prof. Doru Popescu Anastasiu
C. N. "Radu Greceanu" Slatina

Generarea permutărilor. Alfametica

În continuare vom prezenta cei mai importanți algoritmi de generare a permutărilor, începând cu o metodă clasică, simplă şi flexibilă:

Algoritmul L (Generarea permutărilor în ordine lexicografică) Dată fiind o secvență de n elemente $a_1, a_2, \ldots, a_n$, inițial sortate, astfel încât

$a_1 \le a_2 \le \ldots \le a_n$,

acest algoritm generează toate permutările mulțimii $\{a_1, a_2, \ldots, a_n\}$, parcurgându-le în ordine lexicografică.

Exemplu
Pentru secvența 1, 2, 2, 3 avem permutările
1,2,2,3; 1,2,3,2; 1,3,2,2; 2,1,2,3; 2,1,3,2; 2,2,1,3; 2,2,3,1; 2,3,1,2; 2,3,2,1; 3,1,2,2; 3,2,1,2; 3,2,2,1;
în ordine lexicografică. În continuare vom folosi un element auxiliary a_0 care este strict mai mic decât cel mai mare element, a_n.

```
L0. Citiea lui n si a secventei a1, a2, …, an
L1. Se afişează permutarea a1, a2, …, an.
L2. {gasirea lui j}
    j←n-1
    Cât timp j>0 si aj>=aj+1 executa
      j←j-1
    sfarsit cat timp
    daca j=0 atunci
      goto L5
    sfarsit daca
{in acest moment, j este cel mai mic indice pentru care am vizitat deja
toate permutarile care incep cu a1,…, aj. Astfel, urmatoarea permutare in
ordine lexicografica va modifica (incrementa) valoarea lui aj}
L3. {incrementarea lui aj}
    h←n
    Cat timp aj>=ah executa
      h←h-1
    sfarsit cat timp
    aj↔ah
{Deoarece aj+1≥…≥an, ah este cel mai mic element care este mai mare decat
aj si care poate urma in mod justificat dupa a1,…,aj-1 in cadrul unei
permutari. Inainte de interschimbare aveam aj+1≥…≥ah-1≥ah>aj≥ah+1≥…≥ah-1 ;
dupa interschimbare avem aj+1≥…≥ah-1≥aj>ah≥ah+1≥…≥an.}
L4.{inversarea secventei aj+1,…,an}
   k←j+1
   h←n
   cat timp k<h executa
    ak↔ah
    k←k+1
    h←h-1
   sfarsit cat timp
L5. sfarsit
```

În general, succesorul în ordine lexicografică al oricărui model combinatoric a_1, …, a_n poate fi obținut printr-o procedură care cuprinde trei pași:

a) Se găsește cel mai mare indice j astfel încât a_j poate fi mărit.
b) a_j este incrementat cu cantitatea cea mai mică aplicabilă.
c) Se găsește cea mai rapidă modalitate de a extinde lexicografic noul a_1, …, a_j pentru a completa modelul.

Inversiuni

Fie $a_1a_2…a_n$ o permutare a mulțimii {1, 2, …, n}. Dacă $i<j$ și $a_i>a_j$, perechea (a_i, a_j) se numește o *inversiune* a permutării. De exemplu permutarea 3 1 4 2 are trei inversiuni: (3,1), (3,2) și (4,2).

Tabelul de inversiuni $b_1b_2…b_n$ ale permutării $a_1a_2…a_n$ se obține definind b_j ca numărul de elemente de la stânga lui j care sunt mai mari decât j. Rezultă, de exemplu, că permutarea

5 9 1 8 2 6 4 7 3 (1)

are ca tabel de inversiuni

2 3 6 4 0 2 2 1 0, (2)

pentru că la stânga lui 1 se află 5 și 9, la stânga lui 2 se află 5, 9, 8; etc. Această permutare are ăn total 20 de inversiuni. Prin definiție, numerele b_j, vor satisface ăntotdeauna relațiile:

$0 \le b_1 \le n-1$, $0 \le b_2 \le n-2$, …, $0 \le b_{n-1} \le 1$, $b_n=0$. (3)

Cel mai important lucru despre inversiuni este faptul că un tabel de inversiuni determină unic permutarea corespunzătoare. Putem trece de la orice tabel de inversiuni $b_1b_2…b_n$ satisfăcând (3) înapoi la unica permutare care l-a produs, determinând succesiv poziția relativă a elementelor n, n-1, …,1 (în această ordine). De exemplu, putem construi permutarea corespunzătoare lui (2) după cum urmează: scriem numărul 9; apoi îl amplasăm pe 8 după 9, pentru că $b_8=1$. Similar, îl punem pe 7 după după 9 cât și după 8, deoarece $b_7=2$. Numărul 6 trebuie să urmeze după două din numerele deja scrise, pentru că $b_2=2$; rezultatul perțial obținut până acum este deci:

9 8 6 7.

Continuând cu plasarea lui 5 la stânga, pentru că $b_5=0$; îl punem pe 4 după patru din numerele scrise și pe 3 după șase numere (anume în extrema dreaptă), ceea ce ne dă:

5 9 8 6 4 7 3.

Inserăm în mod analog pe 2 și pe 1 și obținem (1).

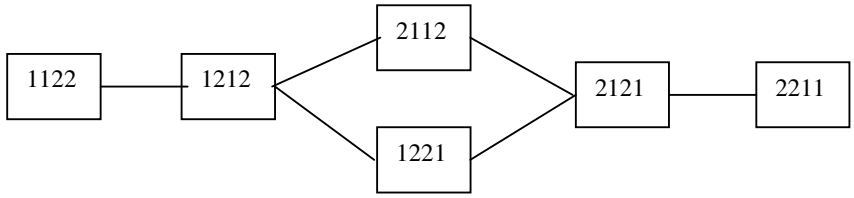
Interschimbarea elementelor alăturate

Cea mai simplă schimbare posibilă în cazul unei permutări este interschimbarea elementelor alăturate și știm că orice permutare poate avea elementele sortate dacă efectuăm o seriea adecvată de asemenea transpoziții. Prin urmare, putem reveni și obține orice permutare dorim parcurgând toate elementele unul după altul și apoi interschimbând elementele alăturate ori de câte ori este cazul.

Se pune o întrebare: este posibilă parcurgerea *tuturor* permutărilor unei multi-mulțimi date, astfel încât la fiecare pas să-și schimbe locul doar două elemente alăturate ?

Din păcate atunci când multi-mulțimea cuprinde elemente care se repetă, nu putem găsi o asemenea secvență.

De exemplu, pentru {1,1,2,2}, prin transpoziții alăturate avem:



(4)

acest graf nu are nici o cale hamiltoniană.

Cele mai multe aplicații lucrează, însă, cu permutări de elemente distincte, iar în acest caz putem construi un algoritm simplu care face posibilă generarea tuturor celor n! permutări, efectuând doar n!-1 interschimbări de elemente alăturate. Mai mult, o altă asemenea transpoziție relizează revenirea la elementul inițial, aşa că avem un ciclu hamiltonian.

Ideea este să luați o asemenea secvență pentru {1,2,…,n-1} şi să inserați numărul n în fiecare permutare, în toate modurile posibile. De exemplu pentru n=4, secvența (123, 132, 312, 321, 231, 213) conduce la formarea coloanelor tabloului

```
1234 1324 3124 3214 2314 2134
1243 1342 3142 3241 2341 2143
1423 1432 3412 3421 2431 2413                    (5)
4123 4132 4312 4321 4231 4213
```

când 4 este inserat pe toate cele patru poziții posibile. Acum obținem secvența dorită citind de sus în jos prima coloană, de jos în sus a doua coloană, de sus în jos a treia, …, de jos în sus ultima: (1234, 1243, 1423, 4123, 4132, 1432, 1342, 1324, 3124, 3142, …, 2143, 2134).
Fiecare transpoziție de elemente alăturate modifică numărul total de inversiuni prin ±1. În realitate, când avem în vedere aşa-zisa tabelă de inversiuni $c_1 \ldots c_n$, unde c_j, este numărul de elemente aflate la dreapta lui j şi care sunt mai mici decât j, descoperim că permutările din (5) au următoarele tabele de inversiuni:

```
0000 0010 0020 0120 0110 0100
0001 0011 0021 0121 0111 0101
0002 0012 0022 0122 0112 0102                    (6)
0003 0013 0023 0123 0113 0103
```

Algoritmul P (Schimbări simple). Dată fiind o secvență de n elemente distincte, $a_1, a_2, \ldots, a_n$, acest algoritm generează toate permutările acestora prin interschimbarea repetată a unor elemente alăturate. Algoritmul utilizează un vector suplimentar $c_1, c_2, \ldots, c_n$ care reprezintă inversiunile în forma descrisă mai sus, parcurgând toate secvențele de numere întregi astfel încât

$0 \le c_j < j$, pentru $1 \le j \le n$. (7)

Un alt vector, $o_1, o_2, \ldots, o_n$, memorează direcțiile în care se chimbă intrările c_j.

```
P1.{initializarea}
   Pentru j=1,…,n executa
    cj←0
    oj←1
   sfarsit pentru
P2. {afisare}
   Se afiseaza permutarea a1…an
P3. {Pregatirea pentru schimbare}
```

```
   j←n
   s←0
    {Urmatorii pasi determina indicele j pentru care valoarea lui c_j este
pe cale sa se schimbe respectand in continuare conditia (7); variabila s
reprezinta numarul de indici k>j astfel incat c_k=k-1}
P4. {Pregatirea pentru schimbare}
    q←c_j+o_j
    daca q<0 atunci
     goto P7
    sfarsit daca
    daca q=j atunci
     goto P6
    sfarsit daca
P5. {Schimbarea}
    a_{j-c[j]+s} ↔ a_{j-q+s}
    c_j←q
    goto P2
P6. {incrementarea lui s}
    daca j=1 atunci
     goto P8
             Altfel
     s←s+1
    sfarsit daca
P7. {schimbarea directiei}
    o_j←-o_j
    j←j-1
    goto P4
P8. Sfarsit
```

Faptul că algoritmul P efectuează exact o interschimbare la fiecare vizită, înseamnă că permutările pe care le generează sunt, în mod alternativ, pare respectiv impare. Prin urmare putem genera toate permutările pare prin simpla evitate a celor impare. În realitate, tabelele c şi o simplifică ținerea evidenței numărului total curent de inversiuni, $c_1+\ldots+c_n$, în timpul execuției algoritmului.

Multe programe trebuie să genereze în mod repetat aceleaşi permutări şi în asemenea cazuri nu trebuie să parcurgem de fiecare dată etapele algoritmului P. Putem pur şi simplu pregăti o listă de transpoziții adecvate, utilizând următoarea metodă:

Algoritmul T (Transpozițiile schimbărilor simple)

```
{Acest algoritm calculează elementele unui tabel t[1], t[2],…,t[n!-1]
astfel incat actiunile algoritmului P sa fie echivalente cu
interschimbarile succesive a_{t[k]}↔a_{t[k]+1}, pentru 1≤k!<n!. Presupunem ca
n≥2.}
T1. {initializarea}
    N←n!
    d←N/2
    t[d]←1
    m←2
T2. daca m=n atunci
     goto T6
             altfel
     m←m+1
     d←d/m
     k←0
    sfarsit daca
```

```
T3. {de sus in jos}
    k←k+d
    j←m-1
    cat timp j>0 executa
     t[k]←j
     j←j-1
     k←k+d
    sfarsit cat timp
T4. {deplasare}
    t[k]←t[k]+1
    k←k+d
T5. {de jos in sus}
    Cat timp j<m-1 executa
     j←j+1
     t[k]←j
     k←k+d
    sfarsit cat timp
    daca k<N atunci
     goto T3
             altfel
     goto T2
    sfarsit daca
T6. sfarsit
```

De exemplu, dacă n=4, obținem tabelul (t[1], t[2], …, t[23])=(3,2,1,3,1,2,3,1,3,2,1,3,1,2,3,1,3,2,1,3,1,2).

Alfametrica

Să considerăm acum un gen de problemă simplă, în care sunt utile permutările:
Cum poate modelul

```
 SEND+
 MORE
------                          (8)
MONEY
```

Reprezenta o sumă corectă, dacă fiecare literă înlocuieşte câte o cifră zecimală?

Asemenea probleme se numesc de obicei "alfametrice", cuvânt inventat de J.A.H. Hunter în 1955, un alt termen pentru acest gen de probleme este "criptaritm", propus de S. Vatriquant în 1931.
Să presupunem că dorim să lucrăm cu o mulțime mare de alfametice complicate, unele dintre ecestea nefiind rezolvabile, pe când altele pot avea mai multe soluții. Atunci, este bine să folosim un algoritm care să încerce toate permutările de cifre, care corespund unui anumit model, determinând permutările care produc o sumă corectă.

Putem la fel de bine să facem o mică generalizare şi să luăm în calcul alfametice sumative în general, lucrând nu numai cu sume simple, cum ar fi (8), ci şi cu exemple cum ar fi:

```
VIOLIN + VIOLIN + VIOLA = TRIO + SONATA
```

În mod echivalent, dorim să rezolvăm probleme cum ar fi

```
2(VIOLIN) + VIOLA - TRIO - SONATA = 0.        (9)
```

unde este dată o sumă de termeni cu coeficienți întregi, iar scopul îl reprezintă obținerea cifrei 0 prin înlocuirea diferitelor litere cu cifre zecimale distincte. Fiecare literă dintr-o asemenea problemă are o "semnătură" obținută prin înlocuirea cu cifra 1 a fiecărei apariții a literei şi cu cifra 0 a celorlalte litere; de exemplu semnătura lui `I` în (9) este

```
2(010010) + 01000 - 0010 - 000000,
```

respectiv 21010. Dacă atribuim în mod arbitrar codurile (1,2,…,10) literelor (V,I,O,L,N,A,T,R,S,X), semnăturile acestora în (9) sunt

s_1=210000, s_2=21010, s_3= - 7901, s_4=210, s_5= - 998, s_6= - 100, s_7= - 1010, s_8= - 100, s_9= - 100000, s_{10}=0. (10)

A fost adăugată o literă suplimentară X, deoarece este necesar să fie 10 litere. Acum problema se reduce la determinarea tuturor permutărilor $a_1 \ldots a_{10}$ ale mulțimii {0,1,…,9}, astfel încât

$$a \cdot s = \sum_{j=1}^{10} a_j s_j = 0 \quad (11).$$

Există de asemenea o condiție suplimentară, deoarece prima cifră a unui număr din alfametică este nenulă. De exemplu, sumele,

7316+ 0823 ------ 08139	6524+ 0735 ------ 07259
5731+ 0647 ------ 06378	2817+ 0368 ------ 03185

şi multe altele sunt considerate a nu fi acceptabile ale lui (8). În general, există o mulțime F a primelor litere, pentru care trebuie să avem

$a_j \neq 0$, pentru orice j din F (12)

Mulțimea F care corespunde relațiilor (9) şi (10) este {1,7,9}.

O metodă de abordare a unei familii de alfametici sumative începe utilizând algoritmul T pentru a pregăti un tabel de 10!-1 transpoziții t[k]. Apoi, pentru fiecare problemă definită de o secvență de semnături (s_1,…,s_{10}) şi de o mulțime de prime litere, F, putem căuta exhaustiv soluții astfel:

```
A1. {Inițializarea}
    a1a2…a10←01…9
    v←Σ_{j=1}^{10} (j-1)s_j
    k←1
    dj←sj+1-sj, pentru 1≤j<10
A2. {Testul}
    daca v=0 si relatia (12) este adevarata atunci
      se afiseaza solutia a1a2…a10
    sfarsit daca
A3. {Interschimbarea}
    daca k=10! Atunci
```

```
      goto A4
            altfel
      j←t[k]
      v←v-(a_{j+1}-a_j)d_j
      a_{j+1}↔a_j
      k←k+1
      goto A2
    sfarsit daca
A4. sfarsit
```

Pasul A3 este justificat de fapltul că interschimbând a_j cu a_{j+1} pur şi simplu este micşorat a·s cu $(a_{j+1}-a_j)(s_{j+1}-s_j)$.

O alfametică se numeşte *pură* dacă are o soluție unică.

(9) nu este pură, pentru că permutările 1764802539 şi 3546281970 rezolvă sistemul format din (11) şi (12). Mai mult $s_6=s_8$ în (10), astfel putem obține încă două soluții interschimbând cifrele atribuite lui A şi R.

Pe de altă parte (8) este o alfametică pură, dar metoda pe care am descris-o va găsi două permutări distincte care reprezintă soluții ale acesteia. Motivul pentru care se întâmplă acest lucru este că (8) implică folosirea a doar opt litere distincte, aşa că o transformăm în soluție utilizând două semnături fictive, $s_9=s_{10}=0$. În general, o alfametică cu m litere distincte va avea 10-m semnături fictive, $s_{m+1}=\ldots s_{10}=0$ şi fiecare dintre soluțiile sale vor fi găsite de (10-m)! ori dacă nu insistăm ca, de exemplu, să fie îndeplinită condiția $a_{m+1}<\ldots<a_{10}$.

Un cadru general

În continuare vom utiliza indici a căror valori încep de la 0, astfel $a_0a_1\ldots a_{n-1}$ va fi notația pentru a nota permutările mulțimii {0,1,…,n-1} în loc să scriem $a_1a_2\ldots a_n$ pentru permutările mulțimii {1,2,…,n}. Mai important este faptul că vom considera schemele de generare a permutărilor, în care cele mai multe operații vor fi efectuate la stânga, astfel încât toate permutările mulțimii {0,1,…,k-1} vor fi generate în cursul primilor k! paşi, unde $1\le k\le n$. De exemplu o asemenea schemă pentru n=4 este

```
0123, 1023, 0213, 2013, 1203, 2103, 0132, 1032, 0312, 3012, 1302, 3102,
0231, 2031, 0321, 3021, 2301, 3201, 1230, 2130, 1320, 3120, 2310, 3210;
                                (13)
```

şi se numeşte "ordine colex inversa" deoarece dacă inversăm şirurile de la dreapta la stânga, obținem 3210, 3201, 3120,…, 0123, în ordine lexicografică inversă. Un alt mod de a privi relația (13) este să considerați intrările de forma $(n-a_n)\ldots(n-a_1)$, unde $a_1a_2\ldots a_n$ parcurge în ordine lexicografică permutările mulțimii {1,2,…,n}.

Inmulțirea permutărilor

Reamintim că o permutare poate fi scrisă pe două linii, de exemplu permutarea p=250143 poate fi scrisă astfel:

$p=\begin{pmatrix}012345\\250143\end{pmatrix}$, fie în formă mai compactă a două cicluri $p=(0\,2)(1\,5\,3)$, cu semnificația că p transformă 0→2, 1→5, 2→0, 3→1, 4→4 şi 5→3; un ciclu de dimensiune 1, cum ar fi 4, nu trebuie indicat. Într-ucât 4 este un punct fix al acestei permutări, spunem că "p îl fixează pe 4". Scriem de asemenea 0p=2, 1p=5 şi aşa mai departe, afirmând că jp este «imaginea lui j prin p«. înmulțirea permutărilor, cum ar fi p înmulțită cu q, unde q=543210, se efectuează astfel

$$pq=\begin{pmatrix}012345\\250143\end{pmatrix}\begin{pmatrix}012345\\543210\end{pmatrix}=\begin{pmatrix}012345\\250143\end{pmatrix}\begin{pmatrix}250143\\305412\end{pmatrix}=\begin{pmatrix}012345\\305412\end{pmatrix}$$

fie în formă de cicluri

$pq = (0\ 2)(1\ 5\ 3)\cdot(0\ 5)(1\ 4)(2\ 3) = (0\ 3\ 4\ 1)(2\ 5)$.

De remarcat că imaginea lui 1 pein pq este 1(pq)=(1p)q=5q=0, etc.

Urmare a unei propuneri făcută de Evariste Galois în 1830, se spune că o mulțime nevidă de permutări G formează un *grup* dacă este închisă la înmulțire, adică dacă pentru orice două permutări p, q din G produsul lor pq este tot în G.

Grupurile de permutări G sunt reprezentate convenabil în calculator, prin intermediul unui tabel Sims, care este o familie de submulțimi S_1, S_2, … ale lui G, cu următoarea proprietate:

S_k conține exact o singură permutare p_{kj} care transpune k→j şi fixează valorile tuturor elementelor care sunt mai mari decât k, ori de câte ori G conține o asemenea permutare. Fie p_{kk}pemutarea identică, aceasta fiind inclusă întotdeauna în G; dar când 0≤j<k, orice permutare adecvată poate fi aleasă să joace rolul lui p_{kj}. Principalul avantaj al unui tabel Sims este acela că oferă o reprezentare convenabilă a întregului grup:

Lema S.

Fie S_1, S_2, …, S_{n-1} un tabel Sims pentru un grup G de permutări ale mulțimii {0, 1, …, n-1}. Atunci fiecare element al lui G are o reprezentare unică, de forma

$q = p_1 p_2 \cdots p_{n-1}, unde \quad p_k \in S_k, \quad pentru \quad 1 \le k < n.$ (14)

Demonstrație

Dacă q este o asemenea reprezentație şi dacă p_{n-1} este permutarea $p_{(n-1)j}$ din S_{n-1}, atunci q transpune n-1→j, deoarece toate elementele lui $S_1 U \ldots U S_{n-2}$ fixează valoarea lui n-1. Reciproc, dacă q transpune n-1→j, avem q=q'$p_{(n/1)j}$, unde $q' = qp^{-}_{(n-1)j}$ este o permutare a lui G, care fixează n-1. Mulțimea G' a tuturor acestor permutări formează un grup, iar S_1, …, $S_{n/2}$ este un table Sims pentru G', prin urmare, rezultatul se obține prin inducție după n.

Pentru cadrul general, prezintă interes grup permutărilor mulțimii {0, 1, …, n-1} şi în acest caz fiecare mulțime S_k a unui tabel Sims va conține k+1 elemente {p(k,0), p(k,1), …, p(k,k)}, unde p(k,0) este permutarea identică, iar celelalte permutări transpun k în valorile {0, 1, …, k-1}, într-o ordine oarecare.(Permutarea p(k,j) nu trebuie să fie identică cu p_{kj} şi, de obicei acestea două sunt diferite.) Fiecare asemenea tabel Sims conduce la un generator de permutări, conform următorului plan:

Algoritmul G (*Generatorul de permutări generale*)

Fie un tabel Sims (S_1, S_2, …, S_{n-1}), în care fiecare S_k are k+1 elemente p(k,j) descrise ca mai înainte. Acest algoritm generează toate permutările $a_0a_1\ldots a_{n-1}$ ale mulțimii {0, 1, …, n-1}, utulizând un tabel de control auxiliar $c_n\ldots c_2c_1$.

```
G1. {initializarea}
    pentru j=1,n-1 executa
     aj←j
     cj+1←0
    sfarsit pentru
G2. {vizitarea}
    {la aceasta etapa, numarul in baza mixta [cn-1,...,c2,c1 / n,...,3, 2] reprezinta
numarul de permutari vizitate pana in prezent}
    Se afiseaza permutarea a0a1…an-1.
G3. {incrementarea cu 1 a lui cn…c2c1}
    k←1
    cat timp (k<n)si(ck>=k)
      ck←0
      k←k+1
    sfarsit cat timp
    daca k=n atunci
     goto G5
             altfel
```

```
     ck←ck+1
    sfarsit daca
G4. {permutarea}
    Se aplica permutarea q(k,ck)u(k-1)¯ lui a0a1…an-1, asa cum se aplica
mai jos, dupa care goto G2.
G5. Sfarsit
```

Aplicarea unei permutări q lui $a_0a_1\ldots a_{n-1}$ înseamnă înlocuirea lui a_j cu a_{jq} pentru toți indicii j, aceasta corespunde unei premultiplicări cu q.
Să definim
$q(k,j)=p(k,j)p(k,j-1)^{-}$, pentru $1\le j\le k$ (15)
$u(k)=p(1,1)\ldots p(k,k)$. (16)
Atunci etapele G3 şi G4 conservă proprietatea:
$a_0a_1\ldots a_{n-1}$ este permutarea $p(1,c_1)p(2,c_2)\ldots p(n-1,c_{n-1})$, (17)
iar Lema S demonstrează că fiecare permutare este vizitată exact o singură dată.

Omiterea blocurilor nedorite

Un avantaj notabil al algoritmului G este acela că parcurge toate permutările lui $a_0a_1\ldots a_{k-1}$ înainte să ajungă la a_k, apoi efectuează alte k! cicluri înainte să interschimbe a_k din nou etc. Prin urmare dacă de fiecare dată ajungem la o aranjare a elementelor finale $a_k\ldots a_{n-1}$, care nu este importantă în cazul problemei la care lucrăm, putem omite rapid toate permutările care se termină cu sufixul nedorit. Mai exact am putea înloci pasul G2 cu următorii sub-paşi:

```
G2.0 {acceptabil?}
     Daca ak…an-1 nu este un sufix acceptabil, se trece la pasul G2.1. In
caz contrar, se efectueaza atribuirea k←k-1. Apoi, in cazul in care k>0,
se repeta pasul curent; in cazul in care k=0, se trece la pasul G2.2
G2.1 {Omiterea acestui sufix}
     In cazul in care ck=k, se aplica p(k,k)¯ lui a0a1…an-1, se efectueaza
atribuirile ck←0, k←k+1 ;I se repeta operatiile pana cand ck<k.
Algoritmul se incheie in cazul in care k=n; in caz contrar, se efectueaza
atribuirea ck←ck+1, se aplica q(k,ck) lui a0a1…an-1 si se revine la pasul
G2.0.
G2.2 {vizitarea}
     Se viziteaza permutarea a0a1…an-1.
```

La pasul G1 ar mai trebui efectuată atribuirea k←n-1. Observați că noii paşi sunt definiți atent, pentru a respecta condiția (17). Algoritmul a devenit mai complicat, deoarece trebuie să cunoaştem permutările q(k,j) şi p(k,k) pe lângă permutările $q(k,j)u(k-1)^{-}$, care apar la pasul G4. Complicațiile suplimentare merită însă adesea efortul, deoarece programul rezultat rulează semnificativ mai rapid.

Bibliografie

1. Donald E. Knuth, Arta programării calculatoarelor, vol. 3, Sortare şi Căutare, Ed Teora, 2002
2. Donald E. Knuth, Arta programării calculatoarelor, vol. 4, fascicola 2, Generarea tuturor tuplurilor şi permutărilor, Ed Teora, 2005