NOPT042 Constraint programming: Tutorial 5 – Advanced constraint modelling

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

```
Picat version 3.2#8
```

The element constraint

Picat doesn't support indexation in constratins, e.g. `X #= L[I]`. Instead, it implements the `element` constraint:

```
element(I, L, x)
```

("X is on the Ith position in the list L"). But this constraint can also be used for "reverse indexing": when we have X and want to find its position in L. The constraint doesn't care about the direction; this is called *bidirectionality*.

For matrices (2D arrays), `Matrix[I,J]#=V` is expressed using the following constraint:

```
matrix_element(Matrix,I,J,V)
```

Example: Langford's number problem

Consider the following problem (formulation slightly modified from the book):

> Consider two sets of the numbers from 1 to N. The problem is to arrange the $2N$ numbers

in the two sets into a single sequence in which the two 1's appear one number apart, the two 2's appear two numbers apart, the two 3's appear three numbers apart, etc.

Try to formulate a model for this problem.

In [2]:
```
!picat langford/langford.pi 8
```

```
CPU time 1.96 seconds. Backtracks: 250746

[1,3,1,6,7,3,8,5,2,4,6,2,7,5,4,8]
```

In [3]:
```
!picat langford/langford2.pi 8
```

```
CPU time 0.0 seconds. Backtracks: 55

solution = [1,3,1,6,7,3,8,5,2,4,6,2,7,5,4,8]
position = [1,9,2,10,8,4,5,7,3,12,6,15,14,11,13,16]
```

In [4]:
```
!picat langford/langford2.pi 12
```

```
CPU time 0.031 seconds. Backtracks: 1346

solution = [1,2,1,3,2,8,9,3,10,11,12,5,7,4,8,6,9,5,4,10,7,11,6,12]
position = [1,2,4,14,12,16,13,6,7,9,10,11,3,5,8,19,18,23,21,15,17,20,22,24]
```

The assignment problem

From Wikipedia:

The assignment problem is a fundamental combinatorial optimization problem. In its most general form, the problem is as follows:

> The problem instance has a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. It is required to perform as many tasks as possible by assigning at most one agent to each task and at most one task to each agent, in such a way that the total cost of the assignment is minimized.

Alternatively, describing the problem using graph theory:

> The assignment problem consists of finding, in a weighted bipartite graph, a matching of a given size, in which the sum of weights of the edges is minimum.

If the numbers of agents and tasks are equal, then the problem is called balanced assignment.

Example: Swimmers

(From: W. Winston, Operations Research: Applications & Algorithms.)

In medley swimming relay, a team of four swimmers must swim 4x100m, each swimmer using a different style: breaststroke, backstroke, butterfly, or freestyle. The table below gives their average times for 100m in each style. Which swimmer should swim which stroke to minimize total time?

Swimmer	Free	Breast	Fly	Back
A	54	54	51	53
B	51	57	52	52
C	50	53	54	56
D	56	54	55	53

Write a general model, generate larger instances, and try to make your model as efficient as possible.

```
In [5]: !cd swimmers && picat instances.pi
```

```
{A,B,C,D}
{Free,Breast,Fly,Back}
{{54,54,51,53},{51,57,52,52},{50,53,54,56},{56,54,55,53}}
{Swimmer1,Swimmer2,Swimmer3,Swimmer4,Swimmer5}
{Style1,Style2,Style3,Style4,Style5,Style6,Style7}
{{46,49,54,64,53,55,55},{54,60,55,47,64,65,49},{65,52,48,60,61,61,62},{59,57,54,47,50,5
0,58},{46,64,50,45,48,57,62}}
```

```
In [6]: !cd swimmers && picat swimmers.pi
```

```
Primal model:
Swimmer A is swims Fly
Swimmer B is swims Back
Swimmer C is swims Free
Swimmer D is swims Breast

Dual model:
Style Free is swum  by C
Style Breast is swum  by D
Style Fly is swum  by A
Style Back is swum  by B

Channeling model:
Swimmer A is swims Fly
Swimmer B is swims Back
Swimmer C is swims Free
Swimmer D is swims Breast
or in the dual view
Style Free is swum  by C
Style Breast is swum  by D
Style Fly is swum  by A
Style Back is swum  by B
```

Modelling functions

In general, how to model a function (mapping) $f : A \to B$? Let's say $A = \{1, \ldots, n\}$ and $B = \{1, \ldots, k\}$.

- as an array:

  ```
  F = new_array(N},
  F :: 1..K.
  ```

- *injective*: `all_different(F)`
- *surjective*: a partition of A into classes labelled by B, to each element of B map a set of elements of A. In Picat we can model set as their characteristic vectors. More on modelling with sets later.
- *partial function*: a dummy value for undefined inputs
- *dual model*: switch the role of variables and values (not a function unless F injective, see above)
- *channelling*: combine the primal and dual models, if it is a bijection, then use `assignment(F, FInv)`

In [7]:
```
!picat functions.pi 4 4
```

```
{1,2,3,4}
{1,2,3,4}
```

Exercises

Homework: stable-marriage

Stable marriage

Given n men and n women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. (When there are no such pairs, the matching is stable.)

The input is the name of a file defining an instance as in the following example. The output should be a list of pairs (2-el arrays) Husband-Wife.

Try the following instance which is given in `stable-marriage-instance.pi`

- A:YZX,
- B:ZYX,
- C:XZY,
- X:BAC,
- Y:CBA,
- Z:ACB

The output of `picat stable-marriage.pi stable-marriage-instance.pi` should be `[{1,3},{2,1},{3,2}]` (meaning that the 1st Man, i.e., A, is married to the third woman, i.e. Z, etc.)

```
In [8]: !cat stable-marriage-instance.pi
```

```
instance(N, RankOfWomenByMen, RankOfMenByWomen) =>
    N = 3,

    % RankOfWomenByMen[2, 3] = 1 means that Woman 3 is the most preferred woman of Man 2
    RankOfWomenByMen = {
        {2, 3, 1},
        {3, 2, 1},
        {1, 3, 2}
    },
    RankOfMenByWomen = {
        {2, 1, 3},
        {3, 2, 1},
        {1, 3, 2}
    }.
```