Computing Lowest Common Ancestors in Directed Acyclic Graphs[1]

Yijie Han

School of Computing and Engineering
University of Missouri at Kansas City
5100 Rockhill Road
Kansas City, MO 64110
hanyij@umkc.edu

Abstract

We show that the lowest common ancestors (LCA) in directed acyclic graphs (DAGs) can be computed in $O(n^{2.575})$ time. Previous best result computes this in $O(n^{2.688})$ time.

Keywords: Algorithms, time complexity, lowest common ancestor(LCA), directed acyclic graph(DAGs), shortest path.

1 Introduction

Finding the lowest common ancestor of a given pair of nodes is a fundamental algorithmic problem. In this paper we study the lowest common ancestor(LCA) problem on directed acyclic graphs (DAGs). A lowest common ancestor of two nodes a and b is a node c which is a common ancestor of a and b and no other node is both a common ancestor of a and b and a proper descendant of c. LCA on trees have been studied extensively. LCA on DAGs has also been investigated by several researchers. Many problems that require finding LCA cannot be solved using tree-LCA algorithms because the structure of DAGs are quite different from that of trees. Nykänen and Ukkonen [6] give a linear-time preprocessing, constant-time-query algorithm for the LCA in arbitrarily directed trees. They ask whether it is possible to preprocess a DAG in $o(n^3)$ time to support $\Theta(k)$-time set-LCA queries, where a set-LCA query returns all k lowest common ancestors of the given pair. Ait-Kaci et al. [2] considered the problem of LCA on lattices and lower semi-lattices (where a node pair has a unique LCA). Bender et al. gave an algorithm [3] for all-pairs-representative LCA in DAGs in $O(n^{2.688})$ time. In this paper we show that all-pairs-representative LCA in DAGs can be computed in $O(n^{2.575})$ time. This time complexity coincides with the current time complexity for computing all-pairs shortest paths for directed unweighted graphs [7].

[1]Research supported in part by NSF grant 0310245.

2 Computing LCA in DAGs

Definition 1: Let $G = (V, E)$ be a DAG, and let $x, y \in V$. Let $G_{x,y}$ be the subgraph of G induced by the set of all common ancestors of x and y. Define $\text{SLCA}(x, y)$ to be the set of outdegree 0 nodes (leafs) in $G_{x,y}$. The lowest common ancestors of x and y are the elements of $\text{SLCA}(x, y)$.

The transitive closure of $G_{tr} = (V, E_{tr})$ of a DAG $G = (V, E)$ is a graph such that $(i, j) \in E_{tr}$ iff there is a path from i to j in G. It is well known that transitive closure can be computed in the same time as matrix multiplication[1]. The current fastest matrix multiplication algorithm runs in $O(n^{2.376})$ time [4]. Therefore transitive closure can also be computed in $O(n^{2.376})$ time.

A source of a DAG is a node of the DAG with indegree 0. A sink of a DAG is a node of the DAG with outdegree 0.

Definition 2: The depth of a node x in a DAG, $\text{depth}(x)$, is the length of the longest path from a source to x.

We answer LCA queries by returning a representative element from $\text{SLCA}(x, y)$. Here we want to return a representative with the greatest depth.

To compute LCA we first obtain G' which is obtained by reversing every edge in G. We compute transitive closure in G and G', call then G_{tr} and G'_{tr}. We now compute the depth of every node in G, this takes at most $O(n^2)$ time. We then sort the nodes by their depth, breaking ties arbitrarily. We now group every consecutive n^t nodes in the sorted list into one group and obtain n^{1-t} groups, where t is a parameter to be fixed later on. Nodes in smaller numbered groups have depth no larger than nodes in greater numbered groups. For each pair of nodes a and b, we first decide the greatest numbered group which contains a common ancestor for a and b, we then check each node in this group to find out whether it is a common ancestor of a and b by using the transitive closure relationship.

To determine whether group g contains a common ancestor for every pair of nodes we use the transitive closure we computed earlier. Let group g contain nodes $x_1, x_2, ..., x_{n^t}$. Use G'_{tr} we construct graph G'_1. G'_1 is a bipartite graph (U, g, E), where $|U| = n$, $|g| = n^t$ and there is an edge (u, x_i) if there is an edge (u, x_i) for any u and $1 \leq i \leq n^t$ in G'_{tr}. We also construct G_1 using G_{tr}.

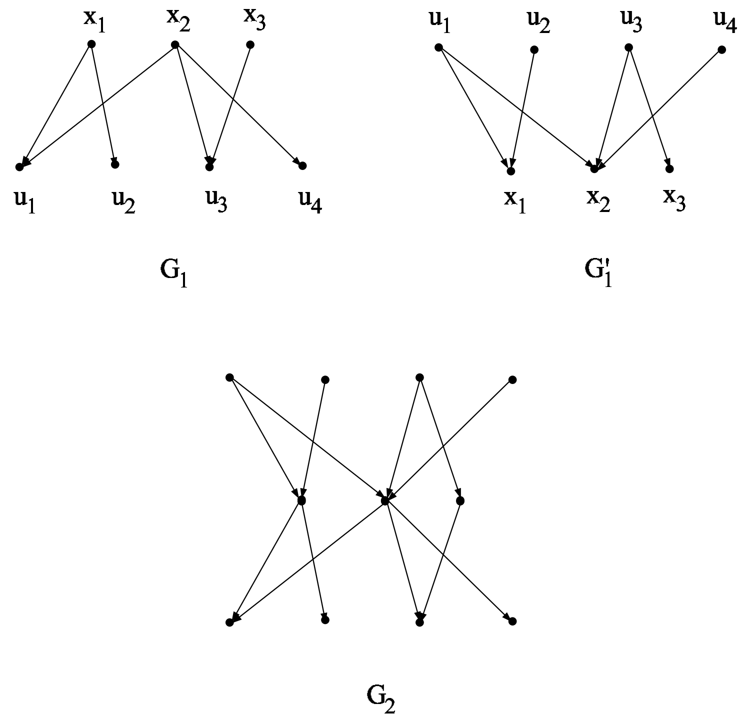

Figure 1:

G_1 is a bipartite graph (g, U, E), where $|g| = n^t$, $|U| = n$ and there is an edge (x_i, u) if there is an edges of (x_i, u) for any u and $1 \leq i \leq n^t$ in G_{tr}. We now combine G_1' and G_1 into one graph G_2 by identifying x_i in G_1' with x_i in G_1. This is shown in Fig. 1. Now we compute the transitive closure in G_2 which requires only a rectangular matrix multiplication of multiplying an $n \times n^t$ matrix with an $n^t \times n$ matrix. This can be done in $O(n^{\omega(1,t,1)})$ time, where $\omega(1,t,1)$ is the exponent of the time for multiplying such two matrices. Huang and Pan have shown [5] that:

Lemma 3[5]:

$$\omega(1,t,1) = \begin{cases} 2 + \frac{\omega-1}{1-\alpha}(t-\alpha) & t > \alpha = 0.294 \\ 2 & t \leq \alpha \end{cases}$$

where $\omega = \omega(1,1,1)$.

The transitive closure in G_2 will tell us for every pair of nodes, whether there is a node in group g which is a common ancestor of the pair. After we do this for every group we can find out, for every pair of nodes a and b, the greatest numbered group $g_{a,b}$ which contains a node which is a common ancestor of a and b. Because we compute rectangular matrix multiplication for every group the exponent of the time complexity is $1 - t + \omega(1,t,1)$.

After we determined $g_{a,b}$ for nodes a and b we then walk through all nodes in $g_{a,b}$ and find the node in $g_{a,b}$ which has the greatest depth and is a common ancestor of a and b. This is done using the transitive closure G_{tr} we computed earlier. This computation involves $O(n^t)$ time for each pair of node or $O(n^{2+t})$ time total.

By balancing the exponents $2 + t$ and $1 - t + \omega(1,t,1)$ we have

$$2 + t = 1 - t + \omega(1,t,1) = 1 - t + 2 + \frac{0.376}{0.706}(t - 0.294)$$

Solving this equation we obtain $t = 0.575$. Therefore the time complexity of our algorithm is $O(n^{2.575})$

Theorem 4: The all-pairs-representative LCA of a DAG can be computed in $O(n^{2.575})$ time. □

References

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman. The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, MA, 1974.

[2] H. Ait-Kaci, R. Boyer, P. Lincoln, R. Nasr. Efficient implementation of lattice operations. *ACM Trans. Program. Lang. Syst.* 11(1), 115-146(1989).

[3] M. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, P. Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms* 57, 75-94(2005).

[4] D. Coppersmith, S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.* **9**, 251-280.

[5] X. Huang, V. Y. Pan. Fast rectangular matrix multiplications and applications. *J. Complexity* 14, 257-299(1995).

[6] M. Nykänen, E. Ukkonen. Finding lowest common ancestors in arbitrarily directed trees. *Inform. Process. Lett.* 50(6), 307-310(1994).

[7] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. of ACM*, Vol. 49, No. 3, 289-317(2002).