CEFET/RJ

RELATÓRIO DO TRABALHO 2 DE AEDs 2

Professora: Laura Silva de Assis
Disciplina: Algoritmo e Estrutura de Dados II

Aluno: Pedro Henrique da Silva Tesch
Matrícula: 2012171GCOM

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.

PALAVRAS CHAVE. BackTracking, Sudoku, Algoritmos.

TÓPICOS:

CEFET/RJ

1. Contextualização

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 abordagem recursiva que possibilita a busca por soluções em problemas complexos, como o Sudoku. Sua principal ideia consiste em explorar todas as combinações possíveis para cada posição vazia do tabuleiro, retornando (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 coluna, na mesma linha 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.

3. Estruturas de Dados Utilizadas

3.1. Matriz Bidimensional

```
1     int sudoku[9][9]
```

3.2. Funções Auxiliares

```
1     void captacaoInicial(int sudoku[9][9]);
2     void imprimir(int sudoku[9][9]);
3     bool encontrarPosicaoVazia(int sudoku[9][9], int* linha, int*
          coluna);
4     bool podeInserir(int sudoku[9][9], int linha, int coluna, int num);
5     bool resolver(int sudoku[9][9]);
```

Esta estrutura de dados e funções auxiliares são fundamentais para o funcionamento solucionador de Sudoku usando backtracking proposto por mim. Estas funções verificam a segurança das modificações, localização de espaços vazios e na impressão adequada do resultado final.

4. Fluxograma

O algoritmo inicia verificando se existem espaços vazios no tabuleiro. Caso haja, ele procura pela próxima posição vazia. Em seguida, tenta atribuir valores de 1 a 9 nessa 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. Esse processo é repetido 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 solucionado 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 finaliza ao preencher todas as posições ou quando não for possível encontrar uma solução.

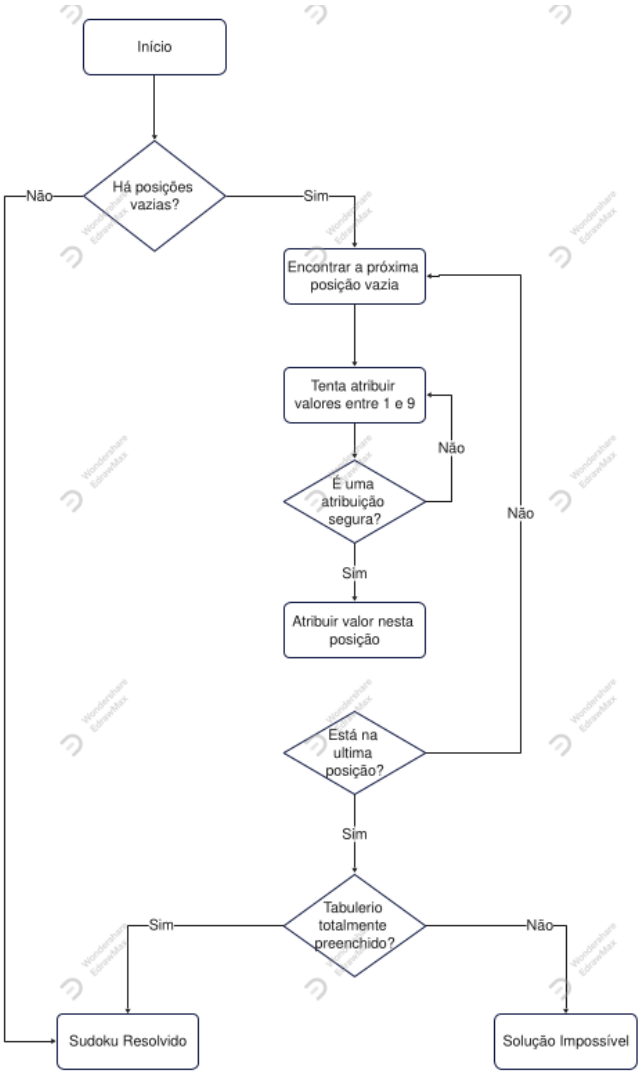

Figura 1: Fluxograma do código

5. Pseudocódigo

Os pseudocódigos a seguir apresentam as funções utilizadas no algoritmo, que incluem a verificação de segurança das atribuições, a busca por posições vazias, a função principal para resolver o Sudoku e a função de impressão do tabuleiro.

5.1. captacaoInicial

```
1  void captacaoInicial(int sudoku[9][9]) {
2     Estrutura de repeticao para as linhas :
3         Estrutura de repeticao aninhada para cada coluna:
4             Pega o valor e atribuindo na posicao correspondente
5  }
```

5.2. Imprimir

```
void imprimir(int sudoku[9][9]):
    Imprime uma linha horizontal
    Para cada linha do tabuleiro:
        Imprime uma linha vertical (Marcando inicio da linha)
        Para cada coluna:
            Imprime o numero daquela posição
            Se a coluna eh multipla de 3:
                Imprime linha vertical (separando cada quadrante)
        Pula uma linha na impressão;
        Se a linha é múltipla de 3:
            Imprime linha horizontal
```

5.3. EncontrarPosiçãoVazia

```
bool encontrarPosicaoVazia(int sudoku[9][9], int* linha, int* coluna) {
    Para cada linha, atualizando o ponteiro da variável linha:
        Para cada coluna, atualizando o ponteiro da variável coluna:
            Se esta posição for igual a zero:
                Retorna true
    Retorn false (caso chegue até o fim do tabuleiro)
}
```

5.4. PodeInserir

```
bool podeInserir(int sudoku[9][9], int linha, int coluna, int num) {
    Para cada posição daquela linha específica:
        Se for igual ao número:
            Retorna false;

    Para cada posição daquela coluna específica:
        Se for igual ao número:
            Retorna false;

    Pega o inicio da linha do quadrante
    Pega o inicio da coluna do quadrante

    Para cada linha do quadrante:
        Para cada coluna do quadrante:
            A posição tendo o mesmo valor que o número:
                Retorna false;

    Retorna true;
}
```

5.5. resolver

```
bool resolver(int sudoku[9][9]) {
    Declara variáveis linha e coluna

    Se não encontrar posição vazia
        Retorna true; indicando que o sudoku está resolvido

    Do 1 ao 9:
        Chama a função PodeInserir e se positivo:
            A posição relativa recebe aquele número

            Chama resolver Recursivamente e se positivo:
                Retorna true;
            Se não:
                0 em cada uma das posições que a recursão acessou

    Retorna false; Caso não tenha conseguido isnerir um número;
}
```

6. Resultados

6.1. Operações X Instâncias

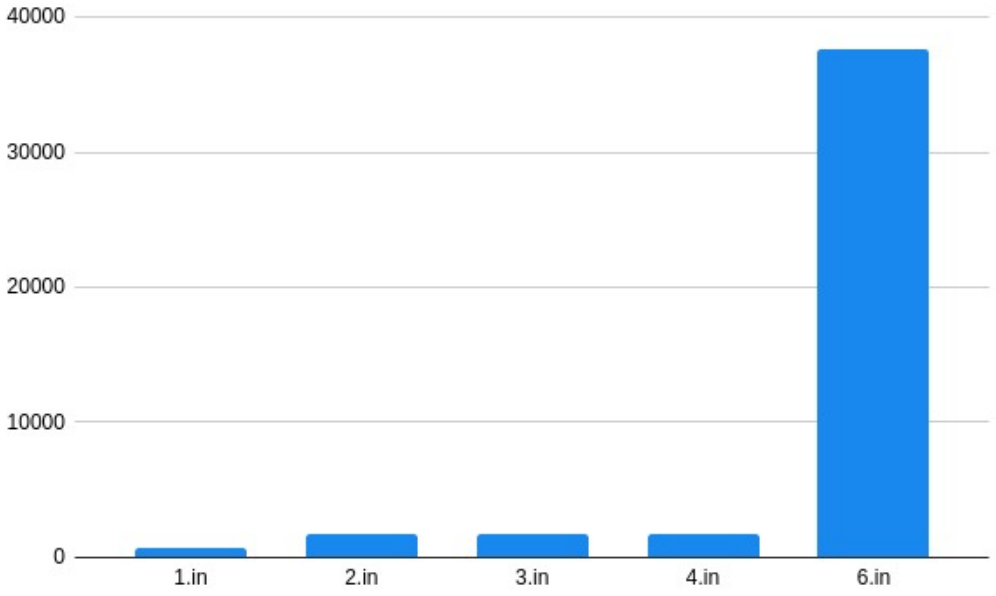

Figura 2: Gráfico de Operações X Instâncias

6.2. Tempo X Instâncias

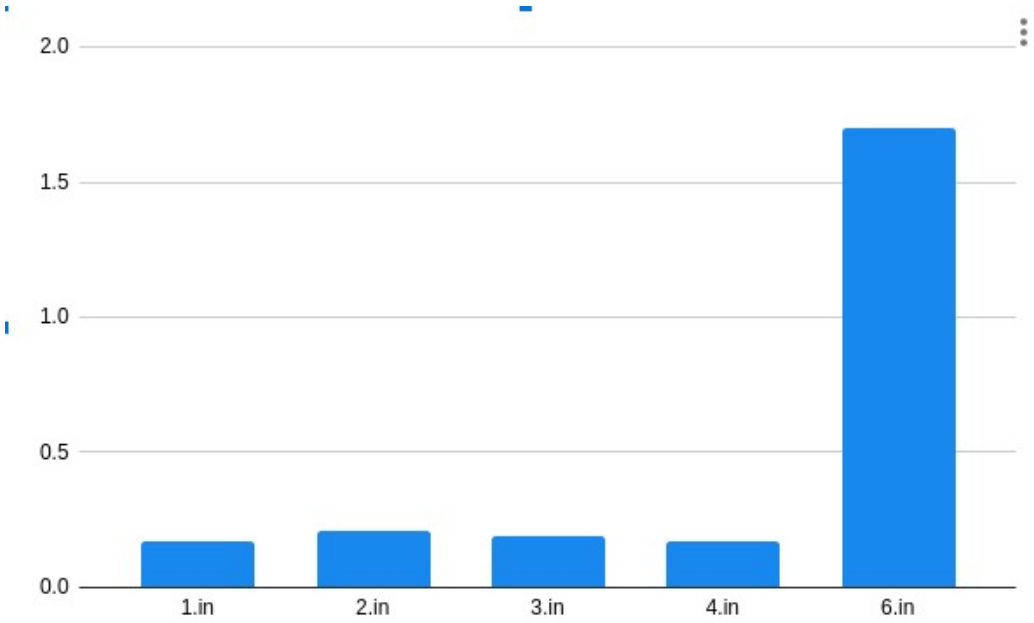

Figura 3: Gráfico de Tempo X Instâncias

7. Conclusão

Neste relatório, discutimos a implementação de um algoritmo em linguagem de programação C para resolver Sudokus usando a técnica de backtracking. O Sudoku é um desafio lógico que envolve preencher uma grade 9x9 com números de 1 a 9, de modo 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, também conhecida por Brute-Force, que tenta resolver o Sudoku por meio de tentativa e erro, voltando a etapas anteriores quando uma solução não é viável. Utilizamos uma matriz bidimensional para representar o tabuleiro do Sudoku e implementamos funções auxiliares para verificar a validade das atribuições, encontrar células vazias e imprimir o tabuleiro.

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

Com esse algoritmo, é possível resolver Sudokus de maneira eficiente, encontrando soluções válidas para tabuleiros incompletos. No entanto, é importante ressaltar 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 fornece uma abordagem sistemática para resolver esse desafio lógico, permitindo a solução eficiente de Sudokus.