A Potential-Based Amortized Analysis of the Union-Find Data Structure

Gregory C. Harfst* Edward M. Reingold†

August 3, 2000

Abstract. We present a simple, new potential-based amortized analysis for the standard union-find data structure.

Key words. Amortized analysis, potential functions, disjoint sets, Tarjan's multiple partitioning, Ackermann's function

AMS(MOS) subject classifications. 68P05, 68Q25, 68R99

The standard union-find data structure is simple to describe, yet has a complex analysis. Traditionally the amortized analysis of this structure is done using the accounting method. We present a simple, new potential-based analysis covering several of the most interesting results.

1 Disjoint Set Data Structure

As usual, we implement the disjoint set data structure as a collection of trees (for a fuller discussion see [1], [2], [3], or [4]). The root of each tree is the canonical element for all the nodes contained within its tree. The data structure supports the operations *makeset*(x), *link*(x, y) and *find*(x).

The operation *makeset*(x) takes an element x and makes it a singleton node with parent$(x) = x$. This makes x the canonical element for the set $\{x\}$ (see Figure 1).

The operation *link*(x, y) combines the tree rooted at x with the tree rooted at y using a heuristic called "union by rank" (see Figure 2). We initially define rank(x) to be zero in the *makeset* operation. When we *link* x with y we make the node with the smaller rank the

*Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, Illinois 61801, USA. Email: gharfst@acm.org

†Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 W. Springfield Avenue, Urbana, Illinois 61801, USA. Supported in part by NSF grant CCR-95-30297. Email: reingold@cs.uiuc.edu

new child of the node with the larger rank. When $\text{rank}(x) = \text{rank}(y)$, we arbitrarily choose one node as the new root and increase its rank by one.

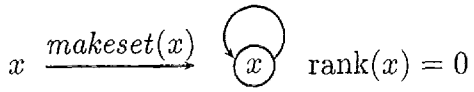

Figure 1: The *makeset* operation creates a singleton node in the data structure using the given element x. The node's rank is initialized to zero.

So after any operation $link(x, y)$ makes x the new parent of y, we always have $\text{rank}(x) > \text{rank}(y)$. This heuristic guarantees that the maximum rank of any node is $\lg n$. Also note that a node's rank is a monotonically increasing function of time. Further, its rank can increase until it becomes the child of another node; after that its rank remains fixed, although the rank of its parent can increase.

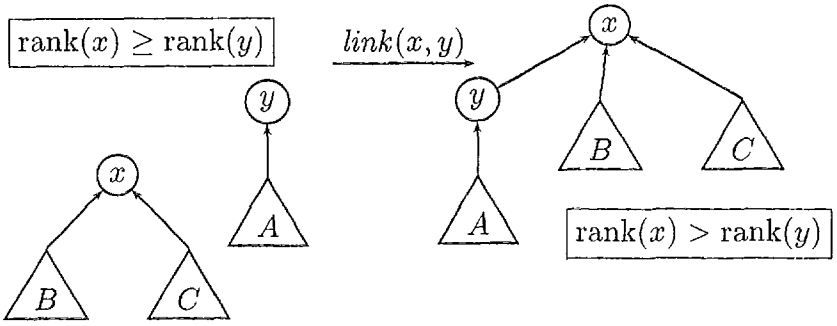

Figure 2: The *link* operation, with triangles denoting subtrees. Before the *link*, node x has a rank that is greater than or equal to that of node y. If $\text{rank}(x) = \text{rank}(y)$ we increase $\text{rank}(x)$ by one. We then make y point to x. This guarantees that $\text{rank}(x) > \text{rank}(y)$ after the *link*.

The operation $find(x)$ follows parent points from a node x up to the root, which designates the canonical element of that set. We then perform a heuristic called path compression (see Figure 3). In doing so we make every node along the find path point directly to the root, thus compressing the find path. $find(x)$ then returns the root as the canonical element of the set.

Note that path compression has no effect on the asymptotic cost of a single operation. We will soon see the impressive effect it has on the amortized cost of a sequence of operations. The path compression also makes the analysis difficult because of the way it changes the tree's structure.

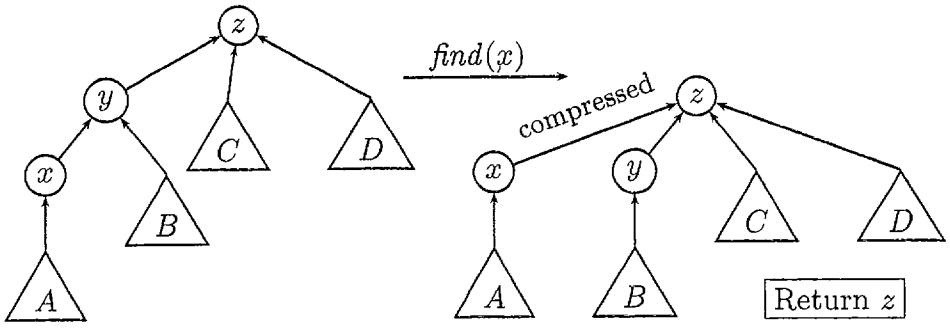

Figure 3: The *find*(x) operation changes the parent pointers of every node on the find path to point directly to the root. Triangles denote subtrees.

2 Analysis

This disjoint set data structure, with the heuristics of union by rank and path compression, has a remarkable amortized behavior. We seek an upper bound on the amortized cost of a sequence of m intermixed *link* and *find* operations on a structure containing n elements. Our heuristic of union by rank gives us a bound of $O(\lg n)$ per operation. Hopcroft and Ullman obtained a bound of $O(m \lg^* n)$ on the structure implemented with both heuristics [2]. Tarjan obtained the actual worst-case bound of $O(m\,\alpha(m,n))$, where $\alpha(m,n)$ represents the functional inverse of Ackermann's function [4]; he achieved this using the accounting method of amortized analysis in conjunction with his "Multiple Partitioning Method."

Here we give a simple analysis of the $O(m \lg^* n)$ and $O(m\,\alpha(m,n))$ bounds using the potential function method of amortized analysis. As shown in Equation (1), we obtain the amortized cost of an operation on a data structure D_i by adding the actual cost of an operation to the change in potential resulting from that operation:

$$\hat{c}_i = c_i + \Phi(D_{i+1}) - \Phi(D_i), \tag{1}$$

where D_i is the data structure after i operations. $\Phi(D_0) = 0$, and $\Phi(D_i) \geq 0$ for all $i \geq 0$. As a result, the amortized cost provides us with an upper bound on the cost of a sequence of m operations on the data structure D. More information on the potential-based method of amortized analysis can be found in [1, chapter 18] or [6].

2.1 The Hopcroft-Ullman Bound

Recall that the $\lg^*$ function is defined in terms of the $\lg^{(i)}$ function, where

$$\lg^{(i)} n = \begin{cases} n & \text{if } i = 0, \\ \lg\left(\lg^{(i-1)} n\right) & \text{if } i > 0 \text{ and } \lg^{(i-1)} n > 0, \\ \text{undefined} & \text{if } i > 0 \text{ and } \lg^{(i-1)} n \leq 0 \text{ or } \lg^{(i-1)} n \text{ is undefined.} \end{cases}$$

Then, we let

$$\lg^* n = \min\{i \geq 0 : \lg^{(i)} n \leq 1\}.$$

We partition the range $[0 \ldots \lg n]$ of possible ranks for a node x into blocks by placing $\text{rank}(x)$ in block $\lg^*(\text{rank}(x))$. We can now define the potential of a single node as

$$\phi(x) = \begin{cases} 3\,\text{rank}(x) & \text{if } x \text{ is a root,} \\ 3\,\text{rank}(x) & \text{else, if } \lfloor \lg \text{rank}(x) \rfloor \\ \quad - \text{rank}(\text{parent}(x)) & \quad = \lfloor \lg \text{rank}(\text{parent}(x)) \rfloor \\ \text{rank}(x) + \lfloor \lg \text{rank}(x) \rfloor & \text{else, if } \text{block}(\text{rank}(\text{parent}(x))) \\ \quad - \lfloor \lg \text{rank}(\text{parent}(x)) \rfloor & \quad = \text{block}(\text{rank}(x)), \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The potential of the whole structure is simply

$$\Phi = \sum_{x \in \text{node}} \phi(x).$$

A few additional comments will provide insight on the intuition behind the potential function. First, the potential of a root node is a constant multiple of its rank, which never exceeds $\lg(n)$. For non-root nodes we consider the magnitude of the difference between $\text{rank}(x)$ and $\text{rank}(\text{parent}(x))$. In the second case, when $\lfloor \lg \text{rank}(x) \rfloor = \lfloor \lg \text{rank}(\text{parent}(x)) \rfloor$, the potential is directly based on this difference, which is not too large. As $\text{rank}(\text{parent}(x)) - \text{rank}(x)$ grows, the third case governs the potential. Here $\text{rank}(x)$ and $\text{rank}(\text{parent}(x))$ both fall in the same block, and hence the magnitude of

$$\lg \text{rank}(\text{parent}(x)) - \lg \text{rank}(x)$$

remains reasonably bounded. In the final case, the rank of the node and its parent are very far apart. This happens only after significant path compression work has been done on the structure and as a result we can assign a zero potential to the node. Finally, note that $\phi(x) \geq 0$ in all cases.

Now Equation (1) can be applied to obtain the amortized cost of each *makeset*, *link* and *find* operation. For the operation *makeset*(x) we have an actual cost of $O(1)$. The node's rank is initialized to zero, and hence

$$\phi(x) = 3\,\text{rank}(x) = 0,$$

so there is no change in potential and hence the amortized cost of *makeset*(x) is $O(1)$.

Consider a *find* operation that starts at a node x_0 and follows the path $x_0, x_1 = \text{parent}(x_0), \cdots, x_l = \text{parent}(x_{l-1})$, where $\text{parent}(x_l) = x_l$. The actual cost of the *find* operation is $O(l)$. Of course the path compression does not effect x_l or x_{l-1}. There are at most $O(\lg^* n)$ nodes whose

ranks are the last in their block on the find path. In other words, there are at most $O(\lg^* n)$ nodes $\hat{x}$ where

$$\text{block}(\text{rank}(\hat{x})) \neq \text{block}(\text{rank}(\text{parent}(\hat{x}))).$$

Now examine what happens to one of the other nodes x_i on the find path, where

$$\text{block}(\text{rank}(x_i)) = \text{block}(\text{rank}(\text{parent}(x_i))), i < l - 1.$$

The potential must fall into either the second or third case of Equation 2. If $\lfloor \lg \text{rank}(x) \rfloor = \lfloor \lg \text{rank}(\text{parent}(x)) \rfloor$ both before and after the path compression then

$$\phi(x) = 3\,\text{rank}(x) - \text{rank}(\text{parent}(x))$$

decreases because $\text{rank}(\text{parent}(x))$ increases. This follows from the monotonicity of the ranks up a find path. Instead, if $\lfloor \lg \text{rank}(x) \rfloor \neq \lfloor \lg \text{rank}(\text{parent}(x)) \rfloor$ both before and after the *find*, then $\phi(x)$ also decreases. The third case is when

$$\phi(x) = 3\,\text{rank}(x) - \text{rank}(\text{parent}(x))$$

before the *find*'s path compression increases $\text{rank}(\text{parent}(x))$ to $\text{rank}(\text{parent}'(x))$ so that $\lfloor \lg \text{rank}(x) \rfloor \neq \lfloor \lg \text{rank}(\text{parent}'(x)) \rfloor$. The condition

$$\lfloor \lg \text{rank}(x) \rfloor = \lfloor \lg \text{rank}(\text{parent}(x)) \rfloor$$

implies that

$$\text{rank}(\text{parent}(x)) < 2\,\text{rank}(x). \tag{3}$$

Also, $\text{rank}(x) > \text{rank}(\text{parent}(x))$ necessitates

$$\lfloor \lg \text{rank}(x) \rfloor \geq \lfloor \lg \text{rank}(\text{parent}(x)) \rfloor. \tag{4}$$

Substituting Equations 3 and 4 into the potential function we obtain

$$\begin{aligned} \phi(x) &= 3\,\text{rank}(x) - \text{rank}(\text{parent}(x)) \\ &> \text{rank}(x) \\ &\geq \text{rank}(x) + \lfloor \lg \text{rank}(x) \rfloor - \lfloor \lg \text{rank}(\text{parent}(x)) \rfloor. \end{aligned}$$

As a result, the potential of every node whose rank is not the last of its block must decrease by at least one. Similarly, the potential of any "last node in block" node cannot increase. This gives us an amortized cost of $O(l) - (l - 2\lg^* n) = O(\lg^* n)$ for the *find* operation.

The *link*(x, y) operation can change the potential of the new child. The actual cost of this operation is $O(1)$. Without loss of generality, suppose the *link* makes x the new child of y. Then $\phi(x)$ changes from $3\,\text{rank}(x)$ to one of the other three cases in Equation 2. By an analysis similar to that done for the *find* operation, $\phi(x)$ must decrease. The *link* can also cause $\text{rank}(y)$ to increase by one; this would increase its potential by $O(1)$. The change in rank might also cause a change in potential in the other children of y. The discussion about

the change in potential via path compression in the *find* operation implies that the potential of these nodes cannot increase by more than $\lg^* n$. Hence the change in potential resulting from a *link* operation is at most $O(\lg^* n)$ and thus the amortized cost is $O(\lg^* n)$.

Clearly then, after rescaling the units of potential, the amortized cost of a sequence of m operations on a structure with n elements is $O(m \lg^* n)$. In the next section we improve this analysis to obtain Tarjan's actual worst case bound by refining our partitioning of the ranks and making corresponding changes to our potential function. We base the new partitioning upon Ackermann's function.

2.2 Ackermann's Function and Tarjan's Partitioning

Ackermann's function is defined as:

$$\begin{aligned} A(1,j) &= 2^j && \text{for } j \geq 1, \\ A(i,1) &= A(i-1,2) && \text{for } i \geq 2, \\ A(i,j) &= A(i-1, A(i,j-1)) && \text{for } i,j \geq 2. \end{aligned}$$

We then define the inverse function $\alpha(m,n)$ for $m \geq n \geq 1$ by

$$\alpha(m,n) = \min\{i \geq 1 | A(i, \lfloor m/n \rfloor) > \lg n\}.$$

This function grows *extremely* slowly. For all practical purposes it is less than or equal to 4. Tarjan provides an insightful discussion of the properties of both Ackermann's function and its inverse α in his analysis [5].

Define a partitioning function $B(i,j)$ using Ackermann's function $A(i,j)$ as:

$$\begin{aligned} B(0,j) &= j && \text{for } j \geq 0, \\ B(i,0) &= 0 && \text{for } i \in [1 \ldots \alpha(m,n)+1], \\ B(i,j) &= A(i,j) && \text{for } i \in [1 \ldots \alpha(m,n)], j \geq 1, \\ B(\alpha(m,n)+1, 1) &= A(\alpha(m,n), \lfloor m/n \rfloor). \end{aligned}$$

This allows us to partition $[0 \ldots \lg n]$ into blocks for each level $i \in [0 \ldots \alpha(m,n)+1]$. We define these blocks as

$$\text{block}(i,j) = [B(i,j) \ldots B(i,j+1) - 1] \bigcap [0 \ldots \lg n] \text{ for } j \geq 0.$$

Figure 4 shows a partitioning for levels 1 through 3 with $n = 2^{16}$. Note that level 1 is the partitioning we defined in our analysis of the $O(m \lg^* n)$ time bound. As the level increases, the partitioning of the spectrum of ranks becomes coarser. As an indication of this coarsening we define b_{ij} to be the number of level $-(i-1)$ blocks whose intersection with block(i,j) is nonempty (see Figure 4). As Figure 4 shows, $b_{2,2} = 12$.

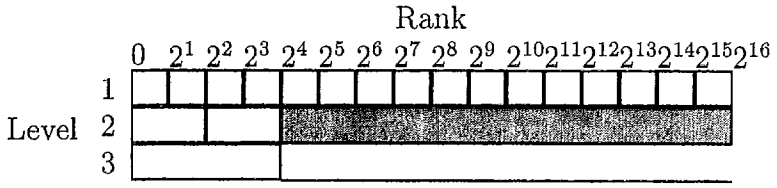

Figure 4: Here the partitioning is drawn on a logarithmic scale. Level 0 is not shown. The shaded block is block(2, 2). Note that $b_{2,2} = 12$. This figure also illustrates how explosively $A(i, j)$ grows and how slow $\alpha(m, n)$ grows.

Define the level of a node x to be the minimum level i such that rank(x) and rank(parent(x)) are in a common block of the level i partition. So if rank(x) = 2^2 and rank(parent(x)) = 2^4-1 then level(x) = 2.

Finally, if level(x) = i and rank(x) $\in$ block($i-1, j$), define $\beta(x)$ as:

$$\beta(x) = \begin{cases} 0 & \text{if level}(x) = 0, \\ j - \sum_{\hat{j}=0}^{k-1} b_{i\hat{j}} \mid \text{rank}(x) \in \text{block}(i,k) & \text{if level}(x) \geq 1. \end{cases}$$

It is best to think of $\beta(x)$ as the "block number" of

$$\text{rank(parent}(x)\text{) in level} -(i-1).$$

So in Figure 4, if rank(x) = 2^4 and rank(parent(x)) = 2^{10}, then $\beta(x) = 7$.

2.3 Potential function

Consider a disjoint set data structure containing n elements after m operations. Since we count the *makeset*(x) as an operation, we always have $m \geq n$. We define the potential function of a node x as:

$$\phi(x) = \begin{cases} (\alpha(m,n)+1)\,\text{rank}(x) & \text{if } x \text{ is a root or rank(x)} = 0, \\ (\alpha(m,n)+1-\text{level}(x))\,\text{rank}(x) - \beta(x) & \text{otherwise.} \end{cases}$$

As before,

$$\Phi = \sum_{x \in \text{node}} \phi(x).$$

We now prove Tarjan's theorem [4] using our potential function.

Theorem *A sequence of m intermixed makeset, link and find operations on a disjoint set data structure containing n elements, implemented with union by rank and path compression, has an amortized running time of $O(m\,\alpha(m,n))$.*

The operation *makeset*(x) adds a node x to the disjoint set with $\text{rank}(x) = 0$. As a result, $\phi(x) = 0$. Since the actual cost of the operation is $O(1)$, and the change in potential is zero, we have an amortized cost of $O(1)$ for *makeset*(x).

As in the $O(m \lg^* n)$ analysis, consider a *find* that starts at a node x_0 and follows the path $x_0, x_1 = \text{parent}(x_0), \cdots, x_l = \text{parent}(x_{l-1})$, where $\text{parent}(x_l) = x_l$. The actual cost of the *find* operation is $O(l)$. Of the l nodes on the find path, some will be the last node of their particular level on that path. Since the maximum level is $\alpha(m,n) + 1$, at most $\alpha(m,n) + 1$ nodes are the last nodes of their level on the find path. In addition, we may have $\text{rank}(x_0) = 0$, in which case $\phi(x_0)$ will always be zero. Further, the *find* does not effect node x_{l-1}.

Consider what happens to the potential of a typical node x in $\text{block}(i,j)$ of the remaining $l - (\alpha(m,n) + 3)$ nodes. The path compression will cause $\text{parent}(x)$ to change and hence $\text{rank}(\text{parent}(x))$ to increase. This change may cause a corresponding change in $\text{level}(x)$ or $\beta(x)$. Suppose $\text{level}(x)$ does not change. Then $\beta(x)$ must increase by at least one. To see this, observe that since x is not the last node with $\text{level}(x) = i$, we must have $\text{rank}(x_l)$ in a larger level $-(i-1)$ block than $\text{rank}(\text{parent}(x))$. Looking to our potential function we see that $\phi(x)$ must decrease in this case.

Suppose that $\text{level}(x)$ does increase. Then $\beta(x)$ may decrease, increase, or remain unchanged. If $\beta(x)$ increases or does not change, then the potential clearly decreases. Now suppose $\beta(x)$ decreases. In the worst case $\beta(x)$ decreases from $b_{i,j} - 1$ to 1. So in order for $\phi(x)$ to decrease we must have $\text{rank}(x) \geq b_{ij}$. Note that $b_{ij} \leq A(i,j)$ since

$$\text{block}(i,j) \subseteq [A(i,j) \ldots A(i,j+1) - 1] \subseteq [0 \ldots A(i-1, A(i,j)) - 1].$$

This follows directly from the definition of $A(i,j)$. Further,

$$A(i,j) \leq \text{rank}(x) \leq A(i,j+1).$$

This follows directly from the definition of $\text{block}(i,j)$ and $\text{level}(x)$. As a result, we do have $b_{ij} \leq \text{rank}(x)$ and $\phi(x)$ must decrease.

Therefore we are guaranteed that the potential of $l - (\alpha(m,n) + 3)$ nodes decreases by at least one. The argument above also guarantees that the potential of the remaining $(\alpha(m,n) + 3)$ cannot increase.

Plugging into Equation (1) we get an amortized cost of at most

$$\hat{c} = l - (l - (\alpha(m,n) + 3)) = \alpha(m,n) + 3.$$

Therefore the amortized cost of a single *find* operation is $O(\alpha(m,n))$.

The *link* operation only involves one pointer operation to make the new child node point to the root. As a result, the actual cost is $O(1)$. When the rank of x and y are different before the *link*, the new child is the only node that changes potential. Since its level must increase from 0 to at least 1 and its β value must also increase, its potential may only decrease.

When $\text{rank}(x) = \text{rank}(y)$ before the *link* the root and its other children may also change potential. The rank of the root will increase by one, so its potential increases by $\alpha(m,n)$. This increase in rank may increase the levels of its children. The relationship between level and β that we discussed in our analysis of the *find* operation implies that their potential cannot increase. Further, since $|\text{rank}(x) - \text{rank}(y)| = 1$, the sum of their changes in potential cannot be greater than a constant. The potential for all the other nodes remains unchanged. Therefore, Equation (1) gives us an amortized cost of $O(\alpha(m,n))$ for a single union operation.

The above argument implies that a sequence of m intermixed *link* and *find* operations on a disjoint set data structure containing n nodes will have a worst case amortized cost of $O(m\,\alpha(m,n))$. As with the Hopcroft-Ullman bound, the units of potential must be rescaled appropriately.

2.4 Alternate Path Compression Heuristics

The heuristic of total path compression just discussed has the aesthetic disadvantage of requiring two passes over the find path. Tarjan and van Leeuwen [8] present two one-pass variants of the path compression heuristic. These are path *splitting* and path *halving*. Splitting makes every node along the path point to its grandparent. Halving makes every other node along the path point to its grandparent.

These heuristics, combined with union by rank, maintain the bound of $O(m\,\alpha(m,n))$. To see this we will redefine the potential of a node x to be $2\phi(x)$. It is easy to see that this has no effect on the amortized cost of the *makeset* or *link* operations.

With the path splitting heuristic the only nodes whose potential does not decrease are those whose level and β values remain unchanged by the *find*. For $\text{level}(x)$ and $\beta(x)$ not to change we must have

$$\text{level}(\text{rank}(x)) > \text{level}(\text{rank}(\text{parent}(x)))$$

before the *find*. Let $y_0, y_1, \ldots, y_j$ be a subsequence of the nodes $x_0, x_1, \ldots, x_l$ consisting of the nodes x_i such that $\text{level}(\text{rank}(x_i)) > \text{level}(\text{rank}(\text{parent}(x_i)))$. There are at most $\alpha(m,n)$ nodes y_i such that $\text{level}(\text{rank}(y_i)) > \text{level}(\text{rank}(y_h))$ for all $h > i$. Let $\hat{\alpha} \leq \alpha(m,n)$ be the number of these "large level" y's in a particular *find*. The fluctuations in levels among the remaining $j - \hat{\alpha}$ nodes guarantee the existence of nodes x_i such that

$$\sum_{x_i \neq y_k, \forall i,k} (\text{level}(\text{rank}(x_{i+1})) - \text{level}(\text{rank}(x_i))) \geq j - \hat{\alpha}.$$

In addition, the potential of any of these x_i not included in the y_i must decrease by at least 2. Therefore the decrease in potential for a single *find* operation with a find path of length l is at least $(l - \alpha(m, n))$. So clearly the amortized cost is still $O(\alpha(m, n))$.

To analyze the implementation with path halving we must redefine level(x) to be the minimum level i such that rank(x) and rank(parent(parent(x))) are in a common block of the level i partition. Again, this does not effect the analysis of *makeset* or *link*. The analysis of the *find* operation becomes identical to that of *find* with path splitting. Hence the amortized cost of a single *find* is once again $O(\alpha(m, n))$.

3 Additional Comments

Kozen [3] provides an excellent analysis of the disjoint set data structure without explicitly using Tarjan's Partitioning method. Tarjan [7] recently developed a potential-based proof of the $O(m\,\alpha(m, n))$ time bound that is based on Kozen's analysis.

References

[1] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, 1990.

[2] John E. Hopcroft and Jeffrey D. Ullman. Set merging algorithms. *SIAM J. Comput.*, 2:294–303, 1973.

[3] Dexter C. Kozen. *The Design and Analysis of Algorithms.* Springer-Verlog, New York, 1990.

[4] Robert E. Tarjan. Efficiency of a good but not linear set union algorithms. *J. Assoc. Comput. Mach.*, 22:215–225, 1975.

[5] Robert E. Tarjan. Data structures and network algorithms. In *CBMS: Conference Board of the Mathematical Sciences, Regional Conference Series*, pages 23–31, 1983.

[6] Robert E. Tarjan. Amortized computational complexity. *SIAM J. Alg. Disc. Meth.*, 6:306–318, 1985.

[7] Robert E. Tarjan, February 1999. Personal communication.

[8] Robert E. Tarjan and Jan van Leeuwen. Worst-case analysis of set union algorithms. *J. Assoc. Comput. Mach.*, 31:245–281, 1984.