Các phương pháp giải bài toán LCA

This document was written by Khuc Anh Tuan-a member of VietNam's IOI team 2006

Như đã trình bày ở mục "Sự liên hệ giữa bài toán LCA và bài toán RMQ", 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 mục 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) :

Đầu vào : 1 cây với n đỉnh.

Chất vấn : với 2 nút u, v bất kỳ của cây T, chất 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, v.

Cách 1 :

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…logN] với A[i][j] cho ta biết nút tổ tiên thứ 2^j của nút i. Xây dựng mảng A mất NlogN 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 <NlogN,LogN>.

Cách 2 :

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

- prevnum[1…n] với prevnum[i] cho ta biết thứ tự gọi thủ tục DFS cho đỉnh i.
- postnum[1…n] với 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 prevnum[u]<=prevnum[v] & postnum[u]>=postnum[v]. Do đó thao tác chất vấn LCA(u,v) thực chất là tìm một đỉnh i sao cho :

- prevnum[i]<=Min(prevnum[u],prevnum[v])
- postnum[i]>=Max(postnum[u],postnum[v])
- prevnum[i] lớn nhất có thể (hoặc 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 postnum[k] với k là đỉnh sao cho 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(prevnum[u],prevnum[v])] phần tử cuối cùng sao cho giá trị của nó không nhỏ hơn Max(postnum[u],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 <NlogN,LogN> như chỉ mất O(N) bộ nhớ.

Cách 3 :

Cũng tương tự cách 2 ta khởi tạo các mảng prevnum[1…n] và postnum[1…n]. Mảng A[1…n] với A[i] cho ta biết đỉnh k sao cho prevnum[k] = i. Như vậy ta cần tìm LCA(u,v) trong mảng con A[1…Min(prevnum[u],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[left…right]. Không mất tính tổng quát giả sử prevnum[u]<prevnum[v], nếu postnum[u]>postnum[v] thì LCA(u,v)=u và đây là trường hợp dễ dàng tìm ra đáp án. Nếu postnum[u]<postnum[v], gọi mid = (left+right)/2. Xét phần tử chính giữa đoạn i = A[mid] sẽ có các khả năng sau :

- postnum[i]>postnum[v] : i sẽ là cha chung của u và v như chưa chắc đã là LCA(u,v). Hiển nhiên prevnum[i]<=prevnum[LCA(u,v)] nên ta gọi đệ quy : Find_LCA(mid, right, u, v).

- postnum[v]>postnum[i]>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).

- postnum[i]<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 = 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 LogN, 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 <NlogN,LogN>. 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 :

Sử dụng Dynamic Tree. 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 Dynamic Tree sẽ là chia cây ban đầu ra thành nhiều cây suy biến. Hình vẽ dưới đây mô tả trực quan hơn về ý tưởng này :

Những cạnh liền là những cạnh thuộc cây suy biến, còn các cạnh nét đứt sẽ là những cạnh nối các cây suy biến với nhau. 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(LogN) 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, còn 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)<=LogN+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 x1…xk. Giả sử x1 = Max(x1…xk). Ta có 2*Max(x2…xk) <= Max(x2…xk)+x1 <= N

- ð Max(x2…xk) <= N/2.

Theo cách xây dựng cây thì :

F(N) = Max(F(x1), F(x2)+1, F(x3)+1, …, F(xk)+1)

Mà :

- F(x1) <= F(N-1) <= LogN+1
- F(x2) +1 <= F(N/2)+1 <= LogN+1
- …
- F(xk) +1 <= F(N/2)+1 <= LogN+1

Vậy F(N) <= LogN+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 u1,v1 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 u1 hay v1 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 <N,LogN>.

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 http://www.spoj.pl/problems/QTREE trên spoj.

Cách 5 :

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,… 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 Kruscal. Độ phức tạp của phương pháp này là (M+N)LogN với M là số thao tác.

Sự liên hệ giữa bài toán LCA và bài toán RMQ

This document was written by Khuc Anh Tuan-a member of VietNam's IOI team 2006

(Dựa theo bài viết của Michael A.Bender và Martin Farach-Colton)

Bài toán LCA (Least Common Ancestor) :

Đầu vào : 1 cây với n đỉnh.

Chất vấn : với 2 nút u, v bất kỳ của cây T, chất 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, v.

Bài toán RMQ (Range Minimum Query) :

Đầu vào : 1 mảng A với n số.

Chất vấn : với 2 chỉ số i và j, chất vấn RMQ(i,j) cho biết chỉ số của phần tử nhỏ nhất trong mảng con A[i…j].

Các bài toán chất vấn nếu cần thời gian chuẩn bị là *f(n)* và thời gian trả lời mỗi chất vấn là *g(n)* thì độ phức tạp của thuật toán sẽ là <*f(n),g(n)*>.

Bổ đề : Nếu có 1 thuật toán với độ phức tạp là <*f(n),g(n)*> cho bài toán LCA thì sẽ có 1 thuật toán với độ phức tạp là <*f(n)+n,g(n)+1*> cho bài toán RMQ.

Chứng minh :

Giả sử ta có mảng A[1…n] là mảng đầu vào của bài toán RMQ. Ta sẽ xây dựng 1 cây theo cách sau : gốc của cây sẽ tương ứng với phần tử nhỏ nhất của mảng A; sau khi loại bỏ phần tử này đi, mảng A sẽ chia thành 2 phần; con trái và con phải của gốc được xây dựng tương tự từ 2 mảng con bên trái và bên phải mảng ban đầu.

Để xây dựng được cây này ta mất thời gian là O(N). Gọi Ti là cây tương ứng với mảng A[1…i], để xây dựng cây Ti+1 ta để ý rằng nút i+1 sẽ thuộc đường đi xuất phát từ gốc và luôn đi sang bên phải (gọi là đường Pr). Vì thế ta sẽ đi theo đường này từ dưới lên trên cho đến khi gặp vị trí thích hợp để chèn nút i+1 vào. Các nút bên dưới sẽ trở thành con trái của nút i+1. Để ý rằng mỗi lần kiểm tra xem 1 vị trí có thích hợp để chèn nút mới vào không, ta sẽ thêm vào hoặc loại bỏ 1 nút từ đường Pr, và 1 nút chỉ có thể được thêm vào và loại bỏ khỏi đường Pr nhiều nhất 1 lần. Vì thế ta xây dựng cây sẽ mất O(N).

Sau khi xây dựng cây, gọi k = LCA(i,j) với i,j bất kỳ. Như thế k sẽ là nút đầu tiên chia cắt i,j. Theo cách xây dựng cây thì k chính là phần tử nhỏ nhất của mảng con A[i…j]. Vì thế RMQ(i,j) = LCA(i,j).

Như thế để tính RMQ(i,j) ta mất O(N) ở bước chuẩn bị để xây dựng cây và O(1) để gọi thủ tục LCA(i,j) trên cây. Bổ đề được chứng minh.

Bổ đề : Nếu có 1 thuật toán với độ phức tạp là <*f(n),g(n)*> cho bài toán RMQ thì sẽ có 1 thuật toán với độ phức tạp là <*f(n)+n,g(n)+1*> cho bài toán LCA.

Chứng minh :

Giả sử ta có 1 cây T là cây đầu vào của bài toán LCA. Dùng thủ tục DFS cho cây này và viết ra các đỉnh của cây mỗi khi đỉnh đó được đi qua. Khi đó ta sẽ nhận được 1 dãy 2*n-1 số bởi vì chúng ta bắt đầu liệt kê từ đỉnh gốc, với mỗi cạnh ta sẽ liệt kê được 2 đỉnh (ở lần đi và lần về). Có n-1 cạnh nên số đỉnh sẽ là 2*(n-1)+1.

Gọi mảng E[1…2n-1] là mảng ghi danh sách các đỉnh được thăm bằng cách DFS trên.

Khởi tạo mảng L[1…2n-1], L[i] cho ta biết khoảng cách từ nút E[i] tới gốc của cây.

Khởi tạo mảng R[1…n], R[i] cho ta biết vị trí xuất hiện của số i trong mảng E. Nếu số i xuất hiện nhiều lần thì ta lấy vị trí nào cũng được.

Dựa vào nhận xét : LCA của 2 nút u và v chính là nút có khoảng cách tới gốc nhỏ nhất trong số các nút xuất hiện từ lúc ta thăm tới u cho đến khi ta thăm tới v.

Thực hiện thao tác i = RMQ(R[u], R[v]) trên mảng L, khi đó i sẽ là chỉ số của phần tử nhỏ nhất trong khoảng L[R[u]…R[v]], E[i] cho ta kết của của phép chất vấn LCA(u,v).

Như thế để tính LCA(u,v) ta mất O(n) để khởi tạo các mảng E,L,R. và mất O(1) để gọi thủ tục RMQ(R[u],R[v]). Bổ đề được chứng minh.

Nhận xét : bài toán RMQ phát sinh khi giải bài toán LCA chỉ là 1 trường hợp đặc biệt của bài toán RMQ tổng quát bởi các phần tử liên tiếp nhau trong mảng hơn kém nhau đúng 1 đơn vị. (Do 2 phần tử liên tiếp là khoảng cách tới gốc của 2 nút có quan hệ cha con với nhau). Bài toán này gọi là ±1RMQ.
Để giải bài toán RMQ ta có thể sử dụng cấu trúc dữ liệu Interval Tree. Khi đó thuật toán sẽ có độ phức tạp là <*NlogN,LogN*> và sử dụng O(N) bộ nhớ. Một thuật toán khác nhanh hơn với độ phức tạp là <*NlogN,1*> sẽ được trình bày dưới đây:
Gọi F[i,j] = RMQ(i,i+2^j-1). Ta có thể khởi tạo mảng F trong thời gian NlogN bằng cách sử dụng QHĐ. Khi đó RMQ(i,j) = F[i,t] nếu A[F[i,t]] < A[F[j-2^t+1,t]] và RMQ(i,j) = F[j-2^t+1,t] nếu ngược lại với t = Trunc(Log(j-i+1)).
Cũng với phương pháp trên và một số cải tiến ta có thể giải bài toán ±1RMQ trong <*N,1*> như sau :
Giải sử A là mảng đầu vào của bài toán ±1RMQ. Chia mảng A thành các block liên tiếp với độ dài là logN/2. Định nghĩa mảng A'[1…2*N/LogN] với A'[i] cho ta biết giá trị nhỏ nhất của block thứ i và B[1…2*N/LogN] với B[i] là vị trí mà giá trị A'[i] xuất hiện trong block đó. Sử dụng thuật toán trên với mảng A' ta sẽ có thuật toán với độ phức tạp là <*N,1*>. Để thực hiện chất vấn RMQ(i,j) trên mảng A, nếu i,j thuộc các block khác nhau thì ta sẽ làm như sau :

- Tìm min của đoạn từ i đến phần cuối của block chứa i
- Tìm min của các block nằm giữa block chứa i và block chứa j
- Tìm min của đoạn từ đầu block chứa j đến j

Thao tác thứ 2 hoàn toàn có thể thực hiện trong <*N,1*> dựa vào mảng A' và B. Vì vậy ta chỉ cần quan tâm đến trường hợp i,j nằm trên cùng 1 block.
Để ý 1 block có 2 đặc trưng : 1 là giá trị của phần tử đầu tiên, 2 là dãy nhị phân ứng với mỗi phần tử của block cho ta biết nó lớn hơn hay nhỏ hơn phần tử trước. Nếu 2 block có cùng đặc trưng 2 thì ta có thể sử dụng kết quả chất vấn của block này để xác định kết quả chất vấn của block kia. Số đặc trưng 2 chỉ vào khoảng 2^(logN/2-1) = O(sqrt(N)) là rất nhỏ, vì thế ta có thể xác định trước các đặc trưng 2 cũng như kết qủa của tất cả các câu chất vấn chỉ mất O(sqrt(N)*LogN*LogN). Với N đủ lớn ta có thể coi sqrt(N)*Log(N)*Log(N) < N (nếu muốn ta có thể sử dụng các block với độ dài là LogN/3, LogN/4 để đạt được bất đẳng thức này). Để xác định đặc trưng 2 của các block trong A cũng chỉ mất O(N) khi khởi tạo. Vì thế bài toán ±1RMQ có thể giải quyết trong <*N,1*> với bộ nhớ là O(N).
Với thuật toán ±1RMQ <*N,1*> ta có thể đưa ra được thuật toán LCA <*N,1*> và thuật toán RMQ tổng quát cũng với thời gian <*N,1*>.