Resolução de Sudoku em JavaScript

Marcos Vinicius Silva

19 de Setembro de 2023

1 Introdução

O Sudoku é um quebra-cabeça lógico numérico que envolve a colocação de números em uma grade 9x9 de células subdivididas em sub-grades 3x3. O objetivo é preencher a grade de modo que cada linha, coluna e sub-grade contenha todos os números de 1 a 9, sem repetições. Resolver Sudoku manualmente pode ser um desafio e, portanto, é interessante desenvolver algoritmos que possam resolvê-los automaticamente.

Este relatório descreve um código JavaScript que aborda o problema de resolução de Sudoku. Ele lê um arquivo de entrada contendo um quebra-cabeça de Sudoku, tenta resolvê-lo e imprime a solução na saída padrão. Este algoritmo é baseado no paradigma de backtracking.

2 Modelagem do Problema

Para resolver um Sudoku usando o paradigma de backtracking, o problema é modelado da seguinte maneira:

- O tabuleiro de Sudoku é representado como uma matriz 9x9, onde cada célula contém um número de 1 a 9 ou é representada por 0 para células vazias. - A resolução do Sudoku é tratada como uma busca recursiva por uma solução válida. - A função `resolve(board)` tenta preencher cada célula vazia com um número válido e verifica se o tabuleiro resultante ainda é válido. Se for, a recursão continua; caso contrário, a recursão retrocede e tenta uma abordagem diferente.

```
const fs = require('fs');
function resolve(board) {
    if (!falta(board)) {
        return true;
    }
    const [lin, col] = falta(board);

    for (let i = 1; i <= 9; i++) {
        if (isAceita(board, lin, col, i)) {
            board[lin][col] = i;
            if (resolve(board)) {
                return true;
            }
```

```
            board[lin][col] = 0;
        }
    }


    return false;
}
```

Além dessa função é utilizado mais duas funções para auxiliar nesse processo sendo elas `falta(board)` onde ela retorna qual cédula esta faltando se o sudoku não estiver sido resolvido.

```
function falta(board) {
    for (let lin = 0; lin < 9; lin++) {
        for (let col = 0; col < 9; col++) {
            if (board[lin][col] === 0) {
                return [lin, col];
            }
        }
    }
    return null;
}
```

Por ultimo a função `isAceita(board, lin, col, num)` que verifica se o numero é aceito na linha e no quadrante, sendo que no quadrante a função que realiza essa verificação é `verificaQuad(board, lin, col, num)`

```
function isAceita(board, lin, col, num) {
    // Verifique se o número não está na mesma linha ou coluna.
    for (let i = 0; i < 9; i++) {
        if (board[lin][i] === num || board[i][col] === num) {
            return false;
        }
    }
    return verificaQuad(board, lin, col, num);
}

function verificaQuad(board, lin, col, num){
     // Verifique se o número não está na mesma sub-grade 3x3.
    let startlin = Math.floor(lin / 3) * 3;
    let startCol = Math.floor(col / 3) * 3;
    for (let i = startlin; i < startlin + 3; i++) {
        for (let j = startCol; j < startCol + 3; j++) {
            if (board[i][j] === num) {
                return false;
            }
        }
    }
    return true;
```

3 Pontos Fortes e Fracos

3.1 Pontos Fortes

- O algoritmo de backtracking é eficaz para resolver Sudoku, uma vez que é capaz de explorar todas as possíveis combinações até encontrar uma solução válida. - O código é relativamente simples e fácil de entender, tornando-o acessível para programadores iniciantes. - A estrutura de dados do tabuleiro é eficiente para verificar a validade das atribuições de números.

3.2 Pontos Fracos

- O algoritmo de backtracking pode ser computacionalmente intensivo em casos de Sudoku extremamente difíceis, levando a um tempo de execução longo. - Não há otimizações específicas implementadas, como heurísticas de escolha de células ou números, o que pode afetar o desempenho em problemas complexos. - O código não lida com entradas inválidas ou tabuleiros malformados, o que pode resultar em comportamento inesperado.

4 Instruções de Uso

Para executar o programa, siga as instruções abaixo:

1. Execute o seguinte comando, substituindo `arquivo.txt` pelo nome do arquivo de entrada contendo o quebra-cabeça de Sudoku:

   ```
   node Sudoku.js arquivo.txt
   ```

2. O programa lerá o arquivo, tentará resolver o Sudoku e imprimirá a solução ou uma mensagem de que nenhuma solução foi encontrada. Certifique-se de ter um arquivo válido no formato descrito para que o programa funcione corretamente.

5 Codigo Completo

```
    const fs = require('fs');


function resolve(board) {
    if (!falta(board)) {
        // Se não há mais células vazias, o Sudoku está resolvido.
        return true;
    }

    const [lin, col] = falta(board);// que falta

    for (let i = 1; i <= 9; i++) {
        if (isAceita(board, lin, col, i)) {
            // Se aceita recebe o valor
            board[lin][col] = i;
```

```
            // Tente resolver o restante do Sudoku com a atribuição atual.
            if (resolve(board)) {
                return true;
            }

            // Se a atribuição atual não levar a uma solução, desfaça-a.
            board[lin][col] = 0;
        }
    }

    // Se nenhum número válido funcionar, retorne falso para retroceder.
    return false;
}
//Verifica Celula vazia e retorna cordenadas se ouver
function falta(board) {
    for (let lin = 0; lin < 9; lin++) {
        for (let col = 0; col < 9; col++) {
            if (board[lin][col] === 0) {
                return [lin, col];
            }
        }
    }
    return null;
}

function isAceita(board, lin, col, num) {
    // Verifique se o número não está na mesma linha ou coluna.
    for (let i = 0; i < 9; i++) {
        if (board[lin][i] === num || board[i][col] === num) {
            return false;
        }
    }
    return verificaQuad(board, lin, col, num);
}

function verificaQuad(board, lin, col, num){
     // Verifique se o número não está na mesma sub-grade 3x3.
    let startlin = Math.floor(lin / 3) * 3;
    let startCol = Math.floor(col / 3) * 3;
    for (let i = startlin; i < startlin + 3; i++) {
        for (let j = startCol; j < startCol + 3; j++) {
            if (board[i][j] === num) {
                return false;
            }
        }
    }
    return true;
```

```
}
function printSudoku(board) {
    for (let i = 0; i < 9; i++) {
        if (i % 3 === 0 && i !== 0) {
            console.log("---------------------"); // Linha horizontal divisória
        }

        let linStr = "";
        for (let j = 0; j < 9; j++) {
            if (j % 3 == 0 && j != 0) {
                linStr += "| "; // Linha vertical divisória
            }
            if (board[i][j] == 0) {
                linStr += "  ";
            } else {
                linStr += board[i][j] + " ";
            }
        }
        console.log(linStr);
    }
}
  if (process.argv.length < 3) {
    console.error('Por favor, forneça o nome do arquivo como argumento.');
    process.exit(1);
  }

  const nomeArquivo = process.argv[2];

// Use fs.readFile para ler o arquivo
fs.readFile(nomeArquivo, 'utf8', (err, data) => {
  if (err) {
    console.error('Erro ao ler o arquivo:', err);
    return;
  }
  // Dividir o conteúdo em linhas e, em seguida, dividir cada linha em números
  const linhas = data.trim().split('\n');
  const sudokuBoard = linhas.map((linha) => linha.split(' ').map(Number));


  printSudoku(sudokuBoard)
  let result = resolve(sudokuBoard);
  if (result) {
      console.log("Sudoku resolvido:");
      printSudoku(sudokuBoard)
  } else {
      console.log("Nenhuma solução encontrada.");

  }
```

```
});
```

6 Conclusão

O código em JavaScript apresentado neste relatório aborda o problema de resolução de Sudoku usando o paradigma de backtracking. Embora apresente pontos fortes, como simplicidade e eficácia, também possui limitações, como tempo de execução prolongado em quebra-cabeças difíceis e a falta de otimizações específicas. Este código pode ser um ponto de partida para a implementação de algoritmos de resolução de Sudoku mais avançados e eficientes.