Efficiency of a Good But Not Linear Set Union Algorithm

ROBERT ENDRE TARJAN

University of California, Berkeley, California

ABSTRACT. Two types of instructions for manipulating a family of disjoint sets which partition a universe of n elements are considered. $FIND(x)$ computes the name of the (unique) set containing element x. $UNION(A, B, C)$ combines sets A and B into a new set named C. A known algorithm for implementing sequences of these instructions is examined. It is shown that, if $t(m, n)$ is the maximum time required by a sequence of $m \geq n$ *FIND*s and $n - 1$ intermixed *UNION*s, then $k_1 m\alpha(m, n) \leq t(m, n) \leq k_2 m\alpha(m, n)$ for some positive constants k_1 and k_2, where $\alpha(m, n)$ is related to a functional inverse of Ackermann's function and is *very* slow-growing.

KEY WORDS AND PHRASES. algorithm, complexity, equivalence, partition, set union, tree

CR CATEGORIES: 4.12, 5.25, 5.32

Introduction

Suppose we want to use two types of instructions for manipulating disjoint sets. $FIND(x)$ computes the name of the unique set containing element x. $UNION(A, B, C)$ combines sets A and B into a new set named C. Initially we are given n elements, each in a singleton set. We then wish to carry out a sequence of $m \geq n$ *FIND*s and $n - 1$ intermixed *UNION*s.

An algorithm for solving this problem is useful in many contexts, including handling EQUIVALENCE and COMMON statements in FORTRAN [3, 6], finding minimum spanning trees [9], computing dominators in directed graphs [14], checking flow graphs for reducibility [13], calculating depths in trees [2], computing least common ancestors in trees [2], and solving an offline minimum problem [7].

Several algorithms have been developed [3, 5–7, 10, 12], notably a very complicated one by Hopcroft and Ullman [7]. It is an extension of an idea by Stearns and Rosenkrantz [12] and has an $O(m \log^* n)$ worst-case running time, where

$$\log^* n = \min \{i \mid \overbrace{\log \log \cdots \log}^{i \text{ times}} (n) \leq 1\}.$$

All other known algorithms are slower, except for the very simple one we analyze here, which has been previously considered in [5, 7, 11].

Each set is represented as a tree.[1] Each vertex in the tree represents an element in the

This work was partially supported by the NSF, Contract No. NSF-GJ-35604X, and by a Miller Research Fellowship at the University of California, Berkeley, and by the Office of Naval Research, Contract NR 044-402, Stanford University.

Author's address: Department of Electrical Engineering and Computer Sciences, Computer Science Division, University of California, Berkeley, CA 94720

[1] For the purposes of this paper, a *tree* T is a directed graph with a unique vertex s, the *root* of T, such that (i) no edge (s, v) exists in T, (ii) if $v \neq s$, there is a unique edge (v, w) in T, and (iii) there are no cycles in T. If (v, w) is an edge of T (denoted by $v \rightarrow w$), w is called the *father* of v (denoted by $w = f(v)$) and v is called a *son* of w. If there is a path from v to w in T (denoted by $v \overset{*}{\rightarrow} w$), then

set, and the root of the tree represents the entire set as well as some element. Each tree vertex is represented in a computer by a cell containing two items: the element corresponding to the vertex, and either the name of the set (if the vertex is the root of the tree) or a pointer to the father of the vertex in the tree. Initially, each singleton set is represented by a tree with one vertex. The basic notion of representing the sets by trees was presented by Galler and Fischer [6].

To carry out *FIND*(x), we locate the cell containing x; then we follow pointers to the root of the corresponding tree to get the name of the set. In addition, we may collapse the tree as follows.

Collapsing Rule. After a *FIND*, make all vertices reached during the *FIND* operation sons of the root of the tree.

Figure 1 illustrates a *FIND* operation with collapsing. Collapsing at most multiplies the time a *FIND* takes by a constant factor and may save time in later finds. Knuth [4] attributes the collapsing rule to Tritter; independently, McIlroy and Morris [8] used it in an algorithm for finding spanning trees.

To carry out *UNION*(A, B, C), we locate the roots named A and B, make one a son of the other, and name the new root C, after deleting the old names (Figure 2). We may arbitrarily pick A or B as the new root, or we may apply a union rule, such as the following:

Weighted Union Rule. If set A contains more elements than set B, make B a son of A. Otherwise make A a son of B.

In order to implement this rule, we must attach a third item to each cell, namely the number of its descendants. Morris apparently first described the weighted union rule [8].

We can easily implement these instructions on a random-access computer. Suppose we carry out $m \geq n$ *FIND*s and $n - 1$ intermixed *UNION*s. Each *UNION* requires a fixed finite time. Each *FIND* requires some fixed amount of time plus time proportional to the length of the path from the vertex representing the element to the root of the corresponding tree. Let $t(m, n)$ be the maximum time required by any such sequence of instructions. (Often in practice m is $O(n)$. Previous researchers have restricted their attention to bounding $\max_{m+n=N} t(m, n)$ by some function of N Any upper or lower bound on $t(kN, N)$ for some constant k gives an upper or lower bound on $\max_{m+n=N} t(m, n)$ and vice versa, only the constants in the bound change.)

If neither the weighting nor the collapsing rule is used, it is easy to show that

$$k_1 mn \leq t(m, n) \leq k_2 mn \tag{1}$$

for suitable positive constants k_1 and k_2. If only the weighting rule is used, it is similarly easy to show that

$$k_1 m \log n \leq t(m, n) \leq k_2 m \log n \tag{2}$$

for some positive constants k_1 and k_2. Fischer [5] gave (1) and (2) for $m = n$. If only the collapsing rule is used, we shall show that

$$t(m, n) \leq km \cdot \max(1, \log(n^2/m)/\log(2m/n)) \tag{3}$$

for some positive constant k. Paterson [11] proved this bound for $m = n$ and Fischer [5] proved that it is tight to within a constant factor when $m = n$.

If we use both the weighting rule and the collapsing rule, the algorithm becomes much harder to analyze. Fischer [5] showed that $t(m, n) \leq km \log \log n$ in this case, and Hop-

---

w is an *ancestor* of v and v is a *descendant* of w (Every vertex is an ancestor and a descendant of itself.) If vertex v has no sons, then v is a *leaf* of T. The *depth* of a vertex v is the length of the path from v to s, and the *height* of v is the length of the longest path from a leaf of T to v. $| T |$ denotes the number of vertices in T, and $v \in T$ means v is a vertex of T.

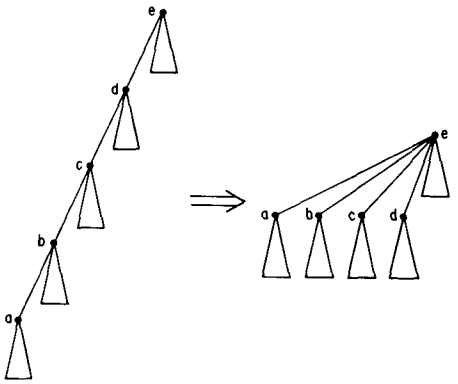

FIG 1. A *FIND* on element a, with collapsing. Triangles denote subtrees Collapsing converts tree T into tree T'

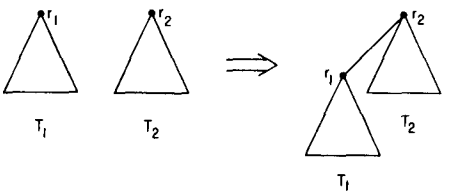

FIG 2 Union of two trees. Root r_1 of T_1 has a descendants; root r_2 of T_2 has b descendants. Root of new tree has $a + b$ descendants.

croft and Ullman [7] improved this bound to $t(m, n) \leq km \log^* n$. Here we show that

$$k_1 m\alpha(m, n) \leq t(m, n) \leq k_2 m\alpha(m, n)$$

for some positive constants k_1 and k_2, where $\alpha(m, n)$ is related to a functional inverse of Ackermann's function and is *very* slow-growing. Thus, $t(m, n)$ is $o(m \log^* n)$ but not $O(m)$.

An Upper Bound

It is useful to think about the set union algorithm in the following way [5]: suppose we perform all $n - 1$ *UNION*s first. Then we have a single tree with n vertices. Each of the original *FIND*s now is a "partial" find in the new tree: to carry out $FIND(x)$ we follow the path in the tree from x to the furthest ancestor of x corresponding to a *UNION* which appears before $FIND(x)$ in the original sequence of operations, and we collapse this path. Thus any sequence of m *FIND*s and $n - 1$ intermixed *UNION*s corresponds to a sequence of m partial finds performed on the tree created by carrying out the $n - 1$ *UNION*s (without any *FIND*s).

Furthermore, let T be any tree created by a sequence of $n - 1$ *UNION*s. For any vertex v, let $r(v)$, the *rank* of v, be the height of v in T. Then any sequence of m partial finds performed on T, such that the ranks of the last vertices on the finds are nondecreasing, corresponds to a sequence of m *FIND*s intermixed with the $n - 1$ *UNION*s used to create T. Thus we can get an upper bound on $t(m, n)$ by bounding the length of m partial finds performed on a tree created by a sequence of $n - 1$ *UNION*s. We can get a lower bound on $t(m, n)$ by bounding the maximum total length of m partial finds, whose last vertices have nondecreasing ranks, performed on a tree created by $n - 1$ *UNION*s.

To get an upper bound, we use a refinement and generalization of ideas in [7]. Since the techniques involved are somewhat complicated, we introduce them by proving the upper bound of (3), generalizing Paterson's result.

Let $t'(m, n)$ be the maximum time used by the set union algorithm for $m \geq n$ *FIND*s and $n - 1$ intermixed *UNION*s, assuming that the algorithm uses the collapsing rule but *not* the weighted union rule. We can bound $t'(m, n)$ by bounding the maximum total length of m partial finds performed on *any* tree with n vertices.

Let T be any tree with n vertices. If $v \in T$, let $r(v)$, the *rank* of v, be the height of v in T. Then $0 \leq r(v) \leq n - 1$, and $v \to w$ in T implies $r(v) < r(w)$. Furthermore, if $f(v) = w$ before a partial find and $f(v) = w' \neq w$ after the find, then $r(w) < r(w')$. Thus ranks strictly increase along any path in any tree formed from T by carrying out

partial finds. (Note. $r(v)$ is defined and fixed with respect to the original tree T, and does not change even though the tree changes when partial finds are carried out.)

To bound the total length of m partial finds performed on T, we shall partition the edges (v, w) on the find paths into various sets, bound the number of edges in each set, and add the bounds.

Let F be the set of edges (v, w) on the m partial find paths. For $1 \leq i \leq z$, where z and b are arbitrary integer parameters to be fixed later, let

$$M_i = \{(v, w) \in F \mid \exists j \text{ such that } b^i j \leq r(v) < r(w) < b^i(j+1) \text{ and } \exists k \text{ such that } r(v) < b^{i-1}k \leq r(w)\}.$$

(Note. $i - 1$ is the most significant position where the b-ary representations of $r(v)$ and $r(w)$ differ.)

Let $M_{z+1} = F - \bigcup_{i=1}^{z} M_i$. Clearly the sets M_i partition F. For $1 \leq i \leq z+1$, let

$L_i = \{(v, w) \in M_i \mid$ Of the edges on the find path containing (v, w), (v, w) is the last one in $M_i\}$.

LEMMA 1. $|L_i| \leq m$.

PROOF. Obvious.

LEMMA 2. $|M_i - L_i| \leq bn$ *for* $1 \leq i \leq z$.

PROOF. Let $v \in T$. Suppose $(v, w) \in M_i - L_i$. Then there is an edge $(v', w') \in M_i$ following (v, w) on the same find path. It follows from the definition of M_i that for some k_0, $r(w) \leq r(v') < b^{i-1}k_0 \leq r(w')$. If $w'' = f(v)$ after this find is performed, it follows that if $r(w) \geq b^{i-1}k$, $r(w'') \geq b^{i-1}(k+1)$.

Suppose that $M_i - L_i$ contains $x(v)$ edges of the form (v, w). Let $w''' = f(v)$ just before the last find corresponding to such an edge is performed. Then by the reasoning above, $r(w''') \geq b^{i-1}(b\lfloor r(v)/b^i \rfloor + x(v) - 1)$. But by the definition of M_i, $b^i(\lfloor r(v)/b^i \rfloor + 1) > r(w''')$. Therefore $x(v) - 1 < b$, and $x(v) \leq b$. Summing over all vertices, $|M_i - L_i| \leq bn$. Q.E.D.

LEMMA 3. $|M_{z+1} - L_{z+1}| \leq n^2/b^z + n$.

PROOF. Let $v \in T$. Suppose $(v, w) \in M_{z+1} - L_{z+1}$. Then there is an edge $(v', w') \in M_{z+1}$ following (v, w) on the same find path. Let $j = \lfloor r(w')/b^z \rfloor$. Then $r(w) \leq r(v') < b^z j \leq r(w') < b^z(j+1)$; otherwise $(v', w') \in M_i$ for some $1 \leq i \leq z$. If $w'' = f(v)$ after this find is performed, $\lfloor r(w'')/b^z \rfloor \geq \lfloor r(w')/b^z \rfloor \geq \lfloor r(w)/b^z \rfloor + 1$. Thus $\lfloor f(v)/b^z \rfloor$ increases by at least one each time an edge $(v, w) \in M_{z+1} - L_{z+1}$ occurs on a find path, and since $0 \leq \lfloor f(v)/b^z \rfloor \leq \lfloor (n-1)/b^z \rfloor$, v can occur in only $\lfloor (n-1)/b^z \rfloor + 1$ edges $(v, w) \in M_{z+1} - L_{z+1}$. Q.E.D.

THEOREM 4. $|F| \leq 3m \cdot max(1, \lceil log(n^2/m)/log\lfloor 2m/n \rfloor \rceil) + 2m + n$.

PROOF.

$$|F| = \sum_{i=1}^{z+1} |L_i| + \sum_{i=1}^{z+1} |M_i - L_i| \leq (z+1)m + bzn + n^2/b^z + n \quad \text{for all } z \geq 1,\ b \geq 1.$$

Pick $b = \lfloor 2m/n \rfloor$ and $z = \max(1, \lceil \log(n^2/m)/\log b \rceil)$. Then $|F| \leq 3m \cdot \max(1, \lceil \log(n^2/m)/\log \lfloor 2m/n \rfloor \rceil) + 2m + n$. Q.E.D.

COROLLARY 5.

$$t'(m, n) \leq k \cdot m \ max(1, log(n^2/m)/log(2m/n)) \quad \textit{for a suitable constant } k. \tag{4}$$

If the algorithm uses the weighted union rule, we can use a subtler version of the same partitioning idea to get an upper bound. Let $t(m, n)$ be the maximum time used by the set union algorithm for $m \geq n$ *FIND*s and $n - 1$ *UNION*s, assuming that the algorithm uses the collapsing rule and the weighted union rule.

Let T be any tree of n vertices formed using the weighted union rule (and no finds).

Let $r(v)$ be defined as before, fixed with respect to T. For $v \in T$, let $d(v)$ be the number of descendants of v, again fixed with respect to T.

LEMMA 6. *If $v \to w$ in T, then $d(w) \geq 2d(v)$.*

PROOF. In the process of forming T, some union makes v a son of w. At this time $d(w) \geq 2d(v)$ because the union obeys the weighted union rule. Subsequent unions cannot change the number of descendants of v and can only increase the number of descendants of w. Q.E.D.

COROLLARY 7. *$d(v) \geq 2^{r(v)}$ and $0 \leq r(v) \leq \log_2 n$ for all $v \in T$.*

PROOF. By induction on $r(v)$, using Lemma 6.

Let the function $A(i, x)$ on integers be defined by

$$\begin{aligned} &A(0, x) = 2x, \quad A(i, 0) = 0 \quad \text{for} \quad i \geq 1, \\ &A(i, 1) = 2 \quad \text{for} \quad 1 \geq 1, \quad A(i, x) = A(i-1, A(i, x-1)) \quad \text{for} \quad i \geq 1, \quad x \geq 2. \end{aligned} \tag{5}$$

$A(i, x)$ is a slight variant of Ackermann's function [1]; it is not primitive recursive. Some important facts about $A(i, x)$ appear below.

$A(0, 2) = 2 \cdot 2 = 4$. $A(i+1, 2) = A(i, A(i+1, 1)) = A(i, 2)$. Therefore, by induction,

$$A(i, 2) = 4 \quad \text{for all } i.$$

$A(1, 1) = 2$. $A(1, x+1) = A(0, A(1, x)) = 2 \cdot A(1, x)$. Therefore,

$$A(1, x) = 2^x \quad \text{for} \quad x \geq 1. \tag{6}$$

$A(2, 1) = 2$. $A(2, x+1) = A(1, A(2, x)) = 2^{A(2,x)}$. Therefore,

$$\begin{aligned} &A(2, x) = 2^{2^{-2}} \Big\} \, x \text{ two's for } x \geq 1. \\ &A(4, 3) = A(3, A(4, 2)) = A(3, 4) = A(2, A(3, 3)) = A(2, A(2, A(3, 2))) \\ &\qquad = A(2, A(2, 4)). \\ &A(0, x) = 2x \geq x. \; A(i, 0) \geq 0. \; A(i, 1) = 2 \geq 1. \end{aligned} \tag{7}$$

$A(i, x) \geq x$ for all x implies $A(i+1, x) = A(i, A(i, x)) \geq A(i, x) \geq x$ for $x \geq 2$. Therefore,

$$A(i, x) \geq x \quad \text{for all } i, x. \tag{8}$$

$A(i+1, 0) = 0 = A(i, 0)$. $A(i+1, 1) = 2 = A(i, 1)$ for $i \geq 0$. $A(i+1, x) = A(i, A(i, x)) \geq A(i, x)$ for $i \geq 0$, $x \geq 2$ by (8). Thus

$$A(i+1, x) \geq A(i, x) \text{ for all } i, x. \tag{9}$$

$A(0, x+1) = 2x + 2 > 2x = A(0, x)$, $A(i, 1) = 2 > 0 = A(i, 0)$, $A(i+1, x+1) = A(i, A(i+1, x)) > A(i, A(i+1, x-1)) = A(i+1, x)$ for $x \geq 1$, if $A(i+1, x) > A(i+1, x-1)$ and $A(i, y+1) > A(i, y)$ for all y. By double induction,

$$A(i, x+1) > A(i, x) \text{ for all } i, x. \tag{10}$$

The following rather weak inequalities will be used in the lower bound proof. By (6), (9), and (10),

$$A(i+1, x+1) = A(i, A(i+1, x)) \geq A(i, 2^x) \quad \text{for} \quad x \geq 1. \tag{11}$$

$$\begin{aligned} A(4k-2, 3) &= A(4k-3, A(4k-2, 2)) = A(4k-3, 4) \\ &= A(4k-4, A(4k-3, 3)) \\ &\geq A(4k-4, 8) \geq A(4k-4, 4) + 4 \text{ for } k \geq 2. \end{aligned} \tag{12}$$

$$\begin{aligned} A(4k-2, 5) &= A(4k-3, A(4k-2, 4)) \geq 2^{A(4k-2, 4)} \\ &\geq 3A(4k-2, 4) \text{ for } k \geq 1. \end{aligned} \tag{13}$$

$$\begin{aligned} A(4k, 4) &= A(4k-1, A(4k, 3)) = A(4k-1, A(4k-1, 4)) \\ &= A(4k-1, A(4k-2, A(4k-1, 3))) \geq 2^{A(k-2,6)} \\ &\geq 4A(4k-2, 6) \text{ if } k \geq 1. \end{aligned} \tag{14}$$

$$\begin{aligned} A(4k, 4s-3) &= A(4k-1, A(4k, 4s-4)) \\ &= A(4k-2, A(4k-1, A(4k, 4s-4)-1)) \\ &\geq A(4k-3, 2^{A(4k,\,4s-4)-1}) \\ &\geq A(4k-3, A(4k, 4s-4)+2) \text{ if } k \geq 1, s \geq 2. \end{aligned} \tag{15}$$

$$\begin{aligned} A(4k, x) &= A(4k-1, A(4k, x-1)) = A(4k-2, A(4k-1, A(4k, x-1)-1)) \\ &\geq A(4k-3, 2^{A(4k,\,x-1)-1}) \\ &\geq A(4k-3, 4A(4k, x-1)) \text{ if } k \geq 1, x \geq 4. \end{aligned} \tag{16}$$

Let $a(i, n) = \min\{j \mid A(i, j) > \log_2 n\}$. The function $a(1, n)$ is $O(\log \log n)$, $a(2, n)$ is $O(\log^* n)$, and $a(3, n)$ is very slow-growing.

For $0 \leq i \leq z$, $0 \leq j < a(i, n)$, where z is an arbitrary parameter to be fixed later, let

$$S_{ij} = \{v \mid A(i, j) \leq r(v) < A(i, j+1)\}.$$

For a fixed value of i, the S_{ij} partition the vertices of T.

LEMMA 8. *$|S_{ij}|$, the number of elements in set S_{ij}, satisfies $|S_{ij}| \leq 2n/2^{A(i,j)}$.*

PROOF. Any two vertices v and w with the same rank k have disjoint sets of descendants in T, and each has at least 2^k descendants by Corollory 7. Thus the number of vertices of rank k is bounded above by $n/2^k$, and

$$|S_{ij}| \leq \sum_{k=A(i,j)}^{A(i,j+1)} (n/2^k) \leq 2n/2^{A(i,j)}. \qquad \text{Q.E.D.}$$

Let m partial finds be performed on T. Let F be the set of edges (v, w) on these find paths. Partition F as follows: if $(v, w) \in F$ and for some i and j, $v \in S_{ij}$ and $w \in S_{ij}$, let $(v, w) \in N_k$, where $k = \min\{i \mid \exists j v, w \in S_{ij}\}$. If for all i and j, either $v \notin S_{ij}$ or $w \notin S_{ij}$, let $(v, w) \in N_{z+1}$. For $0 \leq i \leq z+1$, let

$L_i = \{(v, w) \in N_i \mid$ Of the edges on the find path containing (v, w), (v, w) is the last one in $N_i\}$.

LEMMA 9. $|L_i| \leq m$.

PROOF. Obvious.

LEMMA 10. $|N_0 - L_0| \leq n$.

PROOF. Let $v \in T$. Suppose $(v, w) \in N_0 - L_0$. Then there is an edge $(v', w') \in N_0$ following (v, w) on the same find path. It follows that for some j, $2j \leq r(v) < r(w) < 2j+2 \leq r(w')$, and $r(w') - r(v) \geq 2$. If $f(v) = w''$ after this find is performed, $r(w'') - r(v) \geq 2$; and no finds after this one can contain an edge $(v, w''') \in N_0$. Thus each vertex v is in at most one edge $(v, w) \in N_0 - L_0$. Q.E.D.

LEMMA 11. *For $1 \leq i \leq z$, $|N_i - L_i| \leq \frac{5}{8}n$.*

PROOF. Let $v \in T$ and suppose $v \in S_{ij}$; i.e. $A(i, j) \leq r(v) < A(i, j+1)$. Suppose $(v, w) \in N_i - L_i$. Since $S_{i0} = S_{00}$ and $S_{i1} = S_{01}$ for all i, it must be the case that $j \geq 2$. There is an edge $(v', w') \in N_i$ following (v, w) on the same find path. From the definition of N_i, there is some k_0 such that $r(w) \leq r(v') < A(i-1, k_0) \leq r(w')$. If $w'' = f(v)$ after this find is performed, it follows that if $A(i-1, k) \leq r(w)$, $A(i-1, k+1) \leq r(w'')$.

Suppose that $N_i - L_i$ contains $x(v)$ edges of the form (v, w). Let $w''' = f(v)$ just before the last find corresponding to such an edge is performed. Then by the reasoning above, $A(i-1, x(v)-1) \leq r(w''')$, and by the definition of N_i, $r(w''') < A(i, j+1)$. Since $j \geq 2$, $A(i-1, x(v)-1) < A(i, j+1) = A(i-1, A(i, j))$, and since A is increasing in its second argument, $x(v) - 1 < A(i, j)$; or, $x(v) \leq A(i, j)$.

Thus, applying Lemma 8,

$$|N_i - L_i| \leq \sum_{j=2}^{\infty} A(i,j)\,|S_{i,j}| \leq \sum_{j=2}^{\infty} (2n/2^{A(i,j)})\,A(i,j)$$
$$\leq \sum_{j=2}^{\infty} (2n/2^{2^j})2^j \qquad \text{since } A \text{ is increasing in its first argument}$$
$$\leq \tfrac{5}{8}n. \qquad \text{Q.E.D.}$$

LEMMA 12. $|N_{z+1} - L_{z+1}| \leq na(z, n)$.

PROOF. Let $v \in T$. Suppose $(v, w) \in N_{z+1} - L_{z+1}$. Then there is some edge $(v', w') \in N_i$ following (v, w) on the same find path. From the definition of N_{z+1}, there is a k_0 such that $r(w) \leq r(v') < A(z, k_0) \leq r(w')$. If $w'' = f(v)$ after this find is performed, it follows that if $A(z, k) \leq r(w)$, $A(z, k+1) \leq r(w'')$.

Suppose that $N_{z+1} - L_{z+1}$ contains $y(v)$ edges of the form (v, w). Let $w''' = f(v)$ just before the last find corresponding to such an edge is performed. Then by the reasoning above, $A(z, y(v) - 1) \leq r(w''')$; but $r(w''') \leq \log_2 n$. Thus $y(v) - 1 < a(z, n)$, and $y(v) \leq a(z, n)$. Q.E.D.

Let $\alpha(m, n) = \min\{z \geq 1 \mid A(z, 4\lceil m/n \rceil) > \log_2 n\}$.

THEOREM 13. $|F| \leq (\frac{5}{8}n + m)\alpha(m, n) + 10m + n$.

PROOF.

$$|F| = \sum_{i=0}^{z+1} |N_i - L_i| + \sum_{i=0}^{z+1} |L_i|$$
$$\leq n + \tfrac{5}{8}zn + na(z, n) + (z+2)m \text{ for any } z \geq 1, \text{ by Lemmas 9, 10, 11, and 12}$$
$$\leq (\tfrac{5}{8}n + m)\alpha(m, n) + 10m + n, \text{ by choosing } z = \alpha(m, n), \text{ since}$$
$$a(\alpha(m, n), n) \leq 4\lceil m/n \rceil \leq 8m/n. \qquad \text{Q.E.D.}$$

COROLLARY 14. $t(m, n) \leq k_2 m\alpha(m, n)$ *for some constant* k_2. (17)

The function $\alpha(m, n)$ is maximized when $m = n$; if k is a fixed constant and $m \geq n \cdot a(k, n)$, (17) implies $t(m, n)$ is $O(m)$. If $\log_2 n < A(3, 4)$, $\alpha(m, n) \leq 3$.

$$A(3, 4) = 2^{2^{-^{-^{-^{-2}}}}} \Big\} \text{ 65,536 two's by (7).}$$

Thus for all practical purposes, $\alpha(m, n) \leq 3$.

The next section shows that the upper bound (17) is tight within a constant factor.

A Lower Bound

Let $t(m, n)$ be the maximum time required by the set union algorithm for $m \geq n$ *FIND*s and $n - 1$ *UNION*s, assuming that both collapsing and weighted union are used. To get a nonlinear lower bound on $t(m, n)$, we shall show that for any k, there is a finite $n(k)$ such that a tree with $n(k)$ vertices exists in which $n(k)$ partial finds, each of length k, may be performed. The argument is quite complicated, but is devised to be as general as possible.

Let T be a tree contained in an acyclic graph G, called a *shortcut graph* of T. If $v_0 \to v_1 \to \cdots \to v_k$ is a path in T, we perform a *generalized find* (g-find) in T by adding to G each edge (v_i, v_j), $i < j$, which is not already present in G. The *cost* of the g-find is the length l of the shortest path from v_0 to v_k in G (before the new edges are added). We shall only allow g-finds such that v_k is the root of T or the father of v_k is at distance $l + 1$ from v_0 in G before the new edges are added. We shall provide a nonlinear lower bound on the maximum total cost of a sequence of $m \geq n$ g-finds performed on a tree T with n vertices constructed using *any* union rule, assuming that the shortcut graph is initially T itself. This gives a nonlinear lower bound on $t(m, n)$. It also gives a nonlinear lower bound on the running time of almost any conceivable set union algorithm.

Let T be any tree. Let $T(0) = T$. For any $i \geq 1$, let $T(i)$ be formed from two copies of $T(i-1)$ by making the root of one of them the son of the root of the other. If T is the tree having a single vertex, $T(i)$ is called an S_i tree. S_i has 2^i vertices and 2^{i-1} leaves. Removal of all the leaves from S_i produces S_{i-1}. S_i may be formed using *any* union rule, since the trees combined at each step are identical.

Let G be a shortcut graph of T. Let $G(0) = G$. For any $i \geq 1$, let $G(i)$ be formed from two copies of $G(i-1)$ by adding an edge from the root of $T(i-1)$ embedded in one to the root of $T(i-1)$ embedded in the other. Then $G(i)$ is a shortcut graph of $T(i)$.

THEOREM 15. *Let T be any tree with two or more vertices and $s \geq 1$ leaves, and let G be any shortcut graph of T. If $i \geq A(4k, 4s)$, we can perform a g-find of cost k on at least half the leaves in $T(i)$, starting with shortcut graph $G(i)$.*

PROOF. We prove the theorem by double induction on k and s. Suppose $k \leq 1$ and s is arbitrary. For any $i \geq A(4k, 4s) \geq 0$, each leaf in $T(i)$ is at a distance of one or more from the root of $T(i)$ in any shortcut graph. Thus the theorem is true for $k \leq 1$.

Suppose $k = 2$ and s is arbitrary. Half the leaves in $T(1)$ are at a distance of at least two in $G(1)$ from the root of $T(1)$ and remain that way regardless of what g-finds are done on the other leaves. Thus g-finds of cost two can be done on all these leaves. It follows that if $i \geq A(4k, 4s) = A(8, 4s) \geq 1$, g-finds of cost two can be done on half the leaves of $T(i)$. Thus the theorem is true for $k = 2$.

Suppose the theorem holds for all $k' < k$ and arbitrary s. We prove the theorem holds for k with $s = 1$. We can assume $k \geq 3$. The tree $T(1)$ has two leaves. One is in the copy of T whose root is the root of $T(1)$. Call this the r-leaf of $T(1)$ and call the leaf in the other copy of T the u-leaf of $T(1)$. The u-leaf has a father different from the root of $T(1)$.

If T' is the tree consisting of the path from the father of the u-leaf to the root of $T(1)$, then by the induction hypothesis a g-find of length $k-1$ may be performed in $T'(A(4k-4, 4))$ on half the leaves, starting with shortcut graph $G(1 + A(4k-4, 4))$. Thus in $T(1 + A(4k-4, 4))$ a g-find of length $k-1$ can be performed on the fathers of one-half of the u-leaves, starting with shortcut graph $G(1 + A(4k-4, 4))$. This means that in $T(1 + A(4k-4, 4))$ a g-find of length k can be performed on one-half of the u-leaves, starting with shortcut graph $G(1 + A(4k-4, 4))$. Let G' be the resulting shortcut graph.

Consider the u-leaves of the $T(1)$ trees embedded in $T(1 + A(4k-4, 4))$ on which g-finds have not been performed. There are $2^{A(4k-4, 4)-1}$ such leaves. Each of these has a distinct father and no pair of these fathers is related in $T(1 + A(4k-4, 4))$. It follows by the induction hypothesis that in $T(1 + A(4k-4, 4) + A(4k-4, 2^{A(4k-4, 4)+1}))$ a g-find of length $k-1$ can be performed on one-half of these fathers, starting with shortcut graph $G'(A(4k-4, 2^{A(4k-4, 4)+1}))$. Let $n_1 = 1 + A(4k-4, 4) + A(4k-4, 2^{A(4k-4, 4)+1})$. Then in $T(n_1)$ a g-find of length k can be performed on an additional one-fourth of the u-leaves of the embedded trees $T(1)$, starting with shortcut graph $G'(A(4k-4, 2^{A(4k-4, 4)+1}))$. Let the resulting shortcut graph be G''.

Now consider the r-leaves of the trees $T(1)$ embedded in $T(n_1)$. No g-finds have been performed on these $2^{n_1-1} \geq 2$ leaves. The fathers of all these leaves are distinct and half of them are unrelated to each other. By the induction hypothesis, in $T(n_1 + A(4k-4, 2^{n_1}))$ we may perform a g-find of length $k-1$ on one-half of these unrelated fathers, starting with shortcut graph $G''(A(4k-4, 2^{n_1}))$. It follows that in $T(n_1 + A(4k-4, 2^{n_1}))$, g-finds of length k can be performed on an additional one-eighth of the leaves of the embedded $T(1)$ trees, starting with shortcut graph $G''(A(4k-4, 2^{n_1}))$.

Combining these results, we see that in $T(n_1 + A(4k-4, 2^{n_1}))$, starting with shortcut graph $G(n_1 + A(4k-4, 2^{n_1}))$, we can perform g-finds of length k on one-half of the leaves. Furthermore,

$$
\begin{aligned}
n_1 &= 1 + A(4k-4, 4) + A(4k-4, 2^{A(4k-4, 4)+1}) \\
&\leq 1 + A(4k-4, 4) + A(4k-3, A(4k-4, 4) + 2) && \text{by (11)} \\
&\leq 1 + A(4k-4, 4) + A(4k-2, 4) && \text{by (5), (12)} \qquad (18)
\end{aligned}
$$

and

$$\begin{aligned} n_1 + A(4k-4, 2^{n_1}) &\le n_1 + A(4k-3, n_1+1) && \text{by (11)} \\ &\le n_1 + A(4k-3, A(4k-4, 4)) \\ &\quad + A(4k-2, 4) + 2) && \text{by (9), (10), (13), (18)} \\ &\le 4A(4k-2, 6) \\ &\le A(4k, 4) && \text{by (14).} \end{aligned}$$

Thus the theorem holds for k with $s = 1$, since if we can perform the desired g-finds in $T(n_1 + A(4k-4, 2^{n_1}))$, we can certainly perform them in $T(i)$ if $i \ge A(4k, 4) \ge n_1 + A(4k-4, 2^{n_1})$.

Suppose the theorem holds for all k', s' such that $k' < k$, or $k' = k$ and $s' < s$. We prove the theorem for k and s, using an argument like that above but slightly more complicated.

Consider $T(A(4k, 4s-4))$. Ignoring one leaf in each copy of T, by the induction hypothesis we can perform g-finds of length k on one-half of the remaining leaves, producing a shortcut graph G' from $G(A(4k, 4s-4))$. Now consider the ignored leaves, one per copy of T. Since $A(4k, 4s-4) \ge 1$, $T(A(4k, 4s-4))$ consists of $2^{A(4k, 4s-4)-1}$ copies of $T(1)$. In each copy of $T(1)$ there is one ignored leaf in the copy of T whose root is that of $T(1)$, call this the r-leaf. Call the ignored leaf in the other copy of T in $T(1)$ the u-leaf. In $T(A(4k, 4s-4) + A(4k-4, 2^{A(4k, 4s-4)+1}))$ we can, by the induction hypothesis, perform g-finds of length $k-1$ on one-half of the fathers of the u-leaves (since the fathers of the u-leaves are unrelated). Alternatively we can perform g-finds of length k on one-half of the u-leaves, producing shortcut graph G'' from shortcut graph $G'(A(4k-4, 2^{A(4k, 4s-4)+1}))$.

Let $n_2 = A(4k, 4s-4) + A(4k-4, 2^{A(4k, 4s-4)+1})$. In $T(n_2 + A(4k-4, 2^{n_2}))$ we can perform g-finds of length $k-1$ on the fathers of an additional one-fourth of the u-leaves, or instead g-finds of length k on an additional one-fourth of the u-leaves, producing shortcut graph G''' from shortcut graph $G''(A(4k-4, 2^{n_2}))$.

Let $n_3 = n_2 + A(4k-4, 2^{n_2})$. Now consider the r-leaves of $T(1)$ contained in $T(n_3)$. Half of them have distinct, unrelated fathers. Hence in $T(n_3 + A(4k-4, 2^{n_3}))$ we can perform g-finds of length $k-1$ on the fathers of one-fourth of the r-leaves, or instead g-finds of length k on one-fourth of the r-leaves, starting with shortcut graph $G'''(A(4k-4, 2^{n_3}))$.

Combining, we see that in $T(1)$, if $i \ge n_3 + A(4k-4, 2^{n_3})$, we can perform g-finds of length k on one-half of the leaves, starting with shortcut graph $G(i)$.

But we have

$$\begin{aligned} n_2 &= A(4k, 4s-4) + A(4k-4, 2^{A(4k, 4s-4)+1}) \\ &\le A(4k, 4s-4) + A(4k-3, A(4k, 4s-4) + 2) && \text{by (11)} \\ &\le A(4k, 4s-4) + A(4k, 4s-3) && \text{by (15).} \end{aligned} \tag{19}$$

$$\begin{aligned} n_3 &= n_2 + A(4k-4, 2^{n_2}) \\ &\le A(4k, 4s-4) + A(4k, 4s-3) + A(4k-3, A(4k, 4s-4) \\ &\quad + A(4k, 4s-3) + 1) && \text{by (11), (19)} \\ &\le A(4k, 4s-4) + A(4k, 4s-3) \\ &\quad + A(4k-3, 3A(4k, 4s-3)) && \text{by (10)} \\ &\le A(4k, 4s-4) + A(4k, 4s-3) + A(4k, 4s-2) && \text{by (16)} \\ &\le 3A(4k, 4s-2) && \text{by (10).} \end{aligned} \tag{20}$$

$$\begin{aligned} n_3 &+ A(4k-4, 2^{n_3}) \\ &\le 3A(4k, 4s-2) + A(4k-3, 3A(4k, 4s-2) + 1) && \text{by (11), (20)} \\ &\le 3A(4k, 4s-2) + A(4k, 4s-1) && \text{by (16)} \\ &\le 4A(4k, 4s-1) && \text{by (10)} \\ &\le 2^{A(4k, 4s-1)} \le A(4k-1, A(4k, 4s-1)) = A(4k, 4s) && \text{by (5), (6), (9)} \end{aligned}$$

Thus the theorem holds for k and s, and the theorem holds in general by double induction.

Let $t''(m, n)$ be the maximum cost of a sequence of $m \geq n$ g-finds performed on any tree T of n vertices formed using some union rule, starting with T itself as the shortcut graph.

THEOREM 16. *For some constant* k_1, $k_1 m\,\alpha(m, n) \leq t''(m, n)$.

PROOF. Let T be a tree consisting of a root and s sons of the root. By Theorem 15, a g-find of cost k can be performed on half the leaves of $T(A(4k, 4s))$. It follows that a total of $s \cdot 2^{A(4k,\,4s)-2}$ g-finds of cost $k-1$ can be performed on vertices of $S_{A(4k,4s)}$.

Let m and n satisfy $\alpha(m, n) \geq 2$. Let $k = \lfloor \frac{1}{4}\alpha(m, n) \rfloor - 1$. Then $A(4k, 4\lceil m/n \rceil) \leq \log n$. From n vertices, using any union rule, we can construct one or more copies of $S_{A(4k,\,4\lceil m/n\rceil)}$. We can use up at least half the available vertices forming such trees. Within each such tree, we can perform $\lceil m/n \rceil \cdot 2^{A(4k,\,4\lceil m/n\rceil)-2}$ g-finds, each of cost $k-1$. Thus the total cost of all such finds in all the trees is at least $(m/8)(\lfloor \frac{1}{4}\alpha(m, n) \rfloor - 2)$. This gives the theorem.

COROLLARY 17. *For some positive constant* k_1, $k_1 m\,\alpha(m, n) \leq t(m, n)$.

PROOF. The sequence of g-finds given by Theorem 16 can be interpreted as a sequence of partial finds, each of length at least k. These partial finds can be ordered so that the ranks of their final vertices are nondecreasing. This gives a sequence of $n-1$ *UNION*s and m interspersed *FIND*s of total cost $k_1 m\alpha(m, n)$ for a suitable positive constant k_1.

Thus the bound (17) is tight to within a constant factor.

Conclusions and Open Problems

We have analyzed a known algorithm for computing disjoint set unions, showing that its worst-case running time is $O(m\alpha(m, n))$, where $\alpha(m, n)$ is related to a functional inverse of Ackermann's function, and that this bound is tight to within a constant factor. This is probably the first and maybe the only existing example of a simple algorithm with a very complicated running time. The lower bound given in Theorem 16 is general enough to apply to many variations of the algorithm, although it is an open problem whether there is a linear-time algorithm for the online set union problem. On the basis of Theorem 16, I conjecture that there is *no* linear-time method, and that the algorithm considered here is optimal to within a constant factor.

An interesting though less significant problem is to determine the exact running time of the set union algorithm if the algorithm does not use the weighted union rule. The bound (14) is tight to within a constant factor if $m \leq cn$ or if $m \geq cn^{1+\epsilon}$, for some constants c and ϵ; but a better bound may exist for intermediate values.

REFERENCES

1. ACKERMANN, W Zum Hilbertshen Aufbau der reelen Zahlen *Math. Ann. 99* (1928), 118–133.
2. AHO, A. V , HOPCROFT, J. E , AND ULLMAN, J D On computing least common ancestors in trees. Proc 5th Annual ACM Symp. on Theory of Computing, Austin, Texas, 1973, pp. 253–265
3. ARDEN, B. W , GALLER, B. A , AND GRAHAM, R M. An algorithm for equivalence declarations. *Comm. ACM 4*, 7 (July 1961), 310–314.
4. CHVÁTAL, V., KLARNER, D. A , AND KNUTH, D E. Selected combinatorial research problems Tech Rep STAN-CS-72-292, Comput. Sci Dep , Stanford U., Stanford, Calif , 1972.
5. FISCHER, M J. Efficiency of equivalence algorithms. In *Complexity of Computer Computations*, R. E. Miller and J W Thatcher, Eds., Plenum Press, New York, 1972, pp. 153–168.
6. GALLER, B. A , AND FISCHER, M. J. An improved equivalence algorithm *Comm. ACM 7*, 5 (May 1964), 301–303.
7. HOPCROFT, J , AND ULLMAN, J. D. Set-merging algorithms. *SIAM J. Comput 2* (Dec. 1973), 294–303.
8. HOPCROFT, J Private communication.
9. KERSCHENBAUM, A , AND VAN SLYKE, R Computing minimum spanning trees efficiently. Proc 25th Annual Conf of the ACM, 1972, p.p 518–527.

10 KNUTH, D. E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, Reading, Mass , 1969, pp 353–355.
11. PATERSON, M. Unpublished report, U. of Warwick, Coventry, Great Britain.
12 STEARNS, R E., AND ROSENKRANTZ, D. J. Table machine simulation. 10th Annual SWAT Conf. Proc , 1969, pp 118–128.
13 TARJAN, R Testing flow graph reducibility. Proc 5th Annual ACM Symp. on Theory of Computing, Austin, Texas, 1973, pp. 96–107
14 TARJAN, R. Finding dominators in directed graphs *SIAM J Comput. 3* (March 1974), 62–89.

RECEIVED NOVEMBER 1972; REVISED AUGUST 1974