```
A travelling salesman must visit 7 customers in 7 different locations whose
(symmetric) distance matrix is:

  1  2  3  4  5  6  7
1 - 86 49 57 31 69 50
2    - 68 79 93 24 5
3       - 16  7 72 67
4          - 90 69 1
5             - 86 59
6                - 81

Formulate a mathematical program to determine a visit sequence starting at
ending at location 1, which minimizes the travelled distance, and solve it
with AMPL. Knowing that the distances obey a triangular inequality and are
symmetric, propose a suitable heuristic method.
# tsp.mod
param n > 0, integer;
set V := 1..n;
param d{V,V} >= 0;
param subtours >= 0, integer, default 0;
set S{1..subtours};
var x{V,V} binary;
minimize distance : sum{i in V, j in V : i != j} d[i,j] * x[i,j];
subject to successore {i in V} : sum{j in V : i != j} x[i,j] = 1;
subject to predecessor {j in V} : sum{i in V : i != j} x[i,j] = 1;
subject to subtour_elim {k in 1..subtours} :
  sum{i in S[k], j in V diff S[k]} x[i,j] >= 1;
# tsp.dat
param n := 7;
param d : 1 2 3 4 5 6 7 :=
1         0 86 49 57 31 69 50
2         0 0 68 79 93 24 5
3         0 0 0 16 7 72 67
4         0 0 0 0 90 69 1
5         0 0 0 0 0 86 59
6         0 0 0 0 0 0 81
7         0 0 0 0 0 0 0 ;
model tsp.mod;
data tsp.dat;
for {i in V, j in V : i > j} {
  let d[i,j] := d[j,i];
}
option solver cplex;
solve;
display distance;
#display x;

param i default 1;
printf "Egy kÃ¶r: 1, ";
repeat while (1) {
   let i:= sum {j in V} j*x[i,j];
   if (i==1) then break;
   printf "%d, ",  i;
}
printf "1.\n\n";


let subtours := 1;
let S[1] := { 1, 3, 5 };
solve;

printf "Hamilton kÃ¶r: 1, ";
repeat while (1) {
   let i:= sum {j in V} j*x[i,j];
   if (i==1) then break;
   printf "%d, ",  i;
}
printf "1.\n\n";# tsp.run
model tsp.mod;
```



```
data tsp.dat;
# distance matrix must be symmetric
for {i in V, j in V : i > j} {
  let d[i,j] := d[j,i];
}

option solver cplex;
# don't print CPLEX messages
option solver_msg 0;
# data structures required for the algorithm
let subtours := 0;
param successorvertex{V} >= 0, integer;
param currentvertex >= 0, integer;
param termination binary;
let termination := 0;
# iterative part
repeat while (termination = 0) {
  # solve the problem
  solve;
  let subtours := subtours + 1;
  # find the successors of each vertex
  for {i in V} {
    let successorvertex[i] := sum{j in V : j != i} j * x[i,j];
  }
  # find a subtour
  let currentvertex := 1;
  let S[subtours] := {};
  repeat {
    let S[subtours] := S[subtours] union {currentvertex};
    let currentvertex := successorvertex[currentvertex];
  } until (currentvertex = 1);
  # print the subtour we wish to eliminate
  printf "(sub)tour: (";
  for {i in S[subtours]} {
    printf "%d -> ", i ;
  }
  printf "1)\n";
  # termination condition
  if (card(S[subtours]) >= n) then {
    # Hamiltonian tour, terminate
    let termination := 1;
  }
}
printf "travelled distance in the optimal tour: %f\n", distance;
```