Processamento de Linguagens (3º ano de Mestrado Integrado em Engenharia Informática)

Trabalho Prático Nº1

Relatório de Desenvolvimento

André da Silva Gonçalves (a80368)

Francisco Reinolds (a82982)

João Queirós (a82422)

31 de Março de 2019

Resumo

A resolução do trabalho prático descrita no presente relatório refere-se ao processamento de notícias de um jornal para gerar ficheiros em *html*. Assim, neste trabalho prático pretende-se aplicar os conceitos adquiridos na Unidade Curricular de Processamento de Linguagens associados à utilização de expressões regulares para descrever padrões de textos, para futuramente gerar ficheiros nos formatos referidos, através da ferramenta Flex.

Conteúdo

Capítulo 1

Introdução

No âmbito da unidade curricular de Processamento de Linguagens, foi-nos apresentado um enunciado que contém oito problemas diferentes para resolver, um deles de caráter obrigatório, sendo que o nosso grupo ficou encarregue de resolver o problema dois: Jornal Angolano - Folha 8, v2.

Este relatório inicia-se com uma breve introdução, seguindo-se a descrição dos problemas propostos e o que deve ser desenvolvido para os solucionar. No capítulo 3 é feito uma a exposição da arquitetura da solução, sendo descritas as diversas estruturas de dados utilizadas. De seguida é explanado o processamento do ficheiro e algumas das decisões tomadas para a sua leitura e interpretação. Por fim, são apresentadas algumas conclusões sobre a elaboração do caso de estudo abordado.

O objetivo principal deste relatório é assim expor o trabalho desenvolvido e explicar as razões das decisões tomadas ao longo do projeto.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Nesta seção iremos abordar o enunciado sucintamente de modo a explorar os problemas a resolver, no entanto, deixaremos explicações mais concretas para os capítulos seguintes. Dado um ficheiro com milhares de artigos do jornal angolano "Folha 8", o enunciado consiste:

1. Limpar e normalizar os artigos. Nesta alínea foi nos dito que poderíamos abordá-la de uma maneira mais liberal, já que o conceito de limpeza e normalização não estava bem definido. Ficou então acordado com a equipa docente que teríamos de adotar uma definição de limpeza e normalização e aplicá-la a todos os posts.

2. Criar **HTML** correspondente.

   (a) Criar um ficheiro **HTML** por cada notícia. Nesta sub-alínea, é pretendido que criemos um ficheiro **HTML** com o conteúdo normalizado obtido na alínea anterior mas que este esteja encapsulado por delimitadores do género **XML** de modo a que se observe uma estrutura.

   (b) Juntar um link para o ficheiro com texto "título"em cada indice de "tag". Nesta sub-alínea pretende-se que seja criada para cada tag encontrada, um ficheiro **HTML** da mesma (i.e. *NomeDaTag.html*) e que nesse ficheiro se encontrem ligações para todos os ficheiros html de artigos nos quais é utilizada.

3. Criar uma lista das tags encontradas e respetivo números de ocorrências. Nesta alínea, pretende-se que criemos um ficheiro que contenha os nomes das *Tags* utilizadas e o respetivo número de ocorrências que teve em todos os artigos que foram analisados de modo a que possamos mais facilmente constatar que .

2.2 Especificação dos Requisitos

O ficheiro "Folha 8"contém vários dados relativos a cada notícia, os quais precisam de ser tratados para permitirem uma visualização mais natural, para além de processar estes dados de forma a simplificar a sua análise e permitir uma conclusão rápida.

2.2.1 Dados

O extrato contém informação relativa a uma notícia:

```
<!-- ====================index.html?p=1090 -->
<pub>
#TAG: tag:{Adeptos} tag:{Alexandre Grasseli} tag:{Girabola} tag:{Petro de Luanda}
#ID:{post-1090 post type-post status-publish format-standard has-post-thumbnail hentry
category-desporto tag-adeptos tag-alexandre-grasseli tag-girabola tag-petro-de-luanda}
Desporto

Adeptos do Petro contestam Alexandre Grasseli

--------------------------------------------------------------------------------
PARTILHE VIA:
#DATE: [116eb] Redacção F8 | 22 de Setembro de 2014
[aa18]

A derrota do Petro de Luanda diante do 1º de Maio por 1-0, na última jornada do Girabola,
levou à contestação do treinador Alexandre Grasseli por parte dos adeptos \petrolíferos".

A derrota em casa diante do 1º de Maio levou muitos adeptos do Petro de Luanda a
manifestarem-se contra a continuidade de Alexandre Grasseli. Foram arremessados
objectos para dentro de campo e ouviram-se assobios contra o treinador dos actuais sétimos
classificados no Girabola.

Em declarações à Bola Angola, o treinador desvalorizou a contestação dos adeptos.

\O Petro é um clube grande e vencedor. Esta é uma altura para nos unirmos, pois só
assim é que a vitória pode chegar", afirmou Alexandre Grasseli.

Partilhe este Artigo




Etiquetas: AdeptosAlexandre GrasseliGirabolaPetro de Luanda
</pub>
```

O excerto acima referido apresenta um exemplo de um artigo. Este é delimitado por **<pub>** e contém um título, tags, id, categoria, data e etiquetas. No entanto, alguns artigos não possuem algumas destas caraterísticas, como por exemplo tags ou id. Pretendemos daqui captar, por ordem de aparição:

1. As tags utilizadas no artigo. Estas aparecem quando existe um **#TAG** sendo este seguido na mesma linha pelas tags utilizadas no artigo que se encontram na seguinte forma: **tag:{X}** onde X representa o nome da tag.
2. O id do artigo. Este aparece após um delimitador **#ID** e, na mesma linha, o id encontra-se sob a

forma de: **post-X** onde X representa o id. Esta informação encontra-se encapsulada entre chavetas juntamente com outra informação que decidimos ignorar.

3. A categoria do artigo. Encontra-se imediatamente depois do fecho das chavetas que se encontram após o **#ID**.

4. O título do artigo. É encontrado com um intervalo de `\n` após a categoria do artigo.

5. A data e o autor do artigo. São encontrados após um delimitador **#DATE**, na mesma linha, e um código encapsulado entre parêntes retos que escolhemos também ignorar.

6. O corpo da notícia em si. Encontrado após **#DATE**.

2.2.2 Pedidos

Após o processamento dos dados, esperamos obter diversos ficheiros como já foi mencionado no Capítulo 1, mais detalhadamente:

1. Relativamente à 1ª alínea, obteremos um ficheiro chamado **normalized.txt** que conterá todos os artigos após serem normalizados como resultado do processamento.

2. Relativamente ao 1º ponto da 2ª alínea, obteremos um ficheiro *html* por cada artigo com as respetivas informações encapsuladas com os delimitadores apropriados (i.e. uma tag aparecerá sob o formato: **<tag>X</tag>**), tal como o exemplo que foi fornecido no enunciado.

3. Relativamente ao 2º ponto da 2ª alínea, obteremos um ficheiro *html* por cada tag utilizada, e nesse ficheiro estarão *links* para os diversos artigos nos quais a tag foi utilizada.

4. Relativamente à 3ª alínea, obteremos um ficheiro **lista_tags.html** que conterá o nome de todas as tags (cada nome será um link que remeterá para o ficheiro respetivo da tag obtido no 2º ponto da 2ª alínea) e também o número de ocorrências que lhe corresponde.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

Para a criação dos ficheiros em *html*, é necessário o armazenamento de determinados dados, de maneira que, após o processamento do ficheiro de entrada, seja possível imprimir esses dados para os ficheiros finais respectivos. Desta forma, são necessárias estruturas de dados capazes de suportar estes dados e permitir o seu acesso rápido e eficiente. De modo a tornar o nosso programa o mais célere possível, tomamos partido das seguintes estruturas de dados fornecidas pelo **GLib**, já que foram por eles otimizadas:

1. Uma **GHashTable hash** na qual inserimos como key uma string que corresponde ao nome da tag e, a cada ocorrência da mesma, verificamos se esta já se encontra inserida na hash. Caso esteja, incrementamos o seu valor em 1, caso contrário inserimo-la na tabela com o valor 1.

2. Uma **GTree idPosts** na qual inserimos como key uma string que corresponde ao id do post (i.e. *post-X*, no qual o X é o número identificador do Post) e como valor também o id do post. Desta maneira, verificamos em cada artigo a processar, se de facto é a primeira vez que este ocorre ou se se encontra repetido. Caso seja a primeira ocorrência é processado regularmente e caso contrário, a flag repeated é colocada a 1 de modo a sinalizar que o post que nos encontramos a processar é repetido e como tal deve ser ignorado.

3. Um **GPtrArray tags** que utilizamos para armazenar as tags estão a ser utilizadas em cada post. Este é limpo a cada vez que um artigo é processado de modo a eliminarmos incongruências quer ao nível do número de ocorrências das tags quer em relação a que posts as tags são utilizadas.

Utilizamos também algumas variáveis globais para nos auxiliar com o processamento dos dados, entre elas:

1. Um **FILE* fd** que é utilizado para escrevermos para o ficheiro no qual se encontrarão os artigos normalizados (i.e. **normalized.txt**).

2. Um **FILE* html** que é utilizado para criar e escrever os ficheiros *html* individuais de cada artigo. Este é alterado à medida que iteramos de post em post.

3. Um **FILE* html1** que é utilizado para criar e escrever o ficheiro que conterá tanto a lista das tags encontradas como os seus respetivos números de ocorrências. Aí, cada tag terá um link para o seu ficheiro *html*, ficheiro no qual, encontrar-se-ão ligações para os ficheiros *html* dos artigos nos quais a tag foi utilizada.

4. Um **char* fileName** que é utilizado para alojar o id do artigo atualmente a ser processado.
5. Um **char* fileType** que é utilizado para alojar a extensão ".html', o qual usamos para criar os ficheiros html.
6. Um **char* title** que é utilizado para alojar o título do artigo atualmente a ser processado para posterior escrita em ficheiro.
7. Um **char* category** que é utilizado para alojar a categoria do artigo atualmente a ser processado para posterior escrita em ficheiro.
8. Um **char* directory** que é utilizado para alojar o diretório do ficheiro html relativo ao artigo atualmente a ser processado.
9. Um **int article** que indica o número de artigos já processados.
10. Um **int lim** que indica o número de artigos que o utilizador pretende processar.
11. Um **int nTags** que indica o número de tags que o artigo atualmente a ser processado possui.
12. Um **int flag** que indica se o **GPtrArray tags** deve ter ou não a sua memória libertada.
13. Um **int repeated** que indica se o artigo atual a processar é repetido, e se for esse o caso, temos implementada em cada Start Condition contingências de modo a garantir que o artigo é ignorado de modo a conservar recursos computacionais.

3.2 Algoritmos

O objectivo deste trabalho prático baseia-se no tratamento de um ficheiro *.txt*, para obtenção de informação sobre noticias do jornal angolano "Folha 8". Para tal, começamos por analisar a estrutura base dos artigos no ficheiro fornecido de modo a melhor definirmos a arquitetura da nossa estratégia para a resolução do trabalho prático.

Como resultado dessa análise, obtivemos a solução abaixo descrita, juntamente com o contexto no qual cada uma das suas partes é utilizada, que toma partido da regularidade figurada nos artigos, para resolver o trabalho prático:

1. Todos os artigos começam com uma barra com alguma informação alusiva ao título do artigo e ano de publicação, contudo, escolhemos ignorá-la já que toda esta informação se encontra melhor estruturada mais à frente. Após essa barra, é encontrado um delimitador <*pub*> do género *XML* que indica o início de uma publicação. Ao encontramos esse delimitador procedemos da seguinte forma:
   (a) atualizamos a nossa variável global, **article** que indica o número total de artigos processados.
   (b) abrimos um descritor de ficheiros **html** para um ficheiro **lixo.html**, no qual escrevemos dados de ficheiros que não possuem id, os quais escolhemos ignorar. O ficheiro **lixo.html** será posteriormente removido e o descritor de ficheiros **html** será atualizado seguidamente, quando encontrarmos o id do artigo em processamento.
   (c) começamos a Start Condition **NEWS**.
2. Nesta seção da Start Condition **NEWS**, após o aparecimento do delimitador <*pub*> verificamos uma de duas possíveis situações:

(a) Usualmente, aparecerá um **#TAG** indicando o início de uma sequência de tags relativas ao artigo em questão. Sendo este o caso, analisamos as tags utilizadas no artigo, verificando se existe o caratér "/"na mesma, já que isto causaria uma impossibildade aquando da criação do ficheiro html da mesma e adicionando-a ao array de tags, atualizando a variável global **nTags**. Verificamos também se existem espaços ou `\n` de modo a que, uma vez mais, se evitem incongruências no momento da criação dos links para os ficheiros html das mesmas.

(b) Em casos mais raros, não serão utilizadas tags no artigo, caso esse em que simplesmente ignoramos esta porção do artigo.

3. Ainda na Start Condition **NEWS**, seguidamente do **#TAG**, em artigos que seguem uma estrutura regular, encontraremos um **#ID** que conterá informação relativa a tags usadas, o tipo do post e outros dados irrelevantes. No entanto, é aqui encontrada uma das informações mais importantes relativas ao artigo, o seu id, que se encontra na forma de *post-X* onde o X representa o número do artigo. Após o encontrarmos:

   (a) Verificamos se o seu id é encontrado na nossa **GTree idPosts**. Caso seja, alteramos o valor do **repeated** a 1, de modo a sinalizar que o post em atual "processamento"é na realidade repetido e que deve ser ignorado. Caso não tenha sido encontrado é adicionado e procedemos com os passos abaixo descritos.

   (b) Atualizamos a variável global **fileName** para refeletir o id do post a ser atualmente processado.

   (c) Criamos o ficheiro *html* relativo ao artigo a ser tratado.

   (d) Inserimos cada uma das tags utilizadas no artigo (caso existam) na **GHashTable hash**, de modo a controlar o número de ocorrências de cada tag. Efetuamos este passo aqui, já que desta maneira conseguimos ter a certeza de que o número de ocorrências que cada tag teve está correto e não é afetado de maneira alguma pelo facto de haverem posts repetidos no ficheiro a processar.

   (e) Começamos a Start Condition **ID** que apenas servirá para ignorar o resto do conteúdo encontrado na linha do **#ID**. Após isto é iniciada a Start Condition **CATEGORY**.

4. Na Start Condition **CATEGORY**, imediatamente depois da ocorrência de **#ID**, aparecerá a *Categoria* do artigo a ser processado. Nesta secção escrevemos para o ficheiro normalizado a *Categoria* e guardamo-la para posteriormente ser escrita no ficheiro *html* relativo ao artigo a ser processado. Começamos também a Start Condition **TITLE**.

5. Após o aparecimento da **Categoria**, encontramos o título do artigo, na Start Condition **TITLE**. Nesta porção do programa apenas escrevemos o título do artigo para o ficheiro normalizado e guardamos o título na variável global **title** para que possa após ser escrita no ficheiro *html* individual de cada tag de modo a escrever em que artigos, cada tag foi utilizada. Para finalizar, iniciamos a Start Condition **DATE**;

6. Na Start Condition **DATE**:

   (a) Ignoramos algumas coisas, nomeadamente: uma barra que aparece em todos os artigos, o "PARTILHE VIA:"e uma secção com um código encapsulado entre parênteses retos.

   (b) Encontramos um **#DATE** à frente do qual aparece um código semelhante ao mencionado acima que é ignorado e a data de publicação da publicação e o seu respetivo autor, os quais são imediatamente escritos para o ficheiro *html* individual do artigo, assim como a categoria do mesmo.

   (c) Caso sejam utilizadas tags no artigo, cada uma delas será escrita no ficheiro *html* individual do artigo e também tomamos aproveitamos esta oportunidade para abrir/criar o ficheiro *html*

de cada tag utilizada no artigo presente, e lá escrevemos o link que leva ao artigo na qual ela aparece, leia-se, o artigo em processamento.

(d) Finalizamos, recomeçando a Start Condition **NEWS**.

7. Nesta seção da Start Condition **NEWS**:

   (a) Ignoramos um código encapsulado entre parênteses retos que ocorre ocasionalmente antes da notícia.

   (b) Processamos o texto da notícia escrevendo-o tanto para o ficheiro com os artigos normalizados tal como para o ficheiro *html* individual do artigo em processamento.

   (c) Ignoramos no fim da notícia uma seção **"Etiquetas:..."** à frente da qual aparecem de novo as tags utilizadas na notícia mas escolhemos ignorá-las visto que já as recolhemos anteriormente.

   (d) Eventualmente encontraremos um delimitador *</pub>*, após vários `\n`, indicando o fim da publicação. Aqui, terminamos a escrita no ficheiro *html* individual do artigo e sinalizamos o fim do mesmo no ficheiro com os artigos normalizados. Caso tenham sido utilizadas tags, limpamos o array utilizado, repomos o valor das variáveis globais referentes ao número de tags presentes num artigo (*nTags*) e se o artigo for repetido, colocamos a variável *repeated* a 0. Voltamos também à Start Condition **INITIAL**, na qual voltaremos a encontrar o delimitador *<pub>*, que sinaliza o começo de outro artigo, momento no qual o algoritmo recomeça.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

4.1.1 Alternativas e Decisões

Tal como na maioria dos projetos que envolvem um problema para ser resolvido, fomos deparados com várias estratégias de resolução para o mesmo. Não sendo este caso uma exceção, ao longo da deliberação envolvida no projeto tomamos várias decisões visando sempre a maior eficiência possível do programa culminando na solução em mãos, que achamos ser aquela que melhor retrata o pedido pela equipa docente.

1. Para uma fase inicial, o grupo decidiu apresentar o resultado do processamento dos dados no terminal. Deste modo, deste modo, apenas era necessário executar o *script* (processador de texto) em *flex*, fornecendo o devido ficheiro "Folha 8"em *txt* e toda a informação estaria imediatamente visível. Este método mostrou ser, do nosso ponto de vista, bastante eficaz para verificar se os resultados para as questões explicitadas pelo enunciado deste trabalho prático estavam de acordo com o pretendido.
   Contudo, à medida que o nosso grupo foi integrando respostas a cada vez mais questões, deixou de fazer sentido apresentar todas as informações no terminal, sendo que este estava a ficar sobrecarregado, e a partir daí o grupo começou a realizar a transição para o formato de ficheiros pretendido.

2. Paralelamente, para testar o programa, criamos um pequeno ficheiro com um número reduzido de artigos, no entanto, a partir de certo ponto, decidimos que faría mais sentido realizar testes com um maior número de artigos do que aqueles presentes no ficheiro, mas sem necessitar de processar completamente o ficheiro principal, para isso definimos um parâmetro adicional que deve ser passado ao correr o programa, sendo este o número de artigos a processar. Isto permitiu-nos testar o programa desenvolvido com mais cuidado e segurança.

3. À medida que avançavamos na produção do trabalho e nos sentíamos mais confiantes com o trabalho desenvolvido, fomos realizando testes com um número cada vez mais elevado de artigos, e, eventualmente apercebemo-nos que o documento fornecido possuía artigos repetidos. Enquanto grupo, decidimos que não faria sentido que estes fossem processados novamente, e para tal, foi criada uma estrutura para conter os Id's dos artigos à medida que estes eram processados, sendo que se o Id de um artigo já existisse na estrutura, este era ignorado.

4. Apercebemo-nos também que o documento possui ainda artigos que não contêm *id*, e optamos por ignorar esses artigos. Uma possível alternativa discutida em grupo consistia em atribuir um nome

arbitrário (i.e. *untitled-X*), e processar o restante artigo normalmente. Escolhemos ignorar estes artigos pois ao examinar o seu conteúdo apercebemo-nos que se tratavam de anúncios.

5. Uma das alternativas que ponderamos, foi a de criar um ficheiro normalizado por artigo de modo a que a solução ficasse mais modular. No entanto, como já criávamos o ficheiro html individual por cada artigo, cremos que ficaria melhor se todos artigos normalizados se encontrassem em um só ficheiro **txt**.

4.1.2 Problemas de Implementação

O nosso grupo não se deparou com nenhum problema grave a nível de implementação do código de Expressões Regulares de modo a retirar informação relevante. Planeamos atempadamente todas as informações que seriam possíveis de se obter através da geração dos ficheiros *html* precisamente para evitar qualquer tipo de infortúnios.

A única adversidade que detetamos foi derivada ao facto de inicialmente termos negligenciado a libertação de memória e à medida que aumentavamos o número de artigos a analisar, fomos deparados com alguns problemas advindos desse facto tais como segmentation faults. Desde esta chamada de atenção, e sendo boa prática em programação, dedicamos desde cedo atenção a este problema e como tal, no final, o nosso programa apresenta 0 bytes de memória perdida após o processamento total do ficheiro de 47Mb, como se pode confirmar pelo screenshot abaixo apresentado:

```
francisco@francisco-ThinkPad-P52s:~/Documents/Github/PL$ time valgrind --leak-check=full ./prog folha8.txt 100000
==5714== Memcheck, a memory error detector
==5714== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5714== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5714== Command: ./prog folha8.txt 100000
==5714==
==5714==
==5714== HEAP SUMMARY:
==5714==     in use at exit: 35,614 bytes in 10 blocks
==5714==   total heap usage: 294,032 allocs, 294,022 frees, 326,924,735 bytes allocated
==5714==
==5714== LEAK SUMMARY:
==5714==    definitely lost: 0 bytes in 0 blocks
==5714==    indirectly lost: 0 bytes in 0 blocks
==5714==      possibly lost: 0 bytes in 0 blocks
==5714==    still reachable: 35,614 bytes in 10 blocks
==5714==         suppressed: 0 bytes in 0 blocks
==5714== Reachable blocks (those to which a pointer was found) are not shown.
==5714== To see them, rerun with: --leak-check=full --show-leak-kinds=all
==5714==
==5714== For counts of detected and suppressed errors, rerun with: -v
==5714== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

real    1m37,502s
user    1m35,912s
sys     0m1,364s
```

Figura 4.1: Deteção dos erros de memória pela ferramenta valgrind

4.2 Testes realizados e Resultados

De modo a garantir o bom funcionamento do programa, fizemos questão de testar com quantidades variáveis de artigos e também de artigos que escapem ao formato geral dos posts. Nomeadamente, tal como discriminado acima, testamos extensivamente com artigos que não possuem id, artigos que não possuam tags e também artigos com outras anomalias.

Como tal, eventualmente processamos todo o ficheiro de 47Mb fornecido pela equipa docente e obtivemos resultados com os quais ficamos muito satisfeitos. Mostram-se a seguir alguns dos resultados obtidos

após testes feitos ao ficheiro anteriormente mencionado e os tempos de execução do programa:

```
francisco@francisco-ThinkPad-P52s:~/Documents/Github/PL$ time ./prog folha8.txt 100000

real    0m2,867s
user    0m2,077s
sys     0m0,710s
```

```
francisco@francisco-ThinkPad-P52s:~/Documents/Github/PL$ time ./prog folha8.txt 100000

real    0m2,869s
user    0m2,074s
sys     0m0,724s
```

```
francisco@francisco-ThinkPad-P52s:~/Documents/Github/PL$ time ./prog folha8.txt 100000

real    0m2,902s
user    0m2,142s
sys     0m0,686s
```

Figura 4.2: Tempos obtidos para o processamento total do ficheiro de 47Mb

Capítulo 5

Conclusão

Depois de terminado o primeiro trabalho prático da unidade curricular de Processamento de Linguagens, podemos concluir que a realização do mesmo foi importante, visto que nos ajudou a consolidar a matéria leccionada nas aulas da disciplina, desde o uso de expressões regulares(ERs) para filtrar e transformar textos até ao uso da linguagem **Flex** para desenvolver as soluções necessárias para a resolução do problema.Para além disso, permitiu também trabalhar mais uma vez com a linguagem *html* consolidando assim os nossos conhecimentos.

Como forma de conclusão pode-se referir que o trabalho prático foi, no entender do grupo, realizado com sucesso, visto que realizámos todas as tarefas propostas pela equipa docente, conseguindo lidar até com alguns imprevistos a nível da estrutura do ficheiro.