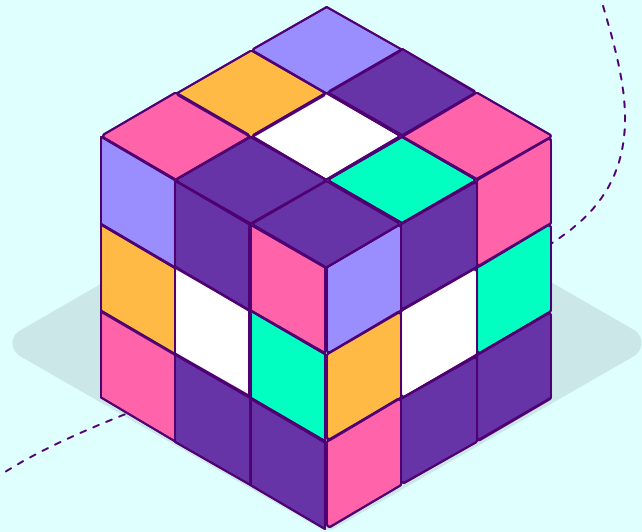


CUBOMÁGICO

Artur Bertoni e Maria Eduarda Fischer



AGENDA

Introdução

01

02

Como Funciona

Resolução

03

04

Conclusão

01

Introdução



Introdução

01

Objetivo

O objetivo deste trabalho é implementar um resolvidor de Cubo Magico (Rubik's Cube Solver).

02

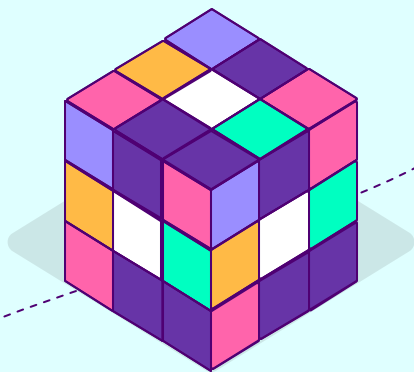
Observações

- ❖ As dimensões do cubo sejam 3x3x3.
- ❖ Seja possível configurar o estado inicial de forma manual.
- ❖ Todos os passos executados pelo resolvidor até a resolução final do cubo sejam mostrados e possam ser auditados.

02

Como Funciona?



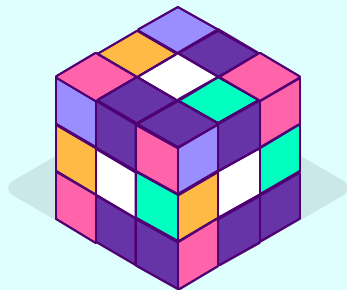


Como funciona ?

O cubo mágico, também conhecido como cubo de Rubik, é um quebra-cabeça tridimensional inventado pelo arquiteto e professor de arquitetura húngaro Ernő Rubik em 1974.

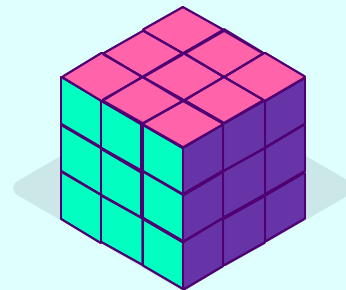
Um cubo mágico $3 \times 3 \times 3$ é um quebra-cabeça tridimensional composto por um cubo formado por 27 cubos menores. Cada face do cubo é composta por 9 desses cubos menores. A estrutura interna do cubo $3 \times 3 \times 3$ é baseada em um mecanismo de eixos que permite a rotação das camadas.

PROBLEMA VS. SOLUÇÃO



PROBLEMA

A partir do embaralhamento de cores, o objetivo restaurar o cubo para sua forma original, com cada face composta por uma única cor.



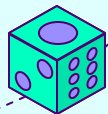
SOLUÇÃO

Algoritmos de padrões e sequências de movimentos que são executados, como permutar duas peças ou mover um cubo para uma posição desejada.

03

Resolução





ESTRUTURA

```
public class UserInterface {  
  
    2 usages arturSantos  
    public void menu(String dir) {  
  
        Cube cube = PersistenceOperations.readCube(dir);  
  
        ProblemA p = new ProblemA(cube);  
        IterativeDepthSearch b = new IterativeDepthSearch(p, maximumDepth: 7, iteration: 1, pruning: true);  
  
        System.out.println(b.toFind());  
    }  
}
```

01

01

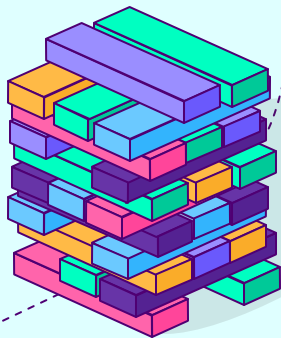
O trecho de código apresentado demonstra a classe UserInterface, responsável por iniciar toda a aplicação.

```
private void initializeReach() {  
    Row r0 = new Row("red", "red", "red");  
    Row r1 = new Row("red", "red", "red");  
    Row r2 = new Row("red", "red", "red");  
    Face frontFace = new Face(r0, r1, r2);  
    r0 = new Row("white", "white", "white");  
    r1 = new Row("white", "white", "white");  
    r2 = new Row("white", "white", "white");  
    Face rightFace = new Face(r0, r1, r2);  
    r0 = new Row("orange", "orange", "orange");  
    r1 = new Row("orange", "orange", "orange");  
    r2 = new Row("orange", "orange", "orange");  
    Face backFace = new Face(r0, r1, r2);  
    r0 = new Row("yellow", "yellow", "yellow");  
    r1 = new Row("yellow", "yellow", "yellow");  
    r2 = new Row("yellow", "yellow", "yellow");  
    Face leftFace = new Face(r0, r1, r2);  
    r0 = new Row("green", "green", "green");  
    r1 = new Row("green", "green", "green");  
    r2 = new Row("green", "green", "green");  
    Face topFace = new Face(r0, r1, r2);  
    r0 = new Row("blue", "blue", "blue");  
    r1 = new Row("blue", "blue", "blue");  
    r2 = new Row("blue", "blue", "blue");  
    Face downFace = new Face(r0, r1, r2);  
  
    reach = new Cube(frontFace, rightFace, backFace, leftFace, topFace, downFace);  
}
```

02

02

Na classe Status, que implementa a interface Serializable, temos o método initializeReach(), que é responsável por gerar o objetivo do programa, ou seja, o cubo resolvido.



ESTRUTURA

```
public String toFind() { //General search
    ArrayList<SearchNode> LS;
    SearchNode current;
    SearchNode start = new SearchNode( parent: null, problem.getStart(), depth: 0, value: 0);
    frontier.insert(start);
    String solution;
    double startTime = System.currentTimeMillis();
    boolean resolved = false;

    do {
        current = frontier.eliminates();
        if (problem.objectiveTest(current.getCurrent())) {
            resolved = true;
        } else {
            LS = problem.successors(current);
            createsTreeNodesList(LS, current);
            spatialComplexity += LS.size();
        }
        showPartialSolution(current);
    } while (!resolved && !frontier.isEmpty());
    if (!this.frontier.isEmpty()) {
        solution = "Final Solution: \n\n" + current;
    } else {
        solution = "Solution not found";
    }
    temporalComplexity = (System.currentTimeMillis() - startTime) / 1000;
    return solution;
}
```

04

```
public abstract class Search {

    4 usages
    protected int advance;
    7 usages
    protected Problem problem;
    3 usages
    protected int spatialComplexity;
    3 usages
    protected double temporalComplexity;
    10 usages
    protected Frontier frontier;
    2 usages
    protected boolean pruning;
    1 usage
    protected int depth;
    4 usages
    protected Hashtable<Integer, Integer> status;

    1 usage - arturSantos
    protected Search(Problem problem, boolean pruning) {
        this.problem = problem;
        frontier = new Frontier();
        status = new Hashtable<>();
        this.pruning = pruning;
        this.advance = 0;
    }
```

03

03

Este trecho de código é uma classe abstrata chamada Search, que fornece operações para aplicar uma estratégia de busca em um caso geral. A classe Search é uma classe pai para diferentes estratégias de busca específicas.

A classe Search possui dois construtores. Ambos inicializam a variável pruning com o valor fornecido e criam uma instância da classe Frontier para a variável frontier. Além disso, eles inicializam a variável status com uma nova instância de Hashtable e definem o valor de advance como zero.

04


O método toFind é a principal função de busca. Ele inicializa um nó de busca chamado start com o estado inicial do problema e o insere na frontier. Em seguida, ele entra em um loop até que um estado objetivo seja encontrado ou a frontier esteja vazia.

Em cada iteração do loop, ele extrai um nó da frontier, verifica se é um estado objetivo e, se não for, gera os sucessores desse nó e os insere na frontier. Além disso, ele chama o método showPartialSolution para exibir a melhor solução encontrada até o momento

04

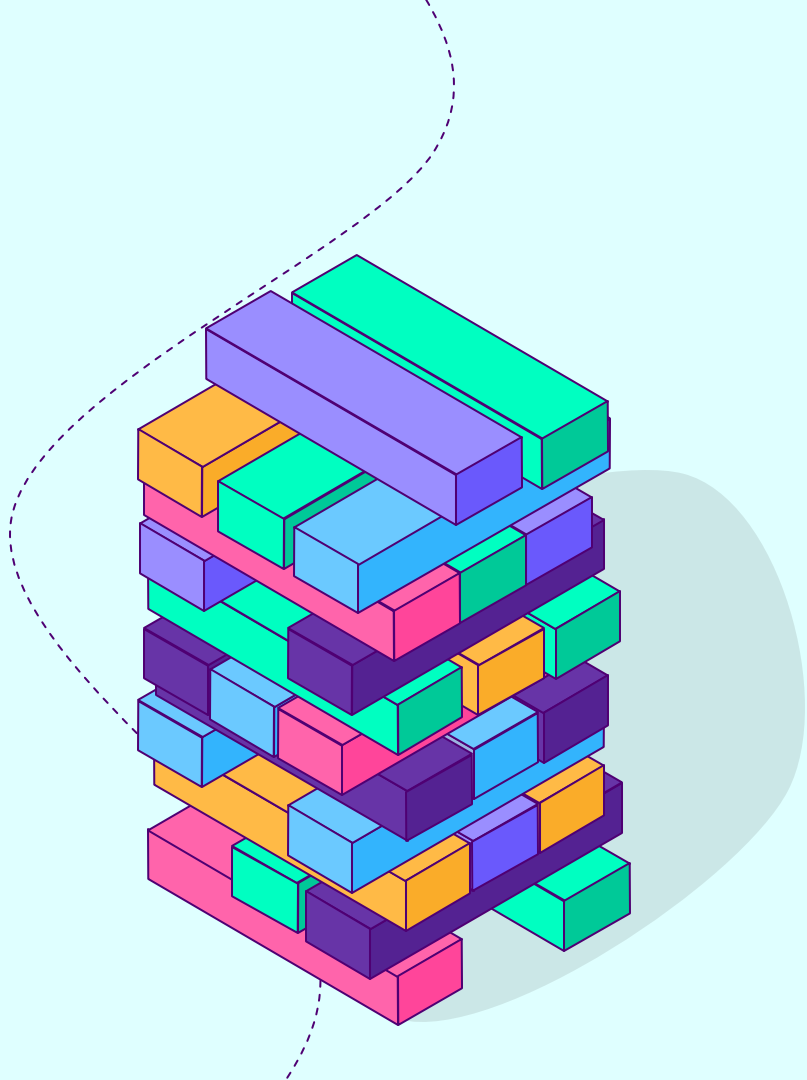
Conclusão



A person wearing a red and white plaid shirt is shown from the chest down, holding a 3x3 Rubik's cube with both hands. The cube is partially solved, showing various colors like green, orange, blue, and yellow. To the right of the person, on a wooden table, is a pyramid structure made of colorful wooden blocks (red, blue, yellow, white). The blocks are arranged in a triangular shape, with the base being the widest and the top being a single block. The background is a plain, light-colored wall.

“ As colors shuffle possibilities,
and algorithms rule chaos; the art
of solving a magic cube builds a
magnetic fascination that spans
generations.”

John Eggâniis



Obrigado!

Você tem alguma
pergunta?