Developing a Linear Algorithm for Cubing a Cyclic Permutation*

Jinyun Xue
David Gries

TR 86-780

September 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

*This research was supported by the NSF under grant DCR-8320274.

Developing a linear algorithm for cubing a cyclic permutation

Jinyun Xue and David Gries
Computer Science Department
Cornell University

September 1986

Abstract

A linear algorithm is developed for cubing a cyclic permutation stored as a function in an array. This continues work discussed in [0] and [1] on searching for disciplined methods for developing and describing algorithms that deal with complicated data structures such as linked lists. Here, a different representation of a cyclic permutation reveals a simple algorithm; then, an equally simple coordinate transformation is used to yield the final algorithm.

Introduction

A permutation of a finite set of elements is a one-to-one function on the set. For example, viewing a function as a set of ordered pairs, the following is a permutation P on the set $\{a, b, c, d\}$:

(0) $P = \{(a,b), (b,d), (c,a), (d,c)\}$.

Thus, $P.a = b$, $P.b = d$, $P.c = a$, and $P.d = c$, where "." denotes function application.

The product $P*Q$ of two permutations P and Q of the same set is defined by

$$(P*Q).r = P.(Q.r) .$$

Further, P^0 is the identity permutation and, for $k \geq 0$, $P^{k+1} = P*P^k = P^k*P$.

[1] This research was supported by the NSF under grant DCR-8320274.

Permutation P is *cyclic* if for each r in its domain the set of values $P^i.r$ for $i \geq 0$ is the whole domain. Any permutation can be viewed as the composition of its cyclic components.

Our problem is the following. Given is an array Π containing a permutation P —i.e. $\Pi.r = P.r$ for all r in the domain of P. We desire an algorithm S that cubes Π —i.e. that changes Π to P^3. Thus, the specification of S is

(1) $\{\Pi = P\}\ S\ \{\Pi = P^3\}$.

Because cubing a permutation can be done by cubing its cyclic components, we restrict our attention to permutations P that are cyclic.

Specification (0) is of little help in the development of S. In order to develop the algorithm, we introduce a second representation of a permutation that does lend insight. We develop an algorithm in terms of this second representation. We then manipulate the algorithm so that a suitable coordinate tranformation can be used to yield an algorithm in terms of array Π.

Remark on notation. Sequences are denoted by capital letters and individual elements by small letters. Catenation of sequences and elements is denoted by juxtaposition. For sequence s, $s = s.0\ s.1\ \ldots\ s.(\#s-1)$, and $s(1..)$ denotes sequence s without its first element.

An algorithm for cubing using a second representation

A cyclic permutation P can be represented by a sequence (consisting of elements of the domain of P) in which the follower of any element r in the sequence is the value $P.r$ (the follower of the last element of the sequence is the first). For example, permutation P given in (0) has the *cyclic representation* $a\ b\ d\ c$.

Second, *any* permutation can be represented by a two-line scheme where the top line is a sequence K giving its domain, the bottom line is a sequence H giving its range, and for each i the pair $(K.i, H.i)$ is in the permutation (see Knuth)[3]. For example, for permutation P of (0) we have:

$$P = \begin{bmatrix} K \\ H \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ b & d & a & c \end{bmatrix}.$$

We sometimes use the more manageable, linear, notation $P = H{:}K$. For example, the permutation given above can be written as $(b\ d\ a\ c){:}(a\ b\ c\ d)$. Note that there are many two-line representations of P, each derived by reordering the top and bottom sequences in the same manner. For example,

$$P = \begin{bmatrix} a & b & c & d \\ b & d & a & c \end{bmatrix} = \begin{bmatrix} b & a & d & c \\ d & b & c & a \end{bmatrix} = \begin{bmatrix} a & b & d & c \\ b & d & c & a \end{bmatrix}.$$

The rightmost representation has a particularly interesting property: the top sequence $K = a\ b\ d\ c$ is itself a cyclic representation of P. Further, the bottom sequence H is derived from K by rotating K one position to the left. Also, P^2 is $\bar{H}{:}K$, where $\bar{H}$ is the result of rotating K two positions to the left. This important property is generalized

further in the following lemma, whose proof is left to the reader.

(2) **Lemma.** Let function *rotl.s* yield sequence s rotated one position to the left. Let K be a cyclic representation of cyclic permutation P. Then, for all i, $i \geq 0$,

$$P^i \;=\; (rotl^i.K) : K \;.$$

This lemma gives us a simple algorithm for cubing a cyclic permutation $P = H{:}K$ where K is itself a cyclic representation of P. By the lemma, $H = rotl.K$, and P^3 can be found simply by rotating H two positions to the left!

An algorithm in terms of H and K

For the moment, suppose the domain of P contains at least three elements. Introducing names for some of the components of P, we write

(3) $P0$: $K = x\ y\ z\ W$.

Lemma (2) tells us that

$P = H{:}K$ is equivalent to $H = y\ z\ W\ x$
$P^3 = H{:}K$ is equivalent to $H = W\ x\ y\ z$,

so that an algorithm that cubes P represented by $H{:}K$ where K is itself a cyclic representation of P can be specified by

$$\{P0 \;\wedge\; H = y\ z\ W\ x\}\ S'\ \{P0 \;\wedge\; H = W\ x\ y\ z\}\ .$$

A simple way to rotate H is to move the pair y,z past one element of $W\ x$ at a time. Introducing two fresh variables U and V, we use the invariant

$$P1\text{: } H = U\ y\ z\ V \;\wedge\; U\ V = W\ x\ .$$

and write the algorithm:

(4)
```
U, V := [ ], W x;
do V ≠ [ ] → var v, V' := V.0, V(1..);
             H := U v y z V';
             U, V := U v, V'
od .
```

Writing an algorithm in terms of Π

Given $P = H{:}K$, the above algorithm changes H to establish $P^3 = H{:}K$. Our desired algorithm is to be in terms of array Π, where Π, H, and K are coupled by the following *representation invariant*:

(5) I: $\Pi = H{:}K$.

We use invariant I to help rewrite algorithm (4) in terms of Π. Note that an iteration of the loop of (4) changes H from

$U\ y\ z\ v\ V'$ to $U\ v\ y\ z\ V'$.

This requires a change in the elements of Π, but which elements? To answer this question, we introduce fresh variables p, q, r to denote elements and fresh variables $\overline{U}$, $\overline{V}$ to denote sequences and define them by

$$P2:\ K\ =\ \overline{U}\ p\ q\ \overline{V}\ \wedge\ \#U\ =\ \#\overline{U}\ \wedge\ r\ =\ (\overline{V}\ \overline{U}\ p).0\ .$$

Then, when H is changed as above, the permutation

$$\begin{bmatrix}\overline{U}\ p\ q\ r\ \overline{V}'\\ U\ y\ z\ v\ V'\end{bmatrix}\ \text{is changed to}\ \begin{bmatrix}\overline{U}\ p\ q\ r\ \overline{V}'\\ U\ v\ y\ z\ V'\end{bmatrix}\quad (\text{for } \overline{V}' \text{ defined by } \overline{V}\ =\ r\ \overline{V}')\ .$$

This means that only the values $\Pi.p$, $\Pi.q$, and $\Pi.r$ have to be changed —to v, y, and z, respectively.

Let us first change algorithm (4) to reflect the introduction of the new variables defined by $P2$. Note that whenever an element is appended to U a change in $\overline{U}$, p, q, r, and $\overline{V}$ is required, because the the lengths of U and $\overline{U}$ are the same (by $P2$).

(6) $U, V := [\,], W\ x;$
$p, q, r, \overline{U}, \overline{V} := x, y, z, [\,], W;$
do $V \neq [\,] \rightarrow$ **var** $v, V' := V.0, V(1..)$;
$H := U\ v\ y\ z\ V';$
$U, V := U\ v, V';$
$\overline{U}, p, q, r, \overline{V} := \overline{U}\ p, q, r, (\overline{V}'\ \overline{U}'\ p).0, \overline{V}'(1..)$
od .

Making the coordinate transformation

We are now ready to replace references to U, V, H, W, V', $\overline{U}$, $\overline{V}$, $\overline{V}'$, y, and z by references to Π using representation invariant I as well as the invariants $P0$, $P1$, and $P2$. Only the variables Π, p, q, r, and an arbitrary element x of the domain of P remain.

First, consider the initialization. Initially, from the fact that K is a cyclic representation of permutation P and from I we have $y = \Pi.x$ and $z = \Pi.y$. Hence, the initialization becomes

$$p := x;\ q := \Pi.x;\ r := \Pi.q\ .$$

The following derivation shows that the expression $V \neq [\,]$ is equivalent to $r \neq x$:

$$\begin{aligned} V = [\,] &\equiv \overline{V} = [\,] && (\text{by } P1 \text{ and } P2)\\ &\equiv r = (\overline{U}\ p).0 && (\text{by } P2)\\ &\equiv r = x && (\text{by } P0 \text{ and } P2)\ . \end{aligned}$$

The statement $H := U\ v\ y\ z\ V'$ calls for a change in Π. Π is being changed from

$$\Pi = \begin{bmatrix}\overline{U}\ p\ q\ r\ \overline{V}\\ U\ y\ z\ v\ \overline{V}'\end{bmatrix}\quad \text{to}\quad \Pi = \begin{bmatrix}\overline{U}\ p\ q\ r\ \overline{V}\\ U\ v\ y\ z\ \overline{V}'\end{bmatrix}\quad \text{where } \#U\ =\ \#\overline{U}\ .$$

Hence, $\Pi.p$, $\Pi.q$, and $\Pi.r$ are the only values of Π that need to be changed, and the change is effected by the assignment

$$\Pi.p, \Pi.q, \Pi.r := \Pi.r, \Pi.p, \Pi.q \ .$$

Finally, in the assignment to p, q, and r, we need to replace the reference to $(\overline{V}\ \overline{U}\ p).0$, by a reference to $\Pi.r$.

The final algorithm is

(7) $p := x;\ q := \Pi.p;\ r := \Pi.q;$
do $r \neq x \rightarrow \Pi.p, \Pi.q, \Pi.r := \Pi.r, \Pi.p, \Pi.q$;
$p, q, r := q, r, \Pi.p$
od .

Let us now consider the cases where the domain of P has size one or two. In both cases, $P = P^3$. In both cases, execution of algorithm (7) stores the same value in p and x and then terminates after 0 iterations of the loop, so Π remains unchanged and contains P^3.

Remark. The extension to compute P^k for some $k \geq 0$ should be obvious.

Acknowledgments

Thanks go to Rett Bull for comments on earlier drafts of this report.

References

[0] Feijen, W.H.J., A.J.M. van Gasteren, and D. Gries. In-situ inversion of a cyclic permutation. TR85-703, Computer Science Department, Cornell University (accepted for publication in IPL).

[1] Gries, D., and J.F. Prins. McLaren's Masterpiece. TR86-729, Computer Science Department, Cornell University (submitted to IPL).

[2] Gries, D. *The Science of Programming.* Springer Verlag, New York, 1981, pp 265-274.

[3] Knuth, D.E. *The Art of Computer PRogramming, Vol 1,* Addison-Wesley, Menlo Park, 1973.