CEFET/RJ
Uned Petrópolis

Backtracking para resolução de Sudokus

Lucas Eckhardt
CEFET/RJ - Uned Petrópolis
lucas.eckhardt@aluno.cefet-rj.br

RESUMO

Este relatório apresenta a implementação de um algoritmo de resolução do Sudoku utilizando a técnica de *backtracking* em linguagem C. Foi desenvolvido em seis etapas, abrangendo desde a contextualização do problema até a conclusão, e incluindo o método de solução, as estruturas de dados utilizadas, o fluxograma do algoritmo, o pseudo-código das funções e a coleta de resultados. Os resultados são apresentados por meio da contagem de operações e do tempo de execução para diferentes instâncias do Sudoku, fornecendo informações sobre o desempenho do algoritmo.

ABSTRACT

This report presents the implementation of a Sudoku solving algorithm using the backtracking technique in C language. It was developed in six steps, covering from problem contextualization to conclusion, and including the solution method, the data structures used, the algorithm flowchart, the pseudocode of the functions, and the collection of results. The results are presented by counting operations and execution time for different instances of Sudoku, providing information about the performance of the algorithm.

1 de junho de 2023

CEFET/RJ
Uned Petrópolis

Sumário

1. Contextualização do Problema

O problema do Sudoku é um desafio de lógica que consiste em preencher uma grade 9x9 com números de 1 a 9, de forma que cada linha, coluna e sub-grade 3x3 contenha todos os números de 1 a 9, sem repetições. O objetivo é encontrar uma solução para o Sudoku, preenchendo todos os espaços em branco.

Neste relatório, abordaremos a implementação de um algoritmo de resolução de Sudoku utilizando a técnica de *backtracking*. O *backtracking* é uma estratégia de busca exaustiva que tenta solucionar um problema através de tentativa e erro, retornando a uma etapa anterior quando uma solução não é viável.

O objetivo é fornecer uma visão detalhada de como o algoritmo funciona, descrevendo as etapas-chave envolvidas na resolução do Sudoku por meio do *backtracking*.

2. Método de Solução

O algoritmo de *backtracking* é uma técnica recursiva que nos permite encontrar soluções para problemas complexos, como o Sudoku. A ideia principal é tentar todas as combinações possíveis para cada posição vazia do tabuleiro, voltando atrás (*backtracking*) quando uma solução não é viável.

A proposta de solução desenvolvida consiste nos seguintes passos:

- Verificar se há alguma posição vazia no tabuleiro. Se todas as posições estiverem preenchidas, retornar true, indicando que o Sudoku foi resolvido com sucesso.
- Encontrar a próxima posição vazia no tabuleiro.
- Tentar atribuir valores de 1 a 9 nessa posição vazia, verificando se cada atribuição é segura. Uma atribuição é considerada segura se o número não estiver presente na mesma linha, na mesma coluna ou no mesmo bloco 3x3.
- Se uma atribuição for segura, atribuir o valor à posição vazia e chamar recursivamente a função para resolver o restante do tabuleiro.
- Se a chamada recursiva retornar true, significa que o restante do tabuleiro foi resolvido com sucesso, então retornar true.
- Se a chamada recursiva retornar false, desfazer a atribuição e tentar o próximo valor possível. Continuar o processo até encontrar uma solução válida ou até esgotar todas as possibilidades.

Este método de solução utiliza a estratégia de busca exaustiva para percorrer todas as combinações possíveis até encontrar uma solução válida para o Sudoku.

3. Estruturas de Dados Utilizadas

3.1. Matriz Bidimensional

```
int grid[N][N];
```

Uma matriz 9x9 é utilizada para representar o tabuleiro do Sudoku. Cada posição da matriz é um elemento do tipo int que armazena um número de 1 a 9 para representar os valores preenchidos no Sudoku, ou o valor 0 para indicar uma posição vazia no tabuleiro.

3.2. Funções Auxiliares

```
int isSafe(int grid[N][N], int row, int col, int num);
int findUnassignedLocation(int grid[N][N], int *row, int *col);
void printGrid(int grid[N][N]);
```

Essas estruturas de dados e funções auxiliares são fundamentais para o funcionamento correto do algoritmo de resolução do Sudoku usando *backtracking*. Eles ajudam na verificação de segurança das atribuições, na localização da próxima posição vazia e na impressão adequada do tabuleiro.

4. Fluxograma

O algoritmo começa verificando se há posições vazias no tabuleiro. Se houver, ele encontra a próxima posição vazia. Em seguida, tenta atribuir valores de 1 a 9 na posição vazia e verifica se essa atribuição é segura, ou seja, se o valor não entra em conflito com outras posições no mesmo bloco, linha ou coluna. Se a atribuição for segura, o algoritmo atribui o valor à posição vazia e chama recursivamente a função para resolver o restante do tabuleiro. Ele repete esse processo até que o restante do tabuleiro seja resolvido ou não haja mais valores possíveis para atribuir. Se o restante do tabuleiro for resolvido, o algoritmo retorna verdadeiro, indicando que o Sudoku foi resolvido com sucesso. Caso contrário, ele desfaz a atribuição anterior, tenta o próximo valor possível e continua o processo. Se nenhum valor for possível, o algoritmo retorna falso, indicando que o Sudoku não pode ser resolvido. O algoritmo termina quando todas as posições forem preenchidas ou quando não for possível encontrar uma solução.

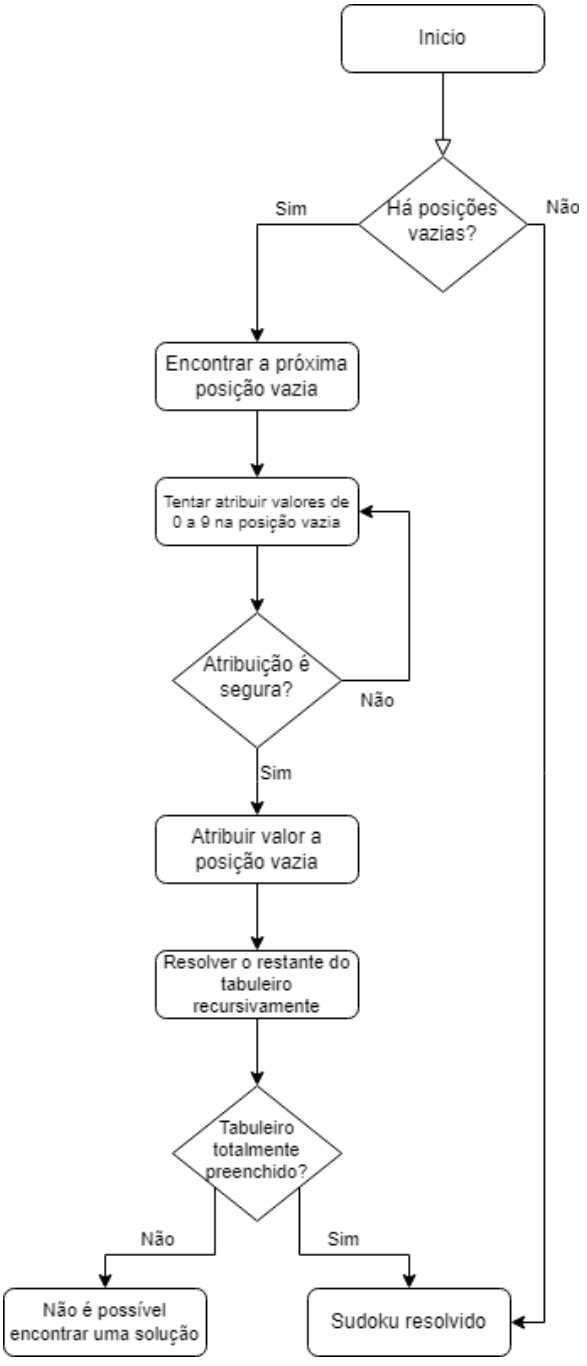

Figura 1: Fluxograma do algoritmo.

5. Pseudocódigo

Estes pseudocódigos descrevem as funções utilizadas no algoritmo, incluindo a verificação de segurança das atribuições, a busca por posições vazias, a função principal de resolução do Sudoku e a função de impressão do tabuleiro.

5.1. Função *isSafe*

```
Função isSafe(grid, row, col, num):
    Para cada coluna na mesma linha:
        Se o número já está presente na coluna:
            Retornar falso

    Para cada linha na mesma coluna:
        Se o número já está presente na linha:
            Retornar falso

    Inicializar startRow e startCol com o valor do canto superior esquerdo do b

    Para cada linha no bloco 3x3:
        Para cada coluna no bloco 3x3:
            Se o número já está presente no bloco 3x3:
                Retornar falso

    Retornar verdadeiro
```

5.2. Função *findUnassignedLocation*

```
Função findUnassignedLocation(grid, row, col):
    Para cada linha no tabuleiro:
        Para cada coluna no tabuleiro:
            Se a posição está vazia:
                Atribuir o valor da linha à variável row
                Atribuir o valor da coluna à variável col
                Retornar verdadeiro

    Retornar falso
```

5.3. Função *solveSudoku*

```
Função solveSudoku(grid):
    Se não há posições vazias no tabuleiro:
        Retornar verdadeiro (Sudoku resolvido com sucesso)

    Encontrar a próxima posição vazia no tabuleiro
    Para cada número de 1 a 9:
        Se a atribuição é segura:
            Atribuir o número à posição vazia
```

```
            Se solveSudoku(grid) retorna verdadeiro:
                Retornar verdadeiro (Sudoku resolvido com sucesso)
            Desfazer a atribuição

    Retornar falso (Sudoku não pode ser resolvido)
```

5.4. Função *printGrid*

```
Função printGrid(grid):
    Para cada linha no tabuleiro:
        Se a linha é múltipla de 3:
            Imprimir linha horizontal
        Para cada coluna no tabuleiro:
            Se a coluna é múltipla de 3:
                Imprimir linha vertical
            Se a posição não está vazia:
                Imprimir número
            Senão:
                Imprimir "x"
        Imprimir nova linha

    Imprimir linha horizontal
```

5.5. Função *main*

```
Função main():
    Inicializar o tabuleiro Sudoku
    Ler o Sudoku a ser resolvido do usuário
    Se solveSudoku(grid) retorna verdadeiro:
        Imprimir "Sudoku resolvido:"
        Chamar a função printGrid(grid)
    Senão:
        Imprimir "Não é possível resolver o Sudoku."

    Retornar 0
```

6. Resultados

6.1. Operações vs Instâncias

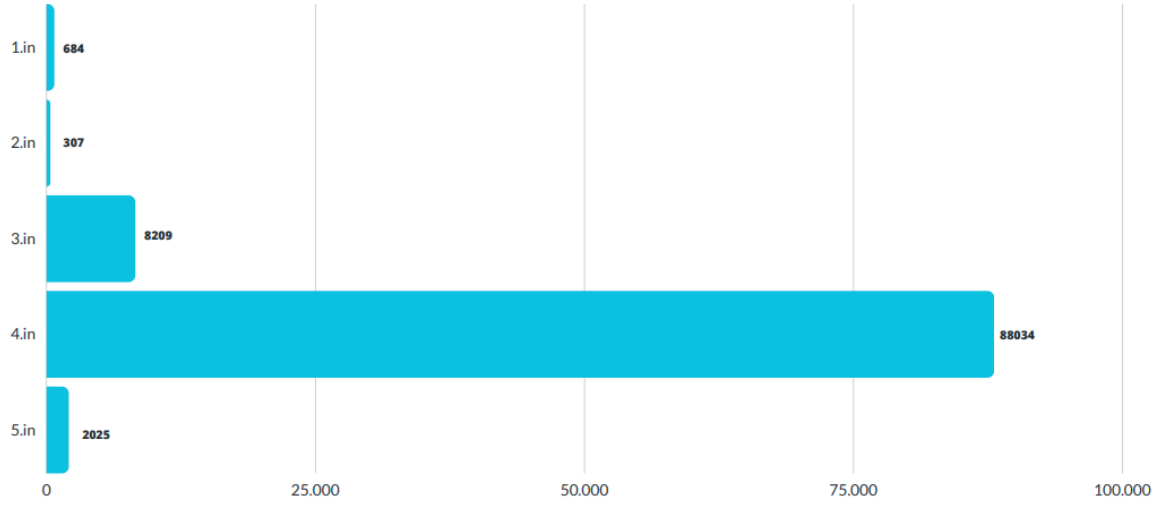

Figura 2: Gráfico Operações vs Instâncias.

6.2. Tempo vs Instâncias

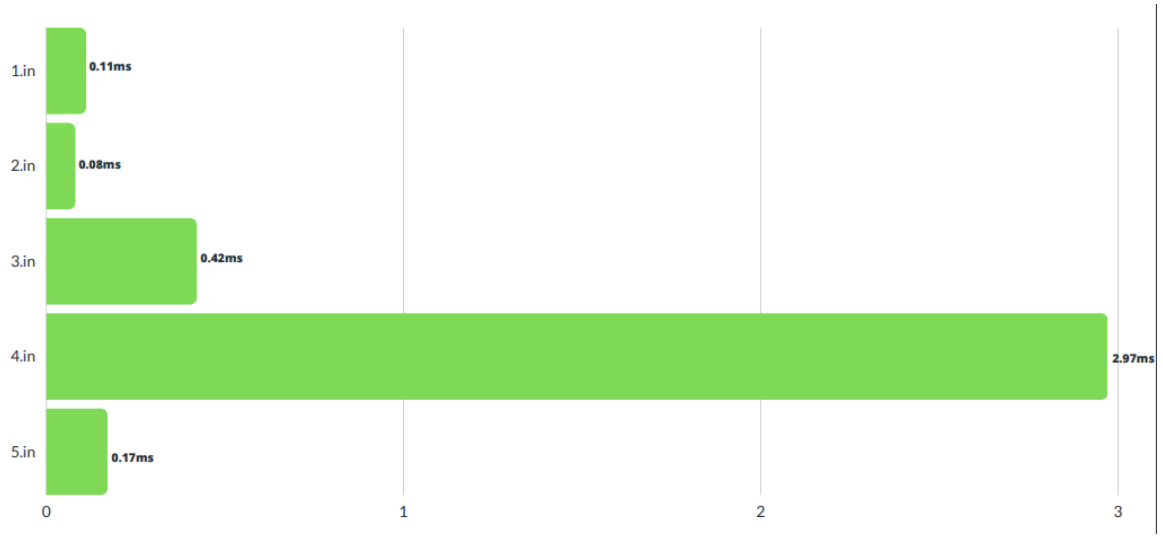

Figura 3: Gráfico Tempo vs Instâncias.

7. Conclusão

Neste relatório, abordamos a implementação de um algoritmo de resolução do Sudoku usando a técnica de *backtracking* em linguagem de programação C. O Sudoku é um desafio de lógica que consiste em preencher uma grade 9x9 com números de 1 a 9, de forma que cada linha, coluna e sub-grade 3x3 contenha todos os números de 1 a 9, sem repetições.

O algoritmo de *backtracking* é uma estratégia de busca exaustiva que tenta solucionar o Sudoku por meio de tentativa e erro, retornando a uma etapa anterior quando uma solução não é viável. Utilizamos uma matriz bidimensional para representar o tabuleiro do Sudoku e implementamos funções auxiliares para verificar a segurança das atribuições, encontrar posições vazias e imprimir o tabuleiro.

Ao longo do relatório, descrevemos as etapas-chave do algoritmo, desde a contextualização do problema até a descrição das estruturas de dados utilizadas, o pseudo-código das funções e o fluxograma do algoritmo.

Através desse algoritmo, é possível resolver Sudokus de forma eficiente, encontrando soluções válidas para tabuleiros incompletos. No entanto, é importante destacar que a eficiência do algoritmo pode variar dependendo da complexidade do Sudoku a ser resolvido.

Em resumo, a implementação do algoritmo de resolução do Sudoku usando *backtracking* proporciona uma abordagem sistemática para solucionar esse desafio de lógica, permitindo a resolução de Sudokus de forma eficiente.