**ALGORITHM 97**

**SHORTEST PATH**

ROBERT W. FLOYD

Armour Research Foundation, Chicago, Ill.

**procedure** shortest path (m, n); value n; integer m, n, In, r;

**comment** Initially m[i, j] is the length of a direct link from point i of a network to point j. If no direct link exists, m[i, j] is initially s10. At completion, m[i, j] is the length of the shortest path from i to j. If none exists, m[i, j] is s10. Reference: WARSHALL, S. A theorem on Boolean matrices, J. ACM 9(1962), 11-12;

begin
  integer i, j, k;
  for i := 1 step 1 until n do
    for j := 1 step 1 until n do
      if m[i, j] < inf then
        m[i, j] := inf;
  integer s := m[i, i];
  for i := 1 step 1 until n do
    for j := 1 step 1 until n do
      if m[i, j] < inf then
        m[i, j] := s;
  end shortest path

Contributions to this department must be in the form stated in the Algorithms Department policy statement (Communications, February, 1960) except that ALGOL 60 notation should be used (see Communications, May 1960).

Contributions should be sent in duplicate to J. H. Wegstein, Computation Laboratory, National Bureau of Standards, Washington 25, D. C. Algorithms should be in the Reference form of ALGOL 60 and written in a style patterned after the most recent algorithms appearing in this department. For the convenience of the printer, please underline words that are delimiters to appear in boldface type.

Although each algorithm has been tested by its contributor, no warranty, expressed or implied, is made by the contributors, the editor, or the Association for Computing Machinery as to the accuracy and functioning of the algorithm and related algorithm material, and no responsibility is assumed by the contributor, the editor, or the association for Computing Machinery in connection therewith.

The reproduction of algorithms appearing in this department is explicitly permitted without any charge. When reproduction is for publication purposes, reference must be made to the algorithm's author and to the Communications issue bearing the algorithm.

**ALGORITHM 98**

**EVALUATION OF DEFINITE COMPLEX LINE INTEGRALS**

JOHN L. PFALTZ

Syracuse University Computing Center, Syracuse, N. Y.

**procedure** COMPLINEINTGRL(A, B, N, RSSUM);

value A, B, N, array RSSUM;

**comment** COMPLINEINTGRL approximates the complex line integral by evaluating the partial Riemann-Stieltjes sum \( \sum_{i=1}^{n} f(z_i) \Delta z_i \) where \( a \leq t \leq b \) and \( z_i \in (z_{i-1}, z_i) \). The programmer must provide 1) the procedures GAMMA(T, Z) to calculate \( \Gamma(t) \) on \( \Gamma \), and FUNCT(Z, F) to calculate function values, and 2) the end points \( A \) and \( B \) of the parametric interval and \( N \) the number of subintervals into which \([a, b]\) is to be partitioned;

begin integer I; real T, DELT; real array ZT, ZTL, DELZ, ZK, PART[1:2]; RSSUM[1] := 0.0; RSSUM[2] := 0.0; DELT := (B - A)/N; T := A;

line: GAMMA(T, ZT);

if T = A then go to next;

for I := 1 step 1 until 2 do

begin
  DELZ[I] := ZT[I] - ZTL[I];
end;

for I := 1 step 1 until 2 do

begin
  ZK[I] := ZT[I] + DELZ[I]/2.0;
end;

for I := 1 step 1 until 2 do

begin
  FUNCT(ZK, FZ);
end;


for I := 1 step 1 until 2 do

begin
  RSSUM[I] := RSSUM[I] + PART[I];
end;

if T < B - (0.25 X DELT) then go to next else go to exit;

next: for I := 1 step 1 until 2 do

begin
  ZTL[I] := ZT[I];
end;

T := T + DELT;

go to line;

exit: end COMPLINEINTGRL.

**ALGORITHM 99**

**EVALUATION OF JACOBI SYMBOL**

STEPHEN J. GARLAND AND ANTHONY W. KNAPP

Dartmouth College, Hanover, N. H.

**procedure** Jacobi(n, m, r); value n, m;

integer n, m, r;

**comment** Jacobi computes the value of the Jacobi symbol \( (n/m) \), where \( m \) is odd, by the law of quadratic reciprocity. The parameter \( r \) is assigned one of the values \(-1, 0, 1\) if \( m \) is odd. If \( m \) is even, the symbol is undefined and \( r \) is assigned the value 2. For odd \( m \) the routine provides a test of whether \( m \) and \( n \) have a nontrivial common factor. In the special case where \( m \) is prime, \( r = -1 \) if and only if \( n \) is a quadratic nonresidue of \( m \);

begin integer s;

Boolean p, q;

**Boolean procedure** parity(x); value x; integer x;

**comment** The value of the function parity is true if \( x \) is odd, false if \( x \) is even;

begin
  parity := x \div 2 \times 2 \neq x
end parity;

Communications of the ACM 345
if \( \neg \text{parity}(m) \) then begin \( r := 2 \); go to exit end;
\( p := \text{true} \);
loop: \( n := n - n + m \times m; \)
\( q := \text{false}; \)
if \( n \leq 1 \) then go to done;
even: if \( \neg \text{parity}(n) \) then begin
\( q := \neg q; \)
\( n := n - 2; \)
go to even
end n now odd;
end if q then if parity \( ((m?2 - 1) \div 8) \) then \( p := \neg p; \)
if \( n = 1 \) then go to done;
if parity \( ((m-1) \times (n-1) + 4) \) then \( p := \neg p; \)
\( s := m; \)
\( m := n; \)
\( n := s; \)
go to loop;
done: \( r := \text{if } n = 0 \text{ then } 0 \text{ else if } p \text{ then } 1 \text{ else } -1; \)
exit: end Jacobi

ALGORITHM 100
ADD ITEM TO CHAIN-LINKED LIST
PHILIP J. KIVIAT

procedure inlist (t,info,m,list,n,first,flag,addr,listfull);
integer n,m,first,flag,t; integer array info,list,addr;
comment inlist adds the information pair \( \{t,\text{info}\} \) to the chain-link structured matrix list \( \{(i,j)\} \), where \( t \) is an order key \( \geq 0 \), and info\( (k) \) is an information vector associated with \( t \). info\( (k) \) has dimension \( m \), list\( (i,j) \) has dimensions \( (n \times (m+3)) \). flag denotes the head and tail of list\( (i,j) \), and first contains the address of the first (lowest order) entry in list\( (i,j) \). addr\( (k) \) is a vector containing the addresses of available (empty) rows in list\( (i,j) \). Initialization: list\((m+3)2 := \text{flag}, \text{for some } i \leq n \). If list\((i,j) \) is filled exit is to listfull;
begin integer i, j, link1, link2;
0: if addr\([1] = 0; \text{then go to listfull;} \); i := 1;
1: if list\([i,1] \leq t \]
then begin
if list\(([i,2] \neq 0 \text{ then begin } \text{link1} := m+2; \)
link2 := m+3; \text{go to 2 end; else begin if } \)
list\([i,m+2] = \text{flag then begin } i := \text{flag; } \)
link1 := m+3; \text{link2} := m+2; \text{go to 3 end; else begin } \)
begin i := i+1; \text{go to 1 end end end; else begin } \)
begin link1 := m+3; \text{link2} := m+2; \text{end end; end if } \)
end if list\([i,link2] \neq \text{flag} \]
then begin
\( k := i; \)
\( i := \text{link}[i,link2]; \)
if \( \text{link2} \neq \text{m+2} \land \text{link}[i,1] \leq t \lor \)
\( \text{link2} \neq \text{m+2} \land \text{link}[i,1] > t \) then go to 4;
else go to 1 end;
else begin
list\([i,link2] := \text{addr}[i] end;
3: \( j := \text{addr}[i]; \)
list\([j,link1] := i; \)
list\([j,link2] := \text{flag}; \)
if \( \text{link2} = \text{m+2} \text{ then first := addr}[i]; \)
go to 5;
4: \( j := \text{addr}[i]; \)
list\([j,link1] := \text{list}[i,link1]; \)
list\([j,link2] := \text{list}[k,link2]; \)
\( \text{addr}[i] := \text{addr}[i] + 1 \);
5: list\([i,1] := t; \)
for i := 1 step 1 until m do
list\([i+1] := \text{info}[i]; \)
for i := 1 step 1 until n-1 do
\( \text{addr}[i] := \text{addr}[i+1]; \)
\( \text{addr}[n] := 0 \)
end inlist

ALGORITHM 101
REMOVE ITEM FROM CHAIN-LINKED LIST
PHILIP J. KIVIAT

procedure outlist (vector,m,list,n,first,flag,addr);
integer n,m,first,flag; integer array vector,list,addr;
comment outlist removes the first entry (information pair with lowest order key) from list\( (i,j) \) and puts it in vector\( (k) \);
begin integer i;
for i := 1 step 1 until m+1 do vector\([i] := \text{list}[first,i]; \)
for i := n-1 step -1 until 1 do addr\([i+1] := \text{addr}[i]; \)
\( \text{addr}[1] := \text{first}; \)
end if \text{list}[first,m+3] = \text{flag} \then
begin list\([1,m+2] := \text{flag}; \)
\( \text{first} := 1; \)
for i := 1 step 1 until n do addr\([i] := i \text{ end; } \)
else begin
\( \text{first} := \text{list}[first,m+3]; \)
\( \text{list}[first,m+2] := \text{flag end}; \)
for i := 1 step 1 until m+3 do list\([addr[1],i] := 0 \text{ end outlist; } \)

ALGORITHM 102
PERMUTATION IN LEXICOGRAPHICAL ORDER
G. F. SCHRACK AND M. SHMRAT
University of Alberta, Calgary, Alberta, Canada

procedure PERMULEX(n,p);
integer n; integer array p;
comment Successive calls of the procedure will generate all permutations \( p \) of \( 1,2,3, \cdots, n \) in lexicographical order. Before the first call, the non-local Boolean variable ‘flag’ must be set to \text{true}. If after an execution of PERMULEX ‘flag’ is \text{false}, additional calls will generate further permutations—if \text{true}, all permutations have been obtained;
begin integer array q[1:n]; integer i, k, t; Boolean flag2;
if flag then
begin for i := 1 step 1 until n do
\( p[i] := i; \)
flag2 := \text{true}; \text{flag} := \text{false};
go to EXIT;
end initialize;
if flag2 then
begin t := \text{p[n]; p[n] := p[n-1]; p[n-1] := t; \text{flag2} := \text{false}; \text{go to EXIT; end}}
end bypass;
flag2 := \text{true}; \text{for i := n-2 step -1 until 1 do if } \text{p[i]} < \text{p[i+1]} \text{ then go to A; \text{flag} := \text{true}; \text{go to EXIT; \text{A: for k := 1 step 1 until n do q[k] := 0; for k := 1 step 1 until n do q[p[k]] := p[k]; for k := p[i]+1 step 1 until n do if } q[k] \neq 0 \text{ then begin i := i+1; p[i] := q[k]; end else if } i \geq n \text{ then go to EXIT;\text{B: p[i] := k; q[k] := 0; for k := 1 step 1 until n do if } q[k] \neq 0 \text{ then begin i := i+1; p[i] := q[k] end else if } i \geq n \text{ then go to EXIT;}}\text{end PERMULEX.}}\)