International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)
Volume 2, Issue 4, April 2015, PP 1-6
ISSN 2349-4840 (Print) & ISSN 2349-4859 (Online)
www.arcjournals.org

Algorithms for Computing LCA in Complete Binary Trees

WANG Jue
Guangdong Neptune High-tech Co. Lt.,
Foshan City, Guangdong Province,
PRC, 528000
88010735@qq.com

LUO Qirong
Department of Mechatronics,
Foshan University, Foshan City,
Guangdong Province, PRC, 528000
1183992513@qq.com

Abstract: *Based on properties of the lowest common ancestor (LCA) of two nodes in a complete binary tree, the paper designs three algorithms as well as their C-language codes to compute the LCAs. The designed algorithms contain both an elementary version that is simple, easily-understood and easily-realized and a high-efficient version that can be applied for developments of embedded systems and SoC. The paper is of a reference to development of embedded systems and related area.*

Keywords: *lowest common ancestor, Algorithm, Embed System, Complete Binary tree.*

1. Introduction

Problem of seeking a lowest common ancestor (LCA) of nodes in a tree was first raised by Alfred Aho and John Hopcroft in 1973. As a basic and common problem in both graph theory and computer science, the problem has been paid attention to because the solution of the problem is helpful to other related problems, as introduced in [1] and [2]. Recent years, complete binary trees that are widely applied in bioinformatics [3], rapid location of data in industrial control [4-6] bring new values for developing algorithms of the problem. The bibliography [7] presents an algorithm for computing an LCA of two neighboring nodes in a complete binary tree, however algorithm for computing that in an arbitrary a complete binary has not been seen.

A recent bibliography [8] has systematically presented properties of LCAs in a complete binary, but the paper does not present an algorithm for the related computations. Therefore, we present related algorithms for it. This paper designs algorithms and *C* language implementation of computing LCA of nodes in a complete binary tree. The paper also makes an analysis of time-complexity for the algorithms.

2. Preliminaries

This section first presents preliminaries for later sections.

2.1 Basic Definitions and Symbols

Symbol $\lfloor x \rfloor$ denotes a floor function of a real number such that satisfies $\lfloor x \rfloor \le x < \lfloor x \rfloor + 1$. Symbol $\{x\}$ is the decimal function such that $x = \lfloor x \rfloor + \{x\}$ and symbol $\lceil x \rceil$ is the ceil function that holds $x \le \lceil x \rceil < x + 1$. Symbol $N_{(k,j)}$ is to express the node at the jth position on the *k*-th level in a binary tree, symbol $G_{(k,i)}^{(l,j)}$ is to express LCA if $N_{(k,i)}$ and $N_{(l,j)}$. The whole paper suggests the maximal depth of the binary tree is *h*.

2.2 Lemmas

Lemma 1 ([9]) For arbitrary real $\delta > 0$, integer $J = \lfloor \log_2 \delta \rfloor + 1$ is the minimal j that fits the equation $\left\lfloor \frac{\delta}{2^j} \right\rfloor = 0$.

Lemma 2([9]) For arbitrary real α, $\delta > 0$ and positive integer j, the condition $\left\lfloor \frac{\delta}{2^j} \right\rfloor = 0$ is necessary for the equation $\left\lfloor \frac{\alpha+\delta}{2^j} \right\rfloor = \left\lfloor \frac{\alpha}{2^j} \right\rfloor$. If a positive integer *J* is the minimal *j* that holds $\left\lfloor \frac{\delta}{2^j} \right\rfloor = 0$, then $\left\lfloor \frac{\alpha+\delta}{2^J} \right\rfloor - \left\lfloor \frac{\alpha}{2^J} \right\rfloor = \begin{cases} 0 \\ 1 \end{cases}$.

Lemma 3([10]) For arbitrary positive integer δ and non-negative *I*, the inequality $(\chi - \frac{1}{2}) \times 2^I < \delta \le (\chi + \frac{1}{2}) \times 2^I$ has a unique solution $\chi = \left\lceil \frac{\delta}{2^I} - \frac{1}{2} \right\rceil$.

Lemma 4 ([8]) Let $N_{(k,\alpha)}$ and $N_{(k,\beta)}$ $(1 < k \le h; 2^{k-1} \le \alpha < \beta < 2^k)$ be two nodes on the *k*-th level in a complete binary tree; if *I* is the smallest *i* that fits the equation $\left\lfloor \frac{\alpha}{2^i} \right\rfloor = \left\lfloor \frac{\beta}{2^i} \right\rfloor$ and $\sigma = \left\lfloor \frac{\alpha}{2^I} \right\rfloor$; the $N_{(k-I,\sigma)}$ is the LCA of $N_{(k,\alpha)}$ and $N_{(k,\beta)}$.

Lemma 5([8]) Let $N_{(k\ \alpha}$ and $N_{(k,\beta)}$ $(1 < k \le h; 2^{k-1} \le \alpha < \beta < 2^k)$ be two nodes on the *k*-th level in a complete binary tree; if *I* is the smallest *i* that fits $0 \le \alpha \bmod 2^i < 2^{i-1}$, integers α and χ are such that satisfy $\sigma = \left\lfloor \frac{\alpha}{2^I} \right\rfloor$ and $(\chi - \frac{1}{2}) \times 2^I < \beta - \alpha \le (\chi + \frac{1}{2}) \times 2^I$, then $G_{(k,\alpha)}^{(k,\beta)} = G_{(k-I,\sigma)}^{(k-I,\sigma+\chi)}$.

Lemma 6([8]) Let $N_{(k,\alpha)}$ and $N_{(k,\beta)}$ $(1 < k \le h; 2^{k-1} \le \alpha < \beta < 2^k)$ be two nodes on the *k*-th level in a complete binary tree; then the two share a direct ancestor if and only if there exists an integer σ such that $i = \left\lfloor \frac{j}{2^\sigma} \right\rfloor, k = l - \sigma$ .This time $N_{(k-1,\lfloor i/2 \rfloor)}$ is the LCA of $N_{(k,i)}$ and $N_{(l,j)}$.

Lemma 7([8]) For valid number (k, j) , if *I* is the smallest *i* that fits $0 \le \alpha \bmod 2^i < 2^{i-1}$ and $\sigma = \left\lfloor \frac{j}{2^I} \right\rfloor$, then $N_{(k-I,\sigma)}$ is the LCA of $N_{(k,j)}$ and $N_{(k,j+1)}$.

3. Algorithm Design and Analysis

This section design concrete algorithms according to previous lemmas. The algorithms can be applied for arbitrary nodes $N_{(m,\alpha)}$ and $N_{(n,\beta)}$ in a complete binary tree, where α and β are sequential indices and $\alpha < \beta$.

3.1 Algorithm Design

3.1.1 Algorthm I Based on Lemma 4

The algorithm is by 4 steps as follows:

```
i.  Compute m, n by $\alpha$ and $\beta$ respectively, $m = \lfloor \log_2 \alpha \rfloor + 1$, $n = \lfloor \log_2 \beta \rfloor + 1$;
ii. Compute $\sigma = |n - m|, \gamma = \left\lfloor \frac{\beta}{2^\sigma} \right\rfloor$;
iii. Compute I by the following loop
    I=1; $\sigma_1 = \left\lfloor \frac{\alpha}{2^I} \right\rfloor, \sigma_2 = \left\lfloor \frac{\gamma}{2^I} \right\rfloor$;
    While ($\sigma_1 \neq \sigma_2$)
        Begin
        I=I+1, $\sigma_1 = \left\lfloor \frac{\alpha}{2^I} \right\rfloor, \sigma_2 = \left\lfloor \frac{\gamma}{2^I} \right\rfloor$;
        End
iv. Compute LCA= $N_{(m-I,\sigma_1)}$.
```

The C++ language for the algorithm I is as follows

```
int FindLCA(int a, int b)
{ int t = 1, temp1 = a, temp2 = b, n, m, I;
   m= (int)(log10(i)/log10(2)); /*compute m*/
   n= (int)(log10(j)/log10(2));  /*compute n*/
   I = n-m; t <<= I;
   temp2 = (int)floor(j/t);
   for (int r = 1;;r++)  {
   temp1 >>= 1;temp2 >>= 1;
   if ( temp1==temp2)return temp2; }
   }
```

3.1.2 Algorithm II Based on Lemma 5 and 6

The algorithm is by 6 steps as follows:

i. Compute *m*, *n* by α and β respectively; $m = \lfloor \log_2 \alpha \rfloor + 1$, $n = \lfloor \log_2 \beta \rfloor + 1$

ii. Compute $\sigma = n - m, \gamma = \left\lfloor \frac{\beta}{2^{\sigma}} \right\rfloor$;

iii. Compute $I = \min(i)\{0 \le \alpha \bmod 2^i < 2^{i-1}\}$;

iv. Compute $\delta = |\gamma - \alpha|$, $\sigma = \left\lfloor \frac{\alpha}{2^I} \right\rfloor$;

v. Compute non-negative integer χ: $\chi = \left\lceil \frac{\delta}{2^I} - \frac{1}{2} \right\rceil$;

vi. If $\chi = 0$, then $N_{(m-I,\sigma)}$ is the LCA, or if $\alpha = \sigma$, then $\gamma = \sigma + \chi$ and goto step iii.

The C++ language for the algorithm II is as follows：

```
int FindLCA(int a, int b)
{  int I, _2i=1,n, m, r,delta, rm;
   int x=b-a;
   while(x!=0)
   { a=(int)floor(a/((double)_2i)); r=a+x;
      m= (int)(log10(a)/log10(2));
       n= (int)(log10(r)/log10(2));
       I = n-m;   _2i=1; _2i<<=I;
      r=(int)floor(r/((double)_2i));
       if (r==a) return a/2;
        else if (r<a) Swap(&a,&r);
       _2i = 1;
    for(int i=1;;i++)  {
    _2i <<= 1; rm=a&(_2i - 1);
   if(rm<_2i/2)  {I=i; break;} }
   delta = abs(r-a);   _2i=1;    _2i<<=I;
```

```
        x=(int)ceil(delta/((double)_2i)-0.5);}
        return (int)floor(a/((double)_2i));}
```

3.1.3 Algorithm III Based on Lemma 1, 2, 6 and 7

The algorithm is by 6 steps as follows:

i. Compute *m*, *n* by α and β respectively, $m=\lfloor \log_2 \alpha \rfloor+1$, $n=\lfloor \log_2 \beta \rfloor+1$;

ii. Compute $\sigma = n-m, \gamma = \left\lfloor \frac{\beta}{2^{\sigma}} \right\rfloor$;

iii. Compute $\delta = |\gamma - \alpha|$;

iv. Compute $I_1 = \lfloor \log_2 \delta \rfloor + 1$, $\sigma_1 = \left\lfloor \frac{\alpha}{2^{I_1}} \right\rfloor, \sigma_2 = \left\lfloor \frac{\gamma}{2^{I_1}} \right\rfloor$;

v. If $\sigma_1 = \sigma_2$, then $N_{(m-I_1,\sigma_1)}$ is what we need; Or if $\alpha \Leftarrow \left\lfloor \frac{\alpha}{2^{I_1}} \right\rfloor$, then goto step vi;

vi. Compute $I_2 = \min(i)\{0 \le \alpha \bmod 2^i < 2^{i-1}\}$, Compute $\sigma = \left\lfloor \frac{\alpha}{2^{I_2}} \right\rfloor$; then $N_{(m-I_1-I_2,\sigma)}$ is LCA

The C++ language for the algorithm III is as follows：

```
int FindLCA(int a, int b)
{ int I,I1,I2, _2i=1,n, m, r,rm;
    int sigama1,sigama2,sigama,delta;
   m= (int)(log10(a)/log10(2));
   n= (int)(log10(b)/log10(2));
  I = n-m;    _2i=1;  _2i<<=I;
  r=(int)floor(b/((double)_2i));
    if (r==a) return a/2; else if (r<a) Swap(&a,&r);
    delta=r-a;
    I1=(int)floor(log10(delta))/log10(2))+1;
  _2i=1;   _2i<<=I1;
sigama1=(int)floor(a/((double)_2i));
sigama2=(int)floor(r/((double)_2i));
 if(sigama1==sigama2) return sigama2;
 a=sigama1;    _2i = 1;
 for(int i=1;;i++){
  _2i <<= 1; rm=a&(_2i - 1);
    if(rm<_2i/2) {I2=i; break;} }
sigama= (int)floor(a/((double)_2i));
return sigama;}
```

3.2 Analysis of Algorithms

The step *iii* in Algorithm I is a loop that wastes most time. By Lemma 4, the number of loops depends on the number of level *m*. Hence it at most needs m loops. Consequently, the time complexity of Algorithm I is $T_1(m) = m = O(m)$.

Algorithm II is a bi-looped one. Its inner loop is to search the minimal I that fits $0 \le \alpha \bmod 2^i < 2^{i-1}$. By Lemma 1, this needs $\lfloor \log_2 \alpha \rfloor + 1$ computations. The other loop is from step *iii* to step *vi* with initial condition $\chi = \delta$ and terminal condition $\chi = 0$. In the loop, it requires to divide χ by 2^I, which needs at most $\lfloor \log_2 \delta \rfloor + 1$ computations. Therefore, the whole time needed is

$$T_2(\alpha, \delta) = (\lfloor \log_2 \delta \rfloor + 1)(\lfloor \log_2 \alpha \rfloor + 1)$$

It is easy to prove the worst case is that $\alpha = \delta$ and this time it holds

$$\max(T_2(\alpha, \delta)) = (\lfloor \log_2 \alpha \rfloor + 1)^2 \le (\log_2 \alpha + 1)^2 = O(\ln^2 \alpha)$$

Particularly, when $\delta = 1$,

$$\max(T_2(\alpha, 1)) = \lfloor \log_2 \alpha \rfloor + 1 = O(\ln \alpha)$$

Since in a complete binary tree, the level of node α lying is $\lfloor \log_2 \alpha \rfloor + 1$, hence we have

$$\max(T_2(m)) = m^2$$

The algorithm III contain a loop in step *vi*, which takes at most $\left\lfloor \log_2 \left\lfloor \frac{\alpha}{2^{I_1}} \right\rfloor \right\rfloor + 1$ computation. Hence the total time needed is $T_3(\alpha) = \left\lfloor \log_2 \left\lfloor \frac{\alpha}{2^{I_1}} \right\rfloor \right\rfloor + 1 \le \log_2 \left\lfloor \frac{\alpha}{2^{I_1}} \right\rfloor + 1 \le \log_2 \frac{\alpha}{2^{I_1}} + 1 = \log_2 \alpha - \lfloor \log_2 \delta \rfloor$.

Namely

$$T_3(\alpha) \le \log_2 \alpha = O(\ln \alpha)$$

4. Conclusion and Future Word

Seen from the time complexities, each of the previous designed three algorithms has its own specialty. From point of view of a programmer, the algorithm I is easy to program because it is simple. However, it is a little rough because we can only know its time complexity to its level while the other two can reach to a node.

It is obviously the algorithm II is the lowest efficiency seen from the time complexity. However it is strictly derived from properties of the floor function and it is worth of more digging out. For example, if the step *vi* is improved by other means, it might be a better one. It remains us a problem whether there is some other better approach to develop.

The algorithm III, we think it the best one because it is a most industrial one. This is because it contains two industrial traits: one is a better time complexity, and the other is its reconstructability.

In fact, the core of the algorithm III is its step *vi*, namely, computation of $I_2 = \min(i)\ \{0 \le \alpha \bmod 2^i < 2^{i-1}\}$. Note that, nodes in a complete binary tree can be previously coded by the way from top to bottom and from left to right. Hence each node α must have a minimal I that fits $0 \le \alpha \bmod 2^i < 2^{i-1}$. If we preprocess and record these Is, it only takes $O(\ln \alpha)$ time. After that preprocess, computation of LCA is turned to be an $O(1)$ inquiry process, which is quite suit for developing embedded system.

Acknowledgements

The research work is supported by the national Ministry of science and technology under project 2013GA780052, Department of Guangdong Science and Technology under projects 2012B011300068, Foshan Bureau of Science and Technology under projects 2013AG10007, Special Innovative Projects from Guangdong Education Department, and Chancheng government under projects 2013B1018 and 2013A1021. The authors sincerely present thanks to them all.

References

[1]. Wikipedia. Lowest Common Ancestor, http://en.wikipedia.org/wiki/ Lowest_common_ ancestor, Dec., 2014

[2]. Czumaj A, Kowaluk M, Lingas A. Faster algorithms for finding lowest common ancestors in directed acyclic graphs[J]. Theoretical Computer Science, 380(1-2), 37(2007)

[3]. Navaro G. A guided tour to approximate string matching. ACM Computing Surveys, 33(1),31(2001)

[4]. WANG Xingbo. Fast algorithms educed from intrinsic properties of node indices of binary trees. Computer Engineering and Applications (In Chinese), 47 (9), 16(2011)
[5]. WANG Xingbo, Study on non-recursive and stack-free algorithms for preorder traversal of complete binary trees, Computer Engineering and design (In Chinese), 32(9), 3077(2011).
[6]. WANG Xingbo, Fast algorithms for traversal of binary trees available for SoC. Computer Engineering and design (In Chinese), 34(3), 873(2012)
[7]. Wang Jue, Fast Algorithm for Finding the Lowest Common Ancestor of Two Neighboring Nodes in a Complete Binary Tree, Journal of Foshan University (Natural Science Edition) (In Chinese), 31(6), 12(2013).
[8]. WANG Xingbo, Properties of the Lowest Common Ancestor in a Complete Binary Tree[J], International Journal of Scientific and Innovative Mathematical Research,3(3),12(2015)
[9]. WANG Xingbo, Some Supplemental Properties with Appendix Application of Floor Function, Journal of Science of Teacher's College and University (In Chinese), 34(3), 7(2014)
[10]. WANG Xingbo, A Mean-value Formula for the Floor Functions on Integers[J], Mathproblems, 2012,2(4),136(2012)

AUTHORS' BIOGRAPHY

WANG Jue was born in 1988 in Hubei, China. He got his Bachelor degree at Chongqing College of Science and Technology, and his Master degree at Huazhong University of Science and Technology. He has been a technician in charge of developing GPS vehicle traveling data recorder since 2010 in Guangdong Neptune High-tech Co. Lt..

LUO Qirong was born in 1990 in Guangdong, China. He got his Bachelor degree at Foshan University. He is now a postgraduate student in the University, studying CNC technologies.