NOPT042 Constraint programming: Tutorial 8 - Global constraints, Routing

What was in the lecture? Nothing, the lecture was canceled.

In [1]:
```
%load_ext ipicat
```

```
Picat version 3.9
```

Exercise: Magic sequence

A magic sequence of length n is a sequence of integers $x_0, \ldots, x_{n-1}$ between 0 and $n-1$, such that for all $i \in \{0, \ldots, n-1\}$, the number i occurs exactly x_i-times in the sequence. For instance, 6,2,1,0,0,0,1,0,0,0 is a magic sequence since 0 occurs 6 times in it, 1 occurs twice, etc.

(Problem from the book.)

We want to maximize the sum of the numbers in the sequence. Write a constraint model that does that (for a given length n).

The constraint `global_cardinality`

```
global_cardinality(List, Pairs)
```

Let `List` be a list of integer-domain variables `[X1, . . ., Xd]`, and `Pairs` be a list of pairs `[K1-V1, . . ., Kn-Vn]`, where each key `Ki` is a unique integer, and each `Vi` is an integer-domain variable. The constraint is true if every element of List is equal to some key, and, for each pair `Ki-Vi`, exactly `Vi` elements of `List` are equal to `Ki`. This constraint can be defined as follows:

```
global_cardinality(List,Pairs) =>
    foreach($Key-V in Pairs)
        sum([B : E in List, B#<=>(E#=Key)]) #= V
    end.
```

---from the guide

In [2]:
```
!time picat magic-sequence/.solution/magic-sequence.pi 4
```

```
CPU time 0.0 seconds. Backtracks: 2

[1,2,1,0]

real	0m0.024s
user	0m0.008s
sys	0m0.013s
```

In [3]:
```
!cat magic-sequence/.solution/magic-sequence.pi
```

```
% Adapted from Constraint Solving and Planning with Picat, Springer
% by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
import cp.

main([N]) =>
  N := N.to_int,
  magic_sequence(N,Sequence),
  println(Sequence).

magic_sequence(N, Sequence) =>
  Sequence = new_list(N),
  Sequence :: 0..N-1,

  % create list: [0-Sequence[1], 1-Sequence[2], ...]
  Pairs = [$I-Sequence[I+1] : I in 0..N-1],
  global_cardinality(Sequence,Pairs),

  time2(solve(Sequence)).
```

In [4]:
```
!time picat magic-sequence/.solution/magic-sequence.pi 64
!time picat magic-sequence/.solution/magic-sequence2.pi 64
!time picat magic-sequence/.solution/magic-sequence2.pi 400
```

```
CPU time 12.676 seconds. Backtracks: 1980

[60,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]

real	0m11.281s
user	0m12.698s
sys	0m0.006s
CPU time 0.006 seconds. Backtracks: 7

[60,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]

real	0m0.048s
user	0m0.035s
sys	0m0.008s
```

```
CPU time 0.235 seconds. Backtracks: 7

[396,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]

real    0m7.006s
user    0m5.943s
sys     0m0.525s
```

In [5]:
```
!cat magic-sequence/.solution/magic-sequence2.pi
```

```
% Adapted from Constraint Solving and Planning with Picat, Springer
% by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
import cp.

main([N]) =>
  N := N.to_int,
  magic_sequence(N,Sequence),
  println(Sequence).

magic_sequence(N, Sequence) =>
  Sequence = new_list(N),
  Sequence :: 0..N-1,

  % create list: [0-Sequence[1], 1-Sequence[2], ...]
  Pairs = [$I-Sequence[I+1] : I in 0..N-1],
  global_cardinality(Sequence,Pairs),

  % extra/redudant (implicit) constraints to speed up the model
  N #= sum(Sequence),
  Integers = [I : I in 0..N-1],
  scalar_product(Integers, Sequence, N),

  time2(solve([ff], Sequence)).
```

In [6]:
```
!time picat magic-sequence/.solution/magic-sequence2.pi 1024
!time picat magic-sequence/.solution/magic-sequence3.pi 1024
```

```
CPU time 3.298 seconds. Backtracks: 7

[1020,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]
real    3m43.417s
user    3m15.928s
sys     0m26.249s
```

```
CPU time 0.73 seconds. Backtracks: 7

[1020,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0]

real	0m2.241s
user	0m3.858s
sys	0m0.498s
```

In [7]:
```
!cat magic-sequence/.solution/magic-sequence3.pi
```

```
% Adapted from Constraint Solving and Planning with Picat, Springer
% by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
import cp.

main([N]) =>
  N := N.to_int,
  magic_sequence(N,Sequence),
  println(Sequence).

magic_sequence(N, Sequence) =>
  Sequence = new_list(N),
  Sequence :: 0..N-1,

  % extra/redudant (implicit) constraints to speed up the model
  N #= sum(Sequence),
  Integers = [I : I in 0..N-1],
  scalar_product(Integers, Sequence, N),

  % % create list: [0-Sequence[1], 1-Sequence[2], ...]
  Pairs = [$I-Sequence[I+1] : I in 0..N-1],
  global_cardinality(Sequence,Pairs),

  time2(solve([ff], Sequence)).
```

The order of constraints

The order might matter to the solver, the above model is an example. When the `cp` solver parses a constraint, it tries to reduce the domains. The implicit (redundant) constranint using `scalar_product` can reduce the model a lot, which is much better to do before parsing `global_cardinality`. (Note: e.g. MiniZinc doesn't preserve the order of constraints during compilation, the behaviour is a bit unpredictable.)

Heuristic: easy constraints and constraints that are strong [in reducing the search space] should go first (?)

Routing

Routing, aka the Arc Routing Problem, refers to a variety of constraint satisfaction problems related to the well-known Vehicle Routing Problem. Such problems appear in a variety of applications such as public transport (bus routing), garbage collection (not that one), snow ploughing, and many others.

Exercise: Knight tour

Given an integer N, plan a tour of the knight on an $N \times N$ chessboard such that the knight visits every field exactly once and then returns to the starting field. You can assume that N is even.

(Hint: For a matrix `M` is a matrix, you can use `M.vars()` to extract its elements into a list.)

The `circuit` constraint

The constraint `circuit(L)` requires that the list L represents a permutation of $1, \ldots, n$ consisting of a single cycle, i.e., the graph with edges $i \to L[i]$ is a cycle. A similar constraint is `subcircuit(L)` which requires that elements for which $L[i] \neq i$ form a cycle.

In [8]:
```
!picat knight-tour/.solution/knight-tour.pi 8
```

```
x = {{11,12,13,10,15,21,22,14},{3,20,26,18,23,4,30,6},{2,1,29,5,27,32,8,7},
{19,9,37,34,35,40,16,38},{43,17,50,53,52,28,24,46},{51,25,49,61,60,36,62,3
1},{59,33,57,58,63,64,45,39},{42,41,44,54,55,56,48,47}}
X:
 11 12 13 10 15 21 22 14
  3 20 26 18 23  4 30  6
  2  1 29  5 27 32  8  7
 19  9 37 34 35 40 16 38
 43 17 50 53 52 28 24 46
 51 25 49 61 60 36 62 31
 59 33 57 58 63 64 45 39
 42 41 44 54 55 56 48 47

Tour:
  1 62  5 10 13 24 55  8
  4 11  2 63  6  9 14 23
 61 64 35 12 25 56  7 54
 34  3 26 59 36 15 22 57
 39 60 37 18 27 58 53 16
 30 33 40 43 46 17 50 21
 41 38 31 28 19 48 45 52
 32 29 42 47 44 51 20 49
```

In [9]:
```
!cat knight-tour/.solution/knight-tour.pi
```

```
/************************************************************
  Adapted from
  knight_tour.pi
  from Constraint Solving and Planning with Picat, Springer
  by Neng-Fa Zhou, Hakan Kjellerstrand, and Jonathan Fruhman
************************************************************/
import cp.

main([N]) =>
  N := N.to_int,
  knight(N,X),
  println(x=X),
  println("X:"),
  print_matrix(X),
  extract_tour(X,Tour),
  println("Tour:"),
  print_matrix(Tour).

% Knight's tour for even N*N.
knight(N, X) =>
  X = new_array(N,N),
  X :: 1..N*N,
  XVars = X.vars(),
  % restrict the domains of each square
  foreach (I in 1..N, J in 1..N)
     D = [-1,-2,1,2],
     Dom = [(I+A-1)*N + J+B : A in D, B in D,
             abs(A) + abs(B) == 3,
             member(I+A,1..N), member(J+B,1..N)],
     Dom.length > 0,
     X[I,J] :: Dom
  end,
  circuit(XVars),
  solve([ff,split],XVars).

extract_tour(X,Tour) =>
  N = X.length,
  Tour = new_array(N,N),
  K = 1,
  Tour[1,1] := K,
  Next = X[1,1],
  while (K < N*N)
    K := K + 1,
    I = 1+((Next-1) div N),
    J = 1+((Next-1) mod N),
    Tour[I,J] := K,
    Next := X[I,J]
  end.

print_matrix(M) =>
  N = M.length,
  V = (N*N).to_string().length,
  Format = "% " ++ (V+1).to_string() ++ "d",
  foreach(I in 1..N)
     foreach(J in 1..N)
        printf(Format,M[I,J])
```

```
    end,
    nl
 end,
 nl.
```

Exercise: Chinese Postman

Given a weighted graph, find the shortest closed path that traverses every edge at least once. (Assume that the graph is undirected, loopless, and the distances are positive integers and satisfy the triangle inequality.) See Wikipedia for a sample solved instance. Unlike the Eulerian Circuit Problem (where each edge has to be traversed exactly once), there is a polynomial-time algorithm for the Chinese Postman. But our goal is to write a constraint model.

In [10]:
```
!cat chinese-postman/instance.pi
```

```
instance(NumVertices, AdjacencyMatrixWithWeights) =>
    NumVertices = 5,
    AdjacencyMatrixWithWeights = [
        [0, 7, 0, 5, 1],
        [7, 0, 6, 0, 3],
        [0, 6, 0, 8, 4],
        [5, 0, 8, 0, 2],
        [1, 3, 4, 2, 0]
    ].
% optimal length: 45, shortest closed path: [1,2,3,4,5,3,2,5,1,5,4,1]
```