have been examined twice. Clearly, half the table will be searched only if we replace the "fixed constant" by a number congruent to $-Q/_p2$. However, even if this is done, there is still the problem that when $Q \equiv 0$, only one table location is examined!

To correct these problems, replace steps (3) and (4) of Bell's algorithm with:

(3) Initialize A with C, where C is defined below.
(4) Increment A by $2Q$.

For this algorithm, we have $a = Q + C$, $b = Q$. We must then choose C so that $C \equiv -Q$ if $Q \not\equiv 0$ and $C \not\equiv -Q$ if $Q \equiv 0$. The algorithm will then search $(p + 1)/2$ locations if $Q \not\equiv 0$, and will search all p locations if $Q \equiv 0$.

The trouble with this algorithm is that it requires testing for $Q \equiv 0$, which means performing an extra division. A seemingly possible way out is to observe that if $(p - a)/_pb \equiv -j$, $b \not\equiv 0$, then the algorithm searches j fewer locations before it starts re-examining locations. We can then try to choose C so that we get j to be small, thereby examining nearly half the table before repeating. However, this requires that we make $C \equiv -(j + 1)Q$. There does not appear to be any simple algorithm for choosing a C satisfying this congruence for a small j when $Q \not\equiv 0$, and choosing $C \not\equiv -Q$ when $Q \equiv 0$.[1] It seems that the division is necessary.

The corrected version of Bell's algorithm still contains a gross inefficiency. For $Q \not\equiv 0$, it decided that the search is a failure after p tries, instead of the necessary $(p + 1)/2$ tries. This is easily corrected by changing the criterion for failure.

In summary, Bell's algorithm requires a correction which adds an extra division to the initialization procedure. This must be considered in evaluating its efficiency. Bell's table comparing the efficiency of his method with that of Maurer's indicates that this extra initialization cost is justified only if checking a single entry is a relatively time consuming operation.

REFERENCES:

1. BELL, JAMES R. The quadratic quotient method: a hash code eliminating secondary clustering. *Comm. ACM 13*, 2 (Feb. 1970), 107–109.
2. MAURER, W. D. An improved hash code for scatter storage. *Comm. ACM 11*, 1 (Jan. 1968), 35–38.

REPLY BY BELL. Before discussing Lamport's comment in detail, let us consider the correct observation on which it is based: Although any quadratic search (including quadratic quotient) hits half of the table entries, sometimes some entries are hit twice before others are hit once.

In other words, $K + ai + bi^2$ may not have maximum period for an arbitrary a and b. The author proves that forcing a to zero will guarantee maximal period.

A much simpler constraint is to let the constant in step (3) of the original algorithm be zero. Then

$$h_i(K) = R + (Q/2)i + (Q/2)i^2$$

and we first return to our original hash address when

$$R = R + (Q/2)i + (Q/2)i^2,$$

that is, when

$$i = -1 \quad \text{or} \quad i = 0 \quad \text{or} \quad Q = 0.$$

The first two cases state that $h(K)$ has a maximum periodicity. The third case is the degenerate one where the quotient is congruent to zero. We could use a division to spot the degenerate case. But by adopting the suggestion of paragraph 3 of Section 3c of the original article we can use

$$(Q \wedge \text{lowbitmask}) + 1$$

in lieu of Q to guarantee that this case does not occur.

Lamport has taken a more complicated approach.

[1] Note added in proof: In his reply below, Bell gives a simple method of choosing $j = 1$.

SCIENTIFIC APPLICATIONS

On the Number of Automorphisms of a Singly Generated Automaton

ZAMIR BAVEL
University of Kansas, Lawrence, Kansas*

Key Words and Phrases: automata, finite automata, singly generated automata, automorphisms, generators, length of state, minimal-length generators, orbit.
CR CATEGORY: 5.22

1. Introduction

Weeg proved in [3] that the number of automorphisms of a strongly connected finite automaton divides the number of states of the automaton. In [1, Th. 6], the author generalized this result to finite singly generated automata by proving that the number of automorphisms of such an automaton A divides the number of generators of A. This brief note improves the latter result. The number of automorphisms of A is shown to divide the number of minimal-length generators of A.

The improvement is of practical value not only in the more general case of singly generated automata but also in the strongly connected case, for the number of states whose length is minimal is usually much smaller than the number of states of the automaton. The improvement is particularly striking when the number of generators (states, in the strongly connected case) is large but only one of them is of minimal length; in that case, the only automorphism is the identity. But without the present result it may be necessary to examine up to half the number of

* Department of Computer Science. This work was supported in part by the National Science Foundation under Grant GJ-639.

states before the triviality of the automorphism group is realized. The utility of this result is enhanced by the fact that the length of a state is easily computed [2].

2. Preliminaries

The notation, definitions, and results in this section are collected mostly from [1]. For a nonempty set Σ, we denote by Σ^* the *free monoid* over Σ, i.e. the set of all strings of finite length of members of Σ including the empty string ϵ.

An *automaton* is a triple $A = (S, \Sigma, \delta)$, where S is a set (of *states*), Σ is a nonempty set (*the input alphabet*), and $\delta : S \times \Sigma^* \to S$ is the *transition function* satisfying: $\forall s \in S$ and $\forall x,y \in \Sigma^*$, $\delta(s, xy) = \delta[\delta(s, x), y]$; and $\delta(s, \epsilon) = s$, $\forall s \in S$.

An automaton $B = (T, \Sigma, \delta')$ is a *subautomaton* of $A = (S, \Sigma, \delta)$, written $B \ll A$, if and only if $T \subseteq S$ and δ' is the restriction of δ to $T \times \Sigma^*$. We use δ for δ', as no ambiguity arises. S_B denotes the set of states of an automaton B.

The set of *successors* of $s \in S$ is $\delta(s) = \{\delta(s, x) : x \in \Sigma^*\}$. The *automaton generated* by $s \in S$ is $\langle s \rangle = (\delta(s), \Sigma, \delta)$; i.e. the subautomaton whose set of states is the set of successors of s. An automaton $A = (S, \Sigma, \delta)$ is *singly generated* if and only if $\exists s \in S$ such that $A = \langle s \rangle$ and in that event s is a generator of $\langle s \rangle$. The *set of generators* of $\langle s \rangle$ is gen $\langle s \rangle = \{r \in S_{\langle s \rangle} : \langle r \rangle = \langle s \rangle\}$.

An automaton is *finite* if and only if its set of states is finite. The cardinality of a set S is denoted by $| S |$.

An automorphism of the automaton $A = (S, \Sigma, \delta)$ is a monic mapping f of S onto S (and the identity mapping on Σ^*) satisfying $f[\delta(s, x)] = \delta[f(s), x]$, $\forall s \in S$, $\forall x \in \Sigma^*$. The *set* (group) *of automorphisms* of an automaton A is denoted by $G(A)$. Where H is a subgroup of $G(A)$ and s is a state of $A = (S, \Sigma, \delta)$, the H-*orbit* of s is $O_H(s) = \{h(s) : h \in H\}$.

For each $u \in \Sigma^*$, where $u = x_1 \cdots x_k$ and $x_i \in \Sigma$, $i \in \{1, \cdots, k\}$, the length of u is $/u/ = /x_1 \cdots x_k/ = k$. The *length* of a state s of A is

$$/s/ = \max_{r \in S_{\langle s \rangle}} \{ \min_{u \in \Sigma^*} \{/u/ : \delta(s, u) = r\}\};$$

i.e. the length of the shortest route to the state farthest from s.

3. A Divisibility Bound on $G(\langle s \rangle)$

The following three results are proved by the author in [1].

LEMMA 1. *An automorphism of $\langle s \rangle$ is completely determined by its value on s.*

LEMMA 2. *Where f is an automorphism of an automaton A and s is a state of A, $\langle f(s) \rangle = f(\langle s \rangle)$.*

LEMMA 3. *Let $A = (S, \Sigma, \delta)$, let $p, q \in S$, and let H be a subgroup of $G(A)$. Then $O_H(p)$ and $O_H(q)$ are either identical or disjoint.*

With the aid of the three lemmas we now have:

THEOREM. *Let $\langle s \rangle = (S, \Sigma, \delta)$ be a finite automaton, let $M = \{m \in$ gen $\langle s \rangle : /m/ \leqq /s/, \forall s \in S\}$, and let H be a subgroup of $G(\langle s \rangle)$. Then $| H |$ divides $| M |$.*

PROOF. Let $r \in$ gen $\langle s \rangle$ and let $f \in G(\langle s \rangle)$. Then $f(r) \in$ gen $\langle s \rangle$, by Lemma 2. Thus, since gen $\langle s \rangle$ is finite, $f($gen $\langle s \rangle)$ = gen $\langle s \rangle$, i.e. automorphisms preserve generators. For any $t \in S$ and any $x,y \in \Sigma^*$, $f[\delta(t, x)] = f[\delta(t, y]$ if and only if $\delta(t, x) = \delta(t, y)$ and hence $/f(t)/ = /t/$, i.e. automorphisms preserve length. Therefore, $f(M) = M$.

By Lemma 1, distinct automorphisms have distinct images on members of gen $\langle s \rangle$ and thus $| O_H(t) | = | H |$, $\forall t \in$ gen $\langle s \rangle$. Thus, by Lemma 3, H partitions M into disjoint subsets of the form $O_H(t)$, and hence $| H |$ divides $| M |$. ❙

REFERENCES

1. BAVEL, Z. Structure and transition-preserving functions of finite automata. *J. ACM 15*, 1 (Jan. 1968), 135–158.
2. ——. Algorithms in the structure and transition-preserving functions of finite automata. Submitted to a technical journal.
3. WEEG, G. P. The structure of an automaton and its operation preserving transformation group. *J. ACM 9*, 3 (July 1962), 345–349.

ALGORITHMS

Remarks on Algorithms with Numerical Constants

C. B. DUNHAM
University of Western Ontario, London, Canada*

Keywords and Phrases: numerical algorithm, numerical constants
CR Categories: 5.10

Algorithms continue to be published in which undefined mathematical constants appear as a finite number of decimal digits. Such constants even appear in algorithms which explicitly claim to be of arbitrary precision; for example, Algorithm 349 [*Comm. ACM 12* (Apr. 1969), 213–214] has an undefined constant *piq* given to 48 decimal digits. Such algorithms are not useful in high precision unless the author defines all constants and tells how they can be obtained. It should be required of all published algorithms that all constants be defined or that working precision be explicitly stated.

[EDITOR'S NOTE. I agree completely with the suggested requirement and will try to enforce it in the future.—*L.D.F.*]

* Computer Science Department