# Mestrado em Engenharia Informática

Raquel Sofia Miranda da Costa **PG47600**

## Métodos Formais em Engenharia de Software

---

# **TPC 2** - SMT solving

# Opção 1: **Futoshiki Puzzle**

O jogo Futoshiki Puzzle, trata-se de um tabuleiro N por N, onde cada coluna e linha deverão ser preenchidas com os números de 1 a N, sendo que nenhum número pode ser repetido nessa mesma linha ou coluna. Além disso, todas as condições que envolvem os símbolos de desigualdade "<" e ">", deverão ser obrigatoriamente respeitadas.

O objetivo deste exercício é desenvolver um programa em Python capaz de resolver o jogo Futoshiki Puzzle, com o auxílio de um SMT solver. Neste caso, será usado o popular solver Z3 da Microsoft.

---

# Ficheiro de Input

O ficheiro de input usado tem o nome de "puzzle.txt". O formato escolhido para este foi o seguinte:

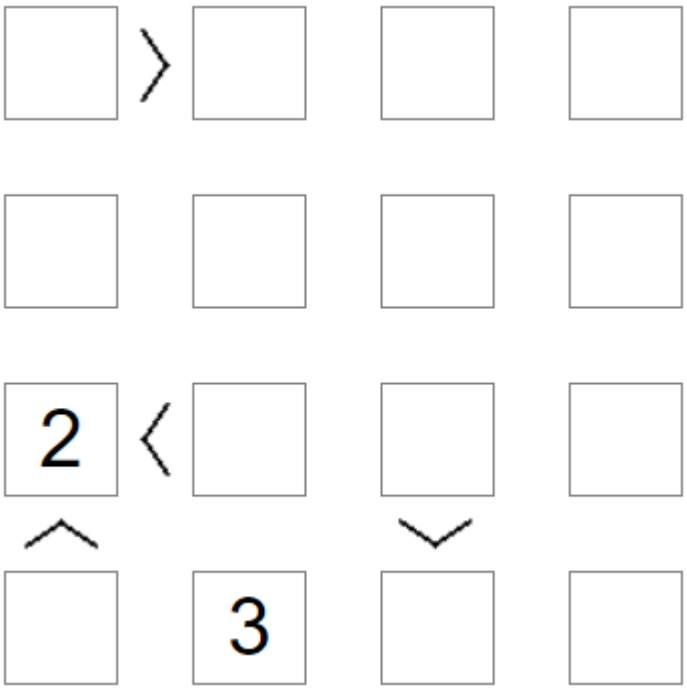
- Na primeira linha do ficheiro deverá especificar-se N, sendo que se trata de um tabuleiro NxN.
- Já nas próximas N linhas deverão ser representadas as posições do tabuleiro e os números que aparecem em cada uma delas inicialmente. Deste modo, é criada uma matriz NxN, onde o número 0 representa as posições do tabuleiro vazias, isto é, sem nenhum número porque serão preenchidas com a solução.
- Finalmente nas últimas linhas são escritas as condições de desigualdade, onde é usado apenas o símbolo ">" para representar as mesmas. Para tal indica-se as posições da matriz que fazem parte da condição. Por exemplo se o número que estiver na posição da

linha 0, coluna 3 for maior que aquele que se encontra na linha 1, coluna 3, i.e, X[0][3] > X[1][3], então é representado da seguinte forma: "0 3 > 1 3".

O exemplo usado retrata um tabuleiro 4x4 com 4 condições de desigualdade e apenas 2 posições preenchidas, com os números 2 e 3. O ficheiro obtido é então o seguinte:

```
4
0 0 0 0
0 0 0 0
2 0 0 0
0 3 0 0
0 0 > 0 1
2 1 > 2 0
3 0 > 2 0
2 2 > 3 2
```

Obtendo o seguinte tabuleiro pra iniciar o jogo:




---

# Programa

Primeiramente, começou-se por instalar o Z3.

```
!pip install z3-solver
```

```
Collecting z3-solver
  Downloading z3_solver-4.8.12.0-py2.py3-none-manylinux1_x86_64.whl (33.0 MB)
     |████████████████████████████████| 33.0 MB 19 kB/s
Installing collected packages: z3-solver
Successfully installed z3-solver-4.8.12.0
```

Também foi instalado o numpy, uma vez que suporta o processamento de matrizes, além de possuir uma grande coleção de funções matemáticas de alto nível que opera sobre estas mesmas. A biblioteca numpy foi necessária apenas para a estética do output do programa desenvolvido. De modo a que o mesmo conseguisse retornar um resultado na forma de matriz, facilitando assim a leitura da solução, uma vez que o formato assemelha-se com o de um tabuleiro.

```
!pip install numpy
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.19
```

De seguida, foi importado o módulo do Z3, e a biblioteca numpy.

```
from z3 import *
```

```
import numpy as np
```

Após ser criado um ficheiro de input com o nome de "puzzle.txt" e preenchido com os devidos dados sobre o tabuleiro a ser resolvido, corre-se então o programa.

```
sol = Solver()

with open("puzzle.txt") as file:
    lines = file.readlines()
        # Array de arrays com as linhas do ficheiro de input
    aux = [line.rstrip().split() for line in lines]

# Variável com o tamanho do tabuleiro
N = int(aux[0][0])

# Leitura da matriz representativa do tabuleiro presente no ficheiro
tab = []
for i in range(N):
    tab.append(aux[i+1])

# Conversão dos números do tabuleiro de string para inteiros
tab = list(map(list, (map(int, x) for x in tab)))

# Parse das condições do tabuleiro de jogo, sabemos que no ficheiro, estas
# posicionam-se após N + 1 linhas. São guardadas no array cond
cond = []
```

```
for i in range(len(aux)-N-1):
    cond.append(aux[N+1+i])

# Matriz NxN de variáveis de inteiros para o puzzle
X = [ [Int("x_%s_%s" % (i,j)) for j in range(N)] for i in range(N)]

# Ciclo responsável por adicionar as condições de desigualdade (caso existam) do tabuleiro
lenCond = len(cond)
for i in range(lenCond):
    sol.add(X[int(cond[i][0])][int(cond[i][1])] > X[int(cond[i][3])][int(cond[i][4])])

# Adição da regras gerais do jogo

# Cada posição contém um número entre 1 e N
sol.add( [ And(X[i][j] >= 1, X[i][j] <= N) for j in range(N) for i in range(N)] )
# A solução tem de conter os números já presentes no tabuleiro inicialmente
sol.add( [ If(tab[i][j] == 0, True, X[i][j] == tab[i][j]) for j in range(N) for i in range
# Cada linha tem de ter números diferentes
sol.add( [ Distinct(X[i]) for i in range(N)] )
# Cada coluna tem de ter números diferentes
sol.add( [ Distinct([X[i][j] for i in range(N) ]) for j in range(N)] )


if sol.check() == sat:
    m = sol.model()
    r = [ [ m.evaluate(X[i][j]) for j in range(N) ]
        for i in range(N) ]
    print(np.matrix(r))
else:
    print ("Puzzle impossível.")
```

```
 [[3 1 2 4]
  [1 2 4 3]
  [2 4 3 1]
  [4 3 1 2]]
```

Primeiramente, é feita a leitura e o parse do ficheiro de input. Os dados do ficheiro são guardados num array de arrays, onde cada array, é uma linha do ficheiro.

Portanto a primeira linha é convertida para inteiro e guardada numa variável N, que representa o tamanho do tabuleiro. Sabe-se então que as próximas N linhas, logo N arrays, serão aqueles que armazenam as linhas do tabuleiro. E, consequente, que a partir das N + 1 linhas, é onde estão representadas as condições de desigualdade do tabuleiro. Sendo assim, são armazenadas na matriz "tab", as linhas do tabuleiro e convertidas de string para int. E no array de arrays "cond" são guardadas as condições de desigualdade do tabuleiro. Ficando assim:

```
[['0', '0', '>', '0', '1'], ['2', '1', '>', '2', '0'], ['3', '0', '>', '2', '0'], ['2', '2', '>',
```

Logo para uma dada condição nº i: cond[i][0] e cond[i][1] representa a linha e a coluna, respetivamente, da posição onde o número deverá ser maior do que aquele que estará representado na linha cond[i][3] e coluna cond[i][4].

De seguida é inicializada a matriz X com variáveis de inteiros, onde iremos definir todas as restrições do jogo.

Após isto, é necessário definir as regras gerais do jogo, assentando em 5 simples regras:

- Cada posição do tabuleiro contém um número entre 1 e N
- A solução tem de conter os números já presentes no tabuleiro inicialmente
- Cada linha tem de ter números diferentes
- Cada coluna tem de ter números diferentes
- É obrigatório respeitar as condições de desigualdade representadas no tabuleiro

Finalmente, o solver irá verificar se existe uma solução que satisfaça as condições impostas. Caso seja verdade, o programa irá devolver no output uma matriz com a solução do jogo.

---

✓ 0 s concluído à(s) 19:25