ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
XII, 1 (1971)

ALGORITHM 11

KRYSTYNA JERZYKIEWICZ (Wrocław)

AN ALGORITHM TO TRANSFORM AN OWN-ARRAY

1. The procedure.

procedure $OWN(ST, NT, IL)$;
value ST, NT;
integer ST, NT, IL;
comment The procedure OWN transforms an array of any dimension in such a way that it changes the proportions of the array but preserves values of the elements common to both arrays. By the proportions of an array we mean here the order of its elements and values of the lower and upper bounds of all the subscripts.

The actual parameters:

ST address of the beginning of the vector a.

NT address of the beginning of the vector a'.
Vectors a and a' contain the information concerning the bound pair list of the array to be transformed and of the transformed one, respectively, i.e.,
$a = [k, d_1, g_1, d_2, g_2 \ldots, d_k, g_k]$, and
$a' = [k, d'_1, g'_1, d'_2, g'_2, \ldots, d'_k, g'_k]$,
where k denotes the dimension of both arrays, d_i and d'_i are the lower bounds of their i-th subscripts, and g_i and g'_i are the upper bounds of these subscripts.

IL after execution of the procedure body the value of this parameter is equal to the number of elements of the transformed array.

The global quantities in the procedure body:

mem the computer core store.

array error label preceding the statement to be executed if there exists such an i $(1 \leqslant i \leqslant k)$ that the inequality $g'_i < d'_i$ holds.

Elements of the array to be transformed are stored in lexicographical order (see, e.g., reference [2]), in successive storage locations of the computer, immediately after the vector a. After the execution of the procedure body the elements of the transformed array are stored also in lexicog-

raphical order, and in the same storage locations, i.e., after the vector a. The vector a' may be stored in any place of the computer store. If, however, the vector a' is stored in the locations which are to be occupied by the transformed array, then after executing the procedure body the elements of this vector become undefined;

```
begin
  Boolean B, B1;
  integer k, kn, p, ps, r1s, r1n, i1, ls, ln, i,
          ds, dn, gs, gn, u, u1, u2, u11, u12;
  k: = mem[ST];
  begin
    integer array l1, xs, xn[1: k];
    p: = ps: = kn: = ST+k+k;
    B: = false;
    B1: = true;
    r1s: = r1n: = i1: = 1;
    ls: = ln: = 0;
    for i: = k step -1 until 1 do
      begin
        u1: = i+i;
        ds: = mem[ST+u1-1];
        gs: = mem[ST+u1];
        dn: = mem[NT+u1-1];
        gn: = mem[NT+u1];
        B: = B ∨ ds > gn ∨ dn > gs;
        B1: = B1 ∧ ds = dn ∧ gs = gn;
        u: = ds-dn;
        u1: = gn-gs;
        ds: = gs-ds+1;
        dn: = gn-dn+1;
        if dn ⩽ 0 then go to array error;
        u11: = u12: = u2: = 0;
        if u < 0 then
          begin
            u12: = u11: = -u;
            u2: = u
          end;
        if u1 < 0 then
          begin
            u2: = u2+u1;
            u12: = u11-u1;
            u1: = 0
```

```
        end;
      xs[i]: = ls: = r1s × u12 + ls;
      ps: = ps + r1s × u11;
      r1s: = r1s × ds;
      ds: = ds + u2;
      l1[i]: = i1: = i1 × ds;
      xn[i]: = ln: = r1n × (dn − ds) + ln;
      kn: = kn − r1n × u1;
      r1n: = r1n × dn
    end i;
IL: = r1n;
kn: = kn + r1n;
if B ∨ B1 then go to end own;
u1: = ls: = l1[k];
for i: = 1 step 1 until i1 do
      begin
        p: = p + 1;
        ps: = ps + 1;
        mem[p]: = mem[ps];
        ls: = ls − 1;
        if ls = 0
          then
          for u: = 2 step 1 until k do
            if i ÷ l1[u] × l1[u] = i
              then
              begin
                ps: = ps + xs[u];
                ls: = u1;
                go to KZ
              end;
KZ: end i;
    ls: = u1;
    for i: = 1 step 1 until i1 do
      begin
        mem[kn]: = mem[p];
        p: = p − 1;
        kn: = kn − 1;
        ls: = ls − 1;
        if ls = 0
          then
          for u: = 2 step 1 until k do
            if i ÷ l1[u] × l1[u] = i
              then begin
```

```
                    kn: = kn − xn[u];
                    ls: = u1;
                    go to KR
                  end;
          KR: end i
            end;
end own:
  end OWN;
```

2. Method used. The realisation of dynamic reservation of own arrays in ALGOL-60 language can be reduced to the discussed here problem of transforming an array. It seems that the lack of algorithms performing such a transformation is one of the reasons that use of own arrays with variable bound pair lists is so often forbidden in hardware representations of ALGOL-60 language. In fact, such an algorithm had been described by Ingerman [1]. However, the algorithm makes use of a recursive procedure and it requires reservation of as much computer storage as is needed to store both arrays. Moreover, the algorithm contains an error due to which the values of elements common to both arrays are, in some instances, not preserved.

Example. In the case of a one-dimensional array with the bound pair list $[1: 2]$, when changing it into $[0: 2]$, Ingerman's algorithm does not preserve the value of the last element of the array.

The procedure *OWN* presented in this paper performs the transformation of arrays. It uses as many storage locations as does the larger of the transformed arrays and, in addition to this, $3 \times k$ locations, where k denotes the dimension of the arrays. A short description of the method employed follows here.

Let A denote a reserved array having the bound pair list equal to

$$[d_1: g_1, d_2: g_2, \ldots, d_k: g_k]$$

and let A' denote the same array after transformation according to the following bound pair list:

$$[d'_1: g'_1, d'_2: g'_2, \ldots, d'_k: g'_k].$$

Let further

$$d''_i = \max(d_i, d'_i), \qquad g''_i = \min(g_i, g'_i)$$

for $i = 1, 2, \ldots, k$. The procedure *OWN* transforms the array A into A' only when for every i the condition $d''_i \leqslant g''_i$ is fulfilled. The transformation of arrays consists in such an arrangement of the elements of array A within array A' that for each system of subscripts j_i $(i = 1, 2, \ldots, k)$ which obey the inequality $d''_i \leqslant j_i \leqslant g''_i$ takes place the equality

$$A[j_1, j_2, \ldots, j_k] = A'[j_1, j_2, \ldots, j_k].$$

Let now A'' denote the array having the bound pair list equal to

$$[d_1'' : g_2'', d_2'' : g_2'', \ldots, d_k'' : g_k''].$$

A'' is therefore the array of elements common to arrays A and A'.

In order to make the transformation the following auxiliary quantities are computed:

ps address of the element $A[d_1'', d_2'', \ldots, d_k'']$,

kn address of the element $A'[g_1'', g_2'', \ldots, g_k'']$,

ll_i $(i = 1, 2, \ldots, k)$ number of successive elements of the array A'' with fixed subscripts smaller than i, i.e., $ll_i = \prod_{j=i}^{k} (g_i'' - d_i'' + 1)$,

xs_i $(i = 2, 3, \ldots, k)$ number of all elements of the array A which occur lexicographically between the elements $A[j_1, j_2, \ldots, j_{i-1}, g_i'', g_{i+1}'', \ldots, g_k'']$ and $A[j_1, j_2, \ldots, j_{i-1}+1, d_i'', d_{i+1}'', \ldots, d_k'']$, i.e.

$$xs_i = \sum_{l=i+1}^{k} xs_l + (g_i - d_i - g_i'' + d_i'') \prod_{l=i+1}^{k} (g_l - d_l + 1),$$

where

$\sum_{l=k+1}^{k} xs_l = 0$ and

$\prod_{l=k+1}^{k} (g_l - d_l + 1) = 1$ throughout this paper,

xn_i $(i = 2, 3, \ldots, k)$ number of all elements of the array A' which occur lexicographically between the elements $A'[j_1, j_2, \ldots, j_{i-1}, g_i'', g_{i+1}'', \ldots, g_k'']$ and $A'[j_1, j_2, \ldots, j_{i-1}+1, d_i'', d_{i+1}'', \ldots, d_k'']$, i.e.,

$$xn_i = \sum_{l=i+1}^{k} xn_l + (g_i' - d_i' - g_i'' + d_i'') \prod_{l=i+1}^{k} (g_l' - d_l' + 1).$$

The transformation of the array A into A' may now be divided in two stages.

a. The array A is transformed into A'', the elements of which are stored in lexicographical order in the locations formerly occupied by A. The ll_1 elements of the array A, which are to be transferred to the array A'', occur in array A in groups of ll_k elements. The address of the first element of the first group is equal to ps. If the number l of the elements already transferred to A'' is divisible by ll_i and is not divisible by ll_{i-1}, then xs_i elements occurring after the last transferred group do not belong to A'', and the address of the first element of the next group is greater by $xs_i + 1$ than the address of the last element of the preceding group.

b. The array A'' is transformed into A'. Elements of A'' are to be transferred to A' also in groups of ll_k elements. The address of the last element of the last group is equal to kn. If the number l of the elements

of A'' already transferred to A' is divisible by ll_i and is not divisible by ll_{i-1}, then the address in A' of the last element of the group to be transferred as the next one is smaller by xn_i+1 than the address of the first element of the lastly transferred group.

The values of the elements of the array A' which do not occur in A are undefined after the transformation.

Example. Let the array A have the bound pair list equal to [1: 3, 2: 4, 3: 5, 4: 6] and let its successive elements be equal to successive natural numbers, the first element being equal to 1. If the bound pair list of A' is equal to [1: 2, 2: 4, 3: 3, 3: 7], then A' is as follows:

x	1	2	3	x
x	10	11	12	x
x	19	20	21	x
x	28	29	30	x
x	37	38	39	x
x	46	47	48	x

If the bound pair list of A' is equal to [1 : 2, 1 : 3, 2 : 4, 3 : 5], then A' is as follows:

x	x	x
x	x	x
x	x	x
x	x	x
x	1	2
x	4	5
x	x	x
x	10	11
x	13	14

x	x	x
x	x	x
x	x	x
x	x	x
x	28	29
x	31	32
x	x	x
x	37	38
x	40	41

3. Application. The procedure *OWN* has been written in the 1204-ALGOL language and tested on a number of examples on the ODRA 1204 digital computer. The time required to execute the procedure body depends on the dimension of the array to be transformed and on the proportions of the common part of the transformed arrays.

Example.

Dimension of the array	Proportions of the common part	The required time (seconds)
5	[1: 3, 1: 3, 1: 3, 1: 3, 1: 2]	2.10
5	[1: 2, 1: 3, 1: 3, 1: 3, 1: 3]	1.62
4	[1: 2, 1: 3, 1: 3, 1: 9]	.90
4	[1: 3, 1: 3, 1: 3, 1: 3]	.72
3	[1: 9, 1: 9, 1: 2]	1.48
3	[1: 2, 1: 3, 1: 27]	.71
2	[1: 4, 1: 4]	.12
1	[1: 27]	.12

The procedure *OWN* had been applied in the storage administration system of the 1204-ALGOL, a hardware representation of ALGOL-60 for the ODRA 1204 digital computer, so that when writing a programme in this language one can use own arrays with variable bound pair lists.

Acknowledgment. The author is greatly indebted to Dr. Stefan Paszkowski for encouragement and valuable suggestions while reading the manuscript.

References

[1] P. Z. Ingerman, *Dynamic declarations*, Comm. ACM 1 (1961), p. 59-60.
[2] S. Paszkowski, *Język ALGOL 60*, PWN, Warszawa 1968, p. 260.

DEPT. OF NUMERICAL METHODS
UNIVERSITY OF WROCŁAW

Received on 30. 5. 1970

KRYSTYNA JERZYKIEWICZ (Wrocław)

ALGORYTM 11

ALGORYTM PRZEKSZTAŁCANIA TABLIC WŁASNYCH

STRESZCZENIE

Procedura *OWN* przekształca tablicę dowolnego wymiaru w ten sposób, że zmienia rozmiar tej tablicy zachowując wartości elementów wspólnych dla obu rozmiarów. Przez rozmiar tablicy rozumiemy tutaj wartości dolnych i górnych granic wskaźników oraz kolejność elementów tablicy.

Parametry aktualne procedury:

ST adres początku wektora a.

NT adres początku wektora a'.

Wektory a i a' zawierają informacje o wykazie par granicznych tablic przekształcanej i przekształconej, odpowiednio:

$a = [k, d_1, g_1, d_2, g_2, \ldots, d_k, g_k]$,

$a' = [k, d'_1, g'_1, d'_2, g'_2, \ldots, d'_k, g'_k]$,

gdzie k oznacza wymiar tablic, d_i i d'_i są dolnymi granicami ich i-tych wskaźników, a g_i i g'_i są górnymi granicami tych wskaźników.

IL po wykonaniu treści procedury wartość tego parametru jest równa liczbie elementów przekształconej tablicy.

Nazwy nielokalne w treści procedury:

mem tablica oznaczająca pamięć maszyny.

array error etykieta, do której przechodzi się, jeśli istnieje takie i $(1 \leqslant i \leqslant k)$, że jest spełniona nierówność $g'_i < d'_i$.

Elementy tablicy przekształcanej są zapamiętane w porządku leksykograficznym (patrz [2]) w kolejnych komórkach pamięci maszyny bezpośrednio po wektorze a. Po wykonaniu treści procedury elementy tablicy przekształconej są zapamiętane także w porządku leksykograficznym i w tym samym miejscu, to znaczy po wektorze a. Wektor a' może być zapamiętany w dowolnym miejscu pamięci maszyny. Jeżeli jednak wektor a' jest zapamiętany na miejscu, które ma zająć przekształcona tablica, to po wykonaniu treści procedury *OWN* elementy tego wektora są nieokreślone.