Inteligência Artificial:
Resolução dos Problemas

Artur Bertoni e Maria Eduarda Fischer

TABLE OF CONTENTS

01

SEND MORE MONEY

Resolução do problema 01

SEND MORE MONEY

A lógica do problema "SEND + MORE = MONEY" é um quebra-cabeça clássico de criptaritmos (também conhecido como "adivinhas alfabéticas") em que cada letra representa um dígito diferente (0-9). O objetivo é encontrar uma atribuição única de dígitos a cada letra, de modo que a soma das palavras SEND e MORE seja igual a MONEY.

O problema pode ser resolvido usando técnicas de tentativa e erro, através de força bruta ou, de forma mais sistemática, aplicando dedução lógica.

SEND MORE MONEY

```java
public static void main(String[] args) {
    int count = 0;

    for (int M = 1; M < 10; M++)
        for (int S = 0; S < 10; S++)
            for (int E = 0; E < 10; E++)
                for (int N = 0; N < 10; N++)
                    for (int D = 0; D < 10; D++)
                        for (int O = 0; O < 10; O++)
                            for (int R = 0; R < 10; R++)
                                for (int Y = 0; Y < 10; Y++) {
                                    String SEND = String.valueOf(S) + E + N + D;
                                    String MORE = String.valueOf(M) + O + R + E;
                                    String MONEY = String.valueOf(M) + O + N + E + Y;

                                    count++;

                                    if ((S ≠ E && S ≠ N && S ≠ D && S ≠ M && S ≠ O && S ≠ R && S ≠ Y)
                                            && (E ≠ N && E ≠ D && E ≠ M && E ≠ O && E ≠ R && E ≠ Y)
                                            && (N ≠ D && N ≠ M && N ≠ O && N ≠ R && N ≠ Y)
                                            && (D ≠ M && D ≠ O && D ≠ R && D ≠ Y)
                                            && (M ≠ O && M ≠ R && M ≠ Y)
                                            && (O ≠ R && O ≠ Y)
                                            && (Integer.parseInt(SEND) + Integer.parseInt(MORE) == Integer.parseInt(MONEY))) {

                                        System.out.println("\n SEND     " + SEND +
                                                "\n MORE     " + MORE +
                                                "\n-----    ----- " +
                                                "\nMONEY    " + MONEY + "\n\n" +
                                                "Número de Iterações: " + count + "\n");
                                    }
}
```

O programa consiste em um conjunto de loops for aninhados nos quais são atribuídos valores a cada letra da palavra. Em seguida, verifica-se se cada letra tem valores distintos entre si e se a soma das palavras coincide. Caso contrário, passa-se para a próxima iteração, alterando o valor de uma das letras, até que todas as possibilidades para aquela letra se esgotem. Em seguida, sobe-se para o próximo loop aninhado e repete-se todo o processo até encontrar um resultado válido.

02

COLORAÇÃO DE GRAFOS

Resolução do problema 02

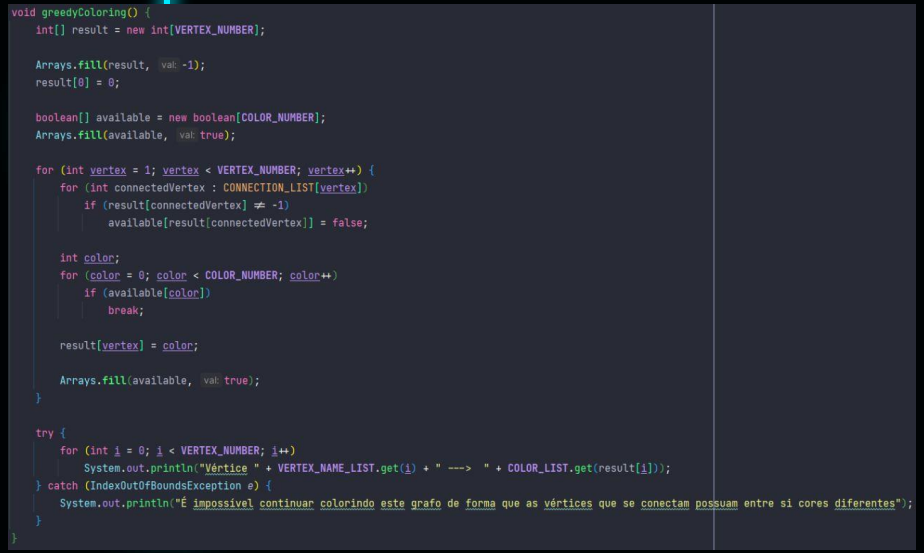
COLORAÇÃO DE GRAFOS

A coloração de grafos é um campo de estudo da teoria dos grafos, que tem suas raízes no século XIX. O problema de coloração de grafos foi formalizado pela primeira vez pelo matemático britânico Arthur Cayley, em 1878. No entanto, a ideia de atribuir cores a objetos e a busca por padrões de coloração remontam a tempos antigos.

O estudo da coloração de grafos ganhou destaque com o trabalho do matemático William Thomas Tutte, na década de 1940. Tutte investigou a coloração de mapas e provou teoremas fundamentais sobre coloração de grafos planares.

Desde então, a coloração de grafos tornou-se um tópico importante na teoria dos grafos e em áreas relacionadas, como otimização combinatória, algoritmos e ciência da computação. O problema de determinar o número mínimo de cores necessárias para colorir um grafo, conhecido como número cromático, é um dos principais problemas estudados nessa área.

COLORAÇÃO DE GRAFOS



```java
void greedyColoring() {
    int[] result = new int[VERTEX_NUMBER];

    Arrays.fill(result, val: -1);
    result[0] = 0;

    boolean[] available = new boolean[COLOR_NUMBER];
    Arrays.fill(available, val: true);

    for (int vertex = 1; vertex < VERTEX_NUMBER; vertex++) {
        for (int connectedVertex : CONNECTION_LIST[vertex])
            if (result[connectedVertex] != -1)
                available[result[connectedVertex]] = false;

        int color;
        for (color = 0; color < COLOR_NUMBER; color++)
            if (available[color])
                break;

        result[vertex] = color;

        Arrays.fill(available, val: true);
    }

    try {
        for (int i = 0; i < VERTEX_NUMBER; i++)
            System.out.println("Vértice " + VERTEX_NAME_LIST.get(i) + " --->  " + COLOR_LIST.get(result[i]));
    } catch (IndexOutOfBoundsException e) {
        System.out.println("É impossível continuar colorindo este grafo de forma que as vértices que se conectam possuam entre si cores diferentes");
    }
}
```

Na parte principal do programa, ocorre a resolução do grafo. Primeiramente, todos os vértices são inicializados como não atribuídos. Em seguida, é atribuída uma cor ao primeiro vértice, e uma lista é criada para armazenar as cores disponíveis para preencher esse vértice. Em um loop, os vértices restantes são coloridos um por um, verificando as cores ao seu redor. As cores ao redor são marcadas como indisponíveis, e, se possível, é atribuída a primeira cor disponível encontrada.

03

SUDOKU

Resolução do problema 03

SUDOKU

A origem do Sudoku remonta a um antigo quebra-cabeça chamado "Quadrado Mágico". O Quadrado Mágico é uma matriz de números onde a soma de cada linha, coluna e diagonal é a mesma.

O desenvolvimento do Sudoku moderno, como o conhecemos hoje, começou no final do século XVIII, na França, com um matemático chamado Leonhard Euler. Euler desenvolveu um jogo chamado "Carré Latin", que era uma variação do Quadrado Mágico. O Carré Latin era composto por uma grade de 9x9, dividida em nove blocos menores de 3x3, e o objetivo era preencher os números de 1 a 9 de forma que não houvesse repetições em nenhuma linha, coluna ou bloco.

SUDOKU

```java
public void play() {
    generateBoard();
    boolean GotHint;

    while (!isGameComplete()) {
        printBoard();

        GotHint = false;

        System.out.print("Digite a linha (0-8), -1 para sair ou -2 para obter uma dica: ");
        int row = SCANNER.nextInt();
        if (row == -1) {
            System.out.println("Jogo encerrado.");
            return;
        } else if (row == -2) {
            if (!getHint()) {
                System.out.println("Não há mais dicas disponíveis.");
            }
            GotHint = true;
        }

        if (!GotHint) {
            System.out.print("Digite a coluna (0-8) ou -1 para sair: ");
            int col = SCANNER.nextInt();
            if (col == -1) {
                System.out.println("Jogo encerrado.");
                return;
            }

            System.out.print("Digite o número (1-9): ");
            int num = SCANNER.nextInt();
            if (!isValidMove(row, col, num)) {
                System.out.println("Movimento inválido. Tente novamente.");
            }
        }
    }

    System.out.println("Parabéns! Você completou o SudokuGame!");
}
```

```java
private boolean isValidMove(int row, int col, int num) {
    if (row < 0 || row ≥ SIZE || col < 0 || col ≥ SIZE) {
        return false;
    }

    if (BOARD[row][col] ≠ EMPTY_CELL) {
        return false;
    }

    if (!isValidPlacement(row, col, num)) {
        return false;
    }

    BOARD[row][col] = num;
    return true;
}
```

A função play() desempenha o papel de executar o jogo, solicitando ao usuário a linha, coluna e número desejados para a jogada. O usuário também tem a opção de digitar 0 para obter uma dica, na qual o programa preenche automaticamente a primeira célula vazia e válida encontrada. Para fornecer a dica, a função getHint() implementa a lógica necessária. O jogo continua até que todas as células sejam corretamente preenchidas ou até que o usuário decida sair, digitando -1.

A função isValidMove() é responsável por verificar se um movimento é válido. Ela avalia se a posição está dentro dos limites do tabuleiro, se a célula está vazia e se o número escolhido atende às regras do Sudoku.

Segue abaixo o link do repositório com a resolução dos problemas:

https://github.com/Artur-Bertoni/Trabalho-Final-IA

Obrigada pela atenção.

Alguma pergunta?