Puzzle: nonogramas

É um puzzle japonês. Ao contrário do Sudoku, este é conhecido por vários nomes, incluindo Griddlers ou Paint by Numbers. Foi inventado em 1987 por Tetsuya Nishio.
O objectivo do puzzle é descobrir, numa grelha, que células serão pintadas e quais ficarão vazias. O que dita que células são pintadas são os números escritos ao lado da grelha. Neste puzzle, os números ditam quantos quadrados consecutivos são preenchidos numa dada linha ou coluna. Por exemplo, uma dica de “4 8 3” significa que existe um conjunto de 4, de 8 e de 3 quadrados pintados, por essa ordem, com pelo menos um espaço em branco a separar os blocos. É tão importante determinar que células são pintadas como que células ficam vazias. A título de exemplo, a seguinte figura 1 mostra um puzzle e a sua solução.

Tarefa

A sua tarefa consiste em escrever um programa que resolva um nonograma. Se optar pela Programação em Lógica, implemente o programa através de um predicado **puzzle/1** que recebe como parâmetro a lista com as restrições do problema e imprime o resultado. Se escolher a Programação Funcional, desenvolva uma função **puzzle : int list -> unit** que recebe como argumento a lista e imprime o resultado.

O seu programa deverá ser acompanhado por um relatório breve, no qual descreve a estrutura da solução que encontrou, assim como as opções que tomou.

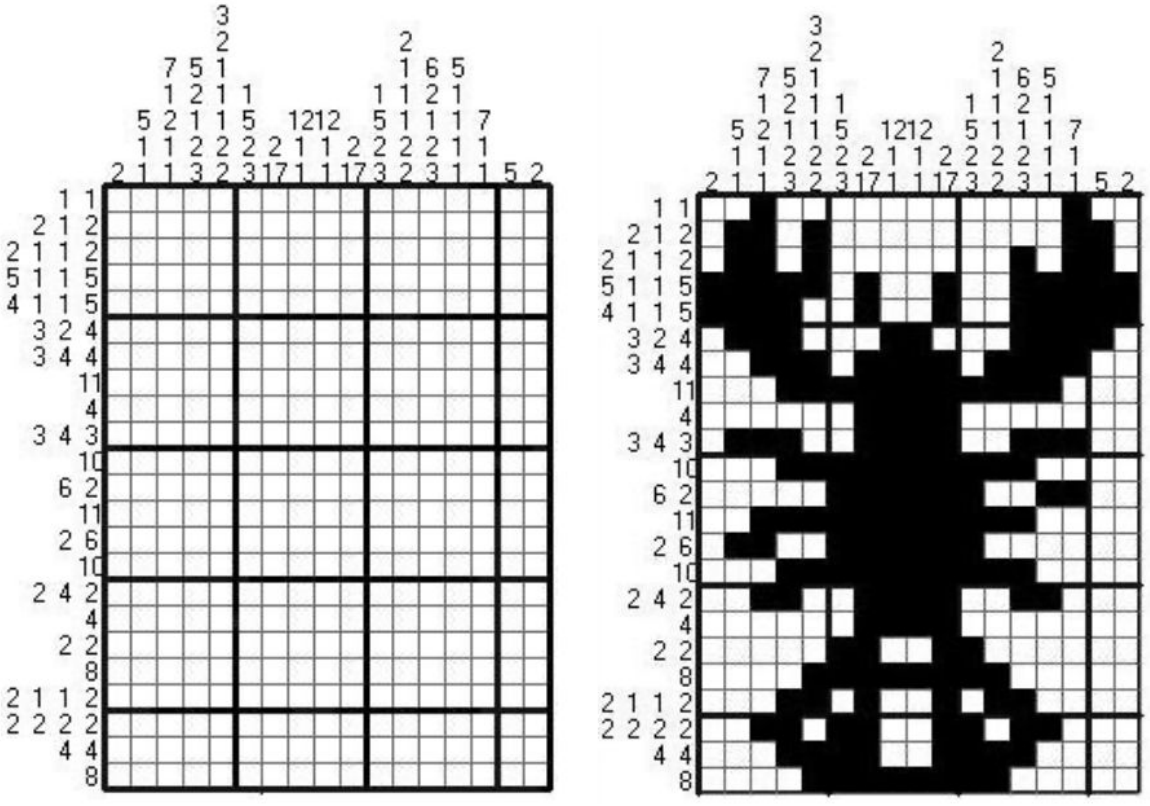

Figura 1: Uma grelha de um puzzle de nonograma e sua respectiva solução.

Os Dados

O argumento será uma lista com duas sub-listas: uma com os números referentes às linhas e outra com os números referentes às colunas. Cada um dos elementos dessas listas será novamente uma lista com os números correspondentes a essa linha ou coluna.

A título de exemplo, eis a lista correspondente ao problema da figura 1.

```
[[[1,1], [2,1,2], [2,1,1,2], [5,1,1,5], [4,1,1,5], [3,2,4], [3,4,4],
  [11], [4], [3,4,3], [10], [6,2], [11], [2,6], [10], [2,4,2], [4],
  [2,2], [8], [2,1,1,2], [2,2,2,2], [4,4], [8]],
 [[2], [5,1,1], [7,1,2,1,1], [5,2,1,2,3], [3,2,1,1,1,2,2], [1,5,2,3],
  [2,17], [12,1,1], [12,1,1], [2,17], [1,5,2,3], [2,1,1,1,2,2],
  [6,2,1,2,3], [5,1,1,1,1], [7,1,1], [5], [2]]]
```

Os Resultados

O resultado de invocar o predicado ou a função puzzle é imprimir no monitor o puzzle resolvido. O programa deverá imprimir a solução de forma legível. Assim, para o caso da figura 1, o programa deveria imprimir:

```
..X.........X..
.XX.X.......XX.
.XX.X.....X.XX.
XXXXX.X..X..XXXXX
XXXX..X..X..XXXXX
.XXX...XX...XXXX.
..XXX.XXXX.XXXX..
...XXXXXXXXXX...
......XXXX......
.XXX..XXXX..XXX..
...XXXXXXXXX....
.....XXXXXX..XX..
..XXXXXXXXXX....
.XX..XXXXXX.....
...XXXXXXXXX....
..XX..XXXX..XX...
......XXXX......
.....XX..XX.....
....XXXXXXXX....
...XX.X..X.XX....
..XX.XX..XX.XX...
...XXXX..XXXX....
....XXXXXXXX....
```

Exemplo 1 (Prog. Lógica)

Segue-se um exemplo ilustrativo do programa pretendido:

```
puzzle([[[1],[3],[1,1,1],[1],[1]],[[1],[1],[5],[1],[1]]]).
```

Que imprime o seguinte resultado:

```
..X..
.XXX.
X.X.X
..X..
..X..
```

Exemplo 2 (Prog. Funcional)

Segue-se um outro exemplo ilustrativo do programa pretendido:

```
puzzle [[[3;3;2]; [5;3;2]; [2;2;2;2]; [2;2;2;2]; [3;2;2;1;2];
[7;3;3;3]; [7;10]; [3;3;9]; [2;2;3;3]; [2;3;2;1]]; [[3]; [6];
[7]; [7]; [2;2]; [3;2]; [7]; [1;7]; [2;3]; [6;1]; [7]; [5]; [4];
[4]; [4]; [4]; [4]; [8]; [7]];;
```

Que deveria imprimir:

```
...XXX.XXX........XX
..XXXXX.XXX.......XX
..XX.XX..XX.......XX
..XX..XX.XX.......XX
.XXX..XX.XX...X...XX
.XXXXXX.XXX.XXX.XXX
.XXXXXX..XXXXXXXXXX
XXX...XXX.XXXXXXXXX.
XX.....XX..XXX.XXX..
XX.....XXX.XX...X...
```