Faculdade de Engenharia da Universidade do Porto

Universidade do Porto

Faculdade de Engenharia

FEUP

Similaridade de ficheiros

Mário Filipe Araújo Ferreira

Pedro José Leal de Sousa

Vítor Filipe Oliveira Teixeira

Mestrado Integrado em Engenharia Informática e Computação

Conceção e Análise de Algoritmos

Professor: Rosaldo Rossetti

28 de Abril de 2014

UNIDADE CURRICULAR:

Conceção e análise de algoritmos

TÍTULO:

Similaridade de ficheiros

DATA:

2 – 6 – 2014

ELEMENTOS DO GRUPO:

Mário Ferreira -> ei12049 – ei12049@fe.up.pt

Pedro Sousa -> ei12179 – ei12179@fe.up.pt

Vítor Teixeira -> ei12060 – ei12060@fe.up.pt

TEMA DO PROBLEMA

O objetivo deste trabalho é construir um programa que proceda à comparação de dois ficheiros de texto, e produza um outro que assinale as alterações efetuadas entre um e outro, usando para tal a definição de "distância de edição", isto é, o esforço em número de operações (adição, eliminação e permutação de caracteres) necessários para tornar o ficheiro a ser comparado no ficheiro de referência.

Essas operações deverão ser assinaladas no output da interface do programa e também guardadas num ficheiro temporário que possibilita ao utilizador guardar esse texto de controlo.

ESTRATÉGIA DE RESOLUÇÃO DO PROBLEMA

Para a resolução do problema proposto foi escolhida a seguinte solução:

- Construir a classe Files que gere o carregamento e gestão de ficheiros de texto.

- Construir a classe FileRevision que implementa os algoritmos necessários para calcular as divergências entre dois textos.

- Esses algoritmos foram o de Levenstein para calcular a distância de edição e um algoritmo dinâmico para calcular a LCS(*longest common subsequence)* entre duas strings.

- Construir a classe Changes que guarda as alterações detetadas pela classe supra referida.

- Demonstrar os elementos da classe acima referida duma forma "user friendly" tanto no standard output como num ficheiro temporário.

FORMALIZAÇÃO DO PROBLEMA

É esperado que o input sejam dois ficheiros de texto, que depois de devidamente carregados e separados de uma forma pré-definida são passados como argumento aos algoritmos de forma a serem tratados e detetadas as transformações entre os dois.

Tais alterações serão depois transmitidas ao utilizador.

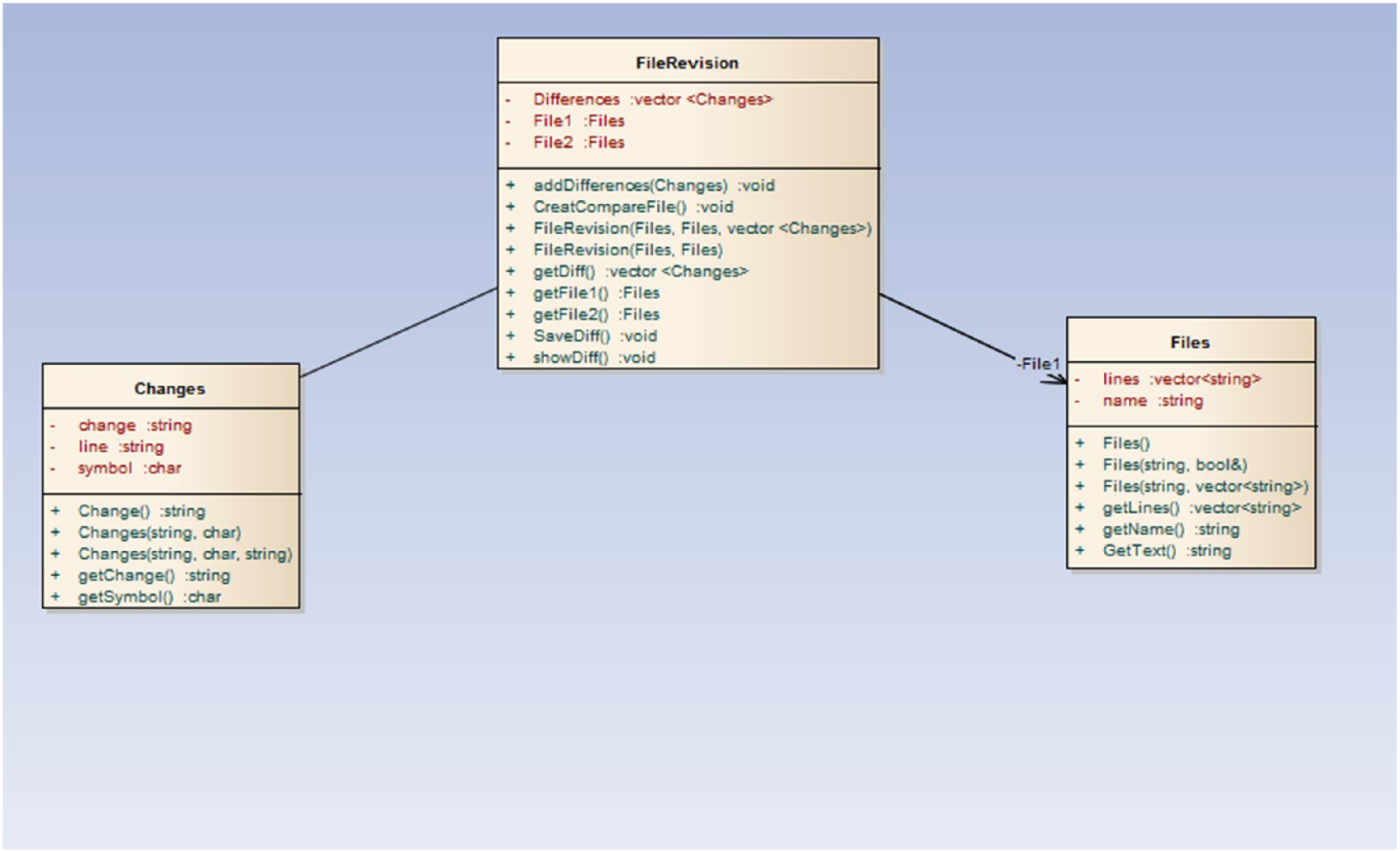
CASOS DE UTILIZAÇÃO

- ➔ Carregamento dos ficheiros: o nome dos ficheiros pretendidos é fornecido ao programa pelo utilizador. Estes são carregados e divididos por linhas e guardados em vetores.

- ➔ Calculo das transformações: através do cálculo da distância de edição entre as várias linhas dos dois ficheiros é detetado se o excerto do texto foi modificado, apagado ou acrescentado. Essas alterações são guardadas em elementos da classe Changes, juntamento com a linha em que ocorreram.

- ➔ Apresentação dos resultados ao utilizador: é criado um texto juntado os excertos que foram alterados, apagados e adicionados e apresentado ao utilizador de forma explícita e clara.

DIAGRAMA DE CLASSES



PRINCIPAL ALGORITMO IMPLEMENTADO

		m	e	i	l	e	n	s	t	e	i	n
	0	1	2	3	4	5	6	7	8	9	10	11
l	1	1	2	3	3	4	5	6	7	8	9	10
e	2	2	1	2	3	3	4	5	6	7	8	9
v	3	3	2	2	3	4	4	5	6	7	8	9
e	4	4	3	3	3	3	4	5	6	6	7	8
n	5	5	4	4	4	4	3	4	5	6	7	7
s	6	6	5	5	5	5	4	3	4	5	6	7
h	7	7	6	6	6	6	5	4	4	5	6	7
t	8	8	7	7	7	7	6	5	4	5	6	7
e	9	9	8	8	8	7	7	6	5	4	5	6
i	10	10	9	8	9	8	8	7	6	5	4	5
n	11	11	10	9	9	9	8	8	7	6	5	4

O principal algoritmo usado foi o do calculo da distancia de edição de levenshtein, que segue o raciocínio expresso no pseudo-código apresentado de seguida:

```
Início

  // tab é uma tabela com lenStr1+1 linhas e lenStr2+1 colunas

  Inteiro:  tab[0..lenStr1, 0..lenStr2]

  // X e Y são usados para iterar str1 e str2

  Inteiro:  X, Y, cost


  Para X de 0 até lenStr1

     tab[X, 0] ← X

  Para Y de 0 até lenStr2

     tab[0, Y] ← Y
```

```
Para X de 1 até lenStr1
    Para Y de 1 até lenStr2
      Se str1[X] = str2[Y] Então cost ← 0
                  Se-Não cost ← 2 // Custo da substituição deve ser 2, "delete" e inserção
      tab[X, Y] := menor(
                  tab[X-1, Y  ] + 1,     // Apagar
                  tab[X  , Y-1] + 1,     // Inserir
                  tab[X-1, Y-1] + cost   // Substituir
                )
  LevenshteinDistance ← tab[lenStr1, lenStr2]
 Fim
```

A complexidade do algoritmo usado no programa é O(n^2), sendo n o numero de linhas do maio ficheiro.

PRINCIPAIS DIFICULDADES

- ➔ Adaptar os algoritmos ao caso do tema proposto
- ➔ Encontrar testes diversificados
- ➔ Detetar e analisar os vários casos possíveis no decorrer do programa
- ➔ Tentar usar o algoritmo de forma mais apropriada para a finalidade de tornar o resultado claro para o utilizador

AUTO-AVALIAÇÃO DO GRUPO

Todos os elementos trabalharam em sintonia visando um objetivo comum ajudando-se mutuamente e tentando dar o seu melhor e colmatar as lacunas alheias. Sendo o grupo constituído por três elementos o trabalho foi dividido em três partes: carregamento dos ficheiros, processamento dos mesmos, output do resultado.

Nenhum elemento se limitou a fazer a sua parte, todos colaboraram em todas as fazes do projeto de forma a melhorar a qualidade do mesmo.