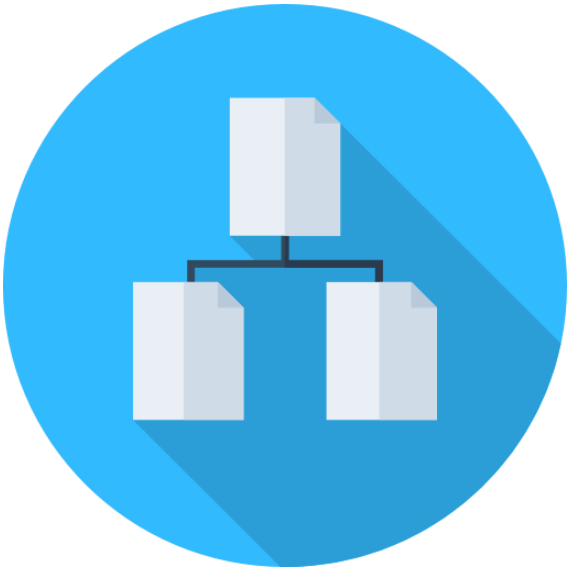


PROJETO ESTRUTURA E ALGORITMIA

Grafo

Rafael Rio, Ricardo Fortuna

Nº22100521, Nº22107919

ÍNDICE

INTRODUÇÃO

Este trabalho prático tem como objetivo desenvolver um programa capaz de detetar cópias de código fonte num conjunto de submissões de estudantes. Para isso, será utilizado um conjunto de dados contendo informações sobre essas submissões. A análise será baseada na correlação entre os códigos fonte, utilizando uma medida adaptada da distância de Levenshtein.

A abordagem proposta consiste em representar as submissões e as suas relações temporais através de um grafo orientado. Cada submissão é representada por um nó no grafo, e as arestas são adicionadas de acordo com a ordem cronológica das submissões. Desta forma, é possível visualizar as relações entre as submissões e identificar possíveis grupos de estudantes que tenham realizado cópias.

Ao analisar o grafo, o programa procura identificar grupos com uma elevada probabilidade de terem copiado, considerando apenas as submissões com uma correlação superior a X%. Estes grupos, representados por componentes fortemente ligados no grafo, podem indicar a ocorrência de plágio ou de trabalho em conjunto de forma desonesta.

.

DESCRIÇÃO

FUNCIONALIDADES DO PROGRAMA

Em termos de funcionalidades o programa contém:

- Leitura de ficheiros CSV (input);
- Processamento e criação das estruturas de dados (process)
- Apresentação no ecrã (output)

ESTRUTURA DO PROGRAMA

O programa não contém uma interface gráfica, porém tem como base estrutural 4package:

O Primeiro package é composto pelo grupo de classes que compõe um grafo:

- Node<t> - Esta Classe é do tipo genérica, mas para este trabalho vai ser definida para receber variáveis do tipo String.
- Edge- é composto por 2 Nodes <String> e um value que guarda o peso da ligação dos 2 Nodes.
- Grafo- é um ArrayList que recebe todas as Edges que o grafo é composto.

O Segundo package é composto pelo corpo do programa:

- copyDetection- é a classe que simula o Main onde recebe os args e passa para a Classe Input
- Input- é a classe responsável pela organização e preparação dos dados, nesta lemos os Csv's, construímos a matriz de adjacência, a header da matriz, tem que ser em tipos de variáveis separadas pois o header é do tipo String enquanto a matriz é do tipo Integer. Esta classe no fim passa os dados para a class Process
- Process – Classe responsável pela criação do grafo e processar os dados relacionados com a matriz de adjacência passando assim para a classe OutPut.
- OutPut – Classe responsável pela conversão de dados de processamento do IDE para a consola para o utilizador visualizar.
- CsvReader- Classe que é responsável por ler o csv e retornar para o Input em formato ArrayList.

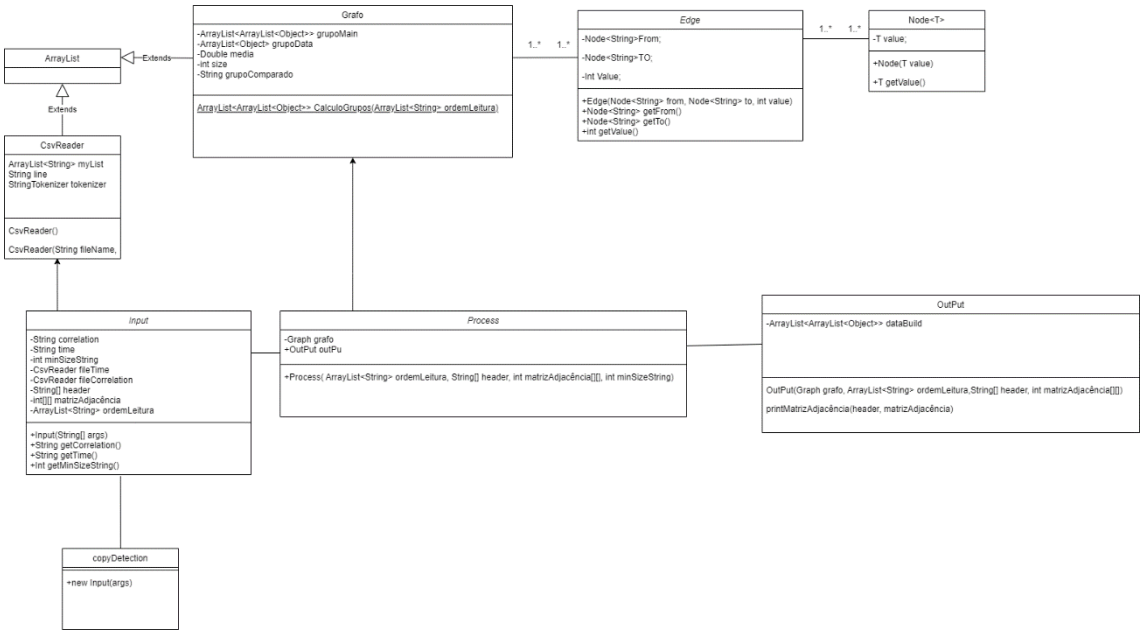
Os outros 2 package são para testes unitários para testar os packages anteriores.

IMPLEMENTAÇÃO DO PROGRAMA

Iremos descrever a implementação do programa. Usando alguns trechos do código, assim como, também o diagrama de classes (UML).

DIAGRAMA DE CLASSES (UML)

Para entendermos melhor o problema, usamos o seguinte diagrama de classes:



CLASSES

CLASSE COPYDETECTION

```
package Body;
/**
* class copyDetection
* @version 1
* @author Rafael Rio
* @author Ricardo Fortuna
*/
public class copyDetection {

      /**
     * O método main é o ponto de entrada do programa.
     * @param args Os argumentos de linha de comando fornecidos ao
programa.
     */
      public static void main(String[] args) {new Input(args);}
}
```

A Classe copyDetection simula o Main onde apenas recebe os argumentos fornecidos pela linha de comandos entregues pelo utilizador e repassa para a classe InPut, como tal é uma classe que não é testada por regra, não existe processamento nem modificação de dados.

CLASSE INPUT

A classe Input é responsável por processar os argumentos de entrada e carregar os dados necessários para o processamento do programa.

Esta classe tem atributos para armazenar os argumentos de entrada, tais como correlation que corresponde ao caminho do ficheiro de correlacionamento, time que corresponde a ordem de entrega do trabalho e minSizeString que vai assumir o valor do threshold que é o valor mínimo a considerar a copia, também possui variáveis para armazenar outros dados necessários para o processamento, como objetos do tipo CsvReader que vai ser a variável que vai guardar o retorno da leitura do ficheiro em causa .A criação de uma matriz de adjacência, uma lista de ordem de leitura que vai absorver o ficheiro time ja tratado, foi escolhido um ArrayList neste caso para poder-se adicionar ou remover quando necessário, mais para a frente vai ser usado uma logica que vai remover os dados da lista consoante que vai sendo usado.

O construtor da classe Input recebe como parâmetro um array de String contendo os argumentos de entrada fornecidos ao programa. Este construtor valida e carrega esses argumentos. Se o número de argumentos for igual a 3, itera sobre os argumentos e verifica cada um deles. Se o argumento for "correlation", atribui o valor da String seguinte ao atributo correlation. Se o argumento for "time", atribui o valor da String seguinte ao atributo time. Se o argumento for "threshold", converte o valor da String seguinte para um número inteiro e atribui ao atributo minSizeString. Caso contrário, lança uma exceção informando que ocorreu um erro nos argumentos de entrada.

Se o número de argumentos não for igual a 3, pressupõe-se que o programa está a ser executado num contexto de teste para o programador. Neste caso, são atribuídos valores predefinidos aos atributos correlation, time e minSizeString. Posteriormente, são criados objetos CsvReader para a leitura dos ficheiros de correlação e tempo, com base nos caminhos dos ficheiros especificados nos atributos. Também são criadas matrizes e listas para armazenar os dados lidos dos ficheiros. Por fim, é criado um objeto de processo, ao qual são passados os dados carregados como argumentos.

```java
package Body;

import java.util.*;

/**
 * A classe Input é responsável por processar os argumentos de entrada e carregar os dados necessários para o processamento.
 * @version 1
 * @author Rafael Rio
 * @author Ricardo Fortuna
 */
public class Input {

    // argumentos do Args do "main"
    private String correlation = null;
    private String time = null;
    private int minSizeString = 0;

    // Variaveis necessarias para o processamento
    private CsvReader fileTime;
    private CsvReader fileCorrelation;
    private String[] header;
    private int[][] matrizAdjacência;
    ArrayList<String> ordemLeitura = new ArrayList<String>();
    Process process;

    /**
     * Construtor da classe Input que valida e carrega os argumentos de entrada.
     * Se os argumentos não estiverem na ordem esperada, ele corrige e atribui aos lugares certos.
     *
     * @param args Os argumentos de entrada fornecidos ao programa.
     */
    public Input(String[] args) {
        if (args.length == 3) {
            for (int i = 0; i < args.length; i++) {
```

```java
    /**
     * Construtor da classe Input que valida e carrega os argumentos de entrada.
     * Se os argumentos não estiverem na ordem esperada, ele corrige e atribui aos lugares certos.
     *
     * @param args Os argumentos de entrada fornecidos ao programa.
     */
    public Input(String[] args) {
        if (args.length == 3) {
            for (int i = 0; i < args.length; i++) {

                String stringDivide[] = args[i].split("=");

                if (stringDivide[0].equals("correlation")) {
                    this.correlation = stringDivide[1];
                } else if (stringDivide[0].equals("time")) {
                    this.time = stringDivide[1];
                } else if (stringDivide[0].equals("threshold")) {
                    this.minSizeString = Integer.parseInt(stringDivide[1]);
                } else
                    throw new IllegalArgumentException("Erro nos Argumentos de entrada");
            }
        }


        else { // Testes para o programador
            if (this.time == null && args.length != 3) {
                this.correlation = ".\\correl.CSV";
                this.time = ".\\time.CSV";
                this.minSizeString = Integer.parseInt("80");
            }

            // Construcao dos objectos
            fileCorrelation = new CsvReader(this.correlation, ",");
            fileTime = new CsvReader(this.time, ",");
```

```
61              header = new String[fileCorrelation.get(0).size()];
62              matrizAdjacência = new int[fileCorrelation.get(0).size() - 1][fileCorrelation.get(0).size() - 1];
63
64              // construçao dos nós
65              for (int i = 0; i < fileCorrelation.get(0).size(); i++) {
66                  this.header[i] = new String(fileCorrelation.get(0).get(i));
67              }
68
69              // contrução do corpo da matrizAdjacência
70              for (int i = 1; i < fileCorrelation.size(); i++) {
71                  for (int x = 1; x < fileCorrelation.size(); x++) {
72                      if (Integer.parseInt(fileCorrelation.get(i).get(x)) < minSizeString) {
73                          matrizAdjacência[i - 1][x - 1] = -1;
74                      } else {
75                          matrizAdjacência[i - 1][x - 1] = Integer.parseInt(fileCorrelation.get(i).get(x));
76                      }
77                  }
78              }
79
80              // contrução da ordem de entrega
81              for (int i = 1; i < fileTime.size(); i++) {
82                  String linha = fileTime.get(i).toString();
83                  String stringDivide[] = linha.split(",");
84                  ordemLeitura.add(stringDivide[0].substring(1));
85              }
86
87              process = new Process(ordemLeitura, header, matrizAdjacência, minSizeString);
88          }
89      }
90
91      /**
92       * Retorna o valor do atributo correlation.
93       *
94       * @return O valor do atributo correlation.
95       */
96      public String getCorrelation() {
97          return correlation;
98      }
99
100     /**
101      * Retorna o valor do atributo time.
102      *
103      * @return O valor do atributo time.
104      */
105     public String getTime() {
106         return time;
107     }
108
109     /** Retorna o valor do atributo minSizeString.
110      *
111      * @return O valor do atributo minSizeString.
112      */
113     public int getMinSizeString() {
114         return minSizeString;
115     }
116
117 }
```

CLASSE PROCESS

A classe Process tem como responsabilidade processar os dados e construir o grafo com base nas informações fornecidas. Ela possui um construtor que recebe como parâmetros uma lista ordemLeitura que representa a ordem de leitura, um array header com os cabeçalhos, uma matriz matrizAdjacência que representa a matriz de adjacência e um valor minSizeString o threshold com o valor minimo aceitavel para considerar copia.

Dentro do construtor, é criado um objeto Graph chamado grafo que será utilizado para construir o grafo com base nos dados fornecidos. Em seguida, é realizada a construção do grafo através de um loop encadeado. O primeiro loop percorre os elementos do array header, exceto o primeiro elemento. O segundo loop também percorre os elementos do array header, exceto o primeiro elemento. Para cada par de elementos, é verificado se o valor da matriz de adjacência na posição correspondente é maior que o tamanho mínimo de String. Se for, é criado um objeto Edge que representa uma aresta do grafo, e essa aresta é adicionada ao grafo.

Após a construção do grafo, é chamado outPut passando como parâmetros o grafo, a lista ordemLeitura, o array header e a matriz matrizAdjacência. Esse objeto OutPut será responsável por imprimir os resultados do processamento, incluindo os grupos calculados e a matriz de adjacência.

Dessa forma, a classe Process desempenha um papel fundamental no processamento dos dados e na construção do grafo, preparando-os para serem exibidos através do objeto OutPut.

```java
package Body;
import Graph.*;
import java.util.*;
/**
 * A classe Process é responsável por processar os dados e construir o grafo com base nas informações fornecidas.
 * @version 1
 * @author Rafael Rio
 * @author Ricardo Fortuna
 */
public class Process {
    private Graph grafo= new Graph();
    OutPut outPut;

     /**
     * Construtor da classe Process que recebe os dados necessários para a construção do grafo.
     *
     * @param ordemLeitura      A lista de ordem de leitura.
     * @param header            O array de cabeçalho.
     * @param matrizAdjacência  A matriz de adjacência.
     * @param minSizeString     O tamanho mínimo de string.
     */
    public Process( ArrayList<String> ordemLeitura, String[] header, int matrizAdjacência[][], int minSizeString) {

        //Construção do grafo
        for(int i=0 ; i < header.length-1  ; i++) {
            for(int x=0 ; x < header.length-1  ; x++) {
                if( matrizAdjacência[i][x]>minSizeString ) {
                    Edge edge=new Edge(new Node<String>(header[i+1]), new Node<String>(header[x+1]) , matrizAdjacência[i][x] );
                grafo.add(edge);
                }
            }
        }
        outPut = new OutPut(grafo, ordemLeitura, header, matrizAdjacência);
    }}
```

CLASSE OUTPUT

A classe OutPut tem como responsabilidade imprimir os resultados do processamento, incluindo os grupos calculados e a matriz de adjacência. Ela possui um construtor que recebe como parâmetros um objeto Graph que representa o grafo a ser processado, uma lista ordemLeitura que contém a ordem de leitura, um array header com os cabeçalhos e uma matriz matrizAdjacência que representa a matriz de adjacência.

Dentro do construtor, é criada uma lista chamada dataBuild do tipo ArrayList<ArrayList<Object>>. Essa lista será preenchida com os grupos calculados pelo grafo, utilizando o método CalculoGrupos do objeto Graph. Em seguida, é realizado um loop para percorrer os elementos da lista dataBuild. Em seguida, é realizado outro loop para percorrer os elementos restantes do grupo. Se o elemento for do tipo Double, é exibido formatado com duas casas decimais utilizando System.out.printf("(%.2f%%)\n", dataBuild.get(i).get(x)). Caso contrário, o elemento é exibido normalmente com System.out.print(dataBuild.get(i).get(x)).

Após a exibição dos grupos, é chamado o método printMatrizAdjacência para imprimir a matriz de adjacência. Esse método recebe como parâmetros o array header contendo os cabeçalhos e a matriz matrizAdjacência. Ele realiza um loop para exibir os cabeçalhos em formato de tabela utilizando System.out.printf("%5s |", header[i]). Em seguida, outro loop é utilizado para percorrer a matriz e exibir os elementos da matriz com System.out.printf("%5d |", matrizAdjacência[i][x]).

O método acima descrito serviu apenas para mostrar a matriz de adjacência para facilitar a compreensão do processo do programa em relação ao grafo. Foi junto ao programa pois foi considerado um elemento importante na construção do programa e como tal vimos que agregou mais valor ao trabalho como ao relatório.

Dessa forma, a classe OutPut é responsável por mostrar os resultados do processamento, exibindo os grupos calculados e a matriz de adjacência.

```java
package Body;
import java.util.ArrayList;
import Graph.Graph;
/**
 * A classe OutPut é responsável por imprimir os resultados do processamento, incluindo os grupos calculados e a matriz de adjacência.
 * @version 1
 * @author Rafael Rio
 * @author Ricardo Fortuna
 */
public class OutPut {
    /**
     * Construtor da classe OutPut que recebe o grafo, a ordem de leitura, o cabeçalho e a matriz de adjacência.
     *
     * @param grafo             O grafo a ser processado.
     * @param ordemLeitura      A lista de ordem de leitura.
     * @param header            O array de cabeçalho.
     * @param matrizAdjacência  A matriz de adjacência.
     */
    public OutPut(Graph grafo, ArrayList<String> ordemLeitura,String[] header, int matrizAdjacência[][]) {

    ArrayList<ArrayList<Object>> dataBuild = new ArrayList<ArrayList<Object>>();

    dataBuild=grafo.CalculoGrupos(ordemLeitura);

    for(int i=0; i< dataBuild.size() ; i++) {
        System.out.println("vertice : " + dataBuild.get(i).get(0));

        for(int x=1 ; x < dataBuild.get(i).size() ; x++){
            if(dataBuild.get(i).get(x)instanceof Double) {
                System.out.printf("(%.2f%%)\n", dataBuild.get(i).get(x));
            System.out.println();
            }else
            System.out.print(dataBuild.get(i).get(x));
        }
```

```
35      }
36      //impressao da matriz
37      printMatrizAdjacência(header, matrizAdjacência);
38      }
39
40      /**
41       * Imprime a matriz de adjacência.
42       *
43       * @param header            O array de cabeçalho.
44       * @param matrizAdjacência  A matriz de adjacência.
45       */
46      public void printMatrizAdjacência(String[] header, int matrizAdjacência[][]) {
47
48          for (int i = 0; i < header.length; i++) {
49              System.out.printf("%5s |", header[i]);
50          }
51
52          for (int i = 0; i < matrizAdjacência.length; i++) {
53              System.out.printf("\n%5s |", header[i + 1]);
54              for (int x = 0; x < matrizAdjacência.length; x++) {
55                  System.out.printf("%5d |", matrizAdjacência[i][x]);
56              }
57          }
58      }
59
60  }
```

CLASSE CSVREADER

Este código define uma classe chamada CsvReader que estende a classe ArrayList< ArrayList<String>> e tem um construtor que recebe o caminho do ficheiro (parâmetro fileName) e o delimitador, isto é, qual o caracter que queremos que ele faça a divisão, no nosso caso escolhemos a "virgula". Ele usa um objeto Scanner para ler o arquivo linha por linha e um objeto StringTokenizer para dividir cada linha em campos separados por vírgulas. A classe divide Linha e coluna separando tudo por palavra em palavra por cada conjunto de índices da estrutura, sendo assim necessário 1 ArrayList para identificar a coluna e outro para identificar a linha.

```java
package Body;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Scanner;
import java.util.StringTokenizer;

public class CsvReader extends ArrayList< ArrayList<String>> {
    /**
     * A classe CsvReader é uma extensão da classe ArrayList e é usada para ler arquivos CSV.
     * @version 1
     * @author Rafael Rio
     * @author Ricardo Fortuna
     */
    private static final long serialVersionUID = 1L;

    /**
     * Construtor padrão da classe CsvReader.
     */
    public CsvReader() {
    }

    /**
     * Construtor da classe CsvReader que recebe o nome do arquivo e o delimitador.
     *
     * @param fileName  O nome do arquivo CSV a ser lido.
     * @param delimiter O delimitador usado para separar os campos no arquivo CSV.
     */
    public CsvReader(String fileName, String delimiter) {
        Scanner scanner = null;
        try {
            scanner = new Scanner(new File(fileName));

              while (scanner.hasNextLine()) {

                  ArrayList<String> myList = new ArrayList<String>();
                  String line = scanner.nextLine();
                  StringTokenizer tokenizer = new StringTokenizer(line, delimiter);

                  while (tokenizer.hasMoreTokens()) {
                      myList.add(tokenizer.nextToken());
                  }

                 add(myList);
              }
              scanner.close();
          } catch (FileNotFoundException e) {
              e.printStackTrace();
          }finally {
              scanner.close();
          }
     }
}
```

CLASSE NODE

A classe Node representa um nó de um grafo. Ela é parametrizada pelo tipo T, que define o tipo de valor armazenado no nó.

A classe possui um atributo value do tipo T, que representa o valor armazenado no nó. O construtor da classe recebe como parâmetro um valor do tipo T e o atribui ao atributo value, a classe possui um método getValue() que retorna o valor armazenado no nó.

Essa classe é utilizada para criar nós no grafo, onde cada nó contém um valor associado. Os nós são utilizados como elementos de ligação entre as arestas do grafo.

```java
package Graph;


/**
 * A classe Node representa um nó em um grafo.
 * @param <T> O tipo de valor armazenado no nó.
 */

public class Node <T>{
    private T value;

    /**
     * Construtor da classe Node.
     * @param value O valor a ser armazenado no nó.
     */
    public Node(T value) {
        this.value = value;
    }

    /**
     * Retorna o valor armazenado no nó.
     * @return O valor armazenado no nó.
     */
    public T getValue(){
        return this.value;
    }
}
```

CLASS EDGE

A classe Edge representa uma aresta em um grafo.

A aresta é definida pela relação entre dois nós, representados pelos atributos “from” e “to”, ambos do tipo Node<String>. O atributo "from" representa o nó de origem da aresta, enquanto o atributo "to" representa o nó de destino da aresta. Além dos nós, a aresta possui um valor associado, representado pelo atributo value, do tipo inteiro. Esse valor representa o peso relevante sobre a aresta. O construtor da classe Edge recebe como parâmetros o nó de origem, o nó de destino e o valor associado à aresta, e os atribui aos respectivos atributos da classe. Possui métodos de acesso para obter o nó de origem, o nó de destino e o valor associado à aresta, através dos métodos getFrom(), getTo() e getValue(), respectivamente.

Essa classe é utilizada para criar as arestas que conectam os nós de um grafo, estabelecendo as relações entre eles. As arestas são elementos importantes para representar a estrutura e as conexões entre os elementos do grafo.

```java
package Graph;

/**
 * A classe Edge representa uma aresta em um grafo.
 */
public class Edge {
        private Node<String> from;
        private Node<String> to;
        private int value;

        /**
         * Construtor da classe Edge.
         *
         * @param from  O nó de origem da aresta.
         * @param to    O nó de destino da aresta.
         * @param value O valor associado à aresta.
         */
        public Edge(Node<String> from, Node<String> to, int value) {
            this.from = from;
            this.to = to;
            this.value = value;
        }

         /**
          * Retorna o nó de origem da aresta.
          *
          * @return O nó de origem da aresta.
          */
         public Node<String> getFrom(){
             return this.from;
         }

         /**
          * Retorna o nó de destino da aresta.
          *
          * @return O nó de destino da aresta.
          */
         public Node<String> getTo(){
             return this.to;
         }

         /**
          * Retorna o valor associado à aresta.
          *
          * @return O valor associado à aresta.
          */
         public int getValue(){
             return this.value;
         }
}
```

A classe Graph representa um grafo, que é uma lista de arestas. Essa classe estende a classe ArrayList<Edge>, ou seja, utiliza uma lista de objetos do tipo Edge para armazenar as arestas do grafo.

O grafo é construído e manipulado através dos métodos fornecidos pela classe Graph. Além dos métodos herdados da classe ArrayList, a classe Graph possui um método específico chamado CalculoGrupos. Este método recebe como parâmetro uma lista de ordem de leitura dos nós.

O objetivo do método CalculoGrupos é calcular os grupos com base na ordem de leitura fornecida. Ele percorre a lista de ordem de leitura e, para cada nó, identifica as arestas que possuem esse nó como nó de origem. Em seguida, ele adiciona os nós de destino dessas arestas ao grupo e calcula a média dos valores associados a essas arestas.

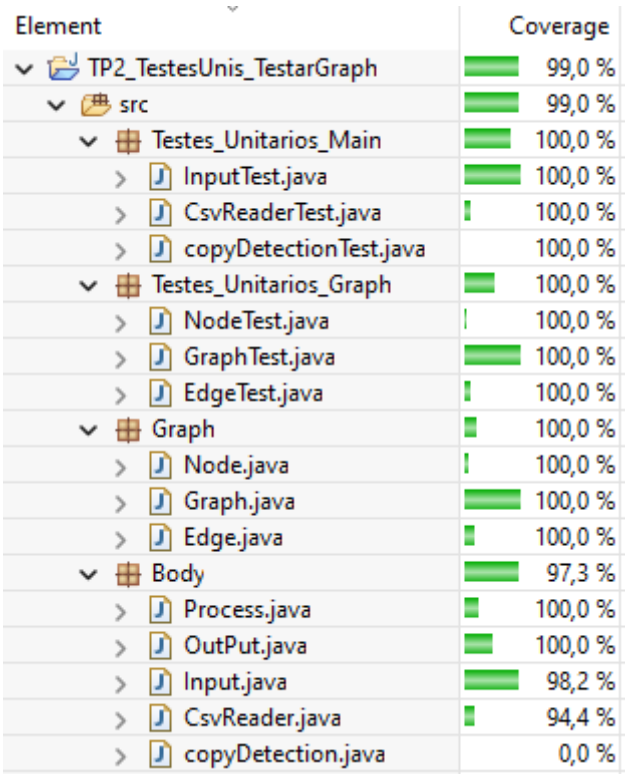
O resultado do cálculo dos grupos é armazenado em uma lista de listas de objetos (ArrayList<ArrayList<Object>>). Cada grupo é representado por uma lista de objetos, onde o primeiro elemento da lista é o nó de referência do grupo, os elementos seguintes são os nós de destino das arestas que partem desse nó, e o último elemento é a média dos valores associados a essas arestas.

Essa classe fornece a estrutura básica para representar e manipular um grafo, permitindo a criação e cálculo de grupos com base nas arestas e na ordem de leitura dos nós.

```java
package Graph;

import java.util.ArrayList;
/**
 * A classe Graph representa um grafo que é uma lista de arestas.
 */
public class Graph extends ArrayList<Edge> {


    ArrayList<ArrayList<Object>> grupoMain = new ArrayList<ArrayList<Object>>();

    /**
     * Serial Version UID para a serialização da classe.
     */
    private static final long serialVersionUID = 1L;

     /**
     * Calcula os grupos com base na ordem de leitura fornecida.
     *
     * @param ordemLeitura A ordem de leitura dos nós.
     * @return Uma lista de grupos, onde cada grupo é representado por uma lista de objetos.
     */
    public ArrayList<ArrayList<Object>> CalculoGrupos(ArrayList<String> ordemLeitura) {

        while (ordemLeitura.size() > 0) {
            ArrayList<Object> grupoData = new ArrayList<>();
            double media = 0;
            int size = 0;

            String grupoComparado = ordemLeitura.get(0).toString();
            grupoData.add(ordemLeitura.get(0).toString());
             for (int x = 0; x < this.size(); x++) {

                if ((grupoComparado.compareTo((this.get(x).getFrom().getValue())) == 0)) {
                    grupoData.add(this.get(x).getTo().getValue() + ", ");

                    media += this.get(x).getValue();
                    size++;

                    ordemLeitura.remove((this.get(x).getTo().getValue()));
                }
            }

            grupoData.add((double)(media / size));
            grupoMain.add(grupoData);
        }

        return grupoMain;
    }
}
```

TESTES UNITÁRIOS

Na criação dos testes unitários conseguimos ter um coverage de 99%, concluímos que é uma cobertura muito a cima do satisfatória, vamos então mostrar na seguinte documentação que houve classes que não foram precisas testar porque não existia processamento de dados, apenas mostrar ao utilizador.



Element	Coverage
TP2_TestesUnis_TestarGraph	99,0 %
src	99,0 %
Testes_Unitarios_Main	100,0 %
InputTest.java	100,0 %
CsvReaderTest.java	100,0 %
copyDetectionTest.java	100,0 %
Testes_Unitarios_Graph	100,0 %
NodeTest.java	100,0 %
GraphTest.java	100,0 %
EdgeTest.java	100,0 %
Graph	100,0 %
Node.java	100,0 %
Graph.java	100,0 %
Edge.java	100,0 %
Body	97,3 %
Process.java	100,0 %
OutPut.java	100,0 %
Input.java	98,2 %
CsvReader.java	94,4 %
copyDetection.java	0,0 %

CLASS NODE

```java
package Testes_Unitarios_Graph;

import Graph.Node;

class NodeTest {

    @Test
    void testNode() {
    Node<String> name = new Node<String> ("Algoritmo");
    assertEquals(name.getValue() ,"Algoritmo");
    }

}
```

CLASS EDGE

```java
package Testes_Unitarios_Graph;

import Graph.*;

class EdgeTest {

    @Test
    void testEdge() {

        Edge graph = new Edge(new Node<String>("algoritmo"),new Node<String>("extrutura"), 20);

        assertEquals(graph.getFrom().getValue() , "algoritmo");
        assertEquals(graph.getTo().getValue() , "extrutura");
        assertEquals(graph.getValue() , 20);

    }

}
```

CLASS GRAPH

```
1  package Testes_Unitarios_Graph;
2
3  import static org.junit.Assert.assertEquals;
12
13 class GraphTest extends ArrayList<Edge> {
14
15     /**
16      *
17      */
18     private static final long serialVersionUID = 1L;
19
20     @Test
21     void CalculoGrupos() {
22
23         Graph grafo = new Graph();
24
25         ArrayList<ArrayList<Object>> grupoMain = new ArrayList<ArrayList<Object>>();
26
27         Edge edge = new Edge(new Node<String>("S1"), new Node<String>("S1"), 100);
28         grafo.add(edge);
29         Edge edge1 = new Edge(new Node<String>("S2"), new Node<String>("S3"), 100);
30         grafo.add(edge1);
31         Edge edge2 = new Edge(new Node<String>("S3"), new Node<String>("S3"), 100);
32         grafo.add(edge2);
33         Edge edge4 = new Edge(new Node<String>("S1"), new Node<String>("S2"), 90);
34         grafo.add(edge4);
35         Edge edge5 = new Edge(new Node<String>("S2"), new Node<String>("S3"), -1);
36         grafo.add(edge5);
37
38         ArrayList<String> ordemLeitura = new ArrayList<String>();
39         ordemLeitura.add("S1");
40         ordemLeitura.add("S2");
41         ordemLeitura.add("S3");
```

```
43         double mediaTeste1 = ((100 + 90) / 2);
44         double mediaTeste2 = 100;
45
46         int contadorTeste=ordemLeitura.size();
47
48         while (ordemLeitura.size() > 0) {
49             ArrayList<Object> grupoData = new ArrayList<>();
50             double media = 0;
51             int size = 0;
52
53             String grupoComparado = ordemLeitura.get(0).toString();
54             grupoData.add(ordemLeitura.get(0).toString());
55
56             for (int x = 0; x < grafo.size(); x++) {
57
58                 if ((grupoComparado.compareTo((grafo.get(x).getFrom().getValue())) == 0)) {
59                     grupoData.add(grafo.get(x).getTo().getValue() + ", ");
60
61                     media += grafo.get(x).getValue();
62                     size++;
63
64                     ordemLeitura.remove((grafo.get(x).getTo().getValue()));
65                     contadorTeste--;
66                 }
67                 assertEquals(ordemLeitura.size(),contadorTeste);
68             }
69
70             grupoData.add((double) (media / size));
71             grupoMain.add(grupoData);
72         }
73         assertEquals(grupoMain.get(0).get(0),"S1");
74         assertEquals(grupoMain.get(0).get(1),"S1"+", ");
75         assertEquals(grupoMain.get(0).get(2),"S2"+", ");
76         assertEquals(grupoMain.get(0).get(3),mediaTeste1);
```

```
78          assertEquals(grupoMain.get(1).get(0),"S3");
79          assertEquals(grupoMain.get(1).get(1),"S3"+", ");
80          assertEquals(grupoMain.get(1).get(2),mediaTeste2);
81     }
82 }
```

CLASS INPUT

```java
package Testes_Unitarios_Main;

import static org.junit.jupiter.api.Assertions.*;

class InputTest {

    private Input input;


    @BeforeEach
    void setUp() {
        // Configurar os argumentos de teste
        String[] args = {"correlation=correlation.csv", "time=time.csv", "threshold=80"};
        input = new Input(args);
    }

    @Test
    void testInpuValidArgs() {
        // Verificar se os argumentos foram carregados corretamente
        assertEquals("correlation.csv", input.getCorrelation());
        assertEquals("time.csv", input.getTime());
        assertEquals(80, input.getMinSizeString());
    }

    @Test
    void testInputForProgrammer() {
        // Configurar argumentos vazios
        String[] args = {};

        // Criar uma instância de Input com argumentos vazios
        input = new Input(args);


        // Verificar se os valores padrão para o programador são aplicados corretamente
        assertEquals(".\\correl.CSV", input.getCorrelation());
        assertEquals(".\\time.CSV", input.getTime());
        assertEquals(80, input.getMinSizeString());
    }


    @Test
    void Input() {

        String[] args = new String[] {"correlation=\\correl.csv","time=\\time.csv",
                "threshold=80" };

        // argumentos do Args do "main"
        String correlation = null;
        String time = null;
        int minSizeString = 0;

        ArrayList<String> suport = new ArrayList<String>();
        ArrayList<String> suport1 = new ArrayList<String>();
        ArrayList<String> suport2 = new ArrayList<String>();
        ArrayList<String> suport3 = new ArrayList<String>();
        ArrayList<String> suportTime = new ArrayList<String>();


        // Variaveis necessarias para o processamento
        CsvReader fileTime;
        CsvReader fileCorrelation=new CsvReader();
        String[] header;
        int[][] matrizAdjacência;

        ArrayList<String> ordemLeitura = new ArrayList<String>();

         // valida e carrega do args do main
         // Se nao for posto pela suposta ordem de Input, ele corrige para os sitios
         // certos
         int contadorTeste=args.length;

        for (int i = 0; i < args.length; i++) {

            String stringDivide[] = args[i].split("=");

            if (stringDivide[0].equals("correlation")) {
                correlation = stringDivide[1];
            } else if (stringDivide[0].equals("time")) {
                time = stringDivide[1];
            } else{
                minSizeString = Integer.parseInt(stringDivide[1]);
            }
            contadorTeste--;
        }
        assertEquals(contadorTeste,0);
            assertEquals(correlation, "\\correl.csv");
            assertEquals(time, "\\time.csv");
            assertEquals(minSizeString, 80);

            // Construcao dos objectos
            suport.add("row");
            suport.add("S1");
            suport.add("S2");
            suport.add("S3");
                fileCorrelation.add(suport);
            suport1.add("S1");
            suport1.add("100");
            suport1.add("90");
            suport1.add("50");
                fileCorrelation.add(suport1);
```

```
            suport2.add("S2");
            suport2.add("90");
            suport2.add("100");
            suport2.add("50");
                fileCorrelation.add(suport2);
            suport3.add("S3");
            suport3.add("50");
            suport3.add("50");
            suport3.add("100");
                fileCorrelation.add(suport3);

            fileTime = new CsvReader();
            suportTime.add("S1");
                    fileTime.add(suportTime);

            suportTime.add("S2");
                    fileTime.add(suportTime);

            suportTime.add("S2");
                    fileTime.add(suportTime);

            header = new String[fileCorrelation.get(0).size()];
            matrizAdjacência = new int[fileCorrelation.get(0).size() - 1][fileCorrelation.get(0).size() - 1];

            // construçao dos nós
            for (int i = 0; i < fileCorrelation.get(0).size(); i++) {
                header[i] = new String(fileCorrelation.get(0).get(i));
            }

            // contrução do corpo da matrizAdjacência
            for (int i = 1; i < fileCorrelation.size(); i++) {
                for (int x = 1; x < fileCorrelation.size(); x++) {
                    if (Integer.parseInt(fileCorrelation.get(i).get(x)) < minSizeString) {
                        matrizAdjacência[i - 1][x - 1] = -1;
                    } else {
                        matrizAdjacência[i - 1][x - 1] = Integer.parseInt(fileCorrelation.get(i).get(x));
                    }
                }
            }

            // contrução da ordem de entrega
            for (int i = 1; i < fileTime.size(); i++) {
                String linha = fileTime.get(i).toString();
                String stringDivide[] = linha.split(",");
                ordemLeitura.add(stringDivide[0].substring(1));
            }

        }
}
```

CLASS PROCESS

CLASS OUTPUTTEST

CLASS CSVREADER

```java
package Testes_Unitarios_Main;

import org.junit.jupiter.api.Assertions;

class CsvReaderTest {

    @Test
    void test() {

            CsvReader csvReader = new CsvReader(
                    "D:\\Work Spasce\\Eclipse\\TP2_TestesUnis_TestarGraph\\src\\Testes_Unitarios_Main\\testFile.CSV",
                    ",");

            Assertions.assertEquals("Key2", csvReader.get(0).get(0));
            Assertions.assertEquals("Value2", csvReader.get(0).get(1));

            Assertions.assertEquals("Key1", csvReader.get(1).get(0));
            Assertions.assertEquals("Value1", csvReader.get(1).get(1));

            Assertions.assertEquals("Key3", csvReader.get(2).get(0));
            Assertions.assertEquals("Value3", csvReader.get(2).get(1));
    }
}

```

CLASS OUTRAS

Existiram classes que não foi necessário testar, como a classe CopyDetection OutPut, Process.

CopyDetection- porque simula o Main e não é testado.

OutPut- apenas serve para mostrar ao utilizador o resultado já calculado e tratado, então não precisa de confirmar aquilo que já foi confirmado noutras classes.

Process- É o movimento de variável para variável, dados que já foram inseridos e não precisam de ser tratados porque são tratados na classe Graph ou na classe Input, então seria repetir o próprio teste.

As afirmações anteriores podemos confirmar no total de covarege atingido onde aparece o quanto do código foi coberto pelos testes, deixando assim apenas as classes Input com 98,2% e CsvReader com 94,4% com a justa causa de não ter sido testado um throw quem em testes unitários sao difíceis de ser testado e “desnecessários” porque já é um teste a logica de ultima situação.

RESULTADOS

A travez do programa criado conseguimos chegar ao resultado igual ao do proposto pelo professor,as imagens seguintes mostramos o resultado da consola, e a matriz de adjacência porque é interessante visualizar os dados processados e com que estrutura o grafo foi criado.

RESULTADO DO PROGRAMA:

```
vertice : S2
S2, (100,00%)

vertice : S6
S131, S6, (97,00%)

vertice : S8
S100, S115, S132, S31, S32, S35, S60, S8, S86, S88, S93, S96, (97,33%)
```

MATRIZ DE ADJACENCIA

row_0	S100	S115	S131	S132	S2	S31	S32	S35	S6	S60	S8	S86	S88	S93	S96
S100	100	98	-1	100	-1	99	99	98	-1	98	100	81	100	99	96
S115	98	100	-1	98	-1	97	97	96	-1	96	98	81	98	97	97
S131	-1	-1	100	-1	-1	-1	-1	-1	94	-1	-1	-1	-1	-1	-1
S132	100	98	-1	100	-1	99	99	98	-1	98	100	81	100	99	96
S2	-1	-1	-1	-1	100	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
S31	99	97	-1	99	-1	100	99	99	-1	99	99	80	99	98	95
S32	99	97	-1	99	-1	99	100	98	-1	98	99	80	99	98	95
S35	98	96	-1	98	-1	99	98	100	-1	99	98	-1	98	97	94
S6	-1	-1	94	-1	-1	-1	-1	-1	100	-1	-1	-1	-1	-1	-1
S60	98	96	-1	98	-1	99	98	99	-1	100	98	-1	98	97	94
S8	100	98	-1	100	-1	99	99	98	-1	98	100	81	100	99	96
S86	81	81	-1	81	-1	80	80	-1	-1	-1	81	100	81	80	81
S88	100	98	-1	100	-1	99	99	98	-1	98	100	81	100	99	96
S93	99	97	-1	99	-1	98	98	97	-1	97	99	82	99	100	94
S96	96	97	-1	96	-1	95	95	94	-1	94	96	81	96	94	100

CONCLUSÃO

Em conclusão, este trabalho prático tem como objetivo desenvolver um sistema de deteção de cópias utilizando grafos como estrutura de dados. O problema abordado envolve a análise de submissões de código-fonte por estudantes em plataformas web, visando identificar correlações e grupos que possam ter realizado cópias.

FONDES DE PESQUISA

https://www.udemy.com/course/testes-unitarios-em-java/

https://www.udemy.com/course/java-curso-completo/