# Homework 4. Towers solver

## Motivation

Towers is an arithmetical-logical puzzle whose goal is to fill in an $N \times N$ grid with integers, so that every row and every column contains all the integers from 1 through $N$, and so that certain extra constraints can be met. These extra constraints are specified by counts of which towers are visible from an edge of the grid, assuming that each grid entry is occupied by a tower whose height is given by the grid entry. There are $4N$ counts since the grid has four edges. For example, if $N$ is 5 and the counts for the top, bottom, left, and right edges are [2,3,2,1,4], [3,1,3,3,2], [4,1,2,5,2], and [2,4,2,1,2] respectively, then there is one solution with rows [2,3,4,5,1], [5,4,1,3,2], [4,1,5,2,3], [1,2,3,4,5], [3,5,2,1,4] respectively, as shown in the corresponding puzzle in Simon Tatham's implementation of the Towers puzzle.

A human doing the abovementioned puzzle can reason it out with arguments like the following. A 5 must be adjacent to each 1 count. The only remaining row or column lacking a 5 is the center row and column, so the center must be a 5. The 5 count at the left of row 4 means that row 4 must be [1,2,3,4,5]. The 4 count for column 5 means that row 1 column 5 must be a 1 or 2, and the 4 count for row 1 therefore means that row 1 must be either [1,3,4,5,2] or [2,3,4,5,1], so we can safely assume row 1 columns 2 and 3 must be [3,4]. For fun, you might try filling out the rest of the puzzle, using similar reasoning.

A computer solving this problem doesn't need to be that smart. It can rely on a small list of solution strategies rather than the ad hoc approach taken by humans.

## Assignment

Three things. First, write a predicate `ntower/3` that accepts the following arguments:

1. $N$, a nonnegative integer specifying the size of the square grid.
2. $T$, a list of N lists, each representing a row of the square grid. Each row is represented by a list of N distinct integers from 1 through N. The corresponding columns also contain all the integers from 1 through N.
3. $C$, a structure with function symbol `counts` and arity 4. Its arguments are all lists of N integers, and represent the tower counts for the top, bottom, left, and right edges, respectively.

*Precondition*. No solution will involve an integer that exceeds the `vector_max` of the GNU Prolog finite domain solver. Also, all arguments must be finite terms of the proper types. Your code need not check these preconditions; it can assume that the preconditions hold. Arguments can contain logical variables, except that the first argument $N$ must be bound to a nonnegative integer.

Second, write a predicate `plain_ntower/3` that acts like `ntower/3` but does not use the GNU Prolog finite domain solver. Instead, `plain_ntower/3` should enumerate the possible integer solutions using standard Prolog primitives such as `member/2` and `is/2`. Although `plain_ntower/3` should be simpler than `ntower/3` and should not be restricted to integers less than `vector_max`, the tradeoff is that it may have worse performance. Illustrate the performance difference on a test case of your own design, measuring performance with `statistics/0` or `statistics/2`. Package up your test case in a predicate `speedup/1`, which runs both `ntower/3` and `plain_ntower/3` and unifies its argument to the floating-point ratio of the latter's total CPU time to the former (hopefully this figure will be greater than 1).

Third, consider the problem of ambiguous Towers puzzle. Write a Prolog predicate `ambiguous(N, C, T1, T2)` that uses `ntower/3` to find a single $N \times N$ Towers puzzle with edges $C$ and two distinct solutions $T1$ and $T2$, and use it to find an ambiguous puzzle. Report the ambiguous puzzle you found.

## Examples

As a trivial example, `ntower(0,T,C)` should generate one solution `N=0, T=[], C=counts([],[],[],[])`. `ntower(1,T,C)` should generate one solution `N=1, T=[[1]], C=counts([1],[1],[1],[1])`. `ntower(2,T,C)` should generate the two solutions `N=2, T=[[1,2],[2,1]], C=counts([2,1],[1,2],[2,1],[1,2])` and `N=2, T=[[2,1],[1,2]], C=counts([1,2],[2,1],[1,2],[2,1])`, in either order. `ntower(3,T,C)` should generate several solutions, one of which is `N=3, T=[[1,2,3],[2,3,1],[3,1,2]], C=counts([3,2,1],[1,2,2],[3,2,1],[1,2,2])`. If the second and third arguments are variables, the number of solutions grows rapidly with $N$: for example, `ntower(3,T,C)` should generate a dozen solutions, and `ntower(4,T,C)` should generate hundreds of solutions.

You should be able to use `ntower/3` to generate edge counts from grids and vice versa. For example, the query:

```
?- ntower(5,
          [[2,3,4,5,1],
           [5,4,1,3,2],
           [4,1,5,2,3],
           [1,2,3,4,5],
           [3,5,2,1,4]],
          C).
```

should output this single result (reindented to fit):

```
C = counts([2,3,2,1,4],
           [3,1,3,3,2],
           [4,1,2,5,2],
           [2,4,2,1,2])
```

Conversely, the query:

```
?- ntower(5, T,
          counts([2,3,2,1,4],
                 [3,1,3,3,2],
                 [4,1,2,5,2],
                 [2,4,2,1,2])).
```

should output this single answer (reindented to fit):

```
T = [[2,3,4,5,1],
     [5,4,1,3,2],
     [4,1,5,2,3],
     [1,2,3,4,5],
     [3,5,2,1,4]]
```

Here “single answer” means that if you attempt to backtrack and get all possible solutions, only the abovementioned single answers will appear.

Here is an example of variables in both the second and third arguments:

```
?- ntower(5,
          [[2,3,4,5,1],
           [5,4,1,3,2],
           Row3,
           [RC41,5|Row4Tail],
           Row5],
          counts(Top, [4|BottomTail],
                 [Left1,Left2,Left3,Left4,5],
                 Right)).
```

It should generate this single answer:

```
BottomTail = [2,2,2,1]
Left1 = 4
Left2 = 1
Left3 = 2
Left4 = 2
RC41 = 3
Right = [2,4,2,2,1]
Row3 = [4,1,5,2,3]
Row4Tail = [2,1,4]
Row5 = [1,2,3,4,5]
Top = [2,3,2,1,5]
```

## Submit

Submit a Prolog file named `tower.pl` containing all the requested code. Also submit a text file `tower-notes.txt` that contains the requested information that is not source code.

---


$Id: hw4.html,v 1.107 2023/02/16 03:22:45 eggert Exp $