EFFICIENCY OF A GOOD BUT NOT LINEAR SET UNION ALGORITHM

by

Robert Endre Tarjan

Memorandum No. ERL-M434

March 28, 1974

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

EFFICIENCY OF A GOOD BUT NOT LINEAR SET UNION ALGORITHM

Robert Endre Tarjan[†]
Department of Electrical Engineering
and Computer Sciences
University of California
Berkeley, California

ABSTRACT

Consider 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. We examine a known algorithm for implementing sequences of these instructions. If t(n) is the maximum time required by any sequence of O(n) instructions, we show that

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

for some constants k_1 and k_2, where $\alpha(n)$ is a functional inverse of Ackermann's function and is very slowly growing.

Keywords and Phrases:

algorithm, complexity, equivalence, partition, set union, tree.

CR Categories: 4.12, 5.25, 5.32.

[†] This work was partially supported by the National Science Foundation, Contract Number NSF-GJ-35604X, and by a Miller Research Fellowship.

EFFICIENCY OF A GOOD BUT NOT LINEAR SET UNION ALGORITHM

Robert Endre Tarjan
Department of Electrical Engineering
and Computer Sciences
Computer Science Division
University of California
Berkeley, California

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 O(n) instructions of the two types.

An algorithm for solving this problem is useful in many contexts, including handling EQUIVALENCE and COMMON statements in FORTRAN [2,4], finding minimum spanning trees [7], computing dominators in directed graphs [13], checking flow graphs for reducibility [12], calculating depths in trees [1], computing least common ancestors in trees [1], and solving an off-line minimum problem [5].

Several algorithms have been developed [2, 3, 4, 5, 11], notably a very complicated one by Hopcroft and Ullman [5]. It is an extension of an idea by Stearns and Rosenkrantz [11] and has an $O(n \log^* n)$ running time, where

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

All other known algorithms are slower, except for the very simple one which we consider here.

Each set is represented as a tree [†]. Each vertex in the tree represents an element in the 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 [2,4].

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:

Collapsing Rule:

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

---

† For the purposes of this paper, a tree T is a directed graph with a unique vertex r, the root of T, such that (i) no edge (v,r) exists in T, (ii) if $w \neq r$, 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 $v \rightarrow w$), v is called the father of w (denoted by $v = f(w)$) and w is called a son of v. If there is a path from v to w in T (denoted by $v \overset{*}{\rightarrow} w$), then v is an ancestor of w and w is a descendant of v. (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 r to v, and the height of v is the length of the longest path from v to a leaf of T. The depth of T is the maximum of the depths of its vertices. $|T|$ denotes the number of vertices in T, and $v \in T$ means v is a vertex of T.

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 [9] attributes the collapsing rule to Tritter; independently McIlroy and Morris used it in an algorithm for finding spanning trees [6].

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. Bob Morris apparently first described the weighted union rule [6].

We can easily implement these instructions on a random-access computer. Suppose we carry out O(n) FIND's and UNION's. Each UNION requires a fixed finite time. Each FIND requires time proportional to one plus the length of the path from the vertex representing the element to the root of the corresponding tree. To simplify the analysis slightly, let us assume that we carry out exactly n FIND's and exactly n - 1 UNION's.

Let t(n) be the maximum time required by any such sequence of instructions. If neither the weighting nor the collapsing rule is used, it is easy to show that:

$$(1) \qquad k_1 n^2 \leq t(n) \leq k_2 n^2$$

for suitable constants k_1 and k_2 [3]. If only the weighting rule is

used, it is similarly easy to show that:

$$(2) \qquad k_1 \, n \log n \leq t(n) \leq k_2 \, n \log n$$

for some constants k_1 and k_2 [3]. If only the collapsing rule is used, we also have:

$$(3) \qquad k_1 \, n \log n \leq t(n) \leq k_2 \, n \log n$$

for some constants k_1 and k_2. Fischer proved the lower bound [3] and Paterson the upper bound [11].

If we use both the weighting rule and the collapsing rule, the algorithm becomes much harder to analyze. Fischer [3] showed that $t(n) \leq k_2 \, n \log \log n$ in this case, and Hopcroft and Ullman [5] improved this bound to $t(n) \leq k_2 \, n \log^* n$. Here we show that

$$(4) \qquad k_1 n \ \alpha(n) \leq t(n) \leq k_2 n \ \alpha(n)$$

for some constants k_1 and k_2, where $\alpha(n)$ is a functional inverse of Ackermann's function and is <u>very</u> slowly growing. Thus, $t(n)$ is $o(n \log^* n)$ but not $O(n)$.

<u>An Upper Bound</u>

It is useful to think about the set union algorithm in the following way: 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. In this

interpretation of the problem, we are interested in bounding the total length of m partial finds performed on a tree generated by $n - 1$ weighted UNION's. (We shall allow the number of FINDS to be different from the number of UNIONS.)

The upper bound argument below is a refinement of Hopcroft and Ullman's method. Let T be any tree with n vertices. Let $d(v)$ be the number of descendants of any vertex v. Let $h(v)$ be the height of v in T.

Lemma 1

If T is formed by weighted unions and $v \to w$ in T, then $d(v) \geq 2d(w)$.

Proof:

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

Suppose T is formed by weighted unions. Lemma 1 implies that $d(v) \geq 2^{h(v)}$ for all $v \in T$. Fix $h(v)$ as the height of v in the original tree T and suppose we perform some partial finds on T, creating a new tree T'. The $h(v)$ values must strictly increase along any path of T', because $v \to w$ in T' implies $v \overset{*}{\to} w$ in T. Furthermore, if v is moved by a find, the height of its father must strictly increase.

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

(5)
$$A(0, x) = 2x$$
$$A(i, 0) = 0 \qquad \text{for } i \geq 1$$
$$A(i, 1) = 2 \qquad \text{for } i \geq 1$$
$$A(i, x) = A(i-1, A(i, x-1)) \qquad \text{for } i \geq 1,\ x \geq 2.$$

A(i,x) is a slight variant of Ackermann's function; it is not primitive recursive. A(i,x) is found by composing A(i-1,y) with itself x - 1 times and taking the value of the resultant function at y = 1.

For $x \geq 1$,

(6)
$$A(1, x) = 2^x, \quad A(2, x) = 2^{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}} \Big\} \ x \text{ two's},$$

and so on.

It is easy to show that $A(i, 2) = 4$ for all i.

Let

(7)
$$a(i, n) = \min \{j \mid A(i, j) > \lfloor \log_2 n \rfloor \}.$$

For fixed z, construct sets S_{ij} for $0 \leq i \leq z$, $0 \leq j < a(i,n)$, composed of the vertices of T, as follows:

(8)
$$S_{ij} = \{v \mid A(i, j) \leq h(v) < A(i, j + 1)\}.$$

For a fixed value of i, the S_{ij}'s are disjoint, and

(9)
$$\bigcup_{j=0}^{a(i,n)-1} S_{ij} = \{v \mid v \in T\}.$$

Lemma 1:

$|S_{ij}|$, the number of elements in set S_{ij}, satisfies

(10)
$$|S_{ij}| \leq \frac{2n}{2^{A(i,j)}}.$$

Proof:

If $v \in T$, $d(v) \geq 2^{h(v)}$. If v and w have height k, their sets of ancestors are disjoint. Thus, for any fixed k, the number of vertices of height k is bounded above by $n/2^k$.

Thus,

$$(11) \qquad |S_{ij}| \leq \sum_{k=A(i,j)}^{\infty} \frac{n}{2^{A(i,j)}} = \frac{2n}{2^{A(i,j)}} .$$

Q.E.D.

Now suppose m partial finds are performed on T. Each find moves all but the last two vertices on the find path. Let N_0 be the number of edges (v,w) on find paths such that v and w are in the same S_{0j}. For $i \leq l \leq z$, let N_i be the number of edges (v,w) on find paths such that v and w are in different S_{kj}'s for $k < i$ and v and w are in the same S_{ij}. Let N_{z+1} be the number of edges (v,w) on find paths such that v and w are in different S_{ij}'s for all $0 \leq i \leq z$. The total length N of all the partial finds satisfies:

$$(12) \qquad N = \sum_{i=0}^{z+1} N_i$$

Lemma 2:

$$(13) \qquad N_{z+1} \leq \min(m,n)(a(z,n)-2) + m.$$

Proof:

There are only $a(z,n)$ sets S_{zj}. The number of edges (v,w) on any single find path such that v and w are in different S_{zj}'s is thus $a(z,n) - 1$ since the heights of vertices along the find path strictly increase. Since there are m finds, $N_{z+1} \leq m(a(z,n) - 1)$.

For any find, there is at most one edge (v,w) on the find path such that v and w are in different S_{zj}'s and no edge (v',w') on the find path satisfies $w' \overset{*}{\rightarrow} v$ and v' and w' are in different S_{zj}'s. This gives at most m edges total. For a fixed w, consider the edges (v,w) on find paths such that v and w are in different S_{zj}'s and (v,w) is not one of the m or less edges counted above. After a find containing such an edge, the father of w jumps to a new S_{zj}. This can happen at most a(z,n) - 2 times, until the father of w is in $S_{z,a(z,n)-1}$. Thus, $N_{z+1} \leq n(a(z,n)-2) + m$. Q.E.D.

Lemma 3:

(14) $N_0 \leq 2n + m$

$N_i \leq \frac{5}{8} n + m$ for $1 \leq i \leq z$. Q.E.D.

Proof:

Each time a given vertex w is in an edge (v,w) on a find path and (v,w) is not the last edge on the find path, the height of the father of w increases by at least one after the find. Thus, after w is in two such edges, its father must be in a different S_{0j}. Thus, $N_0 \leq 2n + m$.

For any i, $1 \leq i \leq z$, there are at most m edges (v,w), one for each find, such that v and w are in different S_{kj}'s for $k < i$, v and w are in the same S_{ij}, and no edge (v',w') on the same find path has $w' \overset{*}{\rightarrow} v$, v' and w' are in different S_{kj}'s for $k < i$, and v' and w' are in the same S_{ij}. Count these edges separately.

Consider any vertex w. Let x be the number of edges (v,w) on find paths such that v and w are in different S_{kj}'s for $k < i$, v and w are in the same S_{ij}, and (v,w) is not counted above. The father of w jumps in height after such an edge appears in a find. After x - 1 such edges appear in finds the height of f(w) is at least A(i-1,x-1). Thus, $A(i-1,x-1) < A(i,j+1)$. Since $A(i,j+1) = A(i-1,A(i,j))$, $x - 1 < A(i,j)$ and $x \leq A(i,j)$.

If we count such edges over all vertices w in sets S_{ij} using (10) and

add the count above, we get

$$(15)\quad N_i \le m + \sum_{j=2}^{\infty} \frac{2n}{2^{A(i,j)}} A(i,j) \le m + \sum_{j=2}^{\infty} \frac{2n}{2^{2^j}} 2^j \le \frac{5}{8} n + m.$$

(The sum starts at $j = 2$ because $S_{00} = S_{i0}$, $S_{01} = S_{i1}$ for all i.) Q.E.D.

By Lemmas 2 and 3,

$$(16)\quad N = \sum_{i=0}^{z+1} N_i \le 2n + 2m + (\frac{5}{8} n + m)z + \min(m,n)(a(z,n) - 2).$$

From (16) we have

$$(17)\quad t(m,n) \le k_2 \min_z (\max(m,n) \cdot z + \min(m,n) \cdot a(z,n))$$

for a suitable constant k_2, where $t(m,n)$ is the worst-case running time required for $n - 1$ unions and m finds.

We have

$$(18)\quad A(k,3) = A(k-1, A(k,2)) = A(k-1,4) = A(k-2, A(k-1,3)) \ge A(k-2,k-2)$$

for all $k \ge 2$ since $A(k-1,3) \ge k - 2$ for all $k \ge 1$.

Thus, $A(k,3)$ grows as fast as $A(k,k)$. Let

$$(19)\quad \alpha(n) = \min \{i \mid A(i,3) > \lfloor \log_2 n \rfloor\}.$$

Suppose we choose $z = \alpha(n)$. Since $A(z,3) > \lfloor \log_2 n \rfloor$, $a(z,n) \le 3$, and

$$(20)\quad t(m,n) \text{ is } O((m+n)\,\alpha(n)) \text{ for all } m \text{ and } n.$$

If m and n are not approximately equal, we can get tighter bounds from (17). For instance, if $m \ge n \cdot a(k,n)$ or $n \le m \cdot a(k,n)$ for some fixed constant k, (17) implies $t(m,n)$ is $O(m+n)$. Thus, the worst case occurs when m and n are approximately equal. The next section shows that the upper bound (20) is tight to within a constant factor when $m = n$.

A Lower Bound

We shall show that for any fixed length k, there is some $B(k)$ such that there are trees with $B(k)$ vertices on which we can perform a partial find of length k on each of a fixed fraction p of the vertices. It follows that $p \cdot k \cdot B(k) \leq t(p\ B(k), B(k))$ for all k, and the function $B(k)$ will give us a lower bound on the running time of the set union algorithm. This bound is independent of the union rule used by the algorithm.

For any $i \geq 0$, let K_i be the tree, all of whose leaves have depth k, such that the number of sons of any vertex of height $j \geq 1$ is 2^j. Figure 3 shows K_2. Let T_0 be the tree with a single vertex. Let T_1 be the tree of three vertices having a root and two sons of the root. For any $i \geq 2$, let T_i be the tree formed from $2^i + 1$ trees T_{i-1} by making the roots of 2^i of them, sons of the root of the last T_{i-1}.

Lemma 4:

(i) T_i contains K_i as a subtree;

(ii) $|T_i| \leq 2\,|K_i|$;

(iii) T_i may be constructed from single vertices using any union rule and at most one find on each vertex which is not a leaf or the root of the embedded K_i.

Figure 4 shows the construction of T_2.

Proof:

For $i \geq 2$, the embedded K_i in T_i consists of the root of T_i plus the embedded trees K_{i-1} in the 2^i trees T_{i-1} whose roots are sons of the root of T_i. Part (i) follows by induction on i. The fraction of vertices in T_i which are not in the embedded K_i is bounded by .

$$(21) \qquad \sum_{i=2}^{\infty} \frac{1}{2^i + 1} \leq \frac{1}{2} \quad ;$$

thus, (ii) holds.

We prove (iii) by induction on i; (iii) clearly holds for $i = 0, 1$. Suppose (iii) is true for $i - 1$. To construct T_i, construct $2^k + 1$ trees T_{i-1} in a way which satisfies (iii). Combine the trees T_{i-1} using any union rule. Then perform finds on the roots of all the trees T_{i-1} except the one which is the root of the new tree. Clearly, the resultant tree is T_i. Since no finds on the roots of the trees T_{i-1} were used in constructing them, T_i was constructed by using at most one find on each of its vertices and no finds on its root. Since the leaves of the embedded K_i are the

leaves of the trees K_{i-1} embedded in 2^i of the trees T_{i-1}, no finds were performed on the leaves of the embedded K_i.

Thus, (iii) holds for i. Q. E. D.

Now we concentrate on K_i trees. Let ℓ_i be the number of leaves in K_i. Then $\ell_0 = 1$ and $\ell_{i+1} = 2^{i+1}\ell_i$. Let n_i be the number of vertices in K_i. Then $n_0 = 1$ and $n_{i+1} = 2^{i+1} n_i + 1$. It is easy to show by induction that

$$\ell_i \leq 2^{i(i+1)/2} \tag{22}$$

$$n_i \leq 2\ell_i - 1$$

Let $K_i(s,h, \{T(v)\})$ denote the tree formed from K_i by deleting all vertices of height less than h and replacing each vertex v of (original) height h by the tree T(v), which contains s or fewer leaves all different from its root. We call $K_i(s, h, \{T(v)\})$ a substituted K_i tree; the leaves of the trees T(v) are the added leaves of $K_i(s,h,\{T(v)\})$. The original height of a vertex w in $K_i(s,h,\{T(v)\})$ is its height in the original K_i if $w \in K_i$, zero if w is a root of some T(v), and undefined if w is a non-root vertex of some T(v). Figure 5 shows a substituted K_2 tree. We will show that for any fixed k, s, and h, there is some B(k,s,h) such that in any $K_b(s,h,\{T(v)\})$ with $b \geq B(k,s,h)$ we can perform a find of length k on each of the added leaves.

Let the function $B(k,s,h)$ on non-negative integers be defined by:

$$(23)\qquad \begin{aligned} B(0,s,h) &= B(1,s,h) = h && \text{if } s \geq 1,\ h \geq 0. \\ B(k,1,h) &= B(k-1,\ 2^{h+1},h+1) && \text{if } k \geq 2,\ h \geq 0. \\ B(k,s,h) &= B(k-1,\ 2^{B(k,s-1,h)(B(k,s-1,h)+1)/2}, \\ &\qquad B(k,s-1,h)) && \text{if } k \geq 2,\ s \geq 2,\ h \geq 0. \end{aligned}$$

It is easy to show by double induction that $B(k,s,h)$ is defined for all integers $k \geq 0$, $s \geq 1$, $h \geq 0$.

Lemma 5:

For any length $k \geq 0$, for any spread $s \geq 1$, for any height $h \geq 0$, for any $b \geq B(k,s,h)$, let $K_b(s,h,\{T(v)\})$ be a substituted K_b tree. Then we can perform a find of length k on each of the added leaves.

Proof:

We prove the Lemma by double induction on k and s. Suppose $k \leq 1$ and s and h are arbitrary. $B(0,s,h) = B(1,s,h) = h$, so $b \geq h$. This just means that $K_b(s,h,\{T(v)\})$ is a non-empty tree. Each added leaf has a depth of one or more, and a find of length zero or one changes the position of no vertices, so the Lemma holds for $k \leq 1$, s and h arbitrary.

Suppose the Lemma holds for all $k' < k$ and arbitrary s' and h', where $k \geq 2$. We prove the Lemma for k, $s = 1$, h arbitrary. In $K_b(1, h, \{T(v)\})$ the fathers of all the added leaves are distinct. For any vertex w in $K_b(1, h, \{T(v)\})$ of original height $h + 1$, let

$T'(w)$ be the tree containing w and all its non-leaf descendants in $K_b(1, h, T(v))$. Each $T'(w)$ contains no more than 2^{h+1} leaves since $s = 1$ and w has 2^{h+1} sons. Since $b \geq B(k, 1, h) \geq h + 1$, $K_b(2^{h+1}, h + 1, \{T'(w)\})$ is a non-empty tree and is contained in $K_b(1, h, \{T(v)\})$. Since $b \geq B(k, 1, h) = B(k-1, 2^{h+1}, h+1)$, we can perform finds of length $k - 1$ on each of the added leaves in $K_b(2^{h+1}, h + 1, \{T'(w)\})$ by the induction hypothesis. Hence, we can perform finds of length $k - 1$ on all the fathers of added leaves in $K_b(1, h, \{T(v)\})$. Since each of these fathers is distinct, we could just as well perform finds of length k on each of the added leaves in $K_b(1, h, \{T(v)\})$, and the Lemma holds for k, $s = 1$, and arbitrary h.

Now suppose the Lemma holds for all k', s', h' such that $k' < k$, $s' \geq 1$, $h' \geq 0$, and for all k, s', h', such that $s' < s$, and $h' \geq 0$, where $k \geq 2$ and $s \geq 2$. We prove the Lemma for k, s, and arbitrary h. Clearly B increases in all its variables, so $b \geq B(k,s,h) \geq B(k, s-1, h)$. For each $T(v)$, let $T'(v)$ be a tree formed from $T(v)$ by deleting one of its leaves $\ell(v)$ and enough other vertices so that $T'(v)$ has one less leaf (not the root) than $T(v)$. For each vertex w in $K_b(s,h,\{T'(v)\})$, let $K_b(s,h,\{T'(v)\})(w)$ denote the subtree of $K_b(s,h,\{T'(v)\})$ consisting of w and all its descendants.

If w is any vertex of original height $B(k, s-1, h)$, $K_b(s,h,\{T'(v)\})(w) = K_{B(k,s-1,h)}(s-1, h, \{T'(v) | w \overset{*}{\rightarrow} v \text{ in } K_b(s,h,\{T'(v)\})\})$. By the induction hypothesis we can perform a find of length k on every added leaf in $K_b(s,h,\{T'(v)\})(w)$. Hence we can perform a find of length k on every added leaf except the $\ell(v)$'s in $K_b(s,h,\{T(v)\})$, resulting

in a tree $K_b(s'', B(k, s-1, h), \{T''(x)\})$ for suitable $T''(x)$'s and a suitable s''.

Given any $T''(x)$, let

$$T'''(x) = \{y \mid \exists\ \ell(v) \in T''(x) \text{ such that } y \overset{*}{\to} f(\ell(v)) \text{ in } T''(x)\}$$

Then $T'''(x)$ contains at most $s''' = 2^{B(k,s-1,h)}B(k,s-1,h) + 1/2$ leaves by (22). Furthermore

$K_b(s''', B(k, s-1, h), \{T'''(x)\})$ is a subtree of

$K_a(s'', B(k, s-1, h), \{T''(x)\})$.

We can perform finds of length $k-1$ on each of the added leaves of $K_b(s''', B(k,s-1,h), \{T'''(x)\})$ by the induction hypothesis, since $b \geq B(k,s,h) = B(k-1, s''', B(k,s-1,h))$. Hence, we can perform finds of length $k - 1$ on all the fathers of $\ell(v)$'s in $K_b(s'', B(k, s-1, h), \{T''(x)\})$. Since each of these fathers is distinct, we could just as well perform finds of length k on all the $\ell(v)$'s.

Hence, we can perform finds of length k on all the added leaves of $K_b(s, h, \{T(v)\})$, the Lemma holds for k, s, and arbitrary h, and by double induction the Lemma holds in general. Q.E.D.

It is easy to prove by induction that

$$A(3, x+2) \geq 2^{x+1} + x + 3 \qquad \text{if } x \geq 1, \text{ and} \tag{24}$$

$$A(3, x-1) \geq 2^{x(x+1)/2} + x + 2 \qquad \text{if } x \geq A(4,4). \tag{25}$$

Lemma 6:

$$A(2k, s+h+2) \geq B(k,s,h) \qquad \text{for all } k \geq 0,\ h \geq 1,\ s \geq 1. \tag{26}$$

Proof:

We prove the Lemma by double induction. Suppose $k \leq 1$, and $s, h \geq 1$.

(27) $B(0,s,h) = B(1,s,h) = h \leq 2(s+h+2) = A(0,s+h+2) \leq A(1, s+h+2)$.

Thus the Lemma holds in this case.

Suppose the Lemma holds for $k' < k$ and arbitrary s', $h' \geq 1$, where $k \geq 2$. We prove the Lemma for k, $s = 1$, $h \geq 1$.

(28) $B(k,1,h) = B(k-1, 2^{h+1}, h + 1)$

$\leq A(2k-2, 2^{h+1} + h + 3)$ by the induction hypothesis

$\leq A(2k-2, A(3, h + 2))$ by (24)

$\leq A(2k-2, A(2k-1, h + 2))$ since $k \geq 2$

$= A(2k-1, h + 3) \leq A(2k, (h+1) + 2)$.

Suppose the Lemma holds for all k', s', h' such that $k' < k$, $s' \geq 1$, $h' \geq 1$, and for all k, s', h' such that $s' < s$, $h' \geq 1$, where $k \geq 2$ and $s \geq 2$. We prove the Lemma for k, s, and $h \geq 1$.

(29) $B(k,s,h) = B(k-1, 2^{B(k,s-1,h)(B(k,s-1,h)+1)/2}, B(k,s-1,h))$

$\leq A(2k-2, 2^{B(k,s-1,h)(B(k,s-1,h)+1)/2} + B(k,s-1,h)+2)$

by the induction hypothesis

$\leq A(2k-2, 2^{A(2k,s+h+1)(A(2k,s+h+1)/2} + A(2k,s+h+1)+2)$

by the induction hypothesis

$\leq$ A(2k-2, A(3, A(2k, s+h+1)-1)) by (25) since $k \geq 2$, and $s \geq 2$

$\leq$ A(2k-2, A(2k-1, A(2k, s+h+1)-1))

= A(2k-1, A(2k, s+h+1))

= A(2k, s + h + 2).

Thus the Lemma holds in this case, and the Lemma holds in general by double induction. Q.E.D.

Lemma 6 gives:

(30) $$B(k, 2, 1) \leq A(2k, 5)$$

and

(31) $$4 \cdot 2^{B(k,2,1)(B(k,2,1)+1)/2} \leq A(2k + 3,3) \quad \text{if } k \geq 1.$$

Lemma 7:

(32) $k_1 \, n \, \alpha(n) \leq t(n,n)$ for some constant k_1.

Proof:

Given n, let $k = \left\lfloor \frac{\alpha(n)}{2} \right\rfloor - 2$. Assume $k \geq 1$.

(33) $$4 \cdot 2^{B(k,2,1)(B(k,2,1)+1)/2} \leq A(2k + 3, 3) \leq \lfloor \log_2 n \rfloor .$$

Starting with n vertices, by Lemma 4 we can construct trees $T_{B(k,2,1)}$ using any union rule and some finds. None of the finds are on leaves of the embedded trees $K_{B(k,2,1)}$. Some vertices may be left over, but we can use up at least $n - \lfloor \log_2 n \rfloor$ of the vertices. By Lemma 5 we can then

perform a find of length k on each of the leaves of the embedded trees $K_{B(k,2,1)}$. If necessary, we can perform additional finds to bring the total up to n.

The length of all the finds is at least $\frac{k}{4}(n - \lfloor \log_2 n \rfloor)$, since at least half the vertices of a tree $T_{B(k,2,1)}$ are in a tree $K_{B(k,2,1)}$, and at least half the vertices of $K_{B(k,2,1)}$ are leaves, by Lemma 4 and (22). Thus $k_1 n \alpha(n) \leq t(n,n)$ for some constant k_1.

Q.E.D.

Conclusions

We have analyzed a known algorithm for computing disjoint set unions, showing that if $t(m,n)$ is the worst-case running time required by $n - 1$ unions and m finds, then $t(m,n)$ is $O(\min(\max(m,n) \cdot \ + \min(m,n) \cdot a(z,n)))$, where a is very slowly growing. If $m = n$, the running time is $O(n \alpha(n))$, and this bound is tight to within a constant factor. This is probably the first, and maybe the only example of a simple algorithm with a very complicated running time. It is an open problem whether there is a linear-time algorithm for the set union problem. I conjecture that there is no linear-time algorithm, and that the algorithm considered here is the fastest possible to within a constant factor.

REFERENCES:

[1] Aho, A.V., J.E. Hopcroft, and J.D. Ullman, "On computing least common ancestors in trees", Proceedings of the 5th Annual ACM Symposium on Theory of Computing, Austin, Texas (1973), 253-265.

[2] Arden, B.W., B.A. Galler, and R.M. Graham, "An algorithm for equivalence declarations", Comm. ACM 4 (July, 1961), 310-314.

[3] Fischer, M.J., "Efficiency of equivalence algorithms", in Complexity of Computer Computations, Miller, R.E., and J.W. Thatcher, (eds.), Plenum Press, New York, 1972, 153-168.

[4] Galler, B.A., and M.J. Fischer, "An improved equivalence algorithm", Comm. ACM 7 (May, 1974), 301-303.

[5] Hopcroft, J. and J.D. Ullman, "Set-merging algorithms", SIAM J. Comput. 2 (December, 1973), 294-303.

[6] Hopcroft, J., private communication.

[7] Kerschenbaum, A., and R. Van Slyke, "Computing minimum spanning trees efficiently", Proceedings of the 25th Annual Conference of the ACM (197), 518-527.

[8] Knuth, D.E., The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Addison-Wesley, Redding, Mass., 1969, 353-355.

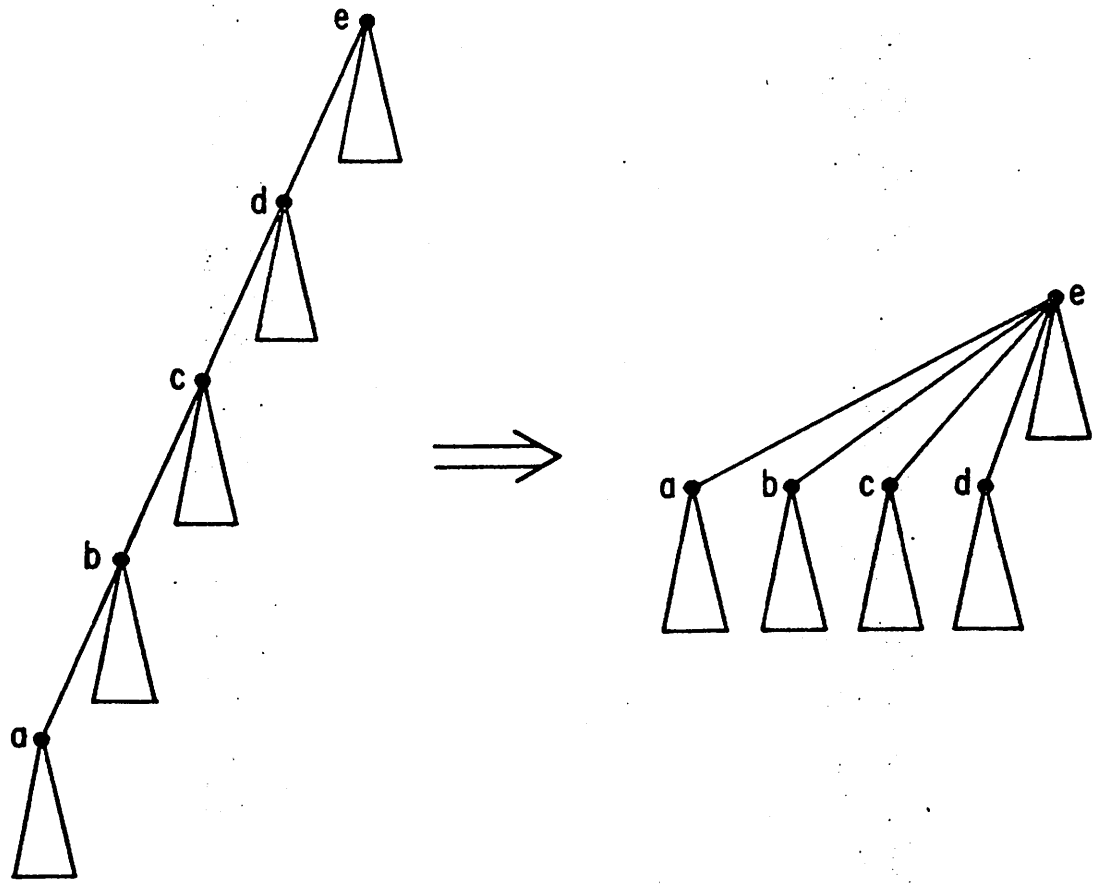
[9] Knuth, D.E.,"Some combinatorial research problems with a computer science flavor", notes by L. Guibas and D. Plaisted from an informal seminar (January, 1972).

[10] Paterson, M., unpublished report, University of Warwick, Coventry, Great Britain.

[11] Stearns, R.E., and D.J. Rosenkrantz, "Table machine simulation", 10th Annual SWAT Conference Proceedings (1969), 118-128.

[12] Tarjan, R., "Testing flow graph reducibility", Proceedings of the 5th Annual ACM Symposium on Theory of Computing, Austin, Texas (1973), 96-107.

[13] Tarjan, R. "Finding dominators in directed graphs", SIAM J. Comput., to appear.



T T'

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

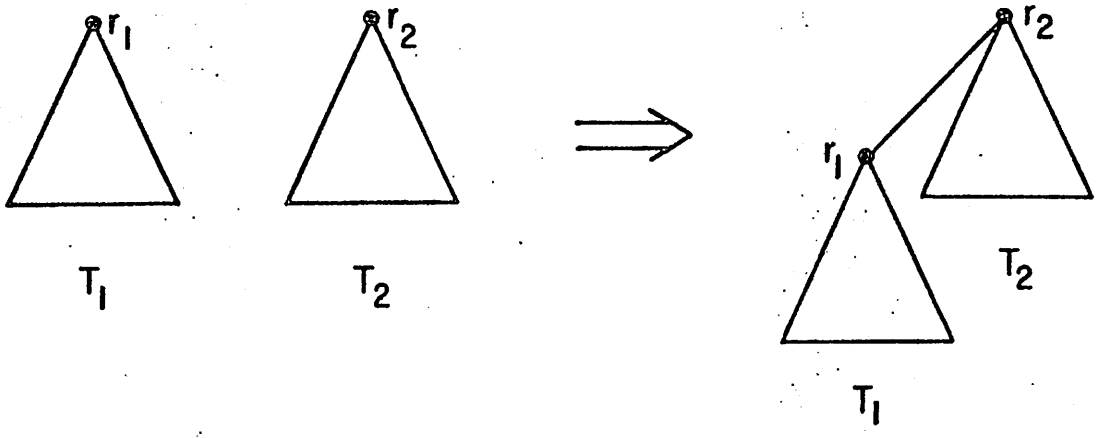


Figure 2: Union of two trees. Root r_1 of T_1 has a descendants; root r_2 of T_2 has b descendants. Root of new tree has a + b descendants.

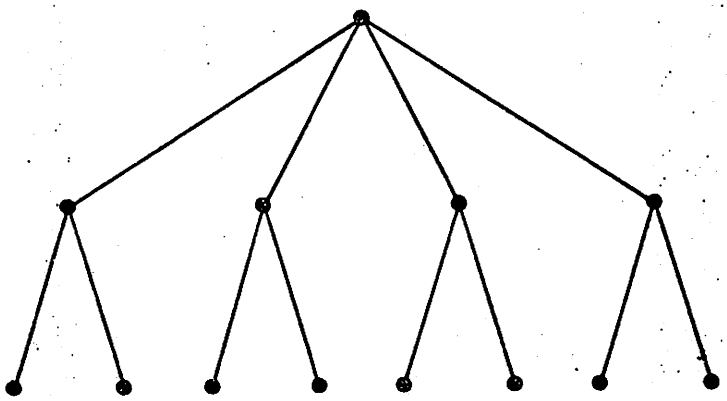


Figure 3: A K_2 tree.

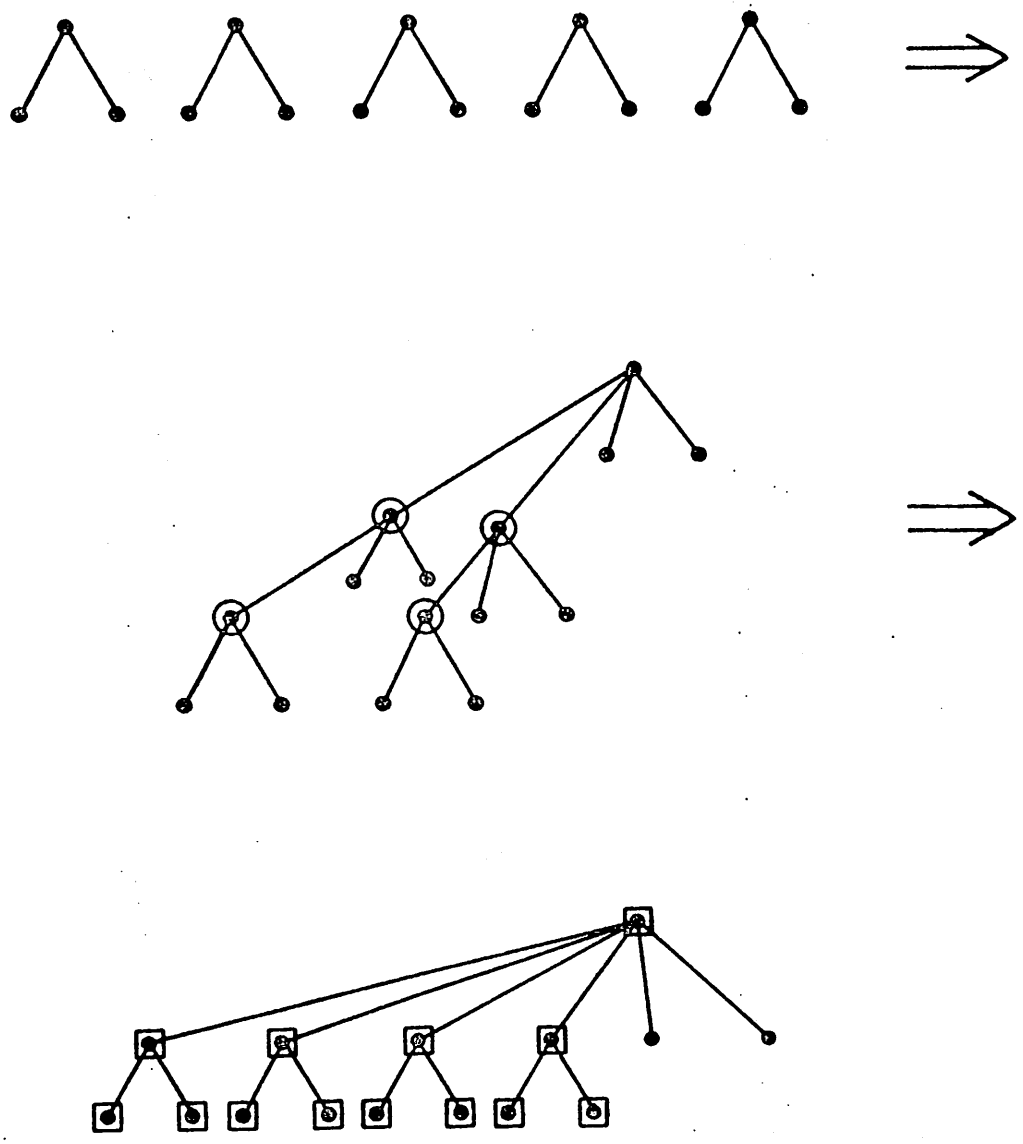
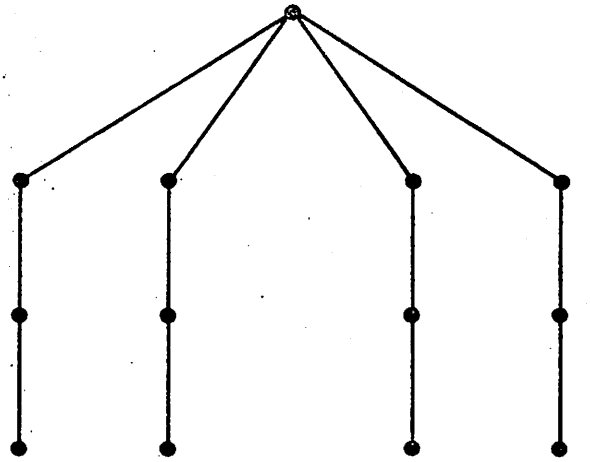


Figure 4: Forming at T_2 tree from T_1 trees. Four finds, on the circled vertices, are used. Vertices of the embedded K_2 tree are boxed.



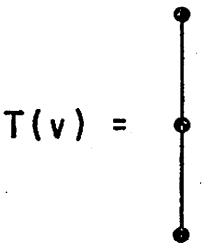


Figure 5: $K_2(1, 1, \{T(v)\})$, with all $T(v)$'s the same.