Centro Educacional Tecnológico Celso Sudckow da Fonseca CEFET/RJ
Engenharia de computação
AEDS II

# Relatório sobre trabalho de Sudoku com Backtracking

Aluno: Ernst Franz Karl Zeidler Neto
Professor orientador: Laura Assis

Petrópolis
26 de junho de 2023

Centro Educacional Tecnológico Celso Sudckow da Fonseca CEFET/RJ
Engenharia de computação
AEDS II

# Relatório

Segundo Relatório de trabalho sobre Sudoku apresentado na turma de AEDS II do Curso Engenharia de computação da Universidade CEFET/RJ como requisito parcial para o trabalho 2.

Aluno: Ernst Zeidler Neto

Professora orientadora: Laura

26 de junho de 2023

# Conteúdo

**1 A contextualização do problema** **1**

**2 O método de solução** **1**

**3 Estruturas de dados que foram utilizadas** **1**
- 3.1 Matriz . . . 2
- 3.2 Contador . . . 2
- 3.3 resolverSudoku . . . 2
- 3.4 verificarPosicaoSegura . . . 2
- 3.5 verificarLinha . . . 2
- 3.6 verificarColuna . . . 2
- 3.7 verificarBloco . . . 3
- 3.8 imprimirSudoku . . . 3

**4 Fluxograma do código** **3**

**5 pseudo-código com descrição das principais rotinas** **4**

**6 Resultado** **5**

**7 Testes** **6**

**8 Conclusões** **9**

# 1 A contextualização do problema

O Sudoku é um quebra-cabeça numérico que consiste em preencher uma grade 9x9 com dígitos de 1 a 9, de modo que cada coluna, cada linha e cada uma das nove sub-grades 3x3 contenha todos os dígitos de 1 a 9, sem repetições entre as linhas, colunas e dentro da sua sub-grade. O objetivo é encontrar uma solução válida para o Sudoku, onde todas as células são preenchidas corretamente.

# 2 O método de solução

Para resolver o Sudoku, foi implementado um algoritmo de backtracking(força bruta). O algoritmo tenta inserir um número em uma célula vazia e verifica se essa inserção é válida de acordo com as regras do Sudoku. Se a inserção for válida, o algoritmo avança para a próxima célula vazia e repete o processo. Se não for possível inserir um número em uma célula vazia(valor = 0), o algoritmo retorna para recorrência anteriormente e altera o valor para o próximo número válido ou setar a célula vazia (valor = 0) e voltando a recorrência e tentar outra opção (outro numero possível e válido). Esse processo é repetido até que todas as células sejam preenchidas corretamente ou não haja mais opções válidas. Para utilizar o método de solução, é preciso ignorar a regra padrão do sudoku em jogos de celular, o usuário que tenta resolver o Sudoku ele possui duas chances ou vidas de errar o numero inserido e fazer em qualquer posição do Sudoku. Se ocorrer o terceiro erro, o jogo é encerrado e como resposta de que o usuário perdeu.

# 3 Estruturas de dados que foram utilizadas

Nesta etapa, foram utilizadas as seguintes estruturas de dados:

- Matriz: Uma matriz 9x9 foi usada para representar o grid do Sudoku.
- Contador: o valor contador de operações foram feitos durante a resolução do Sudoku.
- funções: resolverSudoku, encontrarProximaCelulaVazia, verificarPosicaoSegura, verificarLinha, verificarColuna, verificarBloco, imprimirSudoku.

Podemos dizer que as Funções que foram criadas e são necessarias para resolver o Sudoku que vai ser explicar com detalhes em cada sub-seções.

## 3.1 Matriz

A matriz do Sudoku foi utilizado ou criado de forma dinamicamente e uso do calloc. Uso do calloc da biblioteca stdlib.h para criar a linhas e colunas com valores setados para zero. No final, fazer a liberação da memoria do sistema apos resolver o Sudoku antes de encerrar o programa.

## 3.2 Contador

O contador é criado na main e repassado como parametro de ponteiro e sempre na função resolverSudoku, sempre incrementar em cada etapa de inserção e tentativa dos números.

## 3.3 resolverSudoku

a funcao resolver Sudoku recebe os parâmetros da matriz Sudoku, linha, coluna e contador. durante a sua função recursiva, ele verifica se chegou ao seu fim de linhas e vai retornando as funções recorrentes anteriores, verifica se chegou no fim da coluna e retorna na funcao recorrente com a linha incrementada e seta a coluna no valor inicial. Verifica se na posicao da linha e coluna do Sudoku existe um valor igual a zero, se nao tiver o valor é fixo e faz uma chamada da propria função com a incrementação da coluna. se tiver valor zero, ele vai tentar os numeros de 1 até 9 e fazer a verificação do numero se é seguro com a função verificarPosicaoSegura. Caso nao encontrar o valor seguro, seta para o valor zero e ele retorna na recursão anterior e altera o valor ate encontrar um numero valido e retornar e/ou continuar a resolução do Sudoku da propria função recursiva.

## 3.4 verificarPosicaoSegura

Verifica se é seguro inserir o número num na célula (linha, coluna) do grid e chama as funções de verificarLinha,verificarColuna,verificarBloco.

## 3.5 verificarLinha

Verifica se o número numero já está presente na linha linha do grid e do Sudoku.

## 3.6 verificarColuna

Verifica se o número numero já está presente na coluna coluna do grid e do Sudoku.

### 3.7 verificarBloco

Verifica se o número num já está presente no bloco 3x3 que contém a célula de início (linhaInicio, colunaInicio) do grid.

### 3.8 imprimirSudoku

Imprime o grid do Sudoku.

## 4 Fluxograma do código

- Encontre a próxima célula vazia no grid.
- Se todas as células estiverem preenchidas, retorne verdadeiro (Sudoku resolvido). Para cada número de 1 a 9:
- a. Se o número for válido para a célula atual, atribua-o e avance para a próxima célula.
- b. Se a atribuição levar a uma solução, retorne verdadeiro.
- c. Desfaça a atribuição e tente o próximo número.
- Se nenhuma atribuição de número for válida, retorne falso (Sudoku sem solução).

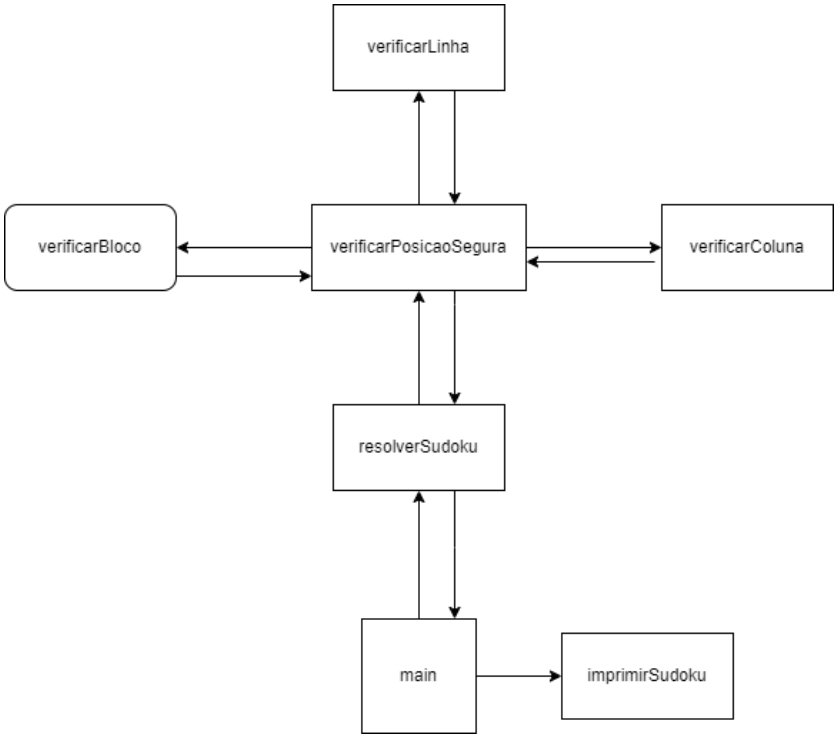


Figura 1: Fluxograma do código

# 5 pseudo-código com descrição das principais rotinas

```
1. Função verificarLinha(grid, linha, num):
    - Para cada coluna no grid:
        - Se o número na célula (linha, coluna) for igual a 'num', retorne falso.
    - Retorne verdadeiro.

2. Função verificarColuna(grid, coluna, num):
    - Para cada linha no grid:
        - Se o número na célula (linha, coluna) for igual a 'num', retorne falso.
    - Retorne verdadeiro.

3. Função verificarBloco(grid, linhaInicio, colunaInicio, num):
    - Para cada célula no bloco 3x3 iniciando em (linhaInicio, colunaInicio):
        - Se o número na célula for igual a 'num', retorne falso.
    - Retorne verdadeiro.

4. Função verificarPosicaoSegura(grid, linha, coluna, num):
    - Retorne o resultado da expressão:
        verificarLinha(grid, linha, num) &&
        verificarColuna(grid, coluna, num) &&
        verificarBloco(grid, linha - linha % 3, coluna - coluna % 3, num).

5. Função encontrarProximaCelulaVazia(grid, linha, coluna):
    - Para cada linha no grid:
        - Para cada coluna no grid:
            - Se a célula (linha, coluna) estiver vazia (valor igual a 0):
                - Atualize 'linha' e 'coluna' com os valores da célula atual.
                - Retorne verdadeiro.
    - Retorne falso.
```

Figura 2: pseudo-código parte 1

# 6 Resultado

Após a execução do algoritmo, o Sudoku será resolvido e o tabuleiro completo será impresso. Além disso, será exibido o número de operações (tentativas de inserção de números) realizadas durante a resolução do Sudoku. Porém, no arquivos de entrada 5.in e 8.in apresentaram saídas diferentes do esperado e o arquivo 7.in não conseguiu gerar um gráfico por levar muito tempo de processamento e muitas operações, esses dados serão apresentados no tópico a seguir.

```
6. Função resolverSudoku(grid, linha, coluna, contador):
    - Se a linha for igual ao tamanho do grid:
        - Retorne verdadeiro (Sudoku resolvido).
    - Se a coluna for igual ao tamanho do grid:
        - Chame resolverSudoku com a próxima linha e a coluna zero.
    - Se a célula (linha, coluna) não estiver vazia:
        - Chame resolverSudoku com a mesma linha, próxima coluna.
    - Para cada número de 1 a 9:
        - Se a atribuição do número na célula (linha, coluna) for válida:
            - Atribua o número à célula.
            - Incremente o contador.
            - Chame resolverSudoku com a mesma linha, próxima coluna.
            - Se o Sudoku for resolvido, retorne verdadeiro.
            - Desfaça a atribuição (atribua 0 à célula).
    - Retorne falso (Sudoku sem solução).

7. Função imprimirSudoku(grid):
    - Para cada linha no grid:
        - Se a linha for igual a 0, 3 ou 6:
            - Imprima uma linha de separação.
        - Para cada coluna no grid:
            - Se a coluna for igual a 0, 3 ou 6:
                - Imprima um "|".
            - Imprima o valor da célula (linha, coluna).
            - Se a coluna for igual a 8:
                - Imprima um "|".
        - Se a linha for igual a 8:
            - Imprima uma linha de separação.
```

Figura 3: pseudo-código parte 2

# 7 Testes

Os testes feitos para gerarem os gráficos, foi utilizado o mesmo algoritmo para versão python e importando a biblioteca matplotlib, podemos gerar os gráficos e apresentar os mesmos resultados do Sudoku, contador e tempo decorrido. Nos arquivos de entrada 5.in, 8.in deram valores divergentes de saída. No arquivo 7.in, o processo demorou mais de 10 segundos para resolver, porem não foi possível gerar o gráfico pelo tempo de processamento no python também.

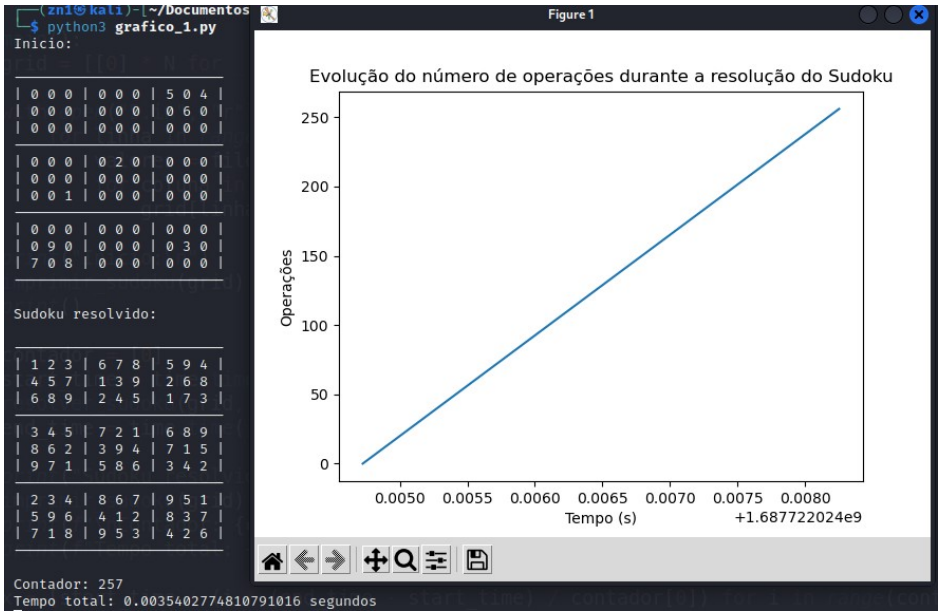


Figura 4: gráfico de 5.in

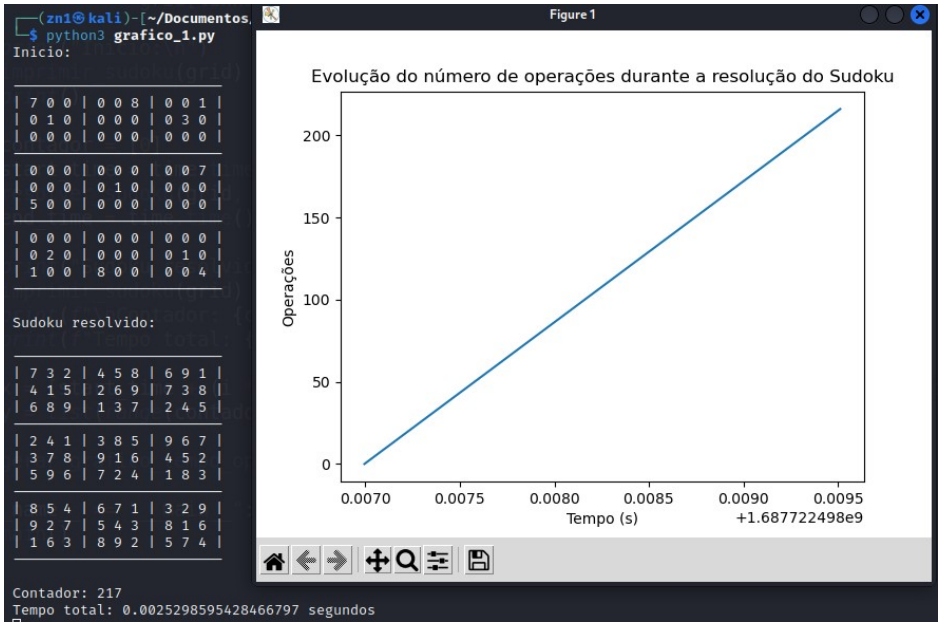


Figura 5: gráfico de 8.in

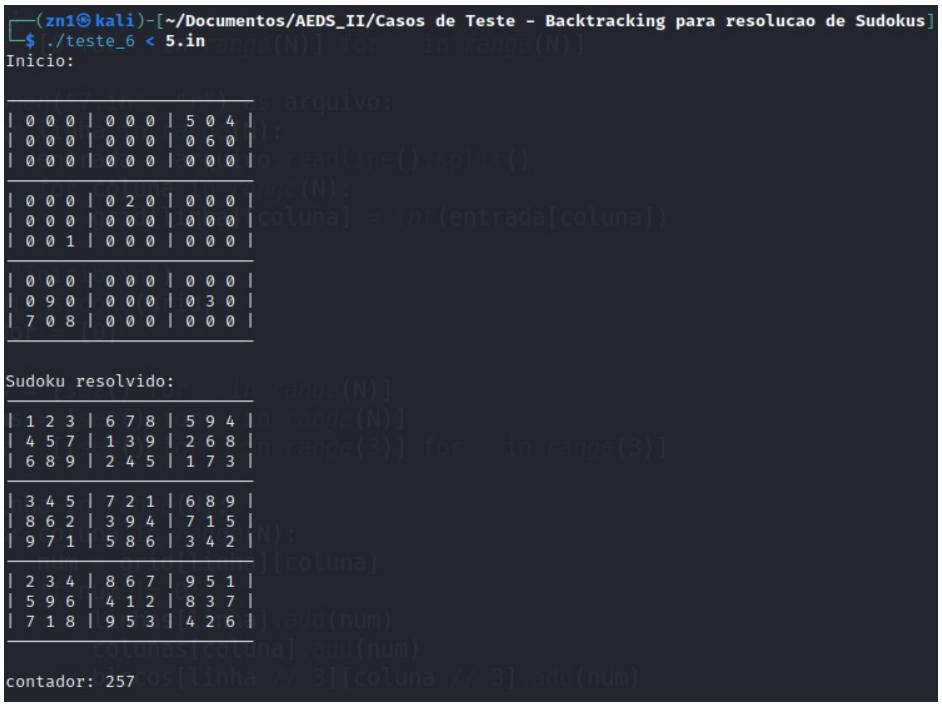


Figura 6: teste de 5.in em C

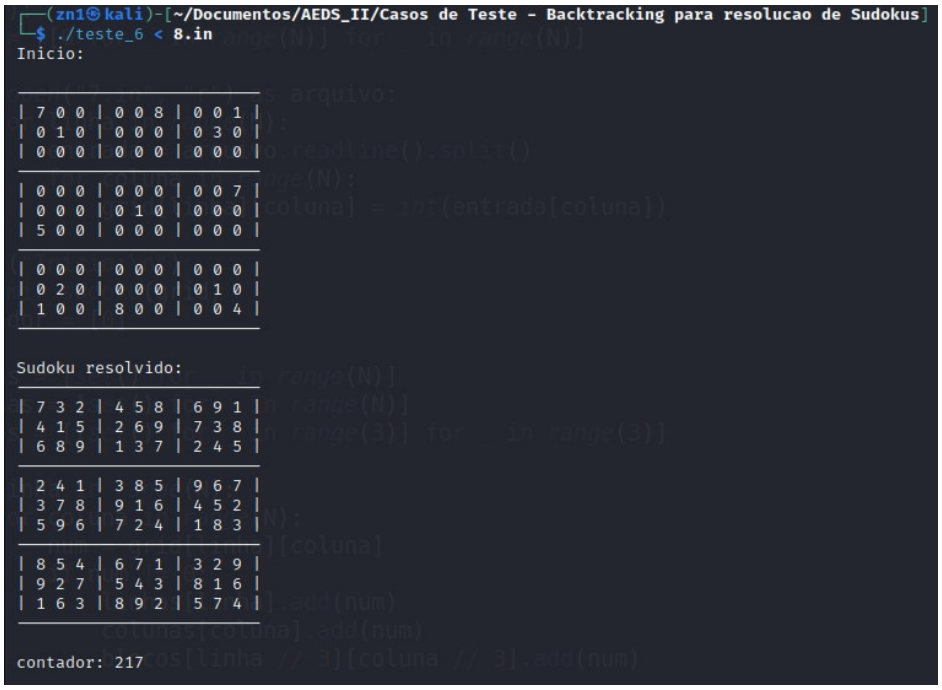


Figura 7: teste de 8.in em C

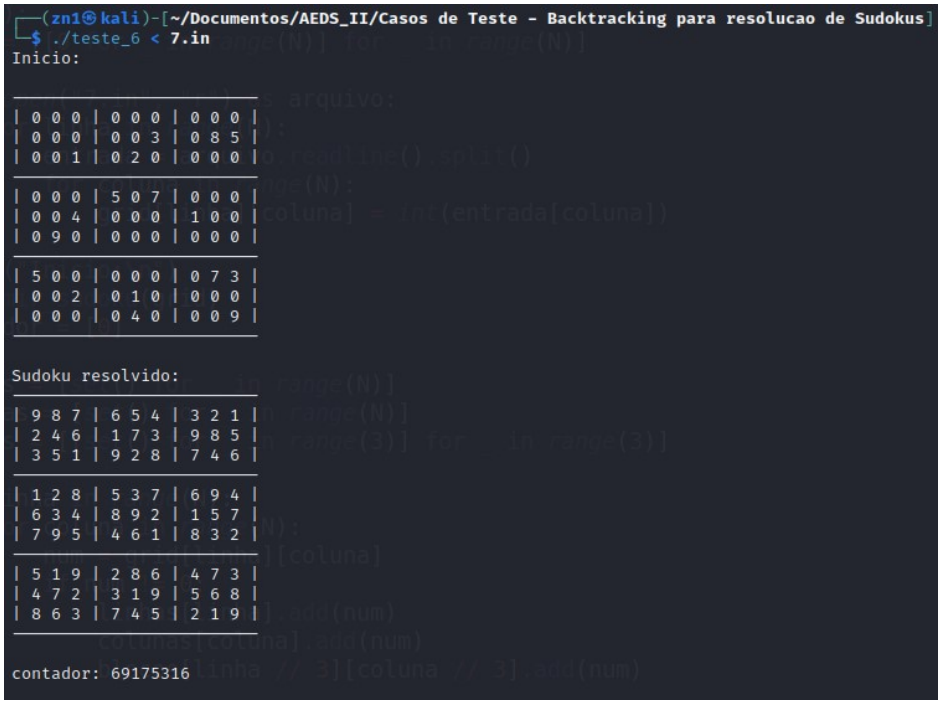


Figura 8: teste de 7.in em C

# 8 Conclusões

O algoritmo implementado é capaz de resolver Sudoku utilizando a técnica de busca por retrocesso. Através da validação das regras do jogo, exceto o limite de tentativas de erro são ignorados e da tentativa de preenchimento das células vazias, é possível encontrar a solução correta. O algoritmo é eficiente e fornece uma solução para qualquer instância válida do jogo Sudoku.