
COLORAÇÃO DE GRAFOS

Eduardo Savian, Marcos Fehlaue, 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 usada 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) execução 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

```
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

```
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

```
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

```
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  
}
```

VERIFICAR SE A COR PODE SER USADA

```
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

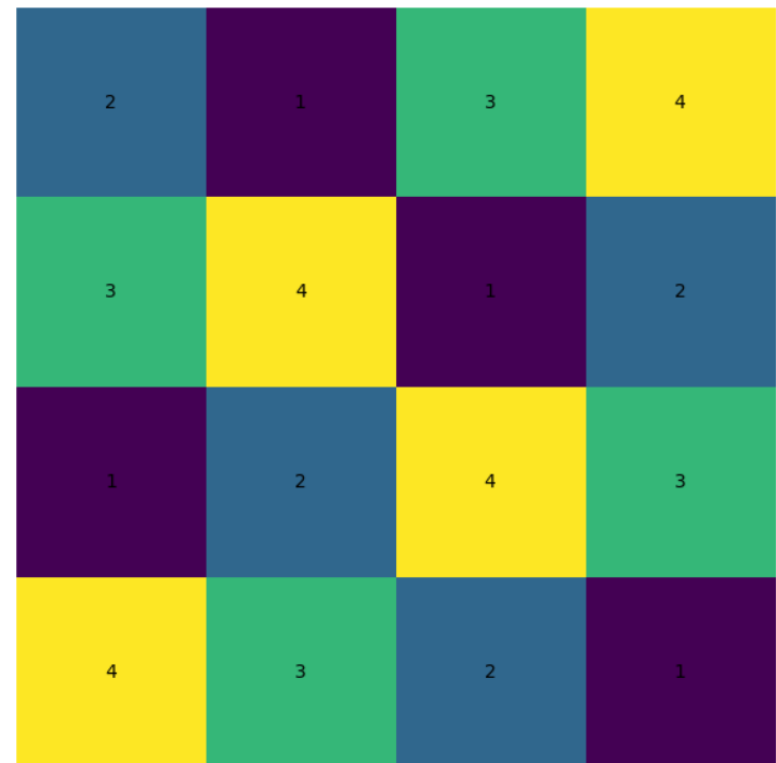
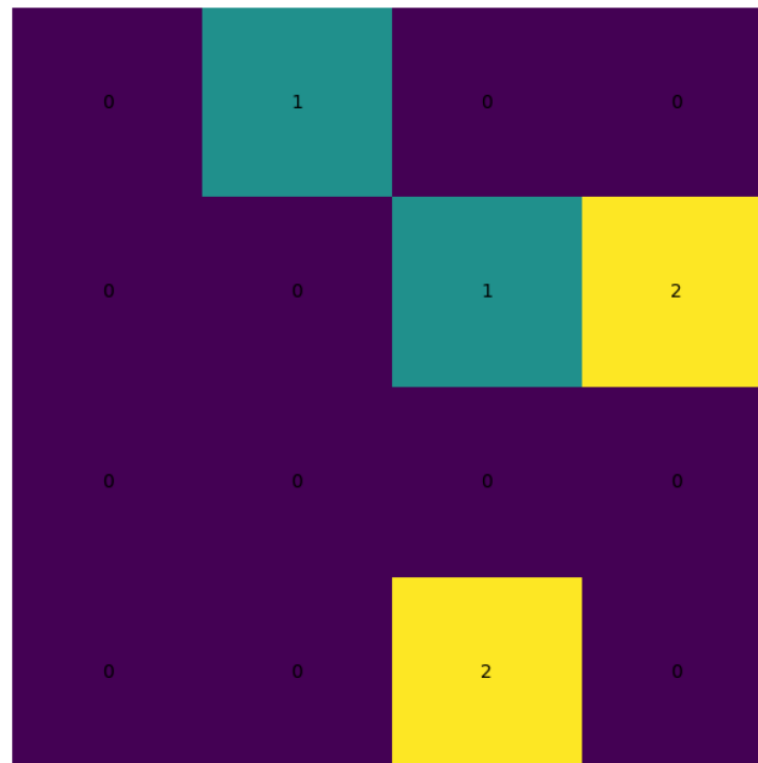
```
fn graph_to_json(graph: &Vec<Vec<i32>>) -> String {  
    let mut res = String::from("");  
  
    for row in 0..graph.len() {  
        res.push_str("");  
        for col in 0..graph.len() {  
            res.push_str(&format!("{}", graph[row][col]));  
        }  
        res.pop();  
        res.push_str(";");  
    }  
    res.pop();  
    res.push_str("-");  
  
    res  
}
```

EXEMPLO 4X4

Enter size of Sudoku board (e.g., 4 for 4x4):

Enter starting row (0-indexed):

Enter starting column (0-indexed):



EXEMPLO 9X9

Enter size of Sudoku board (e.g., 4 for 4x4)

Enter starting row (0-indexed):

Enter starting column (0-indexed):

Plot Sample Sudoku

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

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

EXEMPLO 16X16

Enter size of Sudoku board (e.g., 4 for 4x4)

Enter starting row (0-indexed):

Enter starting column (0-indexed):

[Plot Sample Sudoku](#)

0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0
0	0	10	11	0	0	0	5	0	16	0	0	3	0	0	1
5	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0
0	15	0	0	13	0	0	0	9	0	11	0	0	6	4	0
0	0	0	6	15	0	16	0	7	0	0	0	0	0	0	0
8	0	0	0	0	0	0	11	0	12	0	0	0	0	0	3
0	0	15	16	0	0	0	8	0	0	0	11	0	12	0	14
11	5	9	10	0	0	0	0	0	0	0	0	0	0	0	0
0	0	5	0	14	0	0	0	0	3	0	0	0	0	1	0
6	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0
13	11	0	0	5	0	9	10	0	14	0	0	0	0	15	0
16	0	3	0	0	0	0	0	4	0	0	10	6	14	0	0
0	0	11	0	0	0	0	0	0	2	14	0	0	0	0	0
0	0	0	0	11	0	0	0	0	0	0	0	0	2	0	0
0	0	2	0	0	0	0	0	0	11	0	0	0	0	0	0
9	6	0	5	0	0	0	0	0	0	15	0	11	0	0	0

1	2	4	3	10	6	7	9	5	8	12	13	11	15	14	16
7	8	10	11	4	12	14	5	2	16	15	6	3	9	13	1
5	9	6	13	16	11	2	15	1	4	3	14	7	8	10	12
12	15	16	14	13	1	8	3	9	10	11	7	2	6	4	5
2	1	12	6	15	3	16	14	7	5	4	8	9	10	11	13
8	4	14	7	1	2	5	11	10	12	13	9	15	16	6	3
3	13	15	16	7	9	10	8	6	1	2	11	4	12	5	14
11	5	9	10	6	4	12	13	14	15	16	3	1	7	2	8
4	10	5	9	14	8	6	2	15	3	7	12	16	13	1	11
6	14	1	12	3	15	4	16	11	13	8	2	10	5	7	9
13	11	8	2	5	7	9	10	16	14	6	1	12	3	15	4
16	7	3	15	12	13	11	1	4	9	5	10	6	14	8	2
10	3	11	1	8	5	15	6	12	2	14	16	13	4	9	7
14	16	7	4	11	10	1	12	13	6	9	5	8	2	3	15
15	12	2	8	9	14	13	7	3	11	10	4	5	1	16	6
9	6	13	5	2	16	3	4	8	7	1	15	14	11	12	10

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