LINEAR EXPECTED TIME OF A SIMPLE UNION-FIND ALGORITHM*

Jon DOYLE and Ronald L. RIVEST
Massachusetts Institute of Technology, Artificial Intelligence Laboratory, and Laboratory for Computer Science, 545 Technology Square, Cambridge, Massachusetts 02139, USA

Received 1 March 1976, revised version received 13 August 1976

Analysis of algorithms, data structures, equivalence problem, union-find algorithms, set merging algorithms

This paper presents an analysis of a simple tree-structured disjoint set Union-Find algorithm, and shows that this algorithm requires between n and $2n$ steps on the average to execute a sequence of n Union and Find instructions, assuming that each pair of existing classes is equally likely to be merged by a Union instruction.

Union-Find algorithms are useful in the solution to a number of problems which require the construction of equivalence classes of a set of elements. The common parts of algorithms for constructing spanning trees of graphs, processing EQUIVALENCE statements, and determining the equivalence of finite automata are essentially the operations of combining two equivalence classes into one new equivalence class, and of finding the equivalence class to which an element belongs.

A number of algorithms for executing Union-Find instructions on a RAM are presented in ref. [1]. The fastest of these use tree structures by representing each equivalence class by a tree whose vertices represent the elements of the equivalence class. The simplest such tree-structured algorithm, which is the subject of the following analysis, executes the instruction Union (A, B) by making the root of the tree representing the equivalence class B a son of the root of the tree representing equivalence class A. The instruction Find (x) is executed by tracing up the tree from the vertex representing x until the root of the tree representing the class containing x is reached, and then returning the name of the equivalence class stored there. This algorithm requires at worst $\Theta(s^2)$ steps to execute s instructions on s elements [1], and may in fact require this many, since a chain of length $\Theta(s)$ can be created with $s/2$ union instructions, and $s/2$ subsequent finds can cost $\Theta(s)$ each.

Two improvements to this algorithm are "weighted unions", which make the root of the tree representing the smaller of the two classes being merged a son of the root of the larger, and "collapsing finds", which make the vertices encountered on the way up to the root sons of the root. Either of these two improvements reduces the worst-case running time to $\Theta[s \cdot \ln(s)]$ steps for s instructions on s elements [1], and Tarjan [3] has shown that using both improvements produces a $\Theta(s \cdot \alpha(s))$ running time where $\alpha(s)$ is related to a functional inverse of Ackermann's function. As a corollary of the present analysis, all these improved algorithms also have, in our model of merging probabilities, an expected running time proportional to the number of instructions executed.

The algorithm analyzed in this paper has also been analyzed by Yao [4] for two other models of probabilities. While our model is inappropriate for most known applications, it provides a further illustration of the extreme dependence of the expected running time of this algorithm on the probability distribution of the input instruction sequences.

The algorithm studied here is as follows. There are s elements in the universal set. The names of elements and the names of equivalence classes are integers in the range 1 through s. Initially, each element is alone in an equivalence class. The array FATHER (1:s) is used to store the name of each element's father in the tree. The convention of the algorithm is that if a vertex is its own

* This research was supported by the Fannie and John Hertz Foundation and by the National Science Foundation under research grant no. DCR74-12997 and research grant no. MCS76-14294.

father, then that vertex is the root of a tree representing an equivalence class (whose name is the name of the root).

```
Initialization: for i ← 1 to s do FATHER(i) ← i;
Find(i):        j ← i;
                while FATHER(j) ≠ j do j ← FATHER(j);
                return j;
Union(i,j)      FATHER(j) ← i;
```

For the purposes of this analysis, we take the cost of executing a Union instruction to be 1, and the cost of executing a Find to be the number of times the **while** test is made.

The average time required by this algorithm will be computed by averaging over all legal sequences of n instructions. Such sequences of instructions are of the form

$I_1, \ldots, I_n$

where for each k, $1 \leqslant k \leqslant n$, I_k is either

(a) Find(i) for some i, $1 \leqslant i \leqslant s$, or

(b) Union(i, j) for some i and j, $1 \leqslant i \neq j \leqslant s$, subject to the restriction that no instruction I_l, $1 \leqslant l < k$, is of the form Union(x, i) or Union(x, j).

The following analysis assumes that the probability that two equivalence classes appear in a Union instruction is independent of their sizes. This assumption does not apply to a number of situations in which the use of the Union-Find instructions is primarily to merge the equivalence classes containing two elements chosen independently with uniform probability from s. In such applications the probability that two classes are merged is proportional to the product of their sizes.

Lemma: The average cost of executing a find after u unions have been executed is $1 + H_s - H_{s-u}$.

Proof: Let $D(k, s)$ denote the average depth of the forest of trees (in the universe of s elements) after k unions have been executed, where roots of trees have depth 0. Then $D(u, s)+1$ is the average cost of executing a find after u unions have executed.

After k unions have been performed, there are $s - k$ trees in the forest. The average size of these trees is $s/(s - k)$ elements, so the next union will increase the average depth by

$(1/s)\,[s/(s - k)] = 1/(s - k)$.

Therefore,

$D(k + 1, s) = D(k, s) + 1/(s - k)$,

and since $D(0, s) = 0$,

$D(u, s) = 1/s + \ldots + 1/(s - u + 1) = H_s - H_{s-u}$. □

Let $A(u, n, s)$ be the average cost of all sequences of n instructions with u unions over the universe of s elements, and let $T(n, s)$ denote the average cost of all sequences of n instructions over the universe of s elements.

Theorem: $n \leqslant A(u, n, s) \leqslant 2n$ for all u, n, and s such that $0 \leqslant u \leqslant s \leqslant 1$ and $u \leqslant n$.

Proof: The first inequality follows immediately from the fact that each instruction has at least unit cost. For the second inequality, observe that in sequences of n instructions with u unions, the averaging number of finds between the jth and $j + 1$st unions is just the total number of finds divided by the number of intervals into which the unions decompose the instruction sequence, or $(n - u)/(u + 1)$. Thus the cost due to finds and the constant cost of u due to unions sum to

$$A(u, n, s) = \frac{n - u}{u + 1} \sum_{j=0}^{u} (1 + H_s - H_{s-j}) + u$$

$$= 2n - u - \frac{(n - u)(s - u)}{u + 1} [H_s - H_{s-u-1}]$$

which clearly indicates the inequality of the theorem.

Corollary: $n \leqslant T(n, s) \leqslant 2n$ for all n and s.

Proof: $T(n, s)$ is the average of $A(u, n, s)$, averaged over all values of u, $0 \leqslant u \leqslant \min\{n, s - 1\}$, and so is bounded by the bounds on the size of $A(u, n, s)$. □

We have demonstrated that in one model of probabilities, the simplest tree structured Union-Find algorithm has an expected running time proportional to the number of instructions executed. An immediate consequence of this is that in this model the improved Union-Find algorithms with weighted unions and collapsing finds also have linear expected times.

We thank the referees for suggesting a substantial simplification in the proof of the theorem.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, The Design and Analysis of Computer Algorithms (Addison-Wesley, 1974), 124–139.

[2] D.E. Knuth, The Act of Computer Programming, vol. 1, Fundamental Algorithms (Addison-Wesley, 1968), 58.

[3] R.E. Tarjan, Efficiency of a good but not linear set union algorithm, JACM, 22 (1975) 215–225.

[4] A.C.-C. Yao, On the average behavior of set merging algorithms (extended abstract), Proc. Eighth Ann. ACM Symp. on Theory of Computing (1976), 192–195.