ta cần gọi hàm $countChild(root, 0)$ để đếm số lượng đỉnh thuộc từng cây con. Để tìm centroid, gọi $findCentroid(root, 0)$, với $root$ là gốc của cây (có thể chọn bất cứ đỉnh nào).

```

1  int n;           // n là số đỉnh của cây ban đầu
2  int child[N];    // child[u] là số đỉnh thuộc cây con gốc u
3
4  void countChild(int u, int parent) {
5      child[u] = 1; // cây con gốc u có ít nhất 1 đỉnh là đỉnh u
6      for (int v : adj[u]) { // với mọi v kề u
7          if (v != parent) { // nếu v là con của u
8              countChild(v, u);
9              child[u] += child[v];
10         }
11     }
12 }
13
14 int findCentroid(int u, int parent) {
15     for (int v : adj[u]) {
16         if (v != parent) {
17             if (child[v] > n/2) { // tìm được v thỏa mãn
18                 return findCentroid(v, u);
19             }
20         }
21     }
22     return u; // không có giá trị v nào thỏa mãn, trả về u
23 }

```

Code trên hoạt động với độ phức tạp là $O(n)$ (lưu ý, n là số đỉnh của cây **đang xét**).

Từ định nghĩa hàm $findCentroid(u)$ cũng có thể chứng minh trọng tâm của cây luôn tồn tại. Khi $findCentroid()$ dừng lại tại đỉnh u ($findCentroid(u)$ trả về u) ta biết rằng các cây con có gốc là con của

u đều đã thỏa mãn điều kiện có số đỉnh không vượt quá $n/2$. Đồng thời khi $findCentroid(u)$ được gọi ta cũng biết số lượng đỉnh thuộc cây con gốc u không nhỏ hơn $n/2$, vậy số lượng đỉnh không thuộc cây con gốc u cũng không vượt quá $n/2$. Vậy khi xóa đỉnh u đi thì mọi cây tạo thành đều có số đỉnh không vượt quá $n/2$.

Thuật toán phân tách trọng tâm - Centroid decomposition

Bài toán

Ta sẽ cùng giải quyết một bài toán điển hình như sau: Cho một cây có n đỉnh, đếm số đường đi trên cây có độ dài k .

Phân tích

Nếu thêm điều kiện để bài toán trở thành "Đếm số đường đi trên cây độ dài k đi qua một đỉnh cho trước" thì vấn đề đơn giản hơn khá nhiều.

Để giải quyết bài toán với điều kiện đi qua một đỉnh cho trước, ta chỉ cần chọn đỉnh đó là gốc, lúc này, với mỗi đỉnh v là con trực tiếp của đỉnh gốc, mỗi đỉnh thuộc cây con gốc v có khoảng cách đến gốc là d có thể ghép với tất cả các đỉnh không thuộc cây con gốc v và có khoảng cách đến gốc là $k - d$ để tạo thành một đường đi độ dài k đi qua đỉnh gốc.

Có thể dfs để xây dựng các mảng đếm số lượng đỉnh có khoảng cách đến gốc là $0, 1, 2, 3, \dots$ trên cây và trong mỗi cây con gốc v , khi đó dễ dàng tính được số lượng đường đi thỏa mãn. Độ phức tạp của cách làm này là $O(n)$, với n là số đỉnh của cây đang xét.

Trở lại bài toán ban đầu, làm sao để chuyển từ "số lượng đường đi chứa một đỉnh cố định" thành "số lượng đường đi trên cây"? Khi chọn một đỉnh làm gốc, thấy rằng mọi đường đi trên cây có thể chia thành 2 nhóm: đi qua đỉnh gốc và không đi qua đỉnh gốc. Từ đây ta có ý tưởng như sau: sau mỗi lần đếm số đường đi thỏa mãn đi qua một đỉnh cố định, ta xóa đỉnh đó đi, với mỗi cây mới tạo thành, ta lại thực hiện việc đếm như trên rồi lại xóa đỉnh đi, đến khi mọi đỉnh đều bị xóa.

Cách làm trên cho kết quả chính xác, vì mọi đường đi trên cây đều được xét qua và mỗi đường đi trên cây được xét qua đúng một lần (sau lần đầu tiên, một trong hai đầu mút của đường đi bị xóa hoặc sẽ thuộc về hai cây mới riêng biệt nhau, vì vậy sẽ không được xét lại lần hai).

Tuy nhiên, cách này có độ phức tạp khá lớn trong một số trường hợp. Ví dụ cây là đường thẳng, ta lại liên tục chọn một đầu mút của cây để làm đỉnh cố định, vậy sau mỗi lần xóa, số đỉnh trên cây chỉ giảm đi 1, độ phức tạp tổng sẽ là $O(n + (n - 1) + (n - 2) + (n - 3) + \dots) \approx O(n^2)$.

Thuật toán

Cũng theo ý tưởng trên, nhưng thuật toán phân tách trọng tâm cho cách chọn đỉnh tối ưu hơn, làm giảm độ phức tạp của thuật toán. Cụ thể, thuật toán hoạt động như sau:

1. Chọn trọng tâm của cây làm **gốc** của cây.
2. Đếm số lượng đường đi trên cây thỏa mãn yêu cầu và có chứa gốc của cây.

3. Xóa đỉnh gốc. Nếu trước khi xóa cây có nhiều hơn 1 đỉnh (khi đó tạo thành một hoặc một số cây riêng biệt khác) thì với mỗi cây mới được tạo, trở lại bước 1.

Độ phức tạp của thuật toán bằng $\log(n)$ nhân cho độ phức tạp của bước 2. Nếu bước 2 được thực hiện trong $O(m)$, với m là số đỉnh của cây đang xét lúc đó, thì độ phức tạp tổng sẽ là $O(n \times \log(n))$. Nếu bước 2 được thực hiện trong $O(m \log(m))$, thì độ phức tạp tổng là $O(n \times \log(n)^2)$.

Giải thích

Giả sử ta xếp các cây được xét thành nhiều hàng, bắt đầu từ hàng 0, mỗi hàng gồm một số cây theo quy luật: hàng 0 chứa cây ban đầu, hàng thứ i (i từ 1 trở đi) chứa các cây tạo được từ việc phân tách một cây nào đó ở hàng $i - 1$.

Xếp theo quy luật trên thì tổng kích thước (số đỉnh) của tất cả các cây trên một hàng không vượt quá n (gọi n là số đỉnh của cây ban đầu). Tất cả cây ở hàng thứ i có kích thước không quá một nửa kích thước của cây to nhất ở hàng thứ $i - 1$, hay nói cách khác, mỗi cây ở hàng i có số đỉnh không vượt quá $\frac{n}{2^i}$. Vậy chỉ có thể có nhiều nhất $\log(n)$ hàng.

Vậy tổng số đỉnh của tất cả các cây tạo thành từ thuật toán trên không vượt quá $n \times \log(n)$, đây cũng chính là độ phức tạp của thuật toán.

Cài đặt

```

1  #include<bits/stdc++.h>
2
3  using namespace std;
4
5  const int N = 200005;
6
7  int n, k, child[N], del[N]; // del[u] để kiểm tra đỉnh u có bị xóa hay chưa
8  vector<int> adj[N];
9
10 void countChild(int u, int parent) {
11     child[u] = 1;
12     for (int v : adj[u]) if (v != parent && !del[v]) {
13         countChild(v, u);
14         child[u] += child[v];
15     }
16 }
17
18 int centroid(int u, int parent, int n) {
19     for (int v : adj[u])
20         if (v != parent && child[v] > n/2 && !del[v])
21             return centroid(v, u, n);
22     return u;
23 }
24
25 void updateAns(int root, int n) {
26     //hàm thực hiện bước 2
27 }
28

```


```

29
30 long long solve(int u) {
31     countChild(u, 0);
32
33     int n = child[u];
34     int root = centroid(u, 0, n); // bước 1
35
36     updateAns(root, n); // bước 2
37
38     del[root] = 1;
39     for (int v : adj[root]) if (!del[v])
40         ans += solve(v); // bước 3
41
42     return ans;
43 }
44
45 int main() {
46     cin >> n >> k;
47     for (int i = 1; i < n; ++i) {
48         int u, v; cin >> u >> v;
49         adj[u].push_back(v);
50         adj[v].push_back(u);
51     }
52
53     cout << solve(1);
54
55     return 0;
56 }

```

Mở rộng

Bước 2 là bước quan trọng của thuật toán, có thể kiểm tra xem thuật toán có áp dụng được vào bài toán hay không bằng cách kiểm tra xem bước thứ 2 có khả thi hay không. Bài toán ở bước 2 thường đơn giản hơn nhiều so với bài toán ban đầu.

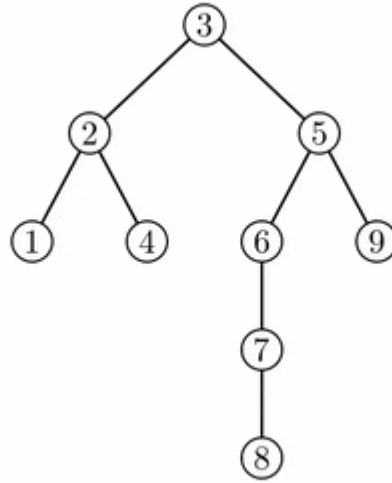
Bạn đọc có thể làm thử bài tập ví dụ tại [đây](#) 

Cây trọng tâm

Cây trọng tâm $T'(V, E')$ của cây $T(V, E)$ có thể được xây dựng như sau:

Thêm vào E' các cạnh nối trọng tâm của T với trọng tâm của mỗi cây mới tạo thành khi xóa trọng tâm của T ra khỏi T . Với mỗi cây mới tạo được từ việc xóa trọng tâm, thực hiện lại việc thêm cạnh và xóa đỉnh như vừa nêu, tiếp tục đến khi tất cả các đỉnh đều bị xóa.

Sau khi xóa tất cả đỉnh trong cây T , ta đã xây dựng được cây T' gọi là cây trọng tâm của cây T ban đầu.



Cây trọng tâm có một số tính chất đặc biệt như:

- ▶ Cây có số lượng đỉnh bằng với cây ban đầu.
- ▶ Độ cao của cây không vượt quá $\log(n)$.
- ▶ $LCA(u, v)$ trên cây trọng tâm cũng thuộc đường đi từ u đến v trên cây ban đầu.

Từ các tính chất đó, ta có một ứng dụng vô cùng quan trọng của cây trọng tâm nói riêng hay thuật toán phân tách trọng tâm nói chung. Ứng dụng như sau:

Giả sử cần tính hàm $f(u, v)$ rất nhiều lần, với $f(u, v)$ là một hàm liên quan đến đường đi u, v , nghĩa là ta có thể tính $f(u, v)$ từ $f(u, k)$ và $f(k, v)$ với k là một đỉnh thuộc đường đi u, v . Khi đó ta có thể tính trước tất cả các giá trị $f(u, p)$ với p là tổ tiên của u **trên cây trọng tâm**. Theo tính chất thứ 2 thì chỉ có $n \log(n)$ cặp (u, p) , có thể áp dụng thuật phân tách trọng tâm để tìm tất cả các giá trị $f(u, p)$ đó. Vậy với hai đỉnh u, v bất kì, ta có thể tính $f(u, v)$ từ $f(u, LCA_{ct}(u, v))$ và $f(v, LCA_{ct}(u, v))$, trong đó LCA_{ct} là tổ tiên chung gần nhất của u và v trên cây trọng tâm.

Nói đơn giản, vì độ cao của cây trọng tâm chỉ là $\log(n)$, vậy chỉ có tất cả $n \log(n)$ đường đi thẳng (đường đi từ một đỉnh đến một tổ tiên của đỉnh đó). Do đó, ta có thể tính trước tất cả giá trị của các đường đi có 2 đầu mút là 2 đầu của đường đi thẳng, từ đó tính giá trị của mọi đường đi trên cây bằng cách chia đường đi đó thành 2 đường đi mà mỗi đường đi có 2 đầu mút là 2 đầu của một đường đi thẳng trên cây trọng tâm.

Đồng thời, việc tìm tổ tiên chung gần nhất trên cây trọng tâm có độ phức tạp vô cùng nhỏ. Độ phức tạp của thuật tìm LCA là $\log(H)$ với H là độ cao của cây. Với cây trọng tâm, $H = \log(n)$, vậy độ phức tạp cho mỗi lần tìm LCA trên cây trọng tâm chỉ là $O(\log(\log(n)))$.

Áp dụng

Lampice - COCI 2019/2020 [↗](#)

Tóm tắt đề bài

Cho một cây có n đỉnh, mỗi đỉnh trên cây mang một kí tự. Tìm độ dài của đường đi dài nhất trên cây mà các kí tự trên đường đi đó tạo thành xâu đối xứng (gọi tắt là đường đi đối xứng).

Phân tích

Thoạt nhìn bài toán giống với các dạng bài dùng thuật phân tách trọng tâm (tìm đường đi thỏa điều kiện...), tuy nhiên bài toán ở bước 2 là "tìm đường đi đối xứng dài nhất chứa đỉnh gốc" vẫn quá khó để giải quyết.

Vì vậy cần áp dụng thêm thuật toán chặt nhị phân để làm bài toán đơn giản hơn nữa.

Thuật toán chặt nhị phân áp dụng được nhờ tính chất: nếu tồn tại đường đi đối xứng độ dài k , thì cũng tồn tại đường đi đối xứng với bất kì độ dài nào nhỏ hơn k và cùng tính chẵn lẻ với k (và vì vậy mà ta cần chặt nhị phân hai lần, một để tìm đường đi đối xứng chẵn dài nhất và một để tìm đường đi đối xứng lẻ dài nhất).

Giải thích đơn giản, nếu tồn tại đường đi đối xứng độ dài k , ta có thể bỏ đi hai đầu mút của đường đi để còn lại đường đi độ dài $k - 2$, và vẫn đối xứng.

Khi đã áp dụng chặt nhị phân, bài toán còn lại "liệu có tồn tại đường đi đối xứng độ dài k hay không?"

Bài toán này có thể giải quyết bằng thuật toán phân tách trọng tâm.

Ta cần giải quyết câu hỏi: "có tồn tại đường đi đối xứng độ dài k và chứa đỉnh gốc của cây hay không?".

Câu hỏi có phần tương tự như vấn đề cần giải quyết ở bài tập trước, tuy nhiên có thêm điều kiện "đối xứng".

Gọi $s(u, v)$ là xâu tạo bởi đường đi từ u đến v .

Ý tưởng như sau, với 2 đỉnh u, v mà đường đi từ u đến v có chứa đỉnh gốc và có độ dài là k , để kiểm tra xâu đối xứng, ta kiểm tra $s(u, v)$ có bằng $s(v, u)$ hay không, hay kiểm tra $s(u, root_u) + s(root, v) = s(v, root_v) + s(root, u)$ hay không? (gọi $root_u$ là con trực tiếp của $root$ mà là tổ tiên của u).

Áp dụng thuật toán *hash* để kiểm tra, ta cần tìm trước các giá trị $hshdown[u]$, $hshup[u]$ với ý nghĩa lần lượt là hash của $s(root, u)$ và hash của $s(u, root_u)$. Từ các giá trị $hshdown$, $hshup$ của u, v và các giá trị k, h , ta có thể áp dụng và biến đổi phương trình $s(u, root_u) + s(root, v) = s(v, root_v) + s(root, u)$ sao cho mỗi vế độc lập về u hoặc v , từ đó giải quyết tương tự như bài tập trước.

Cài đặt

Dưới đây là một code đã ac bài Lampice, bạn đọc có thể tham khảo.

```

1  #include <bits/stdc++.h>
2
3  #define For(i, a, b) for (int i=a;i<=b;++i)
4
5  using namespace std;
6
7  const int N = 200005;
8  const long long base = 35711;
9  const long long mod = 1e9 + 7;
10
11 int n, Len, maxDep, child[N], valid[N];
12 char a[N];
13
14 vector<pair<int, long long>> b;
15 long long pw[N];
16 vector<int> adj[N];

```

```

17 unordered_map<long long, bool> f[N];
18
19 void countChild(int u, int p)
20 {
21     child[u] = 1;
22     for (int v : adj[u]) if (v != p && valid[v])
23     {
24         countChild(v, u);
25         child[u] += child[v];
26     }
27 }
28
29 bool dfs(int u, int p, int h, long long hshdown, long long hshup)
30 {
31     if (h > Len) return false;
32
33     if (p)
34         hshdown = (hshdown * base + a[u]) % mod;
35     hshup = (hshup + 1LL * a[u] * pw[h - 1]) % mod;
36
37     long long x = (hshup * pw[Len - h] - hshdown + mod) % mod;
38     if (!p) f[h][x] = true;
39
40     if (f[Len - h + 1].find(x) != f[Len - h + 1].end() )
41         return true;
42
43     for (int v : adj[u]) if (v != p && valid[v])
44     {
45         if (!p) b.clear();
46
47         if (dfs(v, u, h + 1, hshdown, hshup))
48             return true;
49
50         if (!p)
51             for (pair<int, long long> x : b) f[x.first][x.second] = true;
52     }
53
54     maxDep = max(maxDep, h);
55     b.push_back({h, x});
56
57     return false;
58 }
59
60 bool CD(int u, int n)
61 {
62     countChild(u, 0);
63
64     int flag = 1, half = n / 2;
65     while (flag)
66     {
67         flag = 0;
68     }
69 }

```



```

68         for (int v : adj[u])
69             if (valid[v] && child[v] < child[u] && child[v] > half)
70                 {
71                     u = v;
72                     flag = 1;
73                     break;
74                 }
75     }
76
77     childCounting(u, 0);
78
79     if (dfs(u, 0, 1, 0, 0)) return true;
80
81     For(i, 1, maxDep) f[i].clear();
82     maxDep = 0;
83
84     valid[u] = false;
85     for (int v : adj[u]) if (valid[v])
86         if (CD(v, child[v])) return true;
87     return false;
88 }
89
90 bool check(int len)
91 {
92     Len = len;
93     For(i, 1, n) valid[i] = 1, f[i].clear();
94     return CD(1, n);
95 }
96
97 void solve()
98 {
99     cin >> n;
100     For(i, 1, n) cin >> a[i];
101     For(i, 1, n - 1)
102     {
103         int u, v; cin >> u >> v;
104         adj[u].push_back(v);
105         adj[v].push_back(u);
106     }
107
108     pw[0] = 1;
109     For(i, 1, n) pw[i] = pw[i - 1] * base % mod;
110
111     int l = 0, r = (n - 1) / 2;
112     while (l < r)
113     {
114         int g = (l + r + 1) / 2;
115         if (check(g * 2 + 1)) l = g; else r = g - 1;
116     }
117
118     int ans = r * 2 + 1;
119

```

```

120
121     l = 0, r = n / 2;
122     while (l < r)
123     {
124         int g = (l + r + 1) / 2;
125         if (check(g * 2)) l = g; else r = g - 1;
126     }
127
128     cout << max(ans, r * 2);
129 }
130
131 int main()
132 {
133     ios_base::sync_with_stdio(false);
134     cin.tie(NULL); cout.tie(NULL);
135
136     solve();
137
138     return 0;
}

```

QTREE5

Tóm tắt đề bài

Cho một cây có n đỉnh, ban đầu mỗi đỉnh đều có màu đen.

Thực hiện q truy vấn, mỗi truy vấn thuộc một trong hai loại sau đây:

- 0 u : đổi màu đỉnh u (nếu u đang có màu đen thì đổi thành trắng, nếu u có màu trắng thì đổi thành đen).
- 1 u : tìm khoảng cách từ đỉnh u đến đỉnh màu trắng gần nhất. Nếu không có đỉnh nào màu trắng, in -1 .

Phân tích

Xây dựng cây trọng tâm của cây được cho, gọi $LCA_{ct}(u, v)$ là tổ tiên chung gần nhất của u và v **trên cây trọng tâm**. Gọi $dist(u, v)$ là khoảng cách giữa đỉnh u và đỉnh v **trên cây ban đầu**.

Khoảng cách giữa 2 điểm u, v bất kì có thể phân tích như sau: $dist(u, v) = dist(u, LCA_{ct}(u, v)) + dist(LCA_{ct}(u, v), v)$. Bằng thuật toán phân tách trọng tâm, ta có thể tính trước mọi giá trị $dist(u, p)$ mà p là tổ tiên của u **trên cây trọng tâm**.

Với mỗi truy vấn, đỉnh u là cố định, ta cần xét qua các đỉnh v màu trắng để tìm $dist(u, v)$ nhỏ nhất. Thấy rằng, có thể có nhiều nhất đến n đỉnh v màu trắng, tuy nhiên chỉ có nhiều nhất $\log(n)$ giá trị $LCA_{ct}(u, v)$ khác nhau (tính chất về chiều cao của cây trọng tâm), vì vậy có thể xem xét việc "xử lý chung" cho các đỉnh v mà $LCA_{ct}(u, v)$ cố định.

Với mỗi đỉnh, ta cần 1 multiset để lưu tất cả các khoảng cách từ đỉnh đó đến một đỉnh **con** màu trắng của nó trên cây trọng tâm, tổng kích thước của các multiset không quá $n \log(n)$. Với mỗi truy vấn đổi màu đỉnh u , ta có thể duyệt qua tất cả các đỉnh tổ tiên của u trên cây trọng tâm để cập nhật các multiset cần thiết.

Về truy vấn tìm khoảng cách, ta cũng lại duyệt qua các đỉnh tổ tiên của u trên cây trọng tâm. Tại đỉnh p là tổ tiên của u , gọi s_p là khoảng cách nhỏ nhất từ đỉnh p đến một đỉnh màu trắng, ta có $dist(u, p) + s_p$ là độ dài

nhỏ nhất của một đường đi từ đỉnh u đến một đỉnh màu trắng và có đi qua p , đáp án cho truy vấn là giá trị $dist(u, p) + s_p$ nhỏ nhất khi xét tất cả các đỉnh p . Như vậy, bằng cách xét qua hết tất cả các đỉnh p là tổ tiên của u trên cây trọng tâm, ta đã bao quát tất cả các đường đi từ đỉnh u đến một đỉnh trắng nào đó.

Độ phức tạp của thuật toán là $n \log^2(n)$

Lưu ý, giá trị $dist(u, p) + s_p$ đề cập ở trên có thể là giá trị của một đường đi "không chuẩn" - đường đi đi qua một cạnh nhiều lần. Tuy nhiên đường đi này chắc chắn có độ dài lớn hơn đường đi tối ưu, vì vậy ta chỉ cần quan tâm rằng đường đi tối ưu có được xét qua hay không, nếu có, kết quả tìm được là chính xác.

Cài đặt

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  const int N = 2 * 1e5 + 5;
6  const int oo = 1e9 + 7;
7
8  int n, del[N], par[N], child[N];
9  vector<int> adj[N];
10 multiset<int> s[N];
11 map<int, int> d[N];
12
13 int countChild(int u, int p) {
14     child[u] = 1;
15     for (int v : adj[u]) {
16         if (v == p || del[v]) continue;
17         child[u] += countChild(v, u);
18     }
19     return child[u];
20 }
21
22 int centroid(int u, int p, int m) {
23     for (int v : adj[u]) {
24         if (v == p || del[v]) continue;
25         if (child[v] > m / 2)
26             return centroid(v, u, m);
27     }
28     return u;
29 }
30
31 void calcDist(int u, int p, int root) {
32     for (int v : adj[u]) {
33         if (v == p || del[v]) continue;
34         d[v][root] = d[u][root] + 1;
35         calcDist(v, u, root);
36     }
37 }
38
39 int cd(int u = 1) {

```

```

40     int m = countChild(u, 0);
41     u = centroid(u, 0, m);
42     calcDist(u, 0, u);
43     del[u] = 1;
44     for (int v : adj[u]) {
45         if (del[v]) continue;
46         int x = cd(v);
47         par[x] = u;
48     }
49     return u;
50 }
51
52 void solve()
53 {
54     int n; cin >> n;
55     for (int i = 1; i < n; ++i) {
56         int u, v; cin >> u >> v;
57         adj[u].push_back(v);
58         adj[v].push_back(u);
59     }
60     int root = cd();
61
62     int q; cin >> q;
63     vector<int> col(n + 5, 1);
64     while (q--) {
65         int t, u; cin >> t >> u;
66         if (t == 0) {
67             int p = u;
68             col[u] ^= 1;
69             if (col[u]) {
70                 while (p) {
71                     s[p].erase(s[p].lower_bound(d[u][p]));
72                     p = par[p];
73                 }
74             }
75             else {
76                 while (p) {
77                     s[p].insert(d[u][p]);
78                     p = par[p];
79                 }
80             }
81         }
82         else {
83             int ans = oo, p = u;
84             while (p) {
85                 if (s[p].size()) {
86                     ans = min(ans, d[u][p] + *s[p].begin());
87                 }
88                 p = par[p];
89             }
90             if (ans >= oo) cout << "-1\n";
91         }

```


```
92         else cout << ans << "\n";
93     }
94 }
95 }
96
97 int main()
98 {
99     ios_base::sync_with_stdio(false);
100    cin.tie(NULL);cout.tie(NULL);
101
102    solve();
103
104    return 0;
}
```

Luyện tập

[Fixed-Length Paths II - CSES](#) 

[Digit Tree - Codeforces](#) 

[Race - IOI2011](#) 

Bạn đọc cũng có thể tìm thêm bài tập về Centroid tại mục [tag](#)  trên trang oj.vnoi.info/ .

Được cung cấp bởi [Wiki.js](#)