Declaratieve Talen
Prolog 3

1 Balanced trees

A tree can either be empty (`nil`) or exist of a node (`t`) with a value and two subtrees.

$$\text{Tree} := \text{nil} \mid \text{t}(\text{Tree}, \text{Value}, \text{Tree})$$

Hence, the expression `t(t(nil,2,nil),3,nil)` represents a tree. A tree is balanced if the depths of the left and right subtree differ by at most one, and both subtrees are balanced as well.

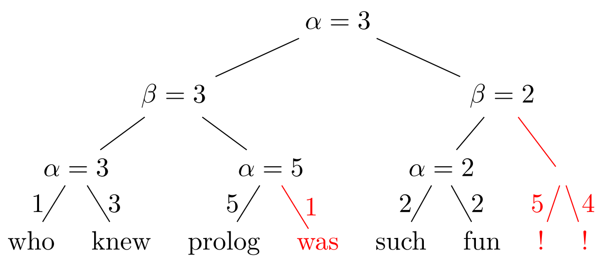
Implement a predicate `balanced/1` that succeeds if a given tree is balanced and fails in all other cases.

```
?- balanced(nil).
true.

?- balanced(t(nil,3,nil)).
true.

?- balanced(t(nil,3,t(nil,4,nil))).
true.

?- balanced(t(nil,3,t(nil,4,t(nil,2,nil)))).
false.
```

Implement a predicate `add_to/3` that adds an element to a balanced tree, and ensures that the tree remains balanced. The tree does not have to be sorted.

```
?- add_to(t(t(nil,3,nil),2,nil),4,Tree).
Tree = t(t(nil,3,nil),2,t(nil,4,nil))
```

Extra: Perform all required changes to store in each node both a value and the depth of the tree at that point. Adapt the `add_to/3` predicate such that its complexity decreases.



Figuur 1: The tree used in the exercise. Pruned branches are higlighted in red.

2 Alpha-Beta Pruning

Assume that we have two players, both playing optimally, one player tries to maximise his score, while the other tries to minimise it. The maximising player plays first. This process can be represented by a minmax tree.

The goal of this exercise is to implement the alpha-beta pruning algorithm. The algorithm maintains two values while exploring the minmax tree: the α-value is the largest score that the maximising player is sure he can achieve, while the β value is the smallest score that the minimising player is sure he can achieve. If, while exploring some node, β becomes less than or equal to α, then further exploration of this node is aborted by the algorithm, as the other player will never pick this node, since he already has a better choice availble. For more information, illustrations and pseudo-code, please consult the relevant Wikipedia article.

For this exercise, represent trees with the following Prolog terms:

- `leaf(S,V)` representing a leaf node with score `S` and value `V`.
- `max(L,R)` for a max-node with subtrees `L` and `R`. `min(L,R)` for a min-node with subtrees `L` and `R`.

Now, implement a predicate `alpha_beta/5` that prunes the tree, by replacing pruned branches with `nil` as soon as $\beta \leq \alpha$. For the tree shown in Figure 1, the output is as follows:

```
% alpha_beta(Tree,Alpha,Beta,Score,NewTree)
?- alpha_beta(max(
               min(
                max(leaf(1,who),leaf(3,knew)),
                max(leaf(5,prolog),leaf(1,was))),
               min(
                max(leaf(2,such),leaf(2,fun)),
                max(leaf(5,!),leaf(5,!)))),
               -10,10,S,T).
S = 3,
T = max(min(max(leaf(1,who),leaf(3,knew)),max(leaf(5,prolog),nil)),
    min(max(leaf(2,such),leaf(2,fun)),nil)).
```

Extra Extend the internal `node` to have more than two subtrees, by using a list instead of two explicit left and right trees.

3 N-queens problem

A chessboard consists of 64 squares, which are divided into 8 rows and 8 columns. In chess, a queen can move horizontally, vertically or diagonally. In each direction, she can travel as far as she wants. The 8-queens problem is a chess problem where 8 queens must be positioned on a chessboard in such a way that no queen can capture another queen in **one** move. In this exercise, we generalize the 8-queen problem to the N-queen problem, where N queens must be positioned on a chessboard consisting of N rows and N columns.

Board representation

The configuration of the queens on a chessboard can be represented as a list, where the index represents the column number and the value represents the row number. For example, the list `[2,4,1,3]` denotes a 4-queens configuration where the queens are in row 2 in column 1, row 4 in column 2, row 1 in column 3, and row 3 in column 4.

The proposed representation guarantees that the queens cannot attack each other in one of the four possible directions (i.e., horizontal, vertical, and diagonal in two directions). Which direction is that? How can the same restriction easily be enforced for one of the three other directions?

Assignment

Implement a predicate `queens(N,L)` that, for a given number of queens `N`, generates a list `L` representing a valid configuration of the N queens on a chessboard comprising N rows and N columns. The predicate should generate **all** solutions.

Naive solution

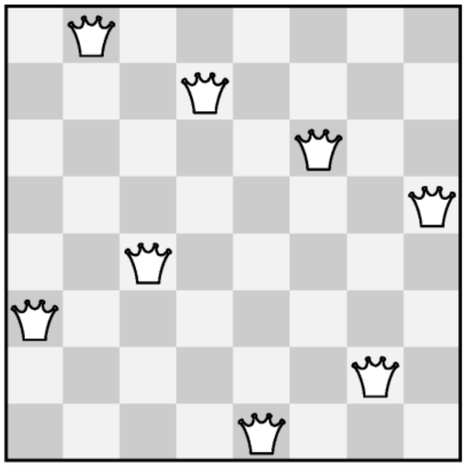
A naive solution to the N-queens problem is to first generate all possible configurations for the queens, and then verify which configurations satisfy the imposed restrictions. The validity of a candidate solution can be verified by checking whether each of the queens cannot attack one of the other queens by moving horizontally, vertically or diagonally. Given that a queen B can attack a queen A if and only if queen A can attack queen B, it suffices to check whether each queen cannot attack any of the queens to her right.

Example

The following is **one** of the 92 solutions for the 8-queens problem, which is also shown visually in Figure 2.

```
?- queens(8,L).
L = [6, 1, 5, 2, 8, 3, 7, 4] ;
```

Table 1 presents the number of solutions for different values of N.



Figuur 2: **One** of the 92 solutions for the 8-queens problem.

Number of queens	1	2	3	4	5	6	7	8	9	10
Number of solutions	1	0	0	2	10	4	40	92	352	724

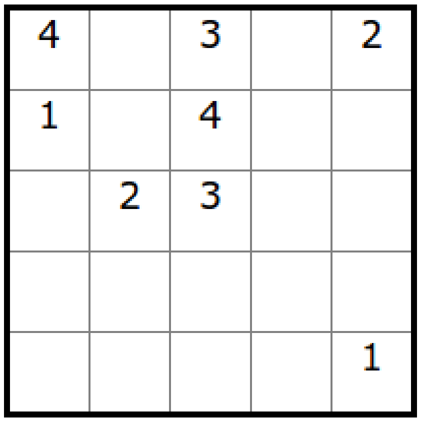
Tabel 1: The number of solutions for a given number of queens.

4 Arukone

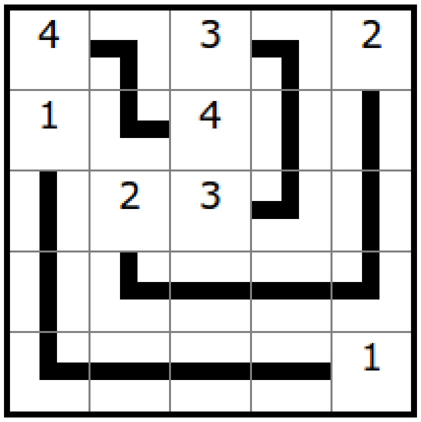
The objective in Arukone[1] puzzles is to find paths connecting pairs of numbers in a square grid. For each pair of numbers, a correct solution to the puzzle consists of an uninterrupted path that does not cross any other path and does not split into multiple paths. Figure 3 shows an example of a simple 5x5 puzzle and its unique solution.

The goal of this exercise is to implement an algorithm that automatically solves **well-designed Arukone puzzles**. A puzzle is well-designed if it has a **unique solution that occupies all the cells in the grid**.

[1]See `http://en.wikipedia.org/wiki/Numberlink` for more background information.



(a) A 5x5 puzzle



(b) Unique solution for the 5x5 puzzle

Figuur 3: A simple 5x5 puzzle and its unique solution.

Representation of puzzles

We represent a puzzle as a list of `link/3` terms. The first argument is a label, the second argument the start location of the path, and the third argument the end location of the path.

```
[
   link(1, pos(2,1), pos(5,5)),
   link(2, pos(1,5), pos(3,2)),
   link(3, pos(1,3), pos(3,3)),
   link(4, pos(1,1), pos(2,3)),
]
```

The list above represents the puzzle in Figure 3. The first `link/3` term means the solution should contain a path between the number 1 in the first column of the second row and the number 1 in the fifth column of the fifth row. Hence, the first argument of a `pos/2` term refers to the row number, while the second argument refers to the column number.

The template file contains a few example puzzles, which are represented as `puzzle/3` terms. The first argument is a unique sequence number, the second argument provides the dimensions of the puzzle, and the third argument contains a list of `link/3` terms as described earlier. We represent the dimensions of a puzzle as a `grid/2` term where the arguments specifiy the number of rows and columns respectively. For example, the term `grid(5,4)` defines a puzzle consisting of five rows and four columns.

Representation of solutions

We represent a solution as a list of `connects/2` terms, which correspond to the paths. The first argument is a label and the second argument a list of positions which the path goes through. The number of `connects/2` terms in a solution should always correspond to the number of `link/3` terms in the puzzle.

```
 [
    connects(1, [ pos(2,1), pos(3,1), pos(4,1), pos(5,1),
                  pos(5,2), pos(5,3), pos(5,4), pos(5,5) ]),

    connects(2, [ pos(1,5), pos(2,5), pos(3,5), pos(4,5),
                  pos(4,4), pos(4,3), pos(4,2), pos(3,2) ]),
    connects(3, [ pos(1,3), pos(1,4), pos(2,4), pos(3,4), pos(3,3) ]),
    connects(4, [ pos(1,1), pos(1,2), pos(2,2), pos(2,3) ])
 ]
```

The list above represents the solution in Figure 3. The last `connects/2` term states that there is a path between the number 4 in position (1,1) and the number 4 in position (2,3), which passes through the positions (1,2) and (2,2).

Approach

We transform Arukone puzzles into graphs, where the task of finding a solution to the puzzle resorts to finding non-overlapping paths. In our graph representation, each node corresponds to a cell and an edge exists between any two nodes whose corresponding cells are adjacent in horizontal or vertical direction. The constraint that two paths cannot cross is enforced by using each node only once.

Assignment

1. Familiarize yourself with the provided template.
2. Implement the `connect/4` predicate. Given a grid, a list of links, and a list of already occupied positions, the predicate should generate the solution to the puzzle. Consult the template file for further details.

Examples

Puzzle 1 (5x5 puzzle with 4 paths)

```
2 _ _ _ _                     2 2 4 4 4
1 2 _ 3 4                     1 2 4 3 4
_ _ _ _ _                     1 4 4 3 3
_ 4 _ 1 _                     1 4 1 1 3
_ _ _ 3 _                     1 1 1 3 3

% 8,659 inferences, 0.001 CPU in 0.001 seconds
```

Puzzle 2 (5x5 puzzle with 3 paths)

```
_ _ _ 3 _                   2 2 2 3 3
_ 3 _ 2 _                   2 3 2 2 3
_ _ _ _ _                   2 3 3 3 3
_ _ _ _ _                   2 2 2 2 2
1 _ _ 1 2                   1 1 1 1 2
```

```
% 362,537 inferences, 0.049 CPU in 0.049 seconds
```