

Các phương pháp giải bài toán LCA

Các phương pháp giải bài toán LCA

Tác giả: Khúc Anh Tuấn

Đôi lời về tác giả: Khúc Anh Tuấn được coi là huyền thoại của Competitive Programming Việt Nam với nhiều thành tích khủng:

- HCB IOI 2006
- Người Việt Nam duy nhất từng đạt được Đỏ Target trên Topcoder (max rating là hơn 3000, chỉ có khoảng chưa đến 100 người trên thế giới đạt được).
- Rank 2 Facebook Hackercup
- Người Việt Nam duy nhất từng lọt vào chung kết Google Code Jam.
- Giữ kỷ lục thi ACM ICPC World final của Việt Nam tính đến thời điểm 2011 (rank 17 toàn cầu).

Bài viết này được đưa lên thư viện VNOI cũ và được mình khôi phục lại sau nhiều năm thất truyền.

Trước khi đọc bài viết này, bạn cần đọc bài viết: [Bài toán RMQ và bài toán LCA](#) để nắm được những khái niệm cơ bản.

Để giải bài toán LCA ta có thể chuyển sang bài toán RMQ tương ứng và có thể giải bằng một số cách khác nhau. Trong bài viết này chúng ta sẽ đề cập tới một số phương pháp giải bài toán LCA một cách trực tiếp.

Bài toán LCA (Least Common Ancestor):

Input: 1 cây với n đỉnh.

Truy vấn: với 2 nút u, v bất kỳ của cây T , truy vấn $LCA(u, v)$ cho biết cha chung gần nhất của 2 đỉnh u, v trong cây T , tức là cho biết đỉnh xa gốc nhất là cha của cả u và v .

Cách 1 - Sparse Table

Từ cây đầu vào ta có thể xây dựng được mảng $F[1..n]$ với $F[i]$ cho ta biết nút cha của nút i . Sau đó ta có thể xây dựng mảng $A[1..n][0.. \log N]$ với $A[i][j]$ cho ta biết nút tổ tiên thứ 2^j của nút i . Xây dựng mảng A mất $O(N \log N)$ sử dụng phương pháp QHĐ. Gọi $d(i)$ là khoảng cách tới gốc của nút i . Để xác định $LCA(u, v)$ ta thực hiện các bước sau:

- Giả sử $d(u) > d(v)$, ta thay u bằng một nút tổ tiên của u đến khi $d(u) = d(v)$.

- ▶ Khi $d(u) = d(v)$ ta thay u và v bằng 2 nút tổ tiên tương ứng sao cho vẫn thỏa mãn $d(u) = d(v)$ đến khi $u = v$. Khi đó ta có được kết quả cần tìm.

Tất nhiên trong quá trình thay một nút bằng nút tổ tiên của nó, ta sẽ sử dụng mảng A để có thể nhảy một lần được nhiều bước. Khi đó độ phức tạp của thuật toán sẽ là $\langle O(N \log N), O(\log N) \rangle$.

Cách 2 - Euler Tour + Interval Tree

Từ cây đầu vào ta sử dụng thủ tục DFS để xây dựng 2 mảng:

- ▶ $\text{prevnum}[1..n]$ với $\text{prevnum}[i]$ cho ta biết thứ tự gọi thủ tục DFS cho đỉnh i .
- ▶ $\text{postnum}[1..n]$ với $\text{postnum}[i]$ cho ta biết thứ tự thoát khỏi thủ tục DFS cho đỉnh i .

Từ 2 mảng prevnum và postnum ta có thể thấy điều kiện cần và đủ để u là cha của v là $\text{prevnum}[u] \leq \text{prevnum}[v]$ và $\text{postnum}[u] \geq \text{postnum}[v]$. Do đó thao tác chất vấn $LCA(u, v)$ thực chất là tìm một đỉnh i sao cho :

- ▶ $\text{prevnum}[i] \leq \min(\text{prevnum}[u], \text{prevnum}[v])$
- ▶ $\text{postnum}[i] \geq \max(\text{postnum}[u], \text{postnum}[v])$
- ▶ $\text{prevnum}[i]$ lớn nhất có thể (hoặc $\text{postnum}[i]$ nhỏ nhất có thể).

2 điều kiện đầu đảm bảo i sẽ là cha chung của u và v , điều kiện thứ 3 đảm bảo i sẽ là đỉnh xa gốc nhất, tức $i = LCA(u, v)$.

Xây dựng mảng $A[1..n]$ với $A[i]$ cho ta biết $\text{postnum}[k]$ với k là đỉnh sao cho $\text{prevnum}[k] = i$. Ta hoàn toàn có thể xây dựng mảng A trong thời gian $O(N)$. Như vậy ta cần tìm trong mảng con $A[1..\min(\text{prevnum}[u], \text{prevnum}[v])]$ phần tử cuối cùng sao cho giá trị của nó không nhỏ hơn $\max(\text{postnum}[u], \text{postnum}[v])$. Ta có thể sử dụng cấu trúc dữ liệu Interval Tree để làm việc này, mỗi nút của cây Interval sẽ lưu giá trị lớn nhất của một đoạn và khi thực hiện thủ tục DFS trên cây Interval ta ưu tiên đi sang cây con bên phải. Khi biết được giá trị postnum (và cả prevnum) của đỉnh cần tìm rồi ta sẽ dễ dàng biết được đỉnh đó.

Độ phức tạp của thuật toán này cũng giống như thuật toán 1 với thời gian là $\langle O(N \log N), O(\log N) \rangle$ như chỉ mất $O(N)$ bộ nhớ.

Cách 3

Cũng tương tự cách 2 ta khởi tạo các mảng $\text{prevnum}[1..n]$ và $\text{postnum}[1..n]$. Mảng $A[1..n]$ với $A[i]$ cho ta biết đỉnh k sao cho $\text{prevnum}[k] = i$. Như vậy ta cần tìm $LCA(u, v)$ trong mảng con $A[1..\min(\text{prevnum}[u], \text{prevnum}[v])]$. Ta có thể sử dụng phương pháp chặt nhị phân kết hợp đệ quy để làm cận (khá tốt) như sau:

- ▶ Xét thủ tục `Find_LCA(left, right, u, v : Integer)` tìm cha chung gần nhất của u, v trong mảng con $A[\text{left}..\text{right}]$. Không mất tính tổng quát giả sử $\text{prevnum}[u] < \text{prevnum}[v]$.
 - ▶ Nếu $\text{postnum}[u] > \text{postnum}[v]$ thì $LCA(u, v) = u$ và đây là trường hợp dễ dàng tìm ra đáp án.
 - ▶ Nếu $\text{postnum}[u] < \text{postnum}[v]$, gọi $\text{mid} = (\text{left} + \text{right})/2$. Xét phần tử chính giữa đoạn $i = A[\text{mid}]$ sẽ có các khả năng sau:

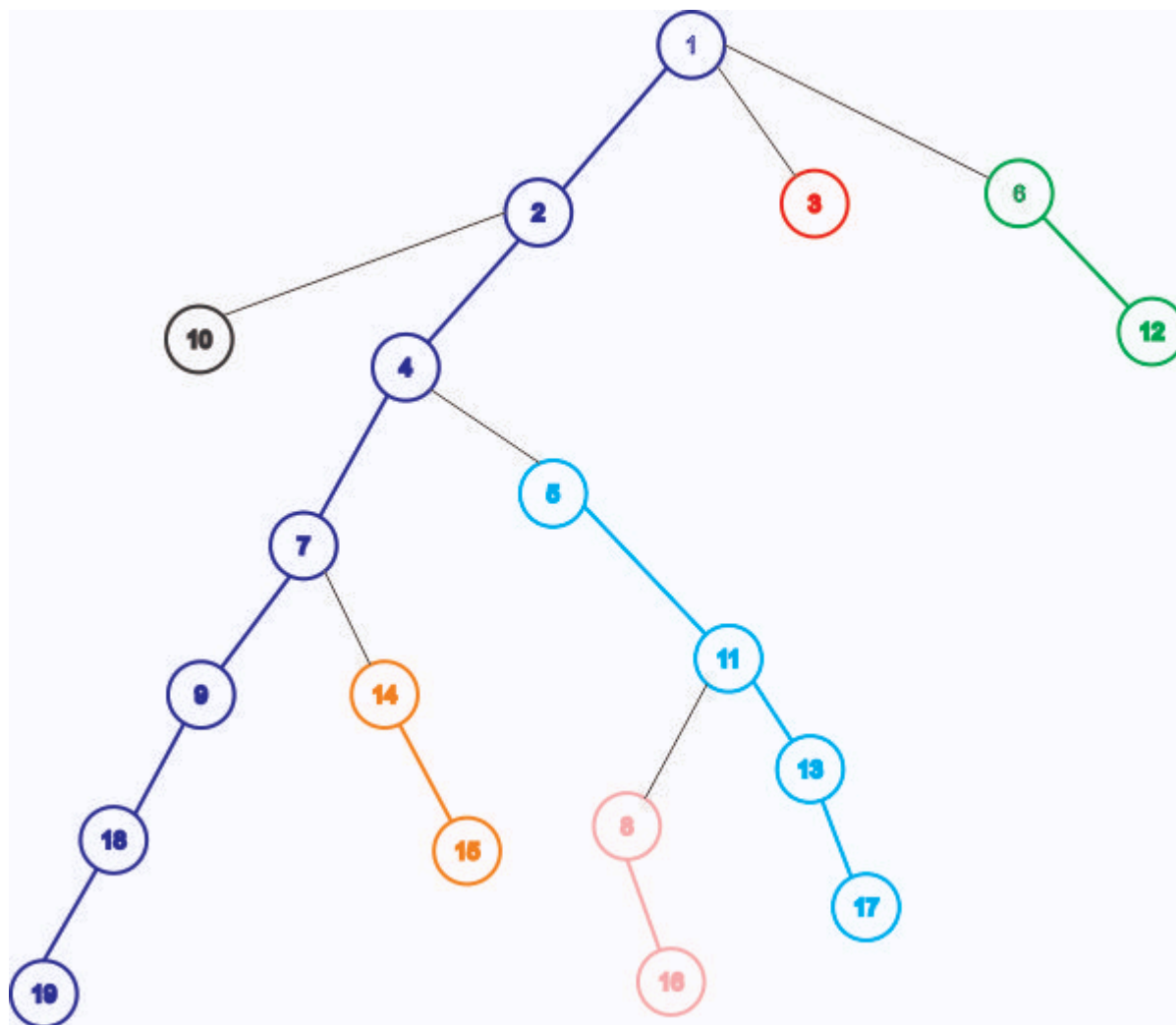
- $\text{postnum}[i] > \text{postnum}[v]$: i sẽ là cha chung của u và v nhưng chưa chắc đã là $LCA(u, v)$. Hiện nhiên $\text{prevnum}[i] \leq \text{prevnum}[LCA(u, v)]$ nên ta gọi đệ quy: `Find_LCA(mid, right, u, v)`.
- $\text{postnum}[v] > \text{postnum}[i] > \text{postnum}[u]$: i là cha của u nhưng không phải là cha của v . Vì vậy $LCA(u, v) = LCA(i, v)$, ta gọi đệ quy: `Find_LCA(left, mid, i, v)`.
- $\text{postnum}[i] < \text{postnum}[u]$: đỉnh i là đỉnh được rẽ nhánh ra từ một nút cha nào đó của u , nhưng ta hoàn toàn chưa biết nút cha này nằm dưới hay trên $LCA(u, v)$. Ta có thể xử lý theo 2 cách: gọi đệ quy `Find_LCA(left, right, cha(u), cha(v))` hoặc lấy $j = \text{Find_LCA(left, mid, i, u)}$ và j sẽ rơi vào 2 trường hợp đầu.

Thuật toán trên nếu chỉ thực hiện 2 trường hợp đầu thì độ phức tạp cho mỗi lần chất vấn là $\log N$, còn nếu chỉ thực hiện trường hợp 3 thì độ phức tạp sẽ là N . Qua khảo sát bằng việc chạy chương trình cho thấy thời gian thực hiện trung bình của thuật toán này ngang với các thuật toán với độ phức tạp $\langle O(N \log N), O(\log N) \rangle$. Thuật toán này tuy có độ phức tạp lớn nhưng lại là phương pháp tiết kiệm bộ nhớ và cài đặt dễ dàng nên đây là thuật toán có ứng dụng cao trong làm bài.

Cách 4 - Heavy Light Decomposition

Sử dụng [Heavy Light Decomposition](#).

Xuất phát từ trường hợp suy biến của cây: mỗi nút của cây chỉ có đúng 1 con (trừ 1 nút lá không có con). Với một cây suy biến ta hoàn toàn có thể tìm $LCA(u, v)$ trong thời gian $O(1)$ (đỉnh nào gần gốc hơn trong 2 đỉnh u, v sẽ là $LCA(u, v)$). Tư tưởng của Heavy Light Decomposition sẽ là chia cây ban đầu ra thành nhiều cây suy biến.



Những đoạn cùng màu là một cây suy biến. Nếu coi mỗi cây suy biến là một đỉnh thì ta sẽ được một cây mới gọi là cây rút gọn. Sau đây là một cách chia cây để cây rút gọn thu được có độ cao $O(\log N)$ với N là số nút của cây ban đầu:

- ▶ Xuất phát từ đỉnh gốc
- ▶ Với mỗi đỉnh nếu là lá thì nó sẽ là kết thúc của một cây suy biến
- ▶ Nếu không ta sẽ phát triển tiếp cây suy biến này xuống đỉnh con có trọng lượng lớn nhất, các đỉnh con khác sẽ là nút gốc của những cây suy biến mới. Trọng lượng của một nút được định nghĩa là số nút nhận nút đó là tổ tiên (hiểu một cách trực quan thì nếu coi mỗi nút có một "sức nặng" thì trọng lượng của một nút chính là "sức nặng" mà nút đó phải gánh).

Chứng minh:

- ▶ Gọi $F(n)$ là hàm cho ta chiều cao tối đa của một cây rút gọn có n đỉnh. Ta sẽ chứng minh $F(n) \leq \log N + 1$.
- ▶ Với $n = 1$ thì $F(1) = \log(1) + 1$.
- ▶ Giả sử điều cần chứng minh đã đúng đến $n - 1$.
- ▶ Với một cây có N đỉnh và nút gốc sẽ có các cây con với số đỉnh là x_1, \dots, x_k . Giả sử $x_1 = \max(x_1 \dots x_k)$. Ta có $2 * \max(x_2 \dots x_k) \leq \max(x_2 \dots x_k) + x_1 \leq N$
 $\implies \max(x_2 \dots x_k) \leq N/2$.

- ▶ Theo cách xây dựng cây thì :

$$F(N) = \max(F(x_1), F(x_2) + 1, F(x_3) + 1, \dots, F(x_k) + 1)$$

Mà:


- ▶ $F(x_1) \leq F(N - 1) \leq \log N + 1$
- ▶ $F(x_2) + 1 \leq F(N/2) + 1 \leq \log N + 1$
- ▶ ...
- ▶ $F(x_k) + 1 \leq F(N/2) + 1 \leq \log N + 1$

Vậy $F(N) \leq \log N + 1$ (Điều phải chứng minh).

Để thực hiện chất vấn $LCA(u, v)$ ta lần lượt nhảy từ u và v trở về gốc của cây. Từ một đỉnh ta thực hiện lần lượt một bước nhảy dài tới gốc của cây suy biến chứa nó và một bước nhảy ngắn tới nút cha (qua đó chuyển sang cây suy biến mới). Sau khi xác định được cây suy biến chứa $LCA(u, v)$, ta có thể xác định được đỉnh u_1, v_1 tương ứng là nút đầu tiên ta gặp khi nhảy từ u, v tới cây suy biến chứa $LCA(u, v)$. Sau đó chỉ cần sử dụng một phép so sánh xem u_1 hay v_1 gần gốc hơn là có thể xác định được $LCA(u, v)$.

Tuy về ý tưởng ta quan tâm nhiều đến việc chia cây ban đầu ra thành nhiều cây suy biến nhưng về mặt cài đặt, ta chỉ cần quan tâm với mỗi nút của cây đầu vào, nút gốc của cây suy biến chứa nó là nút nào. Dễ thấy khi thực hiện thủ tục DFS (có ưu tiên gọi đệ quy tới nút con có trọng lượng lớn nhất trước) các nút sẽ được liệt kê lần lượt theo từng cây suy biến. Vì vậy ta có thể khởi tạo mảng $Head[1..n]$ với $Head[i]$ cho ta biết nút gốc của cây suy biến chứa nút i chỉ với $O(N)$.

Thuật toán này sẽ chạy trong thời gian $\langle O(N), O(\log N) \rangle$.

Thuật toán này khá linh hoạt và có thể mở rộng ra để ứng dụng vào nhiều bài toán khác trên cây. Để ý rằng nếu cây ban đầu có trọng số ở mỗi cạnh, sau khi chia thành các cây suy biến thì cạnh của mỗi cây suy biến sẽ giống như các phần tử liên tiếp của một mảng. Do đó ta hoàn toàn có thể sử dụng các cấu trúc dữ liệu như Interval Tree để quản lý việc thay đổi hay chất vấn thông tin về các cạnh này. Đây chính là ý tưởng để làm bài [QTREE](#) .

Cách 5 - Xử lý offline



Đây là một phương pháp để giải bài toán LCA khi đã biết trước mọi câu hỏi chất vấn. Cách làm này tuy không linh hoạt nhưng thời gian chạy khá nhanh và tiết kiệm bộ nhớ. Tư tưởng của phương pháp này là trả lời các câu chất vấn theo một thứ tự khác dễ dàng hơn. Với mỗi nút của cây ta sẽ lưu nó trong một tập hợp có nhãn. Ban đầu mỗi nút thuộc một tập hợp khác nhau và nhãn của tập hợp chính là chỉ số của nút đó. Sau đó ta thực hiện thủ tục DFS, trước khi thoát ra khỏi thủ tục DFS ta thực hiện 2 thao tác sau :

- ▶ Tìm cha chung của u với các đỉnh v mà thủ tục $DFS(v)$ đã được thực thi. Đỉnh cha chung chính là nhãn của tập hợp chứa v .
- ▶ Hợp nhất tập hợp chứa u với tập hợp chứa $cha(u)$ và lấy nhãn là $cha(u)$.

Ta sẽ chứng minh "Đỉnh cha chung chính là nhãn của tập hợp chứa v ". Giả sử $i = LCA(u, v)$. Sau khi thực thi thủ tục $DFS(v)$ xong, từ v thủ tục DFS phải đi về i và rẽ xuống u để có thể thực hiện $DFS(u)$. Trong quá trình đi về i , nó sẽ hợp nhất v với cha v , ông v, \dots rồi với i . Do đó nhãn của tập chứa v chính là i .

Để thực hiện thao tác hợp nhất 2 tập hợp với thời gian ngắn, ta có thể sử dụng cấu trúc disjoint set giống như trong thuật toán Kruskal. Độ phức tạp của phương pháp này là $(M + N) \log N$ với M là số thao tác.

Các bài tập áp dụng:

- [VNOJ - LUBENICA](#) 
- [SPOJ - LCA](#) 
- [SPOJ - QTREE](#) 
- [VNOJ - QTREE3](#) 
- [VNOJ - VOTREE](#) 

Được cung cấp bởi [Wiki.js](#)