```
  if b = 1 then
  begin
    p := sqrt(z);  d := 0.5 × z × p;  p := 1 − p
  end else
  begin
    d := z × z;  p := w × z
  end;
  y := 2 × w/z;
  for j := b + 2 step 2 until n do
  begin
    d := (1 + a/(j−2)) × d × z;
    p := if a = 1 then p + d × y/(j−1) else (p+w) × z
  end j;
  y := w × z;  z := 2/z;  b := n − 2;
  for i := a + 2 step 2 until m do
  begin
    j:= i + b;  d := y × d × j/(i−2);  p := p − z × d/j
  end i;
  Fisher := p
end Fisher
```

ALGORITHM 323
GENERATION OF PERMUTATIONS IN LEXICOGRAPHIC ORDER [G6]

R. J. Ord-Smith (Recd. 27 Apr. 1967 and 26 July 1967)
Computing Laboratory, University of Bradford, Bradford, Yorkshire, England

KEY WORDS AND PHRASES: permutations, lexicographic order, lexicographic generation, permutation generation
CR CATEGORIES: 5.39

Author's Remark. Lexicographic generation involves more than the minimum of $n!$ transpositions for generation of the complete set of $n!$ permutations of n objects. The actual number of transpositions required can be shown to tend asymptotically to (cosh 1) $n! \doteqdot 1.53n!$ However, lexicographic generation can be described by an algorithm requiring very simple book-keeping. The author is indebted to Professor H. F. Trotter for suggesting an improvement to an original algorithm, which now results in a process more than twice as fast as the previously fastest lexicographic Algorithm 202 [*Comm. ACM 6* (Sept. 1963), 517]. Tabulated results below show *BESTLEX* to be only 9.3 percent slower than the transposition Algorithm 115 [*Comm. ACM 5* (Aug. 1962), 434] when $n = 8$.

The usual practice is adopted of using a nonlocal Boolean variable called *first* which may be assigned the value *true* to initialize generation. On procedure call this is set *false* and remains so until it is again set *true* when complete generation of permutations has been achieved. Table I gives results obtained for *BESTLEX*. The times given in seconds are for an I.C.T. 1905 computer. t_n is the time for complete generation of $n!$ permutations. r_n has the usual definition $r_n = t_n/(n \cdot t_{n-1})$.

TABLE I

Algorithm	t_7	t_8	r_8	*Number of transpositions*
BESTLEX	6	47	0.98	→ 1.53$n!$
202	12.4	100	1.00	?
115	5.6	43	0.98	$n!$

```
procedure BESTLEX (x, n);  value n;  integer n;  array x;
begin own integer array q[2:n];  integer k, m;  real t;
comment own dynamic arrays are not often implemented. The
  upper bound will then have to be given explicitly;
  if first then
  begin first := false;
    for m := 2 step 1 until n do q[m] := 1
  end of initialization process;
  if q[2] = 1 then
  begin q[2] := 2;
    t := x[1];  x[1] := x[2];  x[2] := t;
    go to finish
  end;
  for k := 2 step 1 until n do
    if q[k] = k then q[k] := 1  else go to trstart;
first := true;  k := n;  go to trinit;
trstart:  m := q[k];  t := x[m];  x[m] := x[k];  x[k] := t;
  q[k] := m + 1;  k := k − 1;
trinit:  m := 1;
transpose:  t := x[m];  x[m] := x[k];  x[k] := t;
  m := m + 1;  k := k − 1;
  if m < k then go to transpose;
finish:
end of procedure BESTLEX
```

ALGORITHM 324
MAXFLOW [H]

G. Bayer (Recd. 31 July 1967)
Technische Hochschule, Braunschweig, Germany

KEY WORDS AND PHRASES: network, linear programming, maximum flow
CR CATEGORIES: 5.41

```
procedure maxflow (from, to, cap, flow, v, n, mflow, source, sink,
inf, eps);
  value v, n, source, sink, inf;
  integer v, n, source, sink;  real inf, eps, mflow;
  integer array from, to;  array cap, flow;
comment The nodes of the network are numbered from 1 to sn.
  It is not necessary but reasonable that each number represent a
  node. The data of the network are given by arrays from, to, cap
  in the following manner. There is a maximum possible flow of
  cap[i], nonnegative, leading from from[i] to to[i], i = 1, ··· , v.
    Compute the maximum flow mflow from source to sink,
  (source and sink given by their node numbers). inf represents
  the greatest positive real number within machine capacity.
  flow[i] gives the actual flow from from[i] to to[i]. Flows abso-
  lutely less than eps are considered to be zero. Literature: G.
  Hadley, Linear Programming, Addison-Wesley, Reading (Mass.)
  and London, 1962, pp. 337–344.
    Multiple solutions are left out of account;
begin integer l, j, k, r, lk, ek, u, s;  real gjk, d;
  integer array low, up, klist, labj[1:n], ind[1:v];  real array
  labf[1:n];
comment Note structure of data lists in up and low;
  l := 1;
  for j := 1 step 1 until n do
  begin low[j] := l;
    for r := 1 step 1 until v do
    begin if from[r] = j then
      begin ind[l] := r;
        flow[l] := cap[l];  l := l + 1
      end
```

```
      end;
      up[j] := l - 1
    end;
    mflow := 0.0;
lab:;
    comment Prepare lists for new labeling;
    for j := 1 step 1 until n do
    begin labj[j] := klist[j] := 0;
      labf[j] := 0.0
    end;
    labf [source] := inf;
    comment labeling;
    j := source;  lk := ek := 0;
path:
    u := up[j];
    for s := low[j] step 1 until u do
    begin l := ind[s];
      k := to[l];  gjk := flow[l];
      if labj[k] ≠ 0 ∨ abs(gjk) < eps
        then go to end;
      labj[k] := j;
      labf[k] := if gjk < labf[j] then gjk else labf[j];
      if k = sink then go to reached;
      lk := lk + 1;  klist[lk] := k;
end:
    end;
    ek := ek + 1;  j := klist[ek];
    if j ≠ 0 then go to path else go to max;
    comment sink is labeled, find path and possible
      flow, reduce excess capacities along path;
reached:
    j := sink;  d := labf[j];  mflow := mflow + d;
look:  k := labj[j];  u := up[k];
    for s := low[k] step 1 until u do
    begin l := ind[s];
      if to[l] = j then flow[l] := flow[l] - d
    end;
    u := up[j];
    for s := low[j] step 1 until u do
    begin l := ind[s];
      if to[l] = k then flow[l] := flow[l] + d
    end;
    j := k;  if j ≠ source then go to look;
    go to lab;
max:;  comment maximal flow found;
    for l := 1 step 1 until v do
      flow[l] := cap[l] - flow[l]
end
```

ALGORITHM 325
ADJUSTMENT OF THE INVERSE OF A SYMMETRIC MATRIX WHEN TWO SYMMETRIC ELEMENTS ARE CHANGED [F1]

GERHARD ZIELKE (Recd. 24 Aug. 1967)
Institut für Numerische Mathematik der Martin Luther Universität Halle-Wittenberg, German Democratic Republic

KEY WORDS AND PHRASES: symmetric matrix, matrix inverse, matrix perturbation, matrix modification
CR CATEGORIES: 5.14

```
procedure INVSYM 2 (n, i, j, c, a, b);
  value n, i, j, c;  integer n, i, j;  real c;  array a, b;
comment INVSYM 2 computes the inverse A^-1 = a of a non-
  singular symmetric nth order matrix A = B + c(e_i e_j' + e_j e_i')
  which arises from a symmetric matrix B by a change c in two
  elements B_ij and B_ji = B_ij (i ≠ j). The inverse matrix B^-1 = b
  is assumed to be known. The calculation with the new formula
```

$$a = b - \frac{c}{d}\left[b_{.i}(h_1 b_{j.} + h_2 b_{i.}) + b_{.j}(h_3 b_{j.} + h_1 b_{i.})\right]$$

where

$$h_1 = 1 + cb_{ij}\ ,\quad h_2 = -cb_{jj}\ ,\quad h_3 = -cb_{ii}\ ,\quad d = h_1{}^2 - h_2 h_3$$

requires $n^2 + O(n)$ multiplications, therefore only about the same number of operations as if the well-known Sherman-Morrison formula for a change in one element (see Algorithm 51 [*Comm. ACM 4* (Apr. 1961), 180]) is used. In these equations e_i denotes the ith column and e_i' the ith row of the unit matrix, $b_{.i} = be_i$ denotes the ith column and $b_{i.} = e_i'b$ the ith row of the matrix b;

```
begin integer k, l;  real h1, h2, h3, d;
  array r, s[1:n];
  h1 := 1 + c × b[i, j];  h2 := -c × b[j, j];
  h3 := -c × b[i, i];  d := h1 ↑ 2 - h2 × h3;  d := c/d;
  h1 := h1 × d;  h2 := h2 × d;  h3 := h3 × d;
  for k := 1 step 1 until n do
  begin
    r[k] := h1 × b[j, k] + h2 × b[i, k];
    s[k] := h3 × b[j, k] + h1 × b[i, k]
  end;
  for k := 1 step 1 until n do
  for l := 1 step 1 until k do
    a[k, l] := a[l, k] := b[k, l] - b[k, i] × r[l] - b[k, j] × s[l]
end INVSYM 2
```

MODIFIED SHARE CLASSIFICATIONS

[*Designations follow algorithm titles.*]

A1	Real Arithmetic, Number Theory
A2	Complex Arithmetic
B1	Trig and Inverse Trig Functions
B2	Hyperbolic Functions
B3	Exponential and Logarithmic Functions
B4	Roots and Powers
C1	Operations on Polynomials and Power Series
C2	Zeros of Polynomials
C5	Zeros of One or More Transcendental Equations
C6	Summation of Series, Convergence Acceleration
D1	Quadrature
D2	Ordinary Differential Equations
D3	Partial Differential Equations
D4	Differentiation
E1	Interpolation
E2	Curve and Surface Fitting
E3	Smoothing
E4	Minimizing or Maximizing a Function
F1	Matrix Operations, Including Inversion
F2	Eigenvalues and Eigenvectors of Matrices
F3	Determinants
F4	Simultaneous Linear Equations
F5	Orthogonalization
G1	Simple Calculations on Statistical Data
G2	Correlation and Regression Analysis
G5	Random Number Generators
G6	Permutations and Combinations
G7	Subset Generators and Classifications
H	Operations Research, Graph Structures
I5	Input—Composite
J6	Plotting
K2	Relocation
M1	Sorting
M2	Data Conversion and Scaling
O2	Simulation of Computing Structure
S	Approximation of Special Functions... Functions are Classified S01 to S22, Following Fletcher-Miller-Rosenhead, Index of Math. Tables
Z	All Others