COLORAÇÃO DE GRAFOS

Eduardo Savian, Marcos Fehlauer, Pablo Marques

INTRODUÇÃO

- O Sudoku é um quebra-cabeça lógico com tabuleiro N por N, sendo a raiz quadrada de N um número inteiro, de modo que cada linha, coluna e sub-grade de tamanho da raiz quadrada de N que contenha todos os números de 1 a N sem repetição;
- A técnica usa é o *Backtracking* (Busca Recursiva), que a partir do input do usuário, pinta uma célula vazia com o número 1 e verifica se ele não viola as regras. Se não violar, continua preenchendo as células subsequentes.;
- Se chegar a uma célula onde não podemos preencher nenhum número válido, se é voltado atrás (*backtrack*) e tenta outro número na célula anterior. Esse processo continua até que todas as células estejam preenchidas corretamente.

PONTOS POSITOS E NEGATIVOS

- Forma bastante fácil de implementar um problema que de outra forma seria muito mais complexo de se resolver;
- É examinado todas as opções, garantindo que a solução final seja encontrada;
- Se não por restrições (constraints) executão acontecerá uma busca exaustiva e tenderão à explosão combinatória;
- Necessitam de muita memória no Stack;
- A ordem em que os candidatos são explorados afeta o desempenho. Uma escolha inadequada de ordem pode levar a um tempo de execução mais longo.

PRINCIPAL

```rust
fn main() -> ExitCode {
    let args: Vec<String> = env::args().collect();

    // Less or more args than 3
    if args.len() != 4 { return ExitCode::from(1) }

    // Invalid arg for graph_order
    let graph_order: usize = match args[1].parse() {
        Ok(e) => e,
        _ => return ExitCode::from(2)
    };

    // Graph order different from 4 or 9 or 16
    if [4, 9, 16].contains(&graph_order) == false {
        return ExitCode::from(3);
    }

    let mut graph = generate_partial_sudoku(graph_order);

    // Invalid arg for row number
    let row: usize = match args[2].parse() {
        Ok(e) => if e >= graph_order { return ExitCode::from(4)} else {e}
        _ => return ExitCode::from(5)
    };
```

PRINCIPAL

```
// Invalid arg for col number
    let col: usize = match args[3].parse() {
        Ok(e) => if e >= graph_order { return ExitCode::from(6)} else {e}
        _ => return ExitCode::from(7)
    };

    // Position on graph is not empty
    if graph[row][col] != 0 {
        return ExitCode::from(8);
    }

    println!("{}", graph_to_json(&graph));
    graph_coloring(&mut graph, (row, col), 0);
    println!("{}", graph_to_json(&graph));

    ExitCode::from(0)
}
```

GERAR JOGO SUDOKU PARCIALMENTE COMPLETO

```rust
use rand::seq::SliceRandom;
use rand::thread_rng;
use std::vec::Vec;

fn generate_partial_sudoku(size: usize) -> Vec<Vec<i32>> {
    let base = (size as f64).sqrt() as usize;
    let mut rng = thread_rng();
    let pattern = |r: usize, c: usize| -> usize { (base * (r % base) + r / base + c) % size };
    fn shuffle(s: &mut [usize]) {
        s.shuffle(&mut thread_rng());
    }
    let r_base: Vec<usize> = (0..base).collect();
    let mut rows: Vec<usize> = Vec::new();
    let mut cols: Vec<usize> = Vec::new();
    for g in r_base.iter().copied() {
        let mut r_base_shuffle = r_base.clone();
        shuffle(&mut r_base_shuffle);
        for r in r_base_shuffle.iter().copied() {
            rows.push(g * base + r);
        }
    }
```

GERAR JOGO SUDOKU PARCIALMENTE FEITO

```
for g in r_base.iter().copied() {
        let mut c_base_shuffle = r_base.clone();
        shuffle(&mut c_base_shuffle);
        for c in c_base_shuffle.iter().copied() {
            cols.push(g * base + c);
        }
    }
    let mut nums: Vec<usize> = (1..=base * base).collect();
    shuffle(&mut nums);
    let mut board: Vec<Vec<i32>> = rows
        .iter()
        .map(|&r| cols.iter().map(|&c| nums[pattern(r, c)] as i32).collect())
        .collect();
    let squares = size * size;
    let empties = squares * 3 / 4;
    let mut positions: Vec<usize> = (0..squares).collect();
    positions.shuffle(&mut rng);
    for &p in positions.iter().take(empties) {
        board[p / size][p % size] = 0;
    }
    board
}
```

COLORINDO O GRAFO

```rust
fn graph_coloring(
    graph: &mut Vec<Vec<i32>>,
    start_point: (usize, usize),
    curr_iteration: u64,
) -> bool {
    let (mut row, mut col) = start_point;
    if curr_iteration != 0 {
        let empty = find_non_colored_location(&graph);
        if empty.is_none() {
            return true;
        }
        (row, col) = empty.unwrap();
    }
    for color in 1..=graph.len() {
        if can_this_color_be_used(&graph, row, col, color as i32) {
            graph[row][col] = color as i32;
            let cond: bool = graph_coloring(graph, (0, 0), curr_iteration + 1);
            if cond {
                return true;
            } else {
                graph[row][col] = 0;
            }
        }
    }
    return false;
}
```

ENCONTRAR LOCAL NÃO COLORIDO

```rust
fn find_non_colored_location(graph: &Vec<Vec<i32>>) -> Option<(usize, usize)> {
    for (i, row) in graph.iter().enumerate() {
        if let Some(j) = row.iter().position(|col| *col == 0) {
            return Some((i, j));
        }
    }
    None
}
```

```
0  0  | 0  3
0  0  | 0  0
------------
0  0  | 3  0
0  1  | 2  0

Linha [0 a 3]:
0
Coluna [0 a 3]:
0

------------------------

Pos(0,0) -> c(1)
Pos(0,1) -> c(2)
Pos(0,2) -> c(4)
Pos(1,0) -> c(3)
Pos(1,1) -> c(4)
Pos(1,2) -> c(1)
Pos(1,3) -> c(2)
Pos(2,0) -> c(2)
Iter. 7 | Combinação inválida: Pos(2, 0) => c(2)
Pos(2,0) -> c(4)
Iter. 7 | Combinação inválida: Pos(2, 0) => c(4)
Iter. 6 | Combinação inválida: Pos(1, 3) => c(2)
Iter. 5 | Combinação inválida: Pos(1, 2) => c(1)
Iter. 4 | Combinação inválida: Pos(1, 1) => c(4)
Iter. 3 | Combinação inválida: Pos(1, 0) => c(3)
Pos(1,0) -> c(4)
Pos(1,1) -> c(3)
Pos(1,2) -> c(1)
Pos(1,3) -> c(2)
Pos(2,0) -> c(2)
Pos(2,1) -> c(4)
Pos(2,3) -> c(1)
Pos(3,0) -> c(3)
Pos(3,3) -> c(4)
1  2  | 4  3
4  3  | 1  2
------------
2  4  | 3  1
3  1  | 2  4
```

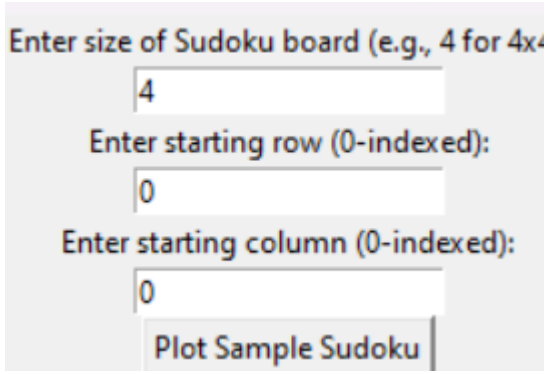
VERIFICAR SE A COR PODE SER USADA

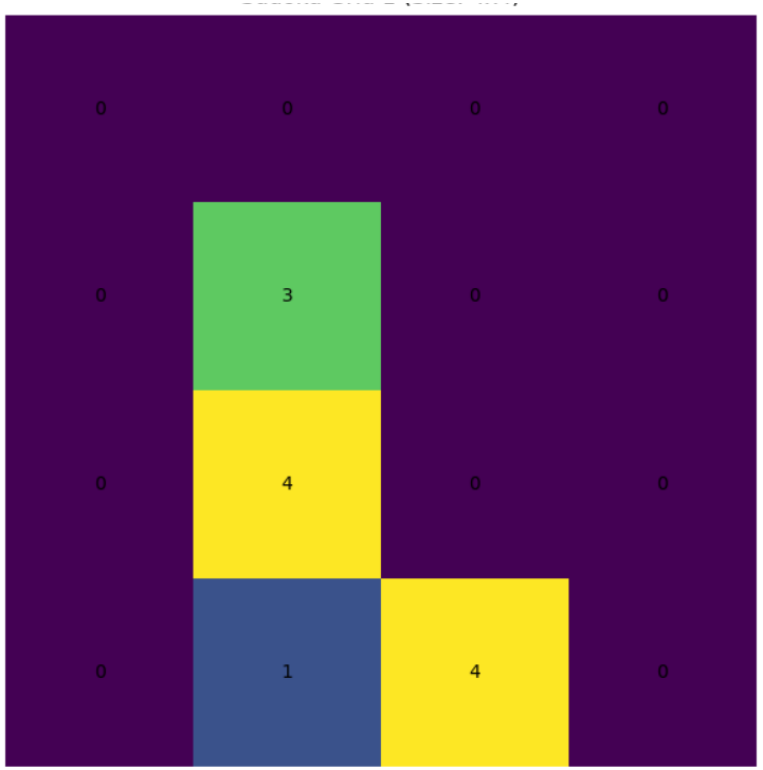
```rust
fn can_this_color_be_used(graph: &Vec<Vec<i32>>, row: usize, col: usize, color: i32) -> bool {
    if graph[row].iter().any(|&c| c == color) {
        return false;
    }
    if graph.iter().any(|row| row[col] == color) {
        return false;
    }
    let block_size: usize = (graph.len() as f64).sqrt() as usize;
    let start_row = (row / block_size) * block_size;
    let start_col = (col / block_size) * block_size;
    for i in 0..block_size {
        for j in 0..block_size {
            if graph[start_row + i][start_col + j] == color {
                return false;
            }
        }
    }
    return true;
}
```

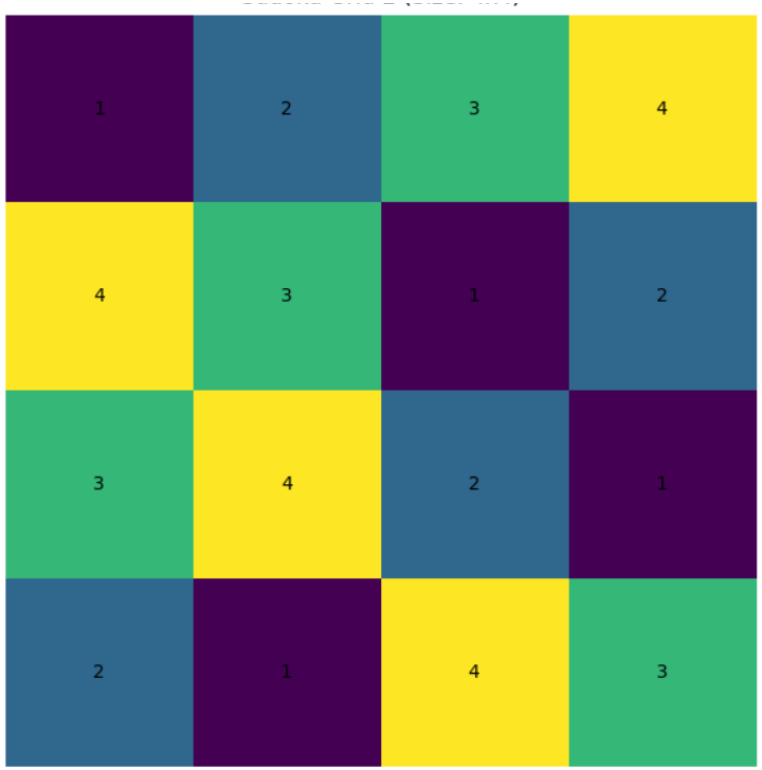
CONVERTER GRAFO PARA JSON

```rust
fn graph_to_json(graph: &Vec<Vec<i32>>) -> String {
    let mut res = String::from("");

    for row in 0..graph.len() {
        res.push_str("");
        for col in 0..graph.len() {
            res.push_str(&format!("{},", graph[row][col]));
        }
        res.pop();
        res.push_str(";");
    }
    res.pop();
    res.push_str("-");

    res
}
```

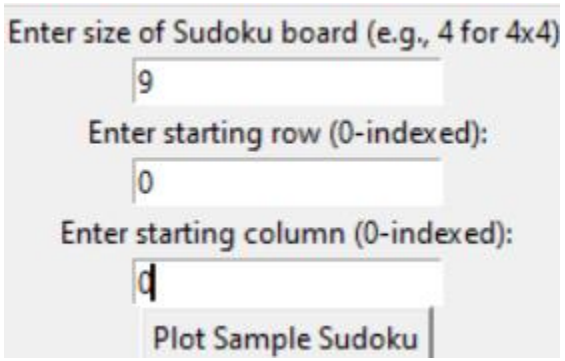
EXEMPLO 4X4

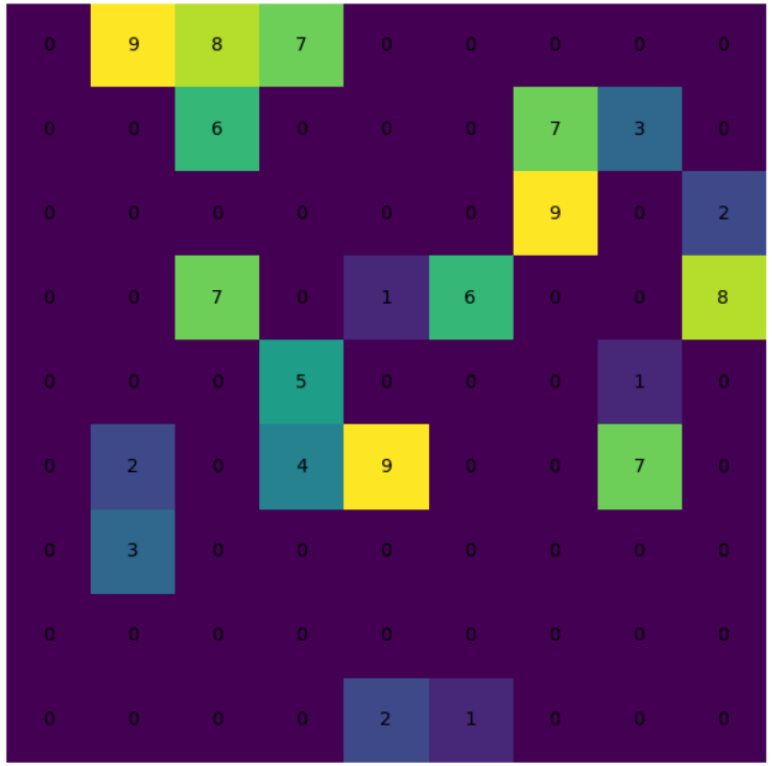

Enter size of Sudoku board (e.g., 4 for 4x
4
Enter starting row (0-indexed):
0
Enter starting column (0-indexed):
0
Plot Sample Sudoku

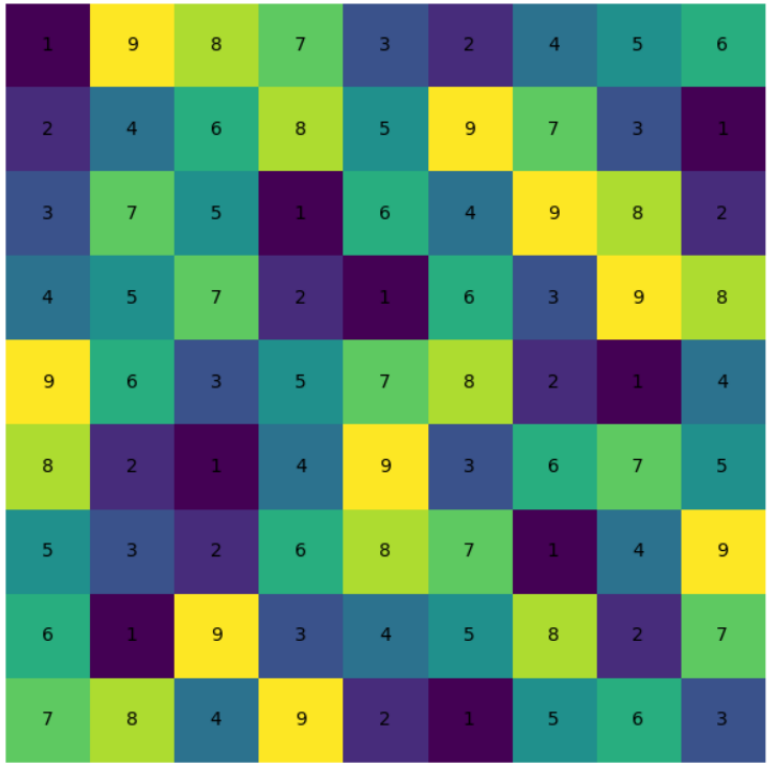

0 0 0 0
0 3 0 0
0 4 0 0
0 1 4 0


1 2 3 4
4 3 1 2
3 4 2 1
2 1 4 3

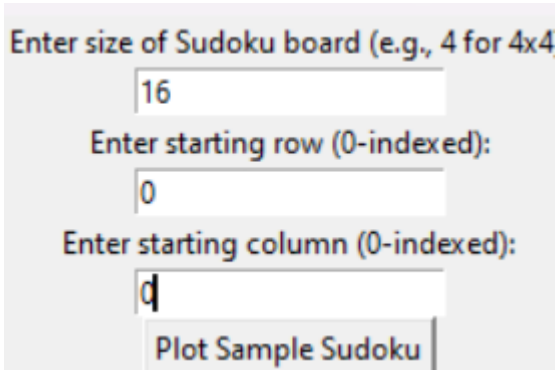
EXEMPLO 9X9

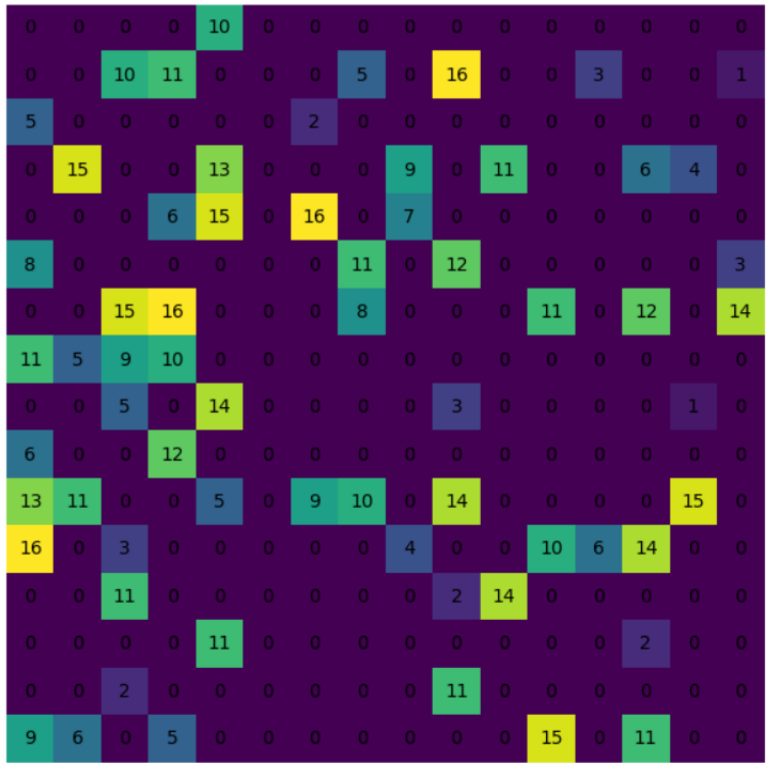

Enter size of Sudoku board (e.g., 4 for 4x4)
9
Enter starting row (0-indexed):
0
Enter starting column (0-indexed):
0
Plot Sample Sudoku

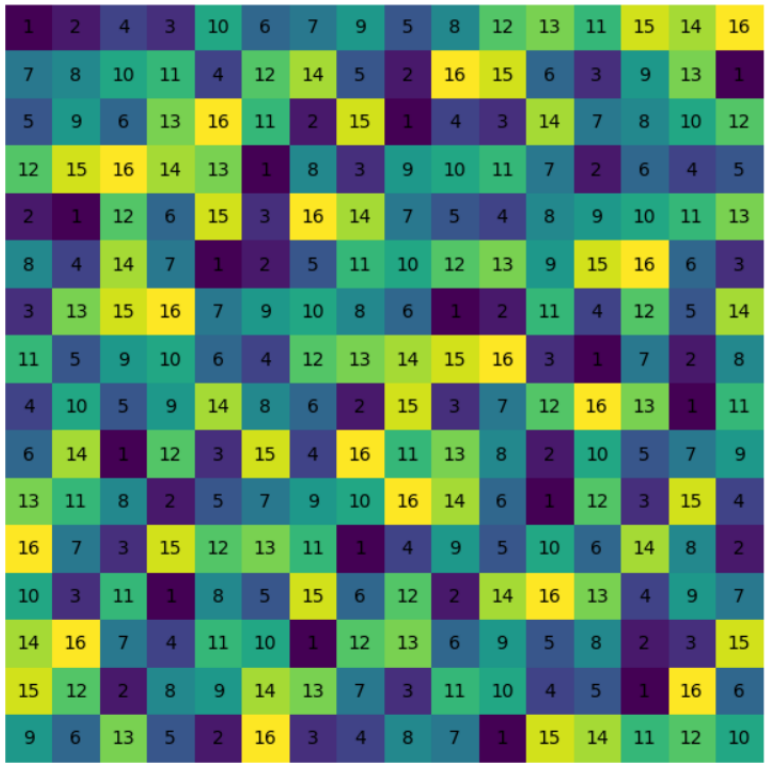

0 9 8 7 0 0 0 0 0
0 0 6 0 0 0 7 3 0
0 0 0 0 0 0 9 0 2
0 0 7 0 1 6 0 0 8
0 0 0 5 0 0 0 1 0
0 2 0 4 9 0 0 7 0
0 3 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 2 1 0 0 0


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

EXEMPLO 16X16







REFERÊNCIAS BIBLIOGRÁFICAS

- AULA, B. –. **Túlio Toffolo – www.toffolo.com.br Marco Antônio Carvalho – marco.opt@gmail.com**. Disponível em: <http://www3.decom.ufop.br/toffolo/site_media/uploads/2011-1/bcc402/slides/10._backtracking.pdf>. Acesso em: 8 maio. 2024.
- **O que é um algoritmo Backtracking?** Disponível em: <https://pt.stackoverflow.com/questions/103184/o-que-%C3%A9-um-algoritmo-backtracking>. Acesso em: 8 maio. 2024.
- WIKIPEDIA CONTRIBUTORS. **Sudoku**. Disponível em: <https://en.wikipedia.org/w/index.php?title=Sudoku&oldid=1222342494>.

OBRIGADO