Exercício Programa 2 – Sudoku – soluções e testes (Entregar até 19/10/2025)
MAC 122 – 2. Semestre de 2025 – BMAC - IMEUSP

Sudoku

O jogo Sudoku consiste em preencher os espaços vazios de uma matriz 9x9 com os algarismos de 1 a 9 de tal maneira que em cada linha, cada coluna e cada quadrado interno de 3x3 contenha todos os algarismos de 1 a 9 sem repetição. A matriz dada possui algumas posições já preenchidas com algum algarismo.

O jogo consiste em preencher os espaços vazios testando as possibilidades até que a matriz esteja totalmente preenchida ou que se conclua não haver solução. Quanto mais posições vazias houver, maior o número de possibilidades, portanto mais difícil é o preenchimento, embora não seja essa uma regra geral.

Para uma dada matriz, pode haver mais de uma solução.

O jogo é normalmente apresentado com graduações de dificuldade quando se joga manualmente (Muito Fácil, Fácil, Difícil, Muito Difícil). O nosso objetivo é um programa que preenche a matriz, testando todas as possibilidades de forma consistente, chegando à solução ou às soluções ou concluindo que não há soluções.

Se quiser exercitar suas habilidades no Sudoku, visite o site: https://www.geniol.com.br/logica/sudoku

O algoritmo para resolver o Sudoku é o mesmo que é usado quando o resolvemos manualmente:

- A partir de uma posição:
  - Procure a próxima posição vazia
    - Se não houver mais posições vazias, chegamos a uma solução
    - Mostre a solução e retorne desta chamada para testar outras possíveis soluções
  - Preencha essa posição vazia com um algarismo dentre os possíveis candidatos
    - Se não houver mais candidatos, zere essa posição e retorne desta chamada. Assim, novas possibilidades serão testadas nas chamadas anteriores.
  - Retomar o processo, chamando novamente a função Sudoku agora com uma posição a mais já preenchida.

Da maneira descrita acima, o algoritmo é recursivo. Uma função Sudoku desta forma deve ter como parâmetros, a matriz, e a linha e coluna do último elemento a partir do qual faremos a busca de alguma posição livre. A primeira chamada desta função, no programa principal seria então:

```
Sudoku(MatrizSudoku, 0, 0)
```

A técnica do algoritmo acima é chamada de **backtracking**. Avançamos no preenchimento das posições guardando as informações necessárias para **retroceder** quando chegamos a uma solução ou a impossibilidade de solução.

Abaixo, exemplo de Sudoku e a matriz Sudoku correspondente. Posições vazias contém zero.

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

```
0 0 4 0 6 0 0 9 0
0 9 0 0 2 7 0 0 0
3 0 0 5 9 0 2 0 1
0 0 0 0 0 6 5 0 0
8 0 5 0 0 0 3 0 9
0 0 2 9 0 0 0 0 0
7 0 3 0 1 4 0 0 5
0 0 0 7 5 0 0 3 0
0 4 0 0 3 0 8 0 0
```

Sobre o Sudoku

Quanto menos posições preenchidas forem fornecidas inicialmente, maior a chance de haver mais de uma solução. Qual é o menor número de posições preenchidas inicialmente que podem ser dadas para que um sudoku tenha uma conclusão única? Com 17 posições inicialmente preenchidas existem soluções únicas conhecidas. No entanto, ninguém jamais encontrou com uma solução única com 16 posições preenchidas inicialmente. Consequentemente, conjecturou-se que a resposta para o problema do número mínimo de pistas é 17. Tal conjectura foi provada pelos autores Gary McGuire, Bastian Tugemann e Gilles Civario descrita no artigo abaixo:

https://arxiv.org/pdf/1201.0749

Embora haja solução única conhecida para 17 posições preenchidas, a quantidade de soluções, mesmo para mais de 17 posições preenchidas tende a ser muito grande.

As funções Sudoku e TestaMatrizSudoku

Estamos interessados em achar todas as soluções para uma determinada matriz. Não importa se a solução é única ou múltipla.

Escreva uma função **`Sudoku(MatrizSudoku, 0, 0)`** que recebe uma matriz **`MatrizSudoku`** parcialmente preenchida e consistente e efetua o preenchimento da mesma de acordo com as regras do jogo, indicando todas as soluções possíveis. Use o algoritmo de backtracking acima.
Para cada solução encontrada, verificar se a matriz foi preenchida corretamente. Para isso escreva uma função **`TestaMatrizSudoku(MatrizSudoku)`** que recebe uma matriz totalmente preenchida e verifica se foi preenchida corretamente (linhas, colunas e quadrados internos com todos os dígitos de 1 a 9).

As funções acima podem ser testadas separadamente antes de fazer o programa completo, garantindo assim que quando forem integradas já estejam funcionando adequadamente.

Arquivos de Teste

Serão fornecidos alguns arquivos de teste com nomes pré-definidos **`sudoku1.txt, sudoku2.txt, ...`**
São arquivos de texto de 9 linhas por 9 colunas com os algarismos separados por espaços. É conveniente que o arquivo de teste esteja no mesmo diretório que o seu programa para evitar ter que dar o caminho completo no open.

A função **`LeiaMatrizLocal (NomeArquivo)`** abaixo sugere uma forma de ler o arquivo de nome **`NomeArquivo`**.

Faça o download dos arquivos de teste em seu diretório.

Teste da função Sudoku

```
def LeiaMatrizLocal(NomeArquivo):
    # retorna a matriz lida ou [] se houver algum erro

    # abrir o arquivo para leitura
    try:
        arq = open(NomeArquivo, "r")
    except:
```

```
        return [] # retorna lista vazia se deu erro

    # inicia matriz Sudoku a ser lida: 9 linhas x 9 colunas
    mat = [9 * [0] for k in range(9)]

    # ler cada uma das linhas do arquivo
    for linha in arq:
        v = linha.split() # elementos separados por espaços
        # verifica se tem 9 elementos, todos entre '0' e '9'
        # . . .
        # transforma de string para int se achar conveniente
        # . . .
    # outras consistências se achar necessário
    # . . .
    # . . .
    # fechar arquivo e retorna a matriz lida e consistida
    arq.close()
    return mat

while True:
    nome_arq = input('Entre com o nome do arquivo')
    matriz = LeiaMatrizLocal(nome_arq)
    # testes das funções Sudoku e TestaMatrizSudoku
    # . . .
    # . . .
```

O que deve ser entregue - depois de fazer os testes da Sudoku e TestaMatrizSudoku e garantir que já funcionam

Os testes anteriores com arquivos são apenas para garantir que as funções Sudoku e TestaMatrizSudoku funcionam. Não fazem parte da entrega do programa.

Escreva uma função **`GeraMatrizSudoku(npp)`** que recebe um número de posições a serem preenchidas **`npp`** e constrói uma matriz de 9 linhas por 9 colunas contendo as **`npp`** posições preenchidas. Devolve a matriz construída. Essa matriz deve estar consistente, ou seja, os **`npp`** números entre 1 e 9 devem estar dispostos de forma que não haja repetições nas linhas, nas colunas e nos 9 quadrados internos.
Para gerar os **`npp`** números entre 1 e 9 (pode haver repetição) e as **`npp`** posições (sem repetição) entre (0,0) e (9,9) use a função **`randrange`** da biblioteca **`random`**.

Em resumo (sugestão):

```
def GeraMatrizSudoku(npp):
  matriz = 9 x 9 zerada
  # gerar npp números entre 1 e 9 (pode ter repetição)
  . . .
  # gerar npp duplas (linha, coluna) sem repetição onde 0 <= linha, coluna <=8
  . . .
  # colocar cada um dos números na respectiva posição
  . . .
  # verificar se ficou consistente (linhas, colunas e quadrados sem repetição)
  # não consistente? começar de novo até conseguir uma matriz consistente
  return matriz
```

O programa a ser entregue:

1. Ler o número de posições **`npp`** a serem preenchidas inicialmente. Tem que funcionar com qualquer número entre 0 e 81, mas o ideal é entre 17 e 30 mais ou menos.
2. **`Matriz = GeraMatrizSudoku(npp):`**
3. **`Sudoku(Matriz, 0, 0)`** – constrói e mostra todas as soluções possíveis. Como pode ser que tenha muitas soluções, pode colocar um limite. Por exemplo parar quando mostrar a vigésima solução. Sem esquecer que dentre da Sudoku tem que estar a **`TestaMatrizSudoku`**, isto é, só é solução depois de verificar linha a linha, coluna a coluna, quadrado a quadrado.
4. Voltar para 1 – repetir até que seja digitado **`"fim"`**

Exemplo de saída do programa

```
Entre com o número de posições a preencher inicialmente:23

 * * * Matriz inicial

0  0  4  0  0  0  0  7  9
5  0  7  0  0  0  0  0  0
0  0  0  0  0  0  0  5  0
0  8  0  0  0  5  7  0  0
0  7  0  0  3  0  8  0  0
2  0  0  0  0  0  0  1  0
0  1  8  6  0  0  0  0  0
4  3  2  8  0  0  0  0  7
0  0  0  0  0  0  4  0  0


* * * Matriz Completa – Solução 1

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

linhas OK  * * * * * *
colunas OK * * * * * *
quadrados OK * * * * *
* * * Matriz Completa e Consistente


* * * Matriz Completa – Solução 2

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

linhas OK  * * * * * *
colunas OK * * * * * *
quadrados OK * * * * *
* * * Matriz Completa e Consistente


* * * Matriz Completa – Solução 3

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

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

linhas OK  * * * * * *
colunas OK * * * * * *
quadrados OK * * * * *
* * * Matriz Completa e Consistente


. . .
. . .

* * * Matriz Completa – Solução 10

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

linhas OK  * * * * * *
colunas OK * * * * * *
quadrados OK * * * * *
* * * Matriz Completa e Consistente

* * * Há mais soluções

Entre com o número de posições a preencher inicialmente:21
```