Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 2

Relatório de Desenvolvimento

Joana Cruz	Maurício Salgado	Rui Azevedo
(xxx)	(xxxx)	(A80789)

24 de Abril de 2019

Resumo

O presente trabalho tem como objectivo aumentar a experiência de uso do ambiente *Linux* e de algumas ferramentas de apoio à programação, aumentar a capacidade de escrever *Expressões Regulares (ER)* para a descrição de padrões de frase, desenvolver sistemática e automaticamente processadores de linguagens regulares que filtrem ou transformem textos e utilizar o sistema de produção para filtragem de texto ***GAWK***.

Conteúdo

Capítulo 1

Introdução

1.1 Preliminares

O objectivo deste relatório é demonstrar o processo de desenvolvimento de um processador de ficheiros, capaz de extrair um conjunto de informações de acordo com os requisitos estabelecidos. Para este efeito, desenvolveu-se um conjunto de *scripts* utilizando o sistema de produção para filtragem de texto, ***GAWK***, de modo analisar e extrair um conjunto de dados de um ficheiro.

O ficheiro a processar contém informação sobre uma colecção de cartas trocadas nos anos de mil e seiscentos aquando da viagem dos navegadores portugueses à Etiópia.

1.2 Enunciado do trabalho e objetivos

Analise com todo o cuidado o ficheiro ***natura.di.uminho.pt/ jj/pl-19/TP2/cartasetiopia.csv*** o qual contém informação diversa sobre uma colecção de cartas trocadas nos anos de mil seiscentos aquando da viagem dos navegadores portuguesas à Etiópia.

Construa então um ou mais programas ***AWK*** que processem esse arquivo de modo a:

- contar o número de cartas por local, relacionando-as com o ano de escrita.
- criar um *index* **HTML** com todos os anos, em que cada ano deve ligar a outra página HTML onde conste, para cada carta desse ano, o título da carta e o seu resumo.
- mostrar a lista das cartas — cada uma identficada pelo número, devidamente associada (em pares número-nome) aos apelidos das pessoas envolvidas no assunto relatado.
- desenhar um grafo (em ***DOT***) que relacione cada autor (identificado pelo seu nome) com o destinatário (também identificado pelo nome).

1.3 Estrutura do relatório

O presente relatório visa apresentar os diferentes passos tomados na conceção de um conjunto de programas que filtram e extraem um conjunto de informações de um ficheiro.

Irá ser contextualizado o problema em questão bem como os objetivos com a realização deste trabalho.

De seguida, apresentámos as características dos dados e decisões tomadas durante o desenvolvimento do projeto.

Por fim iremos apresentar e analisar os diferentes *scripts* desenvolvidos para responder aos requisitos pedidos, apresentando também um excerto do ficheiro a processar bem como os resultados obtidos após a execução desses *scripts*.

1.4 Características dos dados e decisões

O ficheiro a processar tem o formato de um *Comma-Separated-Values (CSV)*. Este tipo de ficheiros armazenam informação em forma tabular em *plain text*, sendo que cada linha do ficheiro representa um registo que contém um ou mais campos.

Uma vez que o ficheiro já vem formatado de uma maneira conveniente, o processo de criação de filtros torna-se mais fácil.

Neste ficheiro, cada registo apresenta informação relativa a uma carta. De seguida apresentámos a estrutura de um registo de uma carta:

$1	**$2**	**$3**	**$4**	**$5**	**$(>=6)**	
ID	Data	Local	Informacao	Apelidos	Resumo	...

Tabela 1.1: Campos do registo de uma carta

Os seis primeiros campos são fixos, ou seja, uma carta tem sempre a informação apresentada na tabela. No entanto, o resumo da carta pode estar repartido por diferentes campos.

Capítulo 2

Implementação

Neste capítulo iremos apresentar os padrões criados em ***GAWK*** de modo a responder aos requisitos estabelecidos. Foram criados, para este fim, quatro ficheiros, cada um contendo um conjunto de padrões que resolvem um requisito.

2.1 Número de cartas por local

O seguinte *script* apresenta um conjunto de filtros que permitem apresentar informação relativa ao número de cartas escritas no determinado local, sendo que estas cartas são relativas a uma data de criação.

Numa primeira fase, definimos o carácter **";"** como o separador de filtros e o carácter **"\n"** como o separador de registos (separador de registos por omissão). De seguida, normalizamos os campos dos diferentes registos, limpando os espaços em branco através da função pré-definida do ***GAWK***, ***gsub***. Como alguns registos de cartas não apresentam informação relativa ao local de emissão, este campo nesses registos foi alterado para a *string* ***NIL***.

Após a normalização dos diferentes campos dos registos, foi criada a estrutura de dados *nr_cartas* que vai guardar a correspondência entre um local, uma data e respetivos números de cartas.

De seguida apresentámos duas instâncias desta estrutura.

```
nr_cartas["Etiópia"]["1626.06.01"] == 2
nr_cartas["Goa"]["1652.10.16"] == 1
...
```

Por último, no fim de percorrer todos os registos, os resultados da filtragem do ficheiro são impressos no seguinte formato:

```
> Local
        Data1       nr_cartas
        Data2       nr_cartas
        ...         ...
        --------------------
        Total =     total_cartas_local
```

De seguida apresentámos o *script* desenvolvido para obter a informação pedida.

```
##############################################
#
# Definicao do Filter Separator
# RS = \n (por omissao)
#
##############################################
BEGIN {FS=";"}
##############################################
#
# Normalizacao de todos os campos do registo
#
##############################################
{
for(i = 1; i <= NF; i++)
    gsub(/^\s+|[]]|\s+$|\s+(?=\s)/,"",$i)
}
##############################################
#
# Se nao existir local coloca o campo com o
# valor de NIL
#
##############################################
length($3)==0 {$3 = "NIL"}
##############################################
#
# Estrutura que armazena o numero de cartas
# por local relacionando com o ano de escrita
#
##############################################
NF>5 {nr_cartas[$3][$2]++}
##############################################
#
# No fim de percorrer todos os registos e
# impresso o resultado
#
##############################################
END{
    for(i in nr_cartas){
        printf("> %s\n",i)
        for(j in nr_cartas[i]){
            total += nr_cartas[i][j]
            printf("\t %4s %6s\n",j, nr_cartas[i][j])
        }
    printf("         %s\n","-----------------")
    printf("\t %4s =    %6s\n\n","Total",total)
    total = 0
    }
}
```

2.2 Gerar ficheiros *HTML*

O objetivo deste requisito é criar um índice ***HTML*** com os anos de todas as cartas contidas no ficheiro. Esses índices ligam a outras páginas ***HTML*** onde constam, para cada carta desse ano, o título da carta e o seu resumo.

Em primeiro lugar, antes do processamento dos registos, foram definidos o separador de registo e de campos. Para além disso, foram definidas as *tags* necessárias para criar o ficheiro ***HTML*** e gravadas no ficheiro *index.html*.

De seguida, procedeu-se à normalização dos campos dos diferentes registos, limpando os espaços em branco e partindo a data de modo a termos somente o campo relativo ao ano da carta. Para além disso, uma vez que o resumo da carta está repartido por vários campos de um registo, foi feita a concatenação destes campos de modo a termos uma única *string* que contenha essa informação.

Após a normalização dos registos, a correspondência *ano -¿ título -¿ resumo* é armazenada numa estrutura de dados *anos*. De seguida apresentámos um exemplo dessa estrutura:

```
ano[Ano][Titulo] = Resumo
```

Por fim, ao fim do processamento de todos os registos, e já com a estrutura devidamente preenchida, essa informação é disposta nas respetivas nas devidas páginas ***HTML*** guardando essa informação no ficheiro *index.hmtl*.

De seguida, é apresentado o *script* desenvolvido para gerar os diversos ficheiro ***HTML***.

```
##############################################
#
# Definicao do Filter Separator
# RS = \n (por omissao)
#
# Definicao de tags HTML
#
##############################################

BEGIN { FS=";"
headerTitle="Processamento de Linguagens-TP2";
bodyTitle="Índice de anos";
headerFormat= "<html><head><meta charset = 'UTF-8'/><title>%s</title></head><body><center>
               <h1>%s</h1></center><ul>\n";
refHtml = "<a href=file:///Users/ruiazevedo/Desktop/Universidade/PL/PL/Fase2/%s>%s</a>\n";
textHtml = "<h3>%s</h3>%s";
endHtml = "</ul></body></html>";
printf headerFormat , headerTitle , bodyTitle > "index.html";
}
```

```
##############################################
#
# Normalizacao de todos os campos do registo
#
# No campo $2 (datas), apenas nos interessa
# o ano
#
##############################################
{
for(i = 1; i <= NF; i++)
    gsub(/^\s+|\s+$|\s+(?=\s)/,"",$i)
gsub(/\.[0-9.]+/,"",$2);
}
##############################################
#
# Agregacao de todos os campos a partir do
# campo $6 numa unica string
#
# Estes campos representao a descricao da carta
# dai necessitarmos de os agregar
#
# Estrutura 'ano' que armazena a correspondencia
# entre uma data um titulo de uma carta e a
# respetiva descricao
#
##############################################
{
for(i=6; i <=NF; i++)
    str=sprintf("%s %s", str, $i);
ano[$2][$4] = str;
str = NULL;
}
##############################################
#
# No fim de percorrer todos os registos são
# gerados os ficheiros HTML
#
##############################################

END {
    for(i in ano){
        printf headerFormat, headerTitle, i > i".html";
            for(j in ano[i])
                printf textHtml, j, ano[i][j] > i".html";
            printf endHtml > i".html";
            printf refHtml, i".html", i > "index.html";
        }
        print endHtml > "index.html";
}
```

2.3 Lista de cartas

O objetivo pretendido deste requisito é apresentar a lista de cartas, cada uma associada aos apelidos das pessoas envolvidas no assunto relatado.

Mais uma vez, o primeiro passo é definir os separadores de registo e de campos, seguido da normalização desses mesmos campos.

A informação relativa aos apelidos das pessoas envolvidas no assunto da carta está presente no campo 5 dos registos. Dado que cada um desses campos contém mais que um apelido, é necessário partir esse campo em n partes, sendo n o número de apelidos presentes nesse campo. A função pré-definida *split* permite com que seja possível partir um campo de um registo em várias *strings*, definindo um separador de campos. O formato do campo 5 dos registo é do tipo:

```
$5 := "     GOA :        AZEVEDO :      FILIPE III :"
```

Para a *string* ser partida em apelidos, desenvolveu-se uma expressão regular que encontra nomes numa string, ignorando espaços em branco insignificantes. Essa expressão pode ser definida da seguinte maneira:

```
regexp(Apelidos) := "\s{2,}|:"
```

Uma vez feita a sua normalização, os dados são impressos com a seguinte estrutura :

```
Número da carta : <nr_carta>
Apelidos : <lista_apelidos>
```

De seguida, apresentámos o *script* desenvolvido para gerar a informação pedida.

```
##############################################
#
# Definicao do Filter Separator
# RS = \n (por omissao)
#
##############################################

BEGIN {FS=";"}


##############################################
#
# Normalizacao de todos os campos do registo
#
##############################################

{
for(i = 1; i <= NF; i++)
if(i != 5 )
       gsub(/^\s+|\s+$|\s+(?=\s)/,"",$i)
}

##############################################
#
# Parte o campo $5 em apelidos através da
# funcao split. Em cada posicao do array
# apelidos temos um apelido
#
##############################################

{split($5,apelidos,/\s{2,}|:/)}
#{split($5,apelidos,/[^a-zA-ZáÁéÉóÓçÇ]+/)}

##############################################
#
# Imprime o resultado ao fim de processar
# cada record
# result := <id,[Apelidos]>
#
##############################################

length(apelidos)>0    { printf("Número da carta: %d\n",$1);
printf("Apelidos: ")

for(i in apelidos)
if(length(apelidos[i])>0)
printf("%s, ", apelidos[i])
print "\n"
}
```

2.4 Grafo *DOT*

É pretendido, neste requisito, que seja gerado um ficheiro ***DOT*** de maneira a gerar um grafo que relacione cada autores com o destinatário das cartas.

O formato do *script* a criar é o seguinte:

```
diagraph grafo{
    Autor -> Destinatário;
    ...
}
```

O primeiro passo é sempre normalizar os diferentes campos do registo e definir os devidos separadores. Para além disso, é inicializada a estrutura de um grafo em notação ***DOT*** escrevendo-a no ficheiro *graph.gv*.

Após a normalização dos dados, é necessário recolher a informação dos diferentes autores e destinatários. Para isto, analisamos o campo 4 dos diferentes registos de modo a encontrar um padrão que nos permita descobrir os autores e destinatários. Após esta análise, podemos verificar que os seguintes padrões:

```
(1) Carta <preposição> <nome> <preposição> <nome>
(2) Certidão da carta <preposição> <nome> <preposição> <nome>
(3) Requerimento <preposição> <nome> <preposição> <nome>

        <preposição> := de || do || a || ao || aos
```

De modo a obter os autores e destinatários corretamente, vamos filtrar todos os registos cujo o campo 4 contém as palavras Carta, Requerimento ou Certidão. Após feita a correspondência apagámos essas palavras do registo ficando apenas com informação no formato:

```
<nome> <preposição <nome>

    <preposição> := a || ao || aos
```

De seguida, partimos essa *string* através da função pré-definida *split*, sendo o separador de campo as diferentes preposições definidas. Após esta filtragem, é escrito no ficheiro *graph.gv* a seguinte *string*:

```
Autor -> Destinatário;
```

Ao fim de processar todos os registos, fechámos a definição da estrutura do grafo escrevendo no ficheiro o carácter **}**.

De seguida, apresentámos o *script* criado para gerar o ficheiro em formato ***DOT***.

```
##############################################
# Definicao do Filter Separator
# RS = \n (por omissao)
#
# Inicio da estrutura de dados de um grafo
#
# digraph grafo {
# size="100,100";
#
# Escrita do inicio da estrutura no ficheiro
# graph.gv
##############################################
BEGIN { FS=";"
        graph = "digraph grafo {\n\tsize=\"100,100\";\n";
        printf graph > "graph.gv";}
##############################################
# Normalizacao de todos os campos do registo
##############################################
{
for(i = 1; i <= NF; i++)
    gsub(/^\s+|\s+$|\s+(?=\s)/,"",$i)
}
##############################################
# Encontra todas as linhas cujo campo $4 (Titulo)
# contenha a palavra carta, requerimento ou
# certidao
# Parte-se a frase na proposicao 'ao' e guarda
# no array 'autores' o resultado
#   autores[1] := Autor
#   autores[2] := Destinatario
#
# Adiciona ao ficheiro graph.gv sd relacoes
# "autor" -> "destinatario";
##############################################
$4 ~ /(Carta|Requerimento|Certidão)/
{
    gsub(/(Carta|Requerimento|Certidão)[ a-z]+(a|d)(e|o)/,"",$4);
    split($4,autores,/ ao?s? /)

    if( contain[autores[1]][autores[2]] == 0 ){
        autor = "\t\"%s\" -> \"%s\";\n";
        printf autor, autores[1], autores[2] > "graph.gv";
        contain[autores[1]][autores[2]]++
}
##############################################
# No fim de percorrer todos os registos fecha
# a definicao da estrutura do grafo
##############################################
END { printf "}" > "graph.gv";}
```

Capítulo 3

Apresentação de resultados

3.1 Input

Nesta secção apresentámos um excerto do conteúdo do ficheiro a processar que contém apenas 17 registos. O ficheiro original contém 67 linhas, logo 67 registos, cada um com o número de campos variável.

Podemos verificar, como já foi referido, que o ficheiro está organizado de forma tabular, um registo por linha e um conjunto de campos por registo separados pelo carácter **;**

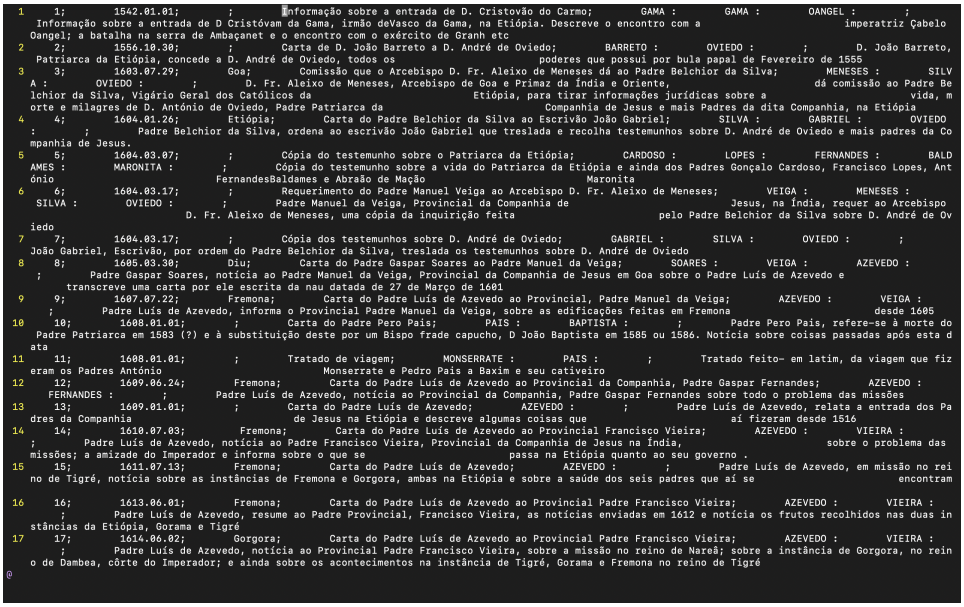

Figura 3.1: Ficheiro com os registos das cartas

3.2 Output's

De seguida, apresentámos excertos dos resultados obtidos após correr o comando:

```
awk -f ex<n>.awk cartasetiopia.csv
    <n> := número do exercício
```

Capítulo 4

Conclusão

A linguagem de programação ***GAWK*** mostrou-se muito útil e versátil para resolver este tipo de trabalhos. Neste caso, uma vez que o ficheiro já estava com formato ***CSV***, o processo de filtrar e extrair informação de ficheiros tornou-se uma tarefa menos custosa, uma vez que o ***GAWK*** apresenta vários mecanismos, intuitivos, de processar estes mesmos ficheiros.

As maiores dificuldades encontradas pelo grupo estão relacionadas com a normalização de dados, uma vez que tivemos que, através de expressões regulares, encontrar/filtrar os dados de forma totalmente genérica.

Podemos concluir que o ***GAWK*** é uma ferramenta muito poderosa e útil para manipular ficheiros/arquivos de dados de uma maneira simples e eficaz.