Instituto de Computação - Unicamp

MC102 - Algoritmos e Programação de Computadores

Laboratório 20 - Sudoku

Prazo de entrega: **28/06/2019 23:59:59**

Peso: **2**

Professor: Luiz Fernando Bittencourt
Monitor: Eduardo de Souza Gama
Monitor: Maurício Gagliardi Palma
Monitor: Jadisha Yarif Ramírez Cornejo
Monitor: João Vitor Vendemiato Fatoretto

Descrição

Sudoku (数独) é um quebra-cabeça baseado na disposição lógica de números. Foi criado por Howard Garns, um projetista e arquiteto de 74 anos aposentado. O objetivo do jogo é dispor números de 1 até 9 em cada uma das células vazias numa grade 9x9, constituída por 9 subgrades 3x3 chamadas de regiões (Figura 1). O quebra-cabeça contém algumas pistas iniciais, que são números inseridos em algumas células, de maneira a permitir uma indução ou dedução dos números em células que estejam vazias. Cada coluna, linha e região só pode ter um número dentre os números de 1 até 9. Resolver o problema requer apenas raciocínio lógico e algum tempo. Wikipedia

Os numerais no jogo sudoku são usados por comodidade; as relações aritméticas entre numerais são absolutamente irrelevantes. Qualquer combinação de símbolos distintos como letras, formas, ou cores podem ser usadas no jogo sem alterar as regras. Por exemplo, algumas variações usam letras, como Scramblets da Penny Press e Sudoku Words da Knight Features Syndicate. Dell Magazines, que fez as primeiras publicações do jogo, têm utilizado números desde a sua primeira publicação em 1979. Wikipedia

Figura 1: Um típico problema de sudoku

(a) O problema original

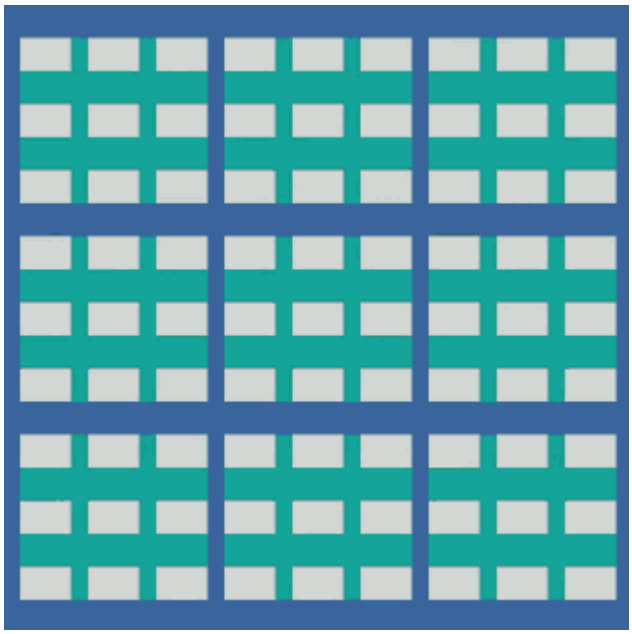
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

(b) A solução destacada em vermelho

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

Nesse laboratório, o objetivo é implementar uma função recursiva que resolve um determinado jogo do Sudoku.

Uma ideia de algoritmo para resolver o problema é: encontrar uma posição vazia corrente, e enumerar os possíveis valores que podem ser atribuídos para esta posição. Para cada um destes valores, atribuir ele na posição vazia e chamar recursivamente a função para que uma nova posição vazia seja avaliada. Se for possível encontrar uma solução então imprimimos a solução. Caso não seja possível, tentamos o próximo valor possível para a posição corrente. Esse algoritmo é chamado de *Backtracking* e pode ser implementado com uma função recursiva. Veja um exemplo do comportamento dessa solução:



Função

```
#!/usr/bin/env python3

# Funcao: print_sudoku
# Essa funcao ja esta implementada no arquivo lab20_main.py
# A funcao imprime o tabuleiro atual do sudoku de forma animada, isto e,
# imprime o tabuleiro e espera 0.1s antes de fazer outra modificacao.
# Voce deve chamar essa funcao a cada modificacao na matriz resposta, assim
# voce tera uma animacao similar a apresentada no enunciado.
# Essa funcao nao tem efeito na execucao no Susy, logo nao ha necessidade de
# remover as chamadas para submissao.
from lab20_main import print_sudoku

# O aluno pode criar outras funcoes que ache necessario

# Funcao: resolve
# Resolve o Sudoku da matriz resposta.
# Retorna True se encontrar uma resposta, False caso contrario
def resolve(resposta):
    # Implementar a funcao e trocar o valor de retorno
    print_sudoku(resposta)
    return False
```

Submissão

Neste laboratório você não precisará se preocupar em ler a entrada a partir da entrada padrão, nem em escrever a saída. O código fornecido no arquivo

`lab20_main.py` se encarrega dessa parte.Seu trabalho é apenas implementar a função descrita.

Você também **não deve** submeter o arquivo `lab20_main.py` para o *SuSy*, somente o arquivo `lab20.py`.

As sessões abaixo, de Entrada e Saída, descrevem os formatos de entrada e saída, mas você não precisa se preocupar com eles.

Entrada

A entrada do programa será de 9x9 dígitos de 0 a 9, onde 0 representa as posições com valores indefinidos que devem ser resolvidas.

Saída

A saída do programa é composta da grade antes e depois da solução, com separadores `-` e `|` para linhas e colunas, respectivamente.

Caso não seja encontrada uma solução, o programa irá imprimir `Nao foi encontrada uma solucao.`. Você pode considerar que todos os testes possuem exatamente uma solução válida.

Exemplos

Teste 01

Entrada

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

Saída

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

Para mais exemplos, consulte os testes abertos no Susy.

Observações

- Você **não deve** submeter o arquivo `lab20_main.py` para o *SuSy*, somente o arquivo `lab20.py`.
- O número máximo de submissões é **10**.

- Para a realização dos testes do SuSy, a execução do código se dará da seguinte forma: (Linux e OSX)

  ```
  python3 lab20_main.py .
  ```

- Você deve incluir, no início do seu programa, uma breve descrição dos objetivos do programa, da entrada e da saída, além do seu nome e do seu RA.
- Indente corretamente o seu código e inclua comentários no decorrer do seu programa.