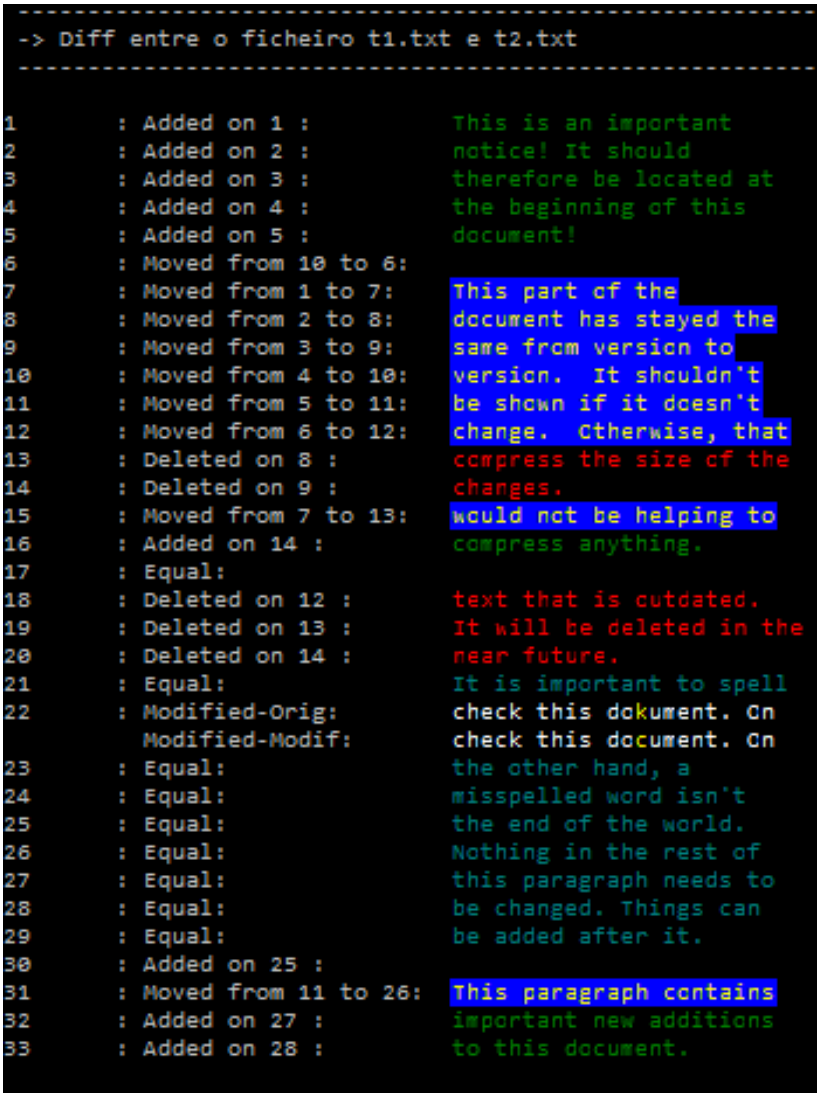
Trabalho de grupo 2 – grupo 8 - tema 1

Comparação de Ficheiros



Trabalho realizado por:

Carlos Miguel Correia da Costa

ei09097 – FEUP

ei09097@fe.up.pt / carlosmccosta@live.com.pt

e entregue a 02/06/2010

Introdução

O presente trabalho tem por objectivo criar um programa capaz de determinar as diferenças entre 2 ficheiros, mostrando na consola (recorrendo a cores), que partes de cada ficheiro foram alteradas e que partes foram movidas e/ou modificadas / eliminadas.

Na implementação da aplicação foi usada programação dinâmica para optimizar o desempenho do programa e os algoritmos principais encapsulados em classes para facilitar a sua reutilização.

Todo o projecto baseia-se no uso de algoritmos de pesquisa aproximada de strings (distância de edição, subsequência de maior comprimento…).

Foi feito uma implementação preliminar de rotinas de patch, ou seja, métodos que permitem a partir do ficheiro original e de um ficheiro com as modificações, obter o ficheiro final (que é isso que faz o SVN, por ex.).

Por falta de tempo não está completamente implementado, mas estava quase.

Só não foi completado porque tive problemas na transformação entre os índices temporários aquando da reconstituição do ficheiro modificado a partir dos índices fixos que foram determinados aquando da análise dos ficheiros.

Descrição da implementação

Para a construções desta aplicação foram implementadas as seguintes classes:

Diff, LCS, EditDistance, LineInfo, ConsoleControl e Exceptions, bem como um namespace com uma biblioteca de funções para a interacção com o utilizador (utils).

De seguida passa-se à descrição das principais estruturas de dados de cada classe, e da funcionalidade de cada uma dessas classes.

1) Diff

Classe que faz toda a gestão do processo de análise dos dois ficheiros.

Incluiu também a interface CLI com o utilizador, apesar do menu está implementado no main.cpp.

2) LCS

Classe que tem os algoritmos de pesquisa pela "longest common sequence", usados para determinar que partes das linhas dos textos são comuns.

3) EditDistance

Classe que contém os algoritmos responsáveis pela determinação do número de operações de edição (eliminação, adição ou substituição) que é necessário fazer para transformar uma string noutra.

Foi usado para determinar o movimento de linhas nos ficheiros, visto que permite determinar se duas strings são semelhantes ou não.

Para ajustar o quanto as strings têm que ser semelhantes para serem consideradas como semelhantes (ou seja, que correspondem a uma mesma linha de texto, que poderá ter sido movida de posição nos ficheiros), é usado o

```
#define PERCENTAGE_TO_CONSIDER_DIFERENTE_LINE 0.66
```

Que define qual o tamanho de edição mínimo que as strings a comparar têm que ter para serem consideras semelhantes. Ou seja, quando mais próximo de 1 mais tolerante ao tamanho de edição o algoritmo é.

4) LineInfo

Classe que contêm a informação relacionada com uma linha de um ficheiro.

É usado para facilitar o emparelhamento de linhas semelhantes aos dois ficheiros e assim evitar o cálculo duplicado da LCS.

5) ConsoleControl

Classe que faz o controlo das cores que aprecem na consola, usando quer a API do Windows quer a API do Linux, usando compilação condicional.

Corresponde a uma abstracção sobre as APIs do Windows e da libraria NCurses do Linux, para permitir a portabilidade e facilidade de manipulação da consola.

6) Exceptions

Classe que contem as excepções que são usadas no programa para tratar de anomalias na execução do código.

É usado sobretudo para tratar de inputs inválidos do utilizador.

Por exemplo quando o utilizador introduz o nome de um ficheiro para ler no programa que não existe no disco, ou quando introduz o nome a dar a um ficheiro onde serão guardados dados do programa, mas cujo nome já está em uso por outro ficheiro naquela pasta…

Notas sobre a complexidade dos algoritmos de pesquisa aproximada em strings (LCS, EditDistance)

O algoritmo de determinação da maior subsequência comum entre duas strings tem complexidade espacial e temporal polinomial, mais precisamente é de O (|P|x|T|), sendo |P| o tamanho de uma das strings (padrão) e |T| será o tamanho da outra string (texto).

Foram implementadas algumas optimizações que faziam reduzir a complexidade temporal e espacial do algoritmo, uma delas sendo a redução do tamanho das strings a analisar tirando as partes comuns no início e fim comuns a ambas as strings.

No entanto esta optimização foi "desactivada" porque não permitia ter a subsequência comum completa, necessária para colorir correctamente as letras alteradas.

Relativamente ao algoritmo de determinação da distância de edição ou distância de Levenshtein, foram feitas duas implementações. Ambas recorrendo a programação dinâmica.

A diferença é que a primeira tem complexidade temporal e espacial de O(|P|x|T|) enquanto que a versão mais optimizada tem uma melhoria na complexidade espacial, ficando então com complexidade espacial de O(|T|).

Notas sobre a implementação

No namespace utils estão as funções genéricas de comunicação da Diff com a interface CLI com o utilizador.

No ficheiro defs.h estão os defines do programa.

No main.cpp é onde está implementado o menu para fazer a interface CLI com a API da classe Diff.

Em Linux é necessário acrescentar a biblioteca ncurses às librarias de compilação do eclipse.

O modelo UML, bem como toda a documentação estão juntos no doxygen gerado.

Descrição da interface com o utilizador

Nota: não são usados acentos porque na consola fica desformatado…

O programa começa por pedir qual os nomes dos ficheiros que serão analisados e onde será guarda a informação relativa à comparação dos ficheiros (a ser usado posteriormente no patch).

Caso o ficheiro de output já existe é perguntado se o utilizador deseja substitui-lo.

```
  -> Introduza o nome do ficheiro original: t1.txt

  -> Introduza o nome do ficheiro modificado: t2.txt

  -> Introduza o nome do ficheiro onde sera guardado o patchFile: p.txt
O nome do ficheiro que introduziu ja existe!

Pretende fazer overwrite ao ficheiro (S/N)? : s
```

De seguida é apresentado o menu do programa

```
########################   CAL - Trabalho 2 - Tema 1   ########################
  >>>                        Comparacao de ficheiros                        <<<
################################################################################


 1 - Comparar ficheiros e criar ficheiro de patch
 2 - Mostar ficheiro original
 3 - Mostar ficheiro modificado
 4 - Carregar ficheiro de patch
 5 - Mostrar ficheiro de patch

 6 - Recarregar ficheiros
 7 - Mudar ficheiros

 0 - Sair



  >>> Opcao:
```

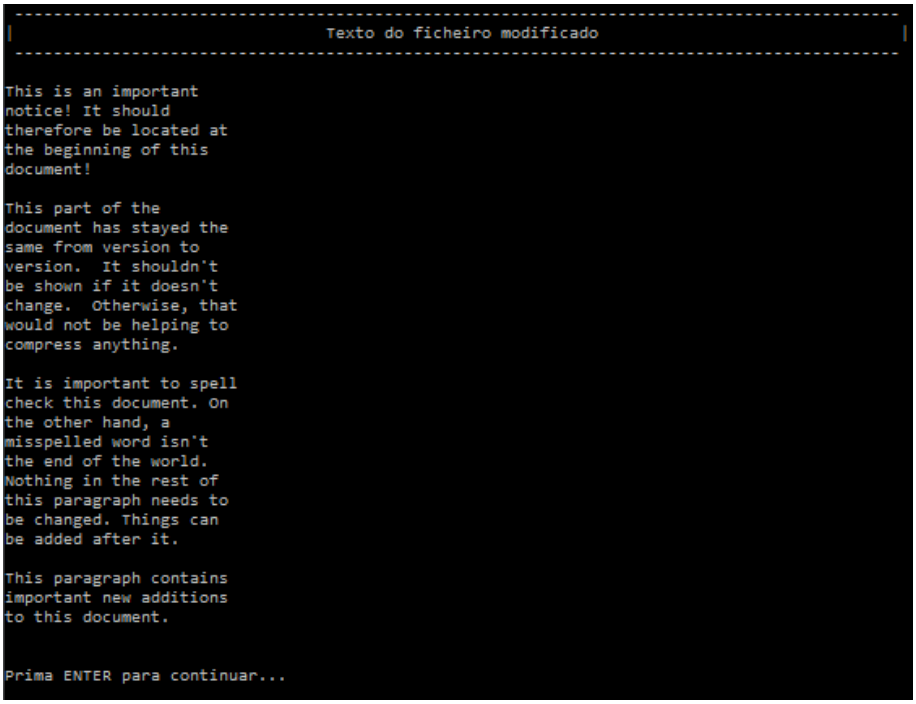
A opção 1 compara os 2 ficheiros mostrando as diferenças entre eles recorrendo a cores (ex imagem na capa do relatório e em baixo).

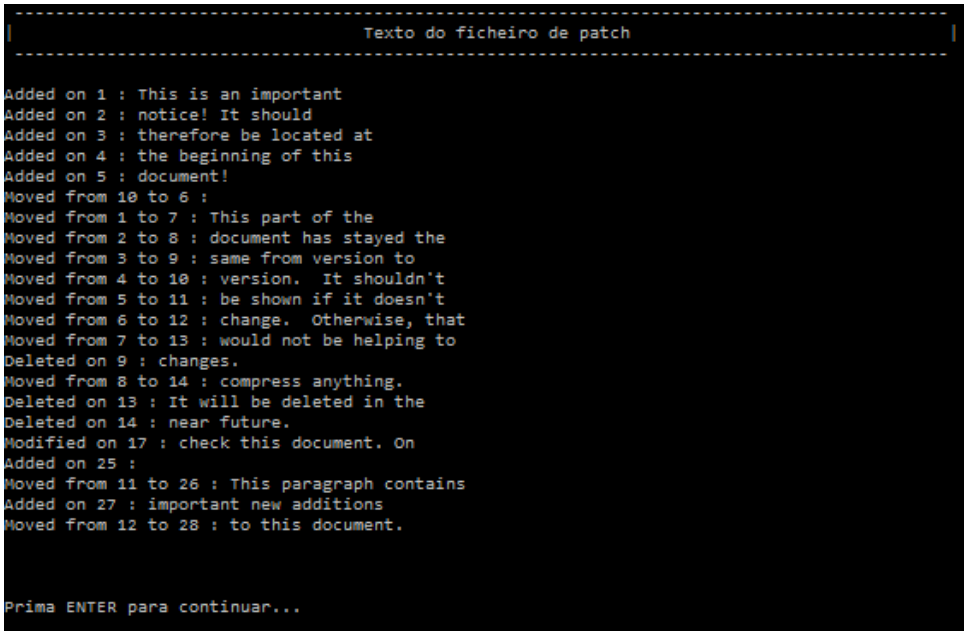
```
-> Diff entre o ficheiro f1.txt e f2.txt

1       : Modified-Orig:        diferent diferent diferent
          Modified-Modif:       iferent diferen different
2       : Modified-Orig:        diferent
          Modified-Modif:       ssdiferente
3       : Equal:
4       : Equal:                same same same same
5       : Equal:                same same same same
6       : Equal:                same same same same
7       : Equal:
8       : Added on 8 :          added added added added
9       : Added on 9 :          added
10      : Added on 10 :         added added added
11      : Moved from 10 to 11:
12      : Moved from 8 to 12:   mcved mcved mcved mcved
13      : Moved from 9 to 13:   mcved mcved mcved mcved
14      : Moved from 13 to 14:
15      : Moved&Modif from 11:  diff diff diff diff
          Moved&Modif to 15:    diffx dxifxf if diff
16      : Moved&Modif from 12:  diffzzz diffzzz diffzzz
          Moved&Modif to 16:    diffzz dffzzz diffFFzz
17      : Moved from 14 to 17:
18      : Moved from 15 to 18:  sss


Prima ENTER para continuar...
```

As opções 2, 3 e 4 mostram respectivamente os conteúdos dos ficheiros original, modificado e de patch que o programa tem em memória.

Exemplos em baixo:

```
|                              Texto do ficheiro original                              |

This part of the
document has stayed the
same from version to
version.  It shouldn't
be shown if it doesn't
change.  Otherwise, that
would not be helping to
compress the size of the
changes.

This paragraph contains
text that is outdated.
It will be deleted in the
near future.

It is important to spell
check this dokument. On
the other hand, a
misspelled word isn't
the end of the world.
Nothing in the rest of
this paragraph needs to
be changed. Things can
be added after it.


Prima ENTER para continuar...
```





A opção 4 permite carregar um ficheiro de patch para possibilitar a aplicação de diferentes patchs e ver as diferenças no merge dos feicheiros (a implementação da obtenção do ficheiro modificado através do ficheiro original e o de patch está quase feita e correspondia à opção que estava em vez da 6, mas como não está 100% funcional, não foi incluída no menu).

A opção 6 serve para recarregar e reanalisar os ficheiros (útil para fazer modificações nos ficheiros que se estava a usar e ver como o programa se comporta).

A opção 7 permite mudar os ficheiros que serão analisados.

Lista de casos de utilização

Os algoritmos implementados neste programa têm muita aplicação nas áreas de controlo de versões (SVN por ex), na análise de sequências de ADN e de proteínas.

São algoritmos relativamente eficientes para os problemas que resolvem e simplificam em muito o trabalho aos programadores e aos bioengenheiros.

Principais dificuldades encontradas

As principais dificuldades encontradas e que levaram à entrega um pouco atrasada do projecto estiveram relacionadas com a implementação da funcionalidade de merge / patch do ficheiro original com um ficheiro com as alterações feitas.

Apesar de não ser explicitamente pedido no enunciado, resolvi proceder à sua implementação porque essa foi a principal razão pela qual escolhi este tema, ou seja, implementar uma ferramenta de Diff de ficheiros para tentar perceber como o SVN funciona e porque é que às vezes faz asneira no merge dos ficheiros submetidos.

As dificuldades focaram-se em como a partir de um ficheiro com as modificações todas analisadas com índices fixos que dão a correspondência das edições entre o ficheiro original e modificado, transformar esses índices de forma a poder reconstituir o ficheiro modificado.

Isto acontece porque no processo de merge o ficheiro original é alterado e como tal os índices também são alterados, tornando inválidas as associações dadas pelo ficheiro de patch.

Foi tentado várias técnicas de actualização dinâmica dos índices aquando do merge, mas até ao momento não está a funcionar a 100%.

Indicação do esforço de cada elemento do grupo

Não se aplica neste trabalho porque decidi fazê-lo sozinho, visto que era um trabalho pequeno e assim pude controlar melhor quando é que estava a trabalhar para este projecto no meu calendário de testes e outros trabalhos para entregar.

Bibliografia

http://en.wikipedia.org/wiki/Diff

http://en.wikipedia.org/wiki/Longest_common_subsequence_problem

http://www.ics.uci.edu/~eppstein/161/960229.html

http://en.wikipedia.org/wiki/Hirschberg%27s_algorithm

http://en.wikipedia.org/wiki/Levenshtein_distance

http://neil.fraser.name/writing/diff/

http://neil.fraser.name/writing/patch/

http://www.ibm.com/developerworks/java/library/j-seqalign/index.html?ca=dgr-jw17dynamicjava&S_TACT=105AGX59&S_CMP=GR

http://www.algorithmist.com/index.php/Longest_Common_Subsequence

Todas as páginas foram revisitadas no dia 2/06/2011 e encontravam-se online.