LCA Queries in Directed Acyclic Graphs

Miroslaw Kowaluk[1,⋆] and Andrzej Lingas[2,⋆⋆]

[1] Institute of Informatics, Warsaw University, Warsaw
kowaluk@mimum.edu.pl
[2] Department of Computer Science, Lund University, 22100 Lund
Fax +46 46 13 10 21
Andrzej.Lingas@cs.lth.se

Abstract. We present two methods for finding a lowest common ancestor (LCA) for each pair of vertices of a directed acyclic graph (dag) on n vertices and m edges.

The first method is surprisingly natural and solves the all-pairs LCA problem for the input dag on n vertices and m edges in time $O(nm)$. As a corollary, we obtain an $O(n^2)$-time algorithm for finding genealogical distances considerably improving the previously known $O(n^{2.575})$ time-bound for this problem.

The second method relies on a novel reduction of the all-pairs LCA problem to the problem of finding maximum witnesses for Boolean matrix product. We solve the latter problem and hence also the all-pairs LCA problem in time $O(n^{2+\frac{1}{4-\omega}})$, where $\omega = 2.376$ is the exponent of the fastest known matrix multiplication algorithm. This improves the previously known $O(n^{\frac{w+3}{2}})$ time-bound for the general all-pairs LCA problem in dags.

1 Introduction

The problem of finding a *lowest common ancestor* (LCA) in a tree, or more generally, in a *directed acyclic graph* (dag) is one of the basic algorithmic problems. An LCA of vertices u and v in a dag is an ancestor of both u and v which has no descendant that is an ancestor of u and v, see Fig. 1 for example. We consider the problem of preprocessing a dag such that LCA queries can be answered quickly for any pair of vertices. It has a variety of important applications, e.g., in object inheritance in programming languages, analysis of genealogical data and lattice operations for complex systems (see [2] for details and further references).

For trees, linear-time preprocessing is sufficient to answer LCA queries in constant time [7]. For general dags, after an $O(n^{\frac{w+3}{2}})$-time preprocessing, LCA queries can be answered in constant time [2] (where n is the number of vertices and $\omega = 2.376$ is the exponent of the fastest known matrix multiplication algo-

⋆ Research supported by KBN grant 4T11C04425.
⋆⋆ Research supported in part by VR grant 621-2002-4049.

L. Caires et al. (Eds.): ICALP 2005, LNCS 3580, pp. 241–248, 2005.

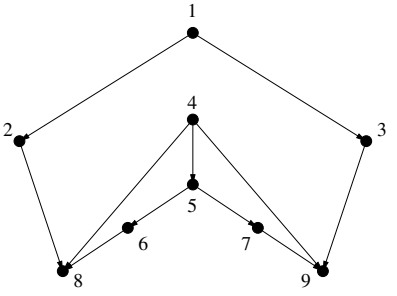

Fig. 1. The LCA of 8 and 9 are 1 and 5

rithm). A lower bound $\Omega(n^w)$ by reduction of the transitive closure problem to all-pairs LCA in dags is also given in [2].

We present two methods of efficiently preprocessing a directed graph on n vertices and m edges in order to answer an LCA query for any pair of vertices in constant time, subsuming the previously known best results from [2].

The first method is surprisingly natural and solves the all-pairs LCA problem for the input dag on n vertices and m edges in time $O(nm)$. For sparse dags, this method is optimal and substantially faster than the known $O(n^{\frac{w+3}{2}})$-time general method from [2]. As a corollary, we obtain an $O(n^2)$-time algorithm for finding genealogical distances considerably improving the previously known $O(n^{2.575})$ time-bound for this problem [2].

The second method efficiently reduces the all-pairs LCA problem to the problem of finding maximum (index) witnesses for Boolean matrix product. We solve the latter problem and hence also the all-pairs LCA problem in time $O(n^{2+\frac{1}{4-\omega}})$. Since $2+\frac{1}{4-\omega} \approx 2.616$ and $\frac{w+3}{2} \approx 2.688$, our result subsumes the previously known $O(n^{\frac{w+3}{2}})$ time-bound for the general all-pairs LCA problem in dags [2].

The first and second methods are respectively described in Sections 2 and 3 whereas Section 4 presents the algorithm for finding genealogical distances. Our paper concludes with final remarks.

2 Optimal Method for Sparse Dags

First, we shall describe preprocessing for answering queries about existence of a common ancestor for arbitrary pair of vertices in constant time.

For the input dag, we shall denote by n and m its number of vertices and edges, respectively. Also for a vertex v in the dag, $indeg(v)$ and $outdeg(v)$ stand respectively for the in-degree and out-degree of v. If $outdeg(v) = 0$ then v is called a *terminal vertex* and if $indeg(v) = 0$ then v is called a *source vertex*.

We may assume without loss of generality that the input dag is connected since otherwise we can decompose it into connected components and solve the

problem for each component separately. For technical reasons, we shall also assume that every vertex is its own ancestor.

The following lemma immediately follows from the definition of a dag.

Lemma 1. *If two vertices have a common ancestor then there is a source vertex that is their common ancestor.*

In the first stage of the preprocessing, for each vertex of the input dag we form a table containing its descendants. In other words, we create the transitive closure of the dag which obviously can be done in time $O(nm)$. For the sake of Section 4, we describe this stage in more details below.

We initialize the tables in time $O(n^2)$ and start from the terminal vertices, filling their tables with single vertices in time $O(n)$. Next we iterate the following step: remove the vertices of out-degree 0 with incident edges and fill the tables for the new vertices v of out-degree 0 by merging the information from the tables associated with the removed direct descendants of v, and taking into account the set of direct descendants of v. We also add v to its table. For each vertex v such an operation takes time $O(n) \times outdeg(v)$. Thus, for the whole graph it takes $O(nm)$ time.

Lemma 2. *The tables of descendants for all vertices can be formed in time* $O(nm)$.

In the second stage of the preprocessing, we determine for each vertex v the set of vertices which have a common ancestor with v. We proceed similarly as in the first stage of preprocessing starting from source vertices instead of the terminal ones. For the source vertices s, the sets are already computed, they are just the sets of descendants of s. Next, we iterate the following step: remove the vertices of in-degree 0 with incident edges and fill the tables for the new vertices v of in-degree 0 by merging the information from the tables associated with the removed direct ancestors of v. For each vertex v such an operation takes time $O(n) \times indeg(v)$. Thus, for the whole graph it takes $O(nm)$ time.

By the *height* of a vertex v in a dag, we shall mean the length of the longest path from a source vertex to v in the dag.

Note that the set of vertices having a common ancestor with a vertex v is the union of the sets of vertices having common ancestors with the ancestors of v (recall that v is also an ancestor of itself). Hence, we obtain the following lemma by induction on the height of v.

Lemma 3. *For all vertices* v*, the tables of vertices having a common ancestor with* v *can be computed in time* $O(nm)$.

In order to answer LCA queries we need to refine the preprocessing slightly. During the second descending phase of the preprocessing we additionally enumerate the vertices in their visiting order. Since an ancestor is always visited before its descendant, we obtain the following lemma.

Lemma 4. *A vertex of a higher number cannot be an ancestor of a vertex of a lower number.*

For all vertices v, in the table keeping vertices w having a common ancestor with v, we keep also the maximum of the numbers assigned to the common ancestors of v and w. To achieve this, when we merge the information from the tables of direct ancestors of v, we pick the maximum number of a common ancestor of a direct ancestor of v and w. Clearly, the refinement can be accomplished within the same asymptotic time $O(mn)$. By induction, we obtain the following lemma.

Lemma 5. *For all vertices v, the tables of vertices w having a common ancestor with v with a pointer to a lowest common ancestor of v and w can be computed in time $O(nm)$.*

Hence, we obtain immediately the following theorem.

Theorem 1. *A dag on n vertices and m edges can be preprocessed for constant-time LCA queries in time $O(nm)$.*

If $m = O(n)$ then the preprocessing is optimal.

Corollary 1. *The all-pairs LCA problem for a dag on n vertices and m edges can be solved in time $O(n(n+m))$.*

3 $O(n^{2+\frac{1}{4-\omega}})$-Time Method for General Dags

If an entry $C[i,j]$ of the Boolean product of two Boolean matrices A and B is equal to 1 then any index k such that $A[i,k]$ and $B[k,j]$ are equal to 1 is a *witness* for $C[i,j]$. If k is the largest possible witness for $C[i,j]$ then it is called the *maximum witness* for $C[i,j]$.

In [3], Galil and Margalit presented an $O(n^{\omega+\epsilon})$-time method for the problem of computing witnesses for all positive entries of the Boolean product of two $n \times n$ Boolean matrices. Their method (too involved to describe shortly) can be viewed as a sequence of algorithms for a generalization of the problem. The first algorithm corresponds to the straightforward cubic method testing all the n witness possibilities for each positive entry of the product. The consecutive algorithms partition the input into blocks. Next, they use the fast algorithm for Boolean matrix product to compute the product of the blocks pairwise, and use the resulting products to partition the problem into subproblems. In the subproblems, for a row of the first input matrix and a column of the second input matrix, only an unique index fragment induced by the block partition and containing a witness is considered. The subproblems are solved recursively by permutting rows and columns and using the previous algorithms from the sequence.

Only the first two algorithms in the sequence of algorithms constructed by their recursive method do not rely on row and column permutations. Therefore, the method does not seem adaptable to produce the maximum witnesses without altering its asymptotic time.

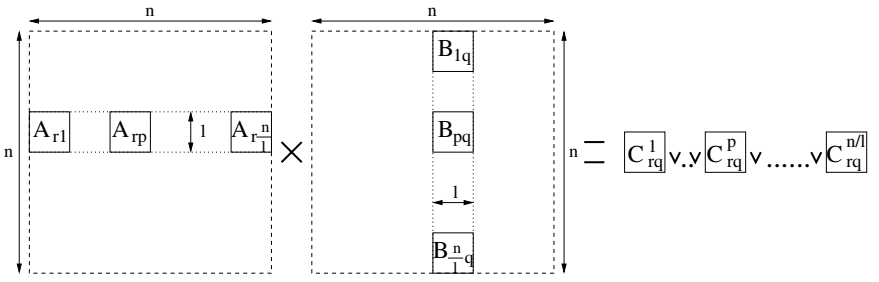

Fig. 2. The relationship between $A'_{rp}s$, $B'_{pq}s$ and $C'_{rq}s$

Our method for maximum witnesses of the Boolean product C of two $n \times n$ Boolean matrices A and B can be viewed as a modification of the second of the algorithms for witnesses of C in the aforementioned sequence of algorithms from [3].

Let l be a positive integer smaller than n. Partition the matrices A and B into $l \times l$ sub-matrices A_{rq}, B_{rq}, where $1 \leq r, q \leq n/l$, such that for $1 \leq r \leq n/l$, the sub-matrices A_{rq}, $1 \leq q \leq n/l$, cover the rows $(r-1)l+1$ through rl of A whereas for $1 \leq q \leq n/l$, the sub-matrices B_{rq}, $1 \leq r \leq n/l$, cover the columns $(q-1)l+1$ through ql of B.

For $1 \leq r, q \leq n/l$, $p = 1, ..., n/l$, compute the Boolean product C^p_{rq} of A_{rp} and B_{pq} using the fast algorithm. The following remark is straightforward.

Remark. Suppose that the (i, j) entry of the product matrix C is positive and $(r-1)l < i \leq rl$ and $(q-1)l < j \leq ql$. Let p' be the maximum value of p such that the entry of C^p_{rq} which is the dot product of the row of A_{rp} corresponding to the i-th row of A and the column of B_{pq} corresponding to the j-th column of B is 1. The maximum witness of the (i, j) entry of the Boolean product of A and B belongs to the interval $[(p'-1)l+1,\ p'l]$.

By this remark, after computing all the products C^p_{rq}, $1 \leq p, r, q \leq n/l$, we need $O(l)$ time per positive entry of C to find the maximum witness. Thus, the total time taken by our method for maximum witnesses is $O((\frac{n}{l})^3 l^\omega + n^2 l)$.

By solving the equation $(\frac{n}{l})^3 l^\omega = n^2 l$, we conclude that for $l = n^{\frac{1}{4-\omega}}$ our method achieves minimum worst-case time complexity at $O(n^{2+\frac{1}{4-\omega}})$. Hence, we obtain the following theorem.

Theorem 2. *The maximum witnesses for all positive entries of the Boolean product of two $n \times n$ Boolean matrices can be computed in time $O(n^{2+\frac{1}{4-\omega}})$.*

The following obvious lemma leads to an efficient reduction of the problem of all pairs LCA in a dag to that of determining maximum witnesses of the Boolean product of two Boolean matrices.

Lemma 6. *Let G be a dag and let G^* be its transitive closure. For vertices u, v in G, let w be its common ancestor of highest rank among all common ancestors*

of u and v in the ordering resulting from a topological sort of G^*. The vertex w is a lowest common ancestor of u and v.

Our algorithm for all pairs LCA in a dag is as follows.

Algorithm 1

1. Compute the transitive closure of the input dag G.
2. Topologically sort the vertices of G and number them by their ranks in the resulting sorting order.
3. Form two Boolean $n \times n$ matrices A and B such that for $i,\ k \in \{1, ..., n\}$ the k-th coordinate of the i-th row of A and the i-th column of B is set to 1 if the k-th vertex is an ancestor of the i-th vertex, or $k = i$, otherwise these two coordinates are set to 0.
4. Find maximum witnesses for the Boolean product C of A and B and for each non-zero entry $C[i, j]$ output the vertex whose number is the index of maximum witness of $C[i, j]$ as the lowest common ancestor of the i-th and j-th vertices.

The correctness of the algorithm follows from Lemma 6. Step 1 can be implemented in time $O(n^\omega)$. Steps 2 and 3 take $O(n^2)$ time. Finally, Step 4 requires $O(n^{2+\frac{1}{4-\omega}})$ time by Theorem 2. Hence, we obtain our second main result.

Theorem 3. *For a dag on n vertices, we can determine for each pair of vertices having a common ancestor their lowest common ancestor in time $O(n^{2+\frac{1}{4-\omega}})$.*

4 Shortest Genealogical Distances

The authors of [2] discuss the so called *pedigree graphs* which are sparse dags used to model human ancestor relations. Since each human has at most two parents, a pedigree graph has maximum in-degree bounded by two. For the fundamental applications of pedigree graphs in the identification of genes associated with genetic diseases the reader is referred to [4, 6]. In these applications, computing the so called *shortest ancestral distance* between a pair of vertices in a pedigree graph is important [2]. The shortest ancestral distance between two vertices u and v in a dag is defined as the length of a shortest path between u and v which passes through a common ancestor of u and v (observe that the common ancestor is not necessarily the lowest one [1]). Bender et al. showed that the all-pairs shortest ancestral distances can be computed in time $O(n^{2.575})$ [2]. In this section, we show that the all-pairs shortest ancestral distances can be optimally computed for sparse dags, in particular, pedigree graphs.

[1] One can also consider the so called shortest ancestral lca distance where the common ancestor is required to be lowest [2].

We can modify our first method to obtain an $O(mn)$-time algorithm to compute the all-pairs shortest ancestral distances as follows. In the ascending phase, for each vertex v, and for each descendent u of v, we additionally compute the shortest directed distance between u and v. This can be easily accomplished within the same asymptotic time $O(mn)$. At the beginning of the descending phase, the previously computed shortest directed distances yield the shortest ancestral distances between sources and their descendents. While descending the shortest ancestral distances between the parents of the current vertex v and each other vertex u are increased by one. Next, the minimum of them and the shortest directed distance between v and u (it can be infinite) is taken as the shortest ancestral distance between v and u. In this way for all pairs of vertices v and u the shortest ancestral distance is computed.

Similarly, the so modified descending phase can be also implemented in time $O(mn)$. We conclude with the following theorem.

Theorem 4. *For a dag on n vertices and m edges, the all-pairs shortest ancestral distances can be computed in time $O(nm)$.*

Corollary 2. *For a pedigree graph on n vertices, the all-pairs shortest ancestral distances can be computed in time $O(n^2)$.*

5 Final Remarks

The problems of finding LCA are classical and central in the area of algorithms and data structures [2, 5, 7]. In spite of the long history of studies devoted to LCA problems, we have succeeded to design two quite natural methods for finding LCA in dags considerably subsuming the previously known best results [2].

The problem of finding maximum witnesses of Boolean matrix product seems to be of interest in its own rights. At first glance it seems that the recursive $O(n^{\omega+\epsilon})$-time method of Galil and Margalit [3] could be adapted to produce the maximum witnesses by considering the fragments containing maximum witnesses in the subproblems without substantially altering its asymptotic time. However, the aforementioned method may permute rows or columns in recursive steps which may disturb the search for maximum witnesses. Thus, the problem of whether or not our $O(n^{2+\frac{1}{4-\omega}})$-time method is optimal is open.

It is also an interesting question whether or not the instances of the problem of finding maximum witnesses of Boolean matrix product occurring in our reduction from the LCA problem in dags are computationally easier than the general ones.

Acknowledgments

The authors are grateful to Pavel Sumazin for inspiration and to Leszek Gąsieniec for some discussions.

References

1. N. Alon and M. Naor. Derandomization, Witnesses for Boolean Matrix Multiplication and Construction of Perfect hash functions. Algorithmica 16, pp. 434-449, 1996.
2. M.A. Bender, G. Pemmasani, S. Skiena and P. Sumazin. Finding Least Common Ancestors in Directed Acyclic Graphs. Proc. the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 845-853, 2001.
3. Z. Galil and O. Margalit. Witnesses for Boolean Matrix Multiplication and Shortest Paths. Journal of Complexity, pp. 417-426, 1993.
4. R.W. Cottingham Jr., R.M. Idury, and A.A. Shäffer. Genetic linkage computations. American Journal of Human Genetics, 53, pp. 252-263, 1993.
5. M. Nykänen and E. Ukkonen. Finding lowest common ancestors in arbitrarily directed trees. Inf. Process. Lett., 50(6), pp. 307-310, 1994.
6. A.A. Shäffer, S.K. Gupta, K. Shriram, and R.W. Cottingham Jr. Avoiding recomputation in linkage analysis. Human Heredity, 44, pp. 225-237, 1994.
7. R.E. Tarjan. Applications of path compression on balanced trees. Journal of the ACM 26(4), pp. 690-715, 1979.