2. Sudoku

Trabalho 1
Pedro Gomes a91647
Francisco Teófilo a93741

Utilizamos a biblioteca OR-Tools para fazer a interface para o SCIP e será utilizada para o `solver`.
Esta biblioteca pode ser instalada com o comando `pip install ortools`.

```
!pip3 install ortools
```

```
Requirement already satisfied: ortools in /usr/local/lib/python3.7/dist-packages (9.1
Requirement already satisfied: absl-py>=0.13 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: protobuf>=3.18.0 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from ab
```

A grelha é a representação em matriz da tabela do **Sudoku**.
Contém os parâmetros N (dimensão da tabela) e α (fração das casas da grelha que são preenchidas).

```
grelha = [
    [8,0,0,0,0,0,0,0,0],
    [0,0,3,6,0,0,0,0,0],
    [0,7,0,0,9,0,2,0,0],
    [0,5,0,0,0,7,0,0,0],
    [0,0,0,0,4,5,7,0,0],
    [0,0,0,1,0,0,0,3,0],
    [0,0,1,0,0,0,0,6,8],
    [0,0,8,5,0,0,0,1,0],
    [0,9,0,0,0,0,4,0,0]
]

n = 3
# α = 25.9%
```

Definimos as funções: `valid`, `print_grelha`, `find_empty` e `solve`.

A Função `valid` testa se é válido colocar um número em determinada posição da grelha, seguindo a ordem de teste *linhas* depois *colunas* e por fim nos quadrados $n * n$. Será utilizada pela função `solve`.

```
def valid(g, num, pos):
```

```
    # Check linha
    for i in range(len(g[0])):
        if g[pos[0]][i] == num and pos[1] != i:
            return False

    # Check coluna
    for i in range(len(g)):
        if g[i][pos[1]] == num and pos[0] != i:
            return False

    # Check gx
    gx_x = pos[1] // n
    gx_y = pos[0] // n

    for i in range(gx_y*n, gx_y*n + n):
        for j in range(gx_x * n, gx_x*n + n):
            if g[i][j] == num and (i,j) != pos:
                return False

    return True
```

A função `print_grelha` tem como objetivo a apresentação mais semelhante e organizada, da grelha de sudoku, como a conhecemos.

```
def print_grelha(g):
    for i in range(len(g)):
        if i % n == 0 and i != 0:
            print("- - - - - - - - - - - -")

        for j in range(len(g[0])):
            if j % n == 0 and j != 0:
                print(" | ", end="")

            if j == (n*n)-1:
                print(g[i][j])
            else:
                print(str(g[i][j]) + " ", end="")
```

A Função `find_empty` é utilizada para encontrar as posições vazias da grelha (==0) e será chamada na função `solve`.

```
def find_empty(g):
    for i in range(len(g)):
        for j in range(len(g[0])):
            if g[i][j] == 0:
                return (i, j)  # linha, coluna

    return None
```

A função `solve` tem como objetivo executar a resolução da grelha do sudoku proposta, utilizando as funções anteriormente definidas. Apresentamos a grelha inicial (por preencher) e a grelha final (resolvida).

```
print_grelha(grelha) # grelha inicial
print("_________________________\n")

def solve(g):
    find = find_empty(g)
    if not find:
        return True
    else:
        linha, coluna = find

    for i in range(1,(n*n)+1):
        if valid(g, i, (linha, coluna)):
            g[linha][coluna] = i

            if solve(g):
                return True

            g[linha][coluna] = 0

    return False

# Importar biblioteca
from ortools.linear_solver import pywraplp
# Criar instância do solver
solver = pywraplp.Solver.CreateSolver('SCIP')
# Invocar o solver
status = solver.Solve()
if status == pywraplp.Solver.OPTIMAL:
  solve(grelha)

print_grelha(grelha)
```

```
 8 0 0  | 0 0 0  | 0 0 0
 0 0 3  | 6 0 0  | 0 0 0
 0 7 0  | 0 9 0  | 2 0 0
 - - - - - - - - - - - -
 0 5 0  | 0 0 7  | 0 0 0
 0 0 0  | 0 4 5  | 7 0 0
 0 0 0  | 1 0 0  | 0 3 0
 - - - - - - - - - - - -
 0 0 1  | 0 0 0  | 0 6 8
 0 0 8  | 5 0 0  | 0 1 0
 0 9 0  | 0 0 0  | 4 0 0
 _________________________

 8 1 2  | 7 5 3  | 6 4 9
 9 4 3  | 6 8 2  | 1 7 5
 6 7 5  | 4 9 1  | 2 8 3
 - - - - - - - - - - - -
 1 5 4  | 2 3 7  | 8 9 6
 3 6 9  | 8 4 5  | 7 2 1
 2 8 7  | 1 6 9  | 5 3 4
 - - - - - - - - - - - -
```

```
5 2 1  | 9 7 4  | 3 6 8
4 3 8  | 5 2 6  | 9 1 7
7 9 6  | 3 1 8  | 4 5 2
```

Utilizamos $N = 3$, $N \in \{3, 4, 5, 6\}$ e α = 0.2, $\alpha \in \{0.0, 0.2, 0.4, 0.6\}$
O programa funciona para qualquer $\alpha \geq 0.2$, uma vez que é o valor mínimo de espaços da grelha preenchidos possíveis para o sudoku ser resolvido.
Para cada índice não atribuído, há 9 opções possibilidades de atribuição, portanto, a complexidade de tempo no pior caso do solucionador do sudoku é exponencial $O(9^{(n*n)})$.

✓ 1 s concluído à(s) 23:52