Relatório do Trabalho de Estruturas de Dados Básicas

Técnicas de Programação Avançada — Ifes — Campus Serra

Alunos: David Vilaça, Harã Heique, Larissa Motta.
Prof. Jefferson O. Andrade

2019/2

Sumário

Lista de Códigos Fonte

Lista de Figuras

1 Introdução

Estrutura de Dados é uma implementação concreta de um tipo abstrato de dado ou um tipo de dado básico ou primitivo. Neste trabalho é proposto a resolução de um conjunto de problemas envolvendo as estruturas de dados básicas vistas na disciplina, onde todos os problemas propostos estão disponíveis no site UVa Online Judge.
OS problemas foram resolvidos usando a linguagem de programação *Java* (versão JDK 11.0.3+) nos Ambientes Integrados de Desenvolvimento (IDE) *Eclipse* e o *Apache Netbeans*. As resoluções estão dispostos na seção 2 deste documento.

2 Problemas

2.1 Warm up

2.1.1 UVa 12247 – Jollo

De acordo com o Código Fonte 1 inicialmente é feito a leitura dos dados de entrada, verificando se a linha é contida toda de 0, caso seja é terminado a execução do programa e caso contrário faz-se a distribuição dos dados de acordo com o problema proposto, sendo as três primeiras cartas da princesa e as 2 últimas do príncipe, como é mostrado entre as linhas 24 e 27.
Para se descobrir se é possível o príncipe ganhar e qual seria o menor valor da carta é feito um loop presente a partir da linha 30 do método *main*, o qual é verificado todas as possibilidades de valores até achar-se o resultado, onde caso não encontrado retorna-se o valor -1, que significa que não tem possibilidade de ganhar. Baseado nisso, o método *worst* do Código Fonte 2 é o que desempenha o papel de checar a pior possibilidade de jogada de maneira recursiva.

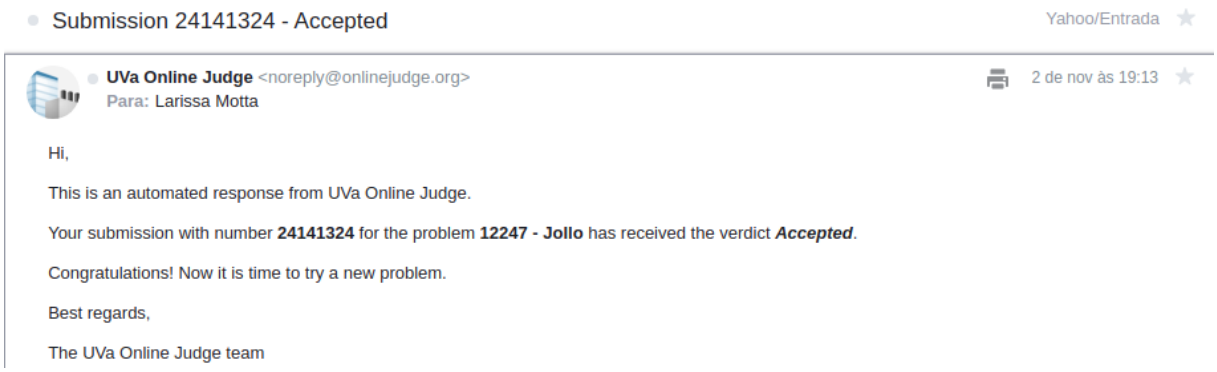

Figura 1: Email de acetação da solucao *12247-Jollo*.

```java
package uva12247;

import java.util.Scanner;
import java.util.Arrays;
import java.util.List;

public class Main {

    private static Scanner scanner;
    private static int[] prince, princess;
    static boolean[] princePlayed, princessPlayed;

    public static void main(String[] args) {
        scanner = new Scanner(System.in);
        String buf = scanner.nextLine();
        int[] line = getLine(buf);
        while (!isLineZeros(line)) {
            int answer = -1;
            princess = new int[3];
            prince = new int[3];
            princePlayed = new boolean[3];
            princessPlayed = new boolean[3];

            princess[0] = line[0];
            princess[1] = line[1];
            princess[2] = line[2];
            prince[0] = line[3];
            prince[1] = line[4];

            for (int i = 1; i < 53 && answer == -1; i++) {
                boolean isValid = true;
                prince[2] = -1;
                for (int j = 0; j < 3; j++) {
                    isValid = isValid && princess[j] != i && prince[j] != i;
                }
                if (isValid) {
                    prince[2] = i;
                    answer = worst(0, 0, -1) ? i : -1;
                }
            }

            System.out.println(answer);

            buf = scanner.nextLine();
            line = getLine(buf);
        }

    }
}
```

Código Fonte 1: Solução para o problema *UVa 12247 – Jollo* (pt 1).

```
    // Recursive get the worst play
    static boolean worst(int player, int lost, int prev) {
            if (lost > 1) {
            return false;
        }
        boolean ret = true;
        for (int i = 0; i < 3; i++) {
            if (player == 0 && !princePlayed[i]) { // prince player
                princePlayed[i] = true;
                ret = ret && worst(1, lost, prince[i]);
                princePlayed[i] = false;
            }
            if (player == 1 && !princessPlayed[i]) { // princess player
                princessPlayed[i] = true;
                int newLost = prev < princess[i] ? lost + 1 : lost;
                ret = ret && worst(0, newLost, -1);
                princessPlayed[i] = false;
            }
        }
        return ret;
    }

    // get all line mapped to int from string input
    public static int[] getLine(String line) {
        List<String> res = Arrays.asList(line.split(" "));
        int[] l = new int[5];
        for (int i = 0; i < 5; i++) {
            l[i] = Integer.parseInt(res.get(i));
        }
        return l;
    }

    // verify if all line elements is zero
    public static boolean isLineZeros(int[] line) {
        for (Integer element : line) {
            if (element != 0) {
                return false;
            }
        }
        return true;
    }
}
```

Código Fonte 2: Solução para o problema *UVa 12247 – Jollo* (pt 2).

2.2 Vetores

2.2.1 UVa 10038 – Jolly Jumpers

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 3: Solução para o problema *UVa 10038 – Jolly Jumpers*.

2.2.2 UVa 11340 – Newspaper

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 4: Solução para o problema *UVa 11340 – Newspaper*.

2.3 Matrizes (Arrays 2D)

2.3.1 UVa 10920 – Spiral Tap

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 5: Solução para o problema *UVa 10920 – Spiral Tap*.

2.3.2 UVa 11581 – Grid successors

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 6: Solução para o problema *UVa 11581 – Grid successors*.

2.4 Ordenação

2.4.1 UVa 10107 – What is the Median?

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 7: Solução para o problema *UVa 10107 – What is the Median?*.

2.4.2 UVa 10258 – Contest Scoreboard

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 8: Solução para o problema *UVa 10258 – Contest Scoreboard*.

2.5 Manipulação de bits

2.5.1 UVa 10264 – The Most Potent Corner

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 9: Solução para o problema *UVa 10264 – The Most Potent Corner*.

2.5.2 UVa 11926 – Multitasking

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 10: Solução para o problema *UVa 11926 – Multitasking*.

2.6 Lista Encadeada

2.6.1 UVa 11988 – Broken Keyboard

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 11: Solução para o problema *UVa 11988 – Broken Keyboard*.

2.7 Pilhas

2.7.1 UVa 00514 – Rails

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 12: Solução para o problema *UVa 00514 – Rails*.

2.7.2 UVa 01062 – Containers

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 13: Solução para o problema *UVa 01062 – Containers*.

2.8 Filas

2.8.1 UVa 10172 – The Lonesome Cargo

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 14: Solução para o problema *UVa 10172 – The Lonesome Cargo*.

2.8.2 UVa 10901 – Ferry Loading III

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 15: Solução para o problema *UVa 10901 – Ferry Loading III*.

2.9 Árvore Binária de Pesquisa (balanceada)

2.9.1 UVa 00939 – Genes

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 16: Solução para o problema *UVa 00939 – Genes*.

2.9.2 UVa 10132 – File Fragmentation

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 17: Solução para o problema *UVa 10132 – File Fragmentation*.

2.10 Conjuntos

2.10.1 UVa 00978 – Lemmings Battle

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 18: Solução para o problema *UVa 00978 – Lemmings Battle*.

2.10.2 UVa 11849 – CD

Introdução breve sobre como o programa funciona

```
1
```

Código Fonte 19: Solução para o problema *UVa 11849 – CD*.