Ramo Estudantil IEEE - UEL

Francisco Barone Gasparini Mugnaini
(francisco.barone@uel.br)

Nicholas Francis Lambert
(nicholas.lambert@uel.br)

Rafael Guerra Waldrigues de Campos Bueno
(rafawaldrigues.05@uel.br)

Henrique Gomes Araujo
(henrique.araujo@uel.br)

RELATÓRIO FINAL:

Projeto Resolvedor de Sudoku

Londrina
2023

Ramo Estudantil IEEE - UEL

Francisco Barone Gasparini Mugnaini
(francisco.barone@uel.br)

Nicholas Francis Lambert
(nicholas.lambert@uel.br)

Rafael Guerra Waldrigues de Campos Bueno
(rafawaldrigues.05@uel.br)

Henrique Gomes Araujo
(henrique.araujo@uel.br)

RELATÓRIO FINAL:

Projeto Resolvedor de Sudoku

Relatório apresentado ao Ramo Estudantil IEEE da Universidade Estadual de Londrina.

Diretor de Projetos: Nathan Andreani Netzel
Gestores de Projetos: Daniel Tresse Dourado, Levi Monteiro dos Santos

Londrina
2023

Barone, Francisco. Bueno, Rafael. Lambert, Nicholas. Araujo, Henrique. **Relatório Final**: Projeto Resolvedor de Sudoku. 2023. 12 folhas. Relatório apresentado ao Ramo Estudantil IEEE da Universidade Estadual de Londrina, Londrina, 2023.

RESUMO

O objetivo desse projeto foi criar um programa que resolvesse um jogo de Sudoku apresentado a ele. O jogo de Sudoku é um quebra-cabeça numérico cujo o intuito é completar uma tabela de dimensões 9 x 9, onde os únicos algarismos que podem ser usados variam de 1 a 9, sendo que esses números não podem coincidir entre si mesmos em qualquer linha, coluna, ou grade 3 x 3(se a tabela 9 x 9 for dividida em 9 grades de 3 x 3 sem sobreposição).

Palavras-chave: Sudoku, Matriz, Algoritmo, Código, Recursividade, LinguagemC.

SUMÁRIO

Sumário

1. INTRODUÇÃO

O jogo Sudoku, originado no Japão, é um exercício mental para quem gosta de problemas lógicos. Assim sendo, a resposta de tal jogo também é complicada de se descobrir. Felizmente, com a computação, esses problemas são facilmente resolvidos, reduzindo o tempo de resolução drasticamente.

Assim sendo, esse trabalho visa criar um método direto para a resolução desse jogo lógico, assim facilitando a verificação da resposta pelo usuário.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Regras do Sudoku

1) A tabela usada precisa ser uma com 9 linhas e 9 colunas
2) A tabela será subdividida em 9 outras de 3 linhas e 3 colunas
3) Cada tabela 3 x 3 precisa conter 9 algarismos e nenhum poderá ser incluso mais de uma única vez em diferentes tabelas
4) Somente poderão ser usados algarismos de 1 a 9 no sistema decimal
5) Qualquer linha ou coluna terá que ter seus algarismos diferentes uns dos outros, ou seja, em uma mesma linha ou coluna, não poderá ter números repetidos.

2.2 Representação Matricial Dentro do Código

A tabela 9 por 9 e subsequentes tabelas formadas por ela, sendo elas as 9 tabelas 3 por 3, são representadas, no algoritmo, por meio de matrizes quadradas de mesma ordem que as tabelas. Partido- se disso, é importante destacar que as matrizes 3 x 3, em prática, não são criadas e representadas, somente são derivadas da matriz principal e percorridas pelo programa.

3. METODOLOGIA

3.1 Objetivos do projeto

3.1.1 Implementação de um sudoku eficaz.

Nosso principal objetivo, é criar um resolvedor de sudoku que seja eficaz e consiga resolver diferentes níveis de dificuldade.

3.1.2 Documentação adequada

Planejamos documentar o código, de forma que todos os processos dele fossem claros, para caso algum outro indivíduo fosse usá-lo no futuro, consiga compreender os processos utilizados.

3.2 Linguagem de programação

A linguagem escolhida, foi C. A escolhemos, por conta de sua alta eficiência, alto controle e pouca abstração, sendo também a linguagem, a qual, os integrantes tinha mais conhecimento.

Consideramos utilizar a linguagem Python, mas a ideia foi descartada, pelo baixo nível de “maestria” dos integrantes com a linguagem.

3.3 Ambiente de desenvolvimento

A IDE utilizada para fazermos os projetos, foi o Visual Studio Code, conhecido por ser uma ótima IDE.

3.4 Estrutura de projeto

O Projeto foi dividido em 3 arquivos de fonte .c, tendo cada um deles, seu módulo. Os arquivos de fonte são divididos em main.c, leitura.c e sudoku.c. Sendo seus módulos, headers.h, leitura.h e sudoku.h respectivamente.

O arquivo, main.c, contém a chamada das funções, para de fato, resolver o sudoku, e gerar a saída. O leitura.c, é responsável por interpretar o arquivo de entrada e transpassa-lo para uma matriz. O sudoku.c é o arquivo, onde são implementadas as funções de resolução do sudoku.

O projeto foi separado desta forma, para que ficasse mais organizado, e facilitasse o entendimento e futuros ajustes.

3.5 Implementação do algoritmo

A implementação do algoritmo, consiste em um algoritmo de força bruta que utiliza recursão para resolver o sudoku. De forma resumida, os valores são testados, até gerar a resposta correta.

3.5.1 Entrada

A entrada do algoritmo, é por meio de um arquivo .txt, o qual terá um sudoku (9x9). Este arquivo será lido, interpretado e o sudoku nele contido, será passado para uma matriz, para que desta forma, seja manipulável.

Exemplo de entrada:

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

Função de leitura/conversão:

```c
void LerSudoku (FILE* sudoku, int** matriz) {

    for (int i=0; i<9; i++) {
        for (int j=0; j<9; j++) {
            if (fscanf(sudoku, "%d", &matriz[i][j]) != 1) return; // Lê cada elemento do sudoku e "transforma em int".
        }
    }
}
```

3.5.2 Resolução do sudoku

A ideia de resolução, consiste em testar os valores possíveis para cada região da grade 9x9. Conforme os valores são testados e conferidos em uma região, passa-se para a próxima, e realiza os mesmos testes. No momento em que, não existirem mais opções possíveis naquela região, significa que pelo menos, algum espaço foi preenchido de forma incorreta. Nesse momento, o código irá “voltar”, utilizando a recursão, tentando “consertar” os valores das regiões.

No instante em que todas as casas forem completas, significa que o sudoku está completo. No caso, uma casa vazia, é reconhecida como sendo igual a 0, da mesma forma na entrada pelo .txt.

Função a qual, testa valores de 1 a 9, em uma região do sudoku:

```c
void TestaValores (int linha, int coluna, int** sudoku) {

    if (linha == 9) return; // Caso termine o sudoku.

    for (int valor = 1; valor <= 9; valor++) { // Percorre os valores a serem testados
        if (ConfereValor(linha, coluna, valor, sudoku)) {
            int aux = sudoku[linha][coluna];
            sudoku[linha][coluna] = valor;

            if (coluna == 8) TestaValores(linha + 1, 0, sudoku); // Caso tenha chego ao fim da linha, pula para a proxima.
            else TestaValores(linha, coluna + 1, sudoku); // Vai para a proxima casa da linha

            if (!ConfereSudoku(sudoku)) sudoku[linha][coluna] = aux;
        }
    }
}
```

Função a qual confere se um valor “valor”, pode ser posto em uma região do sudoku:

```c
bool ConfereValor (int linha, int coluna, int valor, int** sudoku) {

    if (sudoku[linha][coluna] == valor) return true;
    if (sudoku[linha][coluna] != 0) return false;

    for (int col = 0; col < 9; col++) if (sudoku[linha][col] == valor) return false; // Confere se o valor a ser testado, existe na mesma linha.
    for (int lin = 0; lin < 9; lin++) if (sudoku[lin][coluna] == valor) return false; // Confere se o valor a ser testado, existe na mesma coluna.

    int linhaQ = linha/3; // Representa a linha de um dos quadrantes 3x3.
    int colunaQ = coluna/3; // Representa a coluna de um dos quadrantes 3x3.

    // Confere se o valor testado, existe no mesmo quadrante 3x3.
    for (int lin = linhaQ * 3; lin < (linhaQ + 1) * 3; lin++) {
        for (int col = colunaQ * 3; col < (colunaQ + 1) * 3; col++) {
            if (sudoku[lin][col] == valor) return false;
        }
    }

    return true;
}
```

Função a qual confere se o sudoku está completo. Nesse caso, ela não confere se o sudoku está correto, mas sim se a matriz 9x9 está completa, pois a função ConfereValor e TestaValores, já garante que caso o sudoku esteja completo, ele está correto.

```c
bool ConfereSudoku (int** sudoku) {

    for (int i=0; i<9; i++) {
        for (int j=0; j<9; j++) {
            if (*(*(sudoku + j) + i) == 0) return false;
        }
    }

    return true;
}
```

Por fim, o sudoku final é armazenado em uma matriz..

3.5.3 Saída do programa

A saída do programa, se consiste em um arquivo .txt. Ao finalizar a resolução do sudoku, a matriz será escrita em um arquivo chamado saída.txt.

Função de escrita do sudoku:

```c
void EscreveSudoku (int** sudoku) {

    FILE* arq = fopen("saida.txt", "w");

    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            fprintf (arq, "%d ", sudoku[i][j]); // Escreve as linhas do sudoku.
            if ((j+1) % 3 == 0) fprintf (arq, " "); // Escreve um espaçamento entre os espaços 3x3.
        }
        fprintf (arq, "\n");
        if ((i+1) % 3 == 0) fprintf (arq, "\n");
    }

    fclose(arq);
}
```

Exemplo da saída:

```
1   4 5 9  3 6 8  2 7 1
2   7 6 1  5 2 9  8 3 4
3   8 2 3  7 1 4  5 9 6
4
5   3 1 7  2 8 5  6 4 9
6   6 8 5  4 9 1  7 2 3
7   9 4 2  6 3 7  1 8 5
8
9   1 7 8  9 5 3  4 6 2
10  2 9 4  1 7 6  3 5 8
11  5 3 6  8 4 2  9 1 7
12
```

3.6 Testes e validação

Após serem realizados vários testes, for aferido que o solucionador de sudoku, é extremamente assertivo em suas respostas. Por ser um algoritmo de força bruta, acaba não sendo tão eficiente, porém compensa isso com sua simplicidade de implementação.

O único caso que este programa pode gerar uma insatisfação, é caso a matriz 9x9 fornecida no arquivo de entrada .txt, possua mais de uma solução. Quando isso acontece, o programa irá escrever apenas uma das soluções. Uma forma de resolver isso, é alterando a função TestaValores da seguinte forma:

```
void TestaValores (int linha, int coluna, int** sudoku) {

    if (linha == 9) {
        EscreveSudoku(sudoku);
        return;
    }
    for (int valor = 1; valor <= 9; valor++) { // Percorre os valores a serem testados
        if (ConfereValor(linha, coluna, valor, sudoku)) {
            int aux = sudoku[linha][coluna];
            sudoku[linha][coluna] = valor;

            if (coluna == 8) TestaValores(linha + 1, 0, sudoku); // Caso tenha chego ao fim da linha, pula para a proxima.
            else TestaValores(linha, coluna + 1, sudoku); // Vai para a proxima casa da linha

            sudoku[linha][coluna] = aux;
        }
    }
}
```

Desta forma, irá escrever todas as soluções possíveis, porém, a matriz final, não terá um sudoku resolvido armazenado, mas sim, o estado inicial da matriz 9x9.

4. RESULTADOS E DISCUSSÕES

De acordo com o desenvolvimento, pode-se aferir que o resultando desse teste apresenta apenas um dos resultados possíveis para o Sudoku inserido. Entretanto, a programação pode ser modificada para a apresentação de todas as possibilidades, porém como a quantidade de possíveis resoluções são de um valor muito elevado, é inviável sua implementação prática.

5. CONCLUSÕES

No final, após testarmos e conferirmos o resultado chegamos ao consenso de que o código funcionava perfeitamente. Ele lia o arquivo .txt que o entregávamos e chegava a uma resolução correta.

REFERÊNCIAS BIBLIOGRÁFICAS

APÊNDICES

APÊNDICE A

Código Completo:

```c
/*
    Esta biblioteca possui uma função para realizar a leitura de um
    arquivo.txt, e interpretá-lo em forma de sudoku.
*/

#ifndef _LEITURA_H
#define _LEITURA_H

#include "sudoku.h"

/*
    Esta função, recebe um sudoku.txt e uma matriz como parâmetros.
    Seu objetivo, é "passar" os valores do sudoku.txt para a matriz
    "matriz".
*/
void LerSudoku (FILE* sudoku, int** matriz);

#endif
```

```c
#include "leitura.h"

void LerSudoku (FILE* sudoku, int** matriz) {

    for (int i=0; i<9; i++) {
        for (int j=0; j<9; j++) {
            if (fscanf(sudoku, "%d", &matriz[i][j]) != 1) return; // Lê cada elemento do sudoku e "transforma em int".
        }
    }
}
```

```c
/*
    Esta biblioteca, inclui funções para a resolução de um sudoku 9x9,
    além de funções auxiliares, para conferir se está completo e para
    escreve o sudoku.
*/

#ifndef _SUDOKU_H
#define _SUDOKU_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>

/*
    Função a qual resolver um sudoku. Ela recebe uma matriz int,
    a qual, "armazena" o sudoku a ser resolvido.
*/
void ResolveSudoku (int** sudoku);

/*
    Função auxiliar, a qual vai testar os valores possíveis para
    um espaço do sudoku.
*/
void TestaValores (int linha, int coluna, int** sudoku);

/*
    Função booleana, a qual conferese se uma certo valor int "valor"
    1 <= valor <= 9, é um possível valor a ser testado na linha "linha"
    e coluna "coluna" do sudoku "sudoku".

    Retorna true caso o elemento possa ser testado, e false caso contrário.
*/
bool ConfereValor (int linha, int coluna, int valor, int** sudoku);

/*
    Função a qual confere se o sudoku "sudoku" está completo.

    Retorna true caso esteja, e false caso contrário.
*/
bool ConfereSudoku (int** sudoku);

/*
    Função a qual escreve no terminal, o Sudoku "sudoku".
*/
void EscreveSudoku (int** sudoku);

#endif
```

```c
#include "sudoku.h"

void ResolveSudoku (int** sudoku) {

    TestaValores (0, 0, sudoku);
}

void TestaValores (int linha, int coluna, int** sudoku) {

    if (linha == 9) return; // Caso termine o sudoku.

    for (int valor = 1; valor <= 9; valor++) { // Percorre os valores a serem testados
        if (ConfereValor(linha, coluna, valor, sudoku)) {
            int aux = sudoku[linha][coluna];
            sudoku[linha][coluna] = valor;

            if (coluna == 8) TestaValores(linha + 1, 0, sudoku); // Caso tenha chego ao fim da linha, pula para a proxima.
            else TestaValores(linha, coluna + 1, sudoku); // Vai para a proxima casa da linha

            if (!ConfereSudoku(sudoku)) sudoku[linha][coluna] = aux;
        }
    }
}

bool ConfereValor (int linha, int coluna, int valor, int** sudoku) {

    if (sudoku[linha][coluna] == valor) return true;
    if (sudoku[linha][coluna] != 0) return false;

    for (int col = 0; col < 9; col++) if (sudoku[linha][col] == valor) return false; // Confere se o valor a ser testado, existe na mesma linha.
    for (int lin = 0; lin < 9; lin++) if (sudoku[lin][coluna] == valor) return false; // Confere se o valor a ser testado, existe na mesma coluna.

    int linhaQ = linha/3; // Representa a linha de um dos quadrantes 3x3.
    int colunaQ = coluna/3; // Representa a coluna de um dos quadrantes 3x3.

    // Confere se o valor testado, existe no mesmo quadrante 3x3.
    for (int lin = linhaQ * 3; lin < (linhaQ + 1) * 3; lin++) {
        for (int col = colunaQ * 3; col < (colunaQ + 1) * 3; col++) {
            if (sudoku[lin][col] == valor) return false;
        }
    }

    return true;
}

bool ConfereSudoku (int** sudoku) {

    for (int i=0; i<9; i++) {
        for (int j=0; j<9; j++) {
            if (*(*(sudoku + j) + i) == 0) return false;
        }
    }
```

```c
void EscreveSudoku (int** sudoku) {

    FILE* arq = fopen("saida.txt", "w");

    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            fprintf (arq, "%d ", sudoku[i][j]); // Escreve as linhas do sudoku.
            if ((j+1) % 3 == 0) fprintf (arq, " "); // Escreve um espaçamento entre os espaços 3x3.
        }
        fprintf (arq, "\n");
        if ((i+1) % 3 == 0) fprintf (arq, "\n");
    }

    fclose(arq);
}
```

```h
/*
    Esta biblioteca "chama" todas as bibliotecas utilizadas na
    criação do projeto, em resumo, é a biblioteca principal do
    código.
*/

#ifndef _HEADERS_H
#define _HEADERS_H

#include "leitura.h"
#include "sudoku.h"

#endif
```

```c
/*
    Este programa, tem como intuito, resolver um sudoku 9x9 incompleto,
    o qual, será dado como entrada em um arquivo .txt. Em seguida, retornará
    este e irá escrever a resolução do sudoku em outro arquivo .txt.

    O programa será feito na linguagem C.

    Caso não saiba como funciona um sudoku ou como resolvê-lo, acesse
    o site abaixo.
    https://sudoku.com/pt/regras-do-sudoku/

    A estratégia utilizada foi a de BackTracking, para saber mais, vá para:
    http://www3.decom.ufop.br/toffolo/site_media/uploads/2011-1/bcc402/slides/10._backtracking.pdf
*/

#include "headers.h"

int main() {

    // Alocando matriz, para armazenar o sudoku, dinamicamente.
    int** Sudoku = malloc(9 * sizeof(int*));
    for (int i=0; i<9; i++) Sudoku[i] = malloc(9 * sizeof(int));

    FILE* arq = fopen("sudoku.txt", "r");

    LerSudoku(arq, Sudoku); // Passando o sudoku para a matriz.

    ResolveSudoku(Sudoku); // Resolvendo o sudoku.

    // Conferindo se o sudoku de fato foi resolvido.
    if (ConfereSudoku(Sudoku)) EscreveSudoku(Sudoku);
    else printf ("\nSudoku não possui solução\n\n");

    return 0;

}
```

Sudoku.txt = Será o arquivo de entrada do código, segue o exemplo:

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