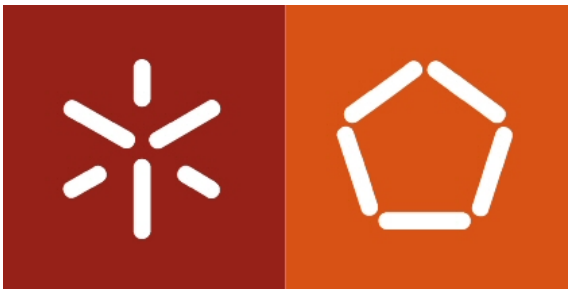


Universidade do Minho

Mestrado Integrado em Engenharia Informática

Processador de Processos de Emigração

Trabalho Prático n.º 2 - GAWK

Processamento de Linguagens

Alexandre Pacheco (A80760)
Diogo Sobral (A82523)
Inês Alves (A81368)

28 de Abril de 2019

Resumo

No âmbito da Unidade Curricular de Processamento de Linguagens, foi proposto ao grupo, como forma de desenvolver os seus conhecimentos à cerca dos conteúdos lecionados nas aulas práticas, a realização de um conjunto de exercícios presentes no ficheiro fornecido. Neste ficheiro estavam presentes vários enunciados de diferentes trabalhos, sendo que o grupo ficou com o enunciado n.º 2.1: Processador de Processos de Emigração.

Deste modo, o objetivo deste trabalho prático passa por aprofundar os conhecimentos relativos à Unidade Curricular em causa.

Conteúdo

1 Introdução

Estando o grupo a frequentar o 3.º ano do Mestrado Integrado em Engenharia Informática, foi-nos proposto, no contexto da Unidade Curricular de Processamento de Linguagens, que aprofundássemos os nossos conhecimentos na área. Posto isto, este trabalho consiste no desenvolvimento de um filtro de texto *GAWK*, através do uso de expressões regulares. Assim, este projeto torna-se uma mais-valia no que toca ao aumento de experiência de uso do ambiente Linux e diversas ferramentas auxiliares, ao aumento da capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases e ainda, a partir destas, desenvolver Processadores de Linguagens Regulares, que filtrem ou transformem textos.

Ao longo deste relatório, iremos apresentar todas as etapas e decisões tomadas durante todo o processo de elaboração do filtro de texto *GAWK* que desenvolvemos.

2 Análise do Ficheiro Recebido

Todos os problemas requerem que seja feita um breve análise do mesmo antes de ser iniciada qualquer tentativa para solucionar os mesmos. Uma vez que este trabalho prático se foca em processamento de texto, é crucial uma boa análise inicial para o sucesso do mesmo. Neste sentido, o primeiro passo a tratar aquando do desenvolvimento da nossa solução, passou pela observação consistente e atenta do ficheiro dado.

Ao olhar brevemente para o ficheiro recebido, pudemos constatar como o mesmo estava estruturado. As duas primeiras linhas continham informação, não só sobre sobre a estrutura, mas também sobre a informação representada e as restantes continham todos os dados que estavam guardados na estrutura. Para compreender melhor os campos da estrutura e, visto que o ficheiro continha a extensão .cvs, decidimos abrir o mesmo no excel. Aqui, apercebemo-nos que a primeira linha continha o nome dado a um conjunto de campos e a segunda linha continha os nomes para cada campo do conjunto, como podemos ver a seguir.

```
IdEmigrante;NOME;NATURALIDADE;;;DNasc;FILIAÇÃO;;;;CONJUGE;
BI;;Lugar;Freg;Concelho;;NomePai;IdPai;NomeMae;IdMae;Nome;IdConj
```

Por fim, ao analisar as linhas individualmente, verificamos que, embora haja vários **;** seguidos em alguns casos, todas a informações que necessitámos sobre um dado processo estão contidas na mesma linha e entre **;**. Assim, para todos os exercícios propostos, usamos como *Field Separator* este mesmo caracter: **;** e como *Row Separator* **n**.

3 Exercício 1

3.1 Enunciado e Descrição do Problema

Como já foi referido, a resolução destes exercícios requer uma abordagem inicial ao ficheiro *emigra.csv*, que contém informações sobre processos de pedido de passaporte para emigrar, registados no início do sec. XX. Neste exercício, foi-nos proposta a implementação de um programa *Awk* que processasse o ficheiro de modo a obter o número de processos registados por Concelho e por Freguesia.

3.2 Estratégia de Implementação

Ora, para começar, criou-se um *array bidimensional*:

```
NR > 2 {conta[$5][$4]++}
```

Como podemos constatar, a partir da análise do ficheiro dado, o campo $5 refere-se ao concelho, enquanto que o campo $4 se refere à freguesia. Posto isto, com esta estrutura, podemos inferir que, a cada concelho, estão associadas várias freguesias, que vão sendo incrementadas. Desta forma, sabemos quantos processos, por freguesia, tem cada concelho.

De notar também que a cláusula $NR > 2$ significa que iremos apenas ter em conta os concelhos e respetivas freguesias, a partir da terceira linha, que é onde começam as informações a que necessitamos de aceder.

De seguida, para realizar, efetivamente, a contagem e respetiva atribuição das diversas freguesias, nos diferentes concelhos, foi implementado o seguinte código:

```
for (concelho in conta){
    printf(linha,concelho,"") > ficheiro;

    printf("<table style=ẅidth:30%¨><tr> <th>%s</th><th>N.° de processos
    registados por freguesia</th> </tr>","Freguesia") > ficheiro;

for (freguesia in conta[concelho])
    printf (tabela,freguesia,conta[concelho][freguesia]) > ficheiro;

    print "</table>" > ficheiro;
}
```

Com o intuito de tornar a visualização do resultado deste exercício esteticamente apelativo e de fácil compreensão, o grupo decidiu colocar os resultados numa tabela, como iremos mostrar mais adiante e como se torna óbvio com este código.

Atentemos agora na parte colorida do código acima. Desta forma, é fácil entender a implementação realizada neste exercício: por cada concelho encontrado no *array* **conta**, é realizada uma impressão na tabela de todas freguesias que esse dado concelho possui, acompanhadas pela quantidade de processos de emigração registados nessa dada freguesia.

Posteriormente, de forma análoga, fazemos a contagem dos processos registados por concelho com a implementação do seguinte código:

```
for (concelho in conta){
    r = 0
    for (freguesia in conta[concelho])
        r+=conta[concelho][freguesia]

    printf (tabela, concelho, r) > ficheiro;
};
printf (tabela,"Total:", NR-2) > ficheiro;
```

Ou seja, percorremos todas as freguesias dos diversos concelhos, onde guardamos, numa variável **r** que vai sendo incrementada, a quantidade de processos registados nessa mesma freguesia desse concelho. Posteriormente, fazemos o mesmo para a freguesia seguinte e por ai adiante. No fim, imprimimos ainda uma linha com o total de processos registados.

3.3 Resultados obtidos

Vejamos, então, os resultados obtidos a partir desta implementação.

Freguesias

Ponte da Barca

Freguesia	N.º de processos registados por freguesia
Asias	1

Fare

Freguesia	N.º de processos registados por freguesia
Fareja	1

Fafe

Freguesia	N.º de processos registados por freguesia
Aboim	1
Vilacova	1
Agrela	1
Fornelos	1
Varzeacova	1
Aões S. Romão	1
Ribeiros	1
S. Gens	2

Figura 1: Número de processos registados nas diversas freguesias

Concelhos

Concelho	N.º de processos registados por concelho
Ponte da Barca	1
Fare	1
Fafe	54
Guimarães	1
Total:	57

Figura 2: Número de processos registados nos diversos concelhos

4 Exercício 2

4.1 Enunciado e Descrição do Problema

Assim como no exercício anterior, recorremos à análise do ficheiro dado no enunciado deste projeto. Desta vez, neste exercício, foi-nos proposto que implementássemos um programa *Awk* que processasse este mesmo ficheiro de modo a calcular a frequência de processos por ano, bem como relaciona-los com os concelhos.

4.2 Estratégia de Implementação

Como já foi dito, neste exercício queríamos calcular a frequência, por ano, dos processos de emigração presentes no ficheiro e relaciona-los com os concelhos. Para isto, tal como no exercício anterior, bastava-nos realizar uma tabela (ou outra estrutura bem organizada) que fornecesse a informação pedida, no entanto, o grupo, de forma unânime, optou por elevar um pouco a dificuldade deste exercício e criar um *index HTML* com todos os anos presentes no ficheiro. Neste, cada ano liga a uma outra página **HTML** que contém, então, a quantidade de processos encontrados nesse ano, nos diversos concelhos existentes.

Mais uma vez, a análise do ficheiro foi crucial na medida em que nos conseguimos aperceber que existiam três diferentes formas de separar os sub-campos, no campo referente à data do processo. Imaginemos que nos estamos a referir à data do dia 26 de março de 1924. Para este dia, existem três formatos possíveis de representação:

```
26/03/1924
1924.26.03
1924-08-07
```

Assim, de modo a contemplar todas as hipóteses, foram criadas três expressões regulares, uma para cada uma das hipóteses de representação:

```
NR > 2 && ($6 ~/[0-9]*\.[0-9]*\.[0-9]/) {split($6,data,".");
conta[data[1]][$5]++;}

NR > 2 && ($6 ~/[0-9]*\/[0-9]*\/[0-9]/) {split($6,data,"/");
conta[data[3]][$5]++;}

NR > 2 && ($6 ~/[0-9]*-[0-9]*-[0-9]/) {split($6,data,"-");
conta[data[1]][$5]++;}
```

Posto isto, desenvolvemos o seguinte código para concretizar o exercício:

```
for (key in conta){

    ficheiroAno = "results/Ano/freq" key ".html";

    print "<li>" "<a href=Äno/freq" key ".html" "Ï>Ano " key"</li>"
    > ficheiro;

    print header > ficheiroAno;

    print "<h1 align=c̈enter¨> Processos do Ano " key "  </h1>" > ficheiroAno;

    printf("<table style=ẅidth:30%¨><tr> <th align="center">%s</th>
    <th align="center"> N.º de processos registados por concelho</th> </tr>",
    "Concelho") > ficheiroAno;

for (concelho in conta[key])
    printf (tabela,concelho,conta[key][concelho]) > ficheiroAno;

    print "</table>" > ficheiroAno;
    print "</body> </html>" > ficheiroAno;
}
```

Vejamos novamente o código colorido acima, onde podemos deduzir que *"key"* se refere a cada ano que se encontra no *array bidimensional* criado previamente.

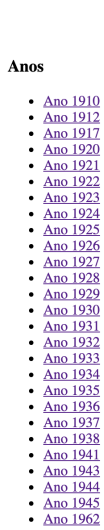
Com as expressões regulares já mencionadas sabemos que o *array* **conta** se encontra preenchido com o número de processos de emigração registados num determinado ano. Agora, queremos associar a este mesmo, um concelho, de forma a sabermos quantos processos foram, então, registados, nos diversos anos presentes no ficheiro. Para isso, percorremos o *array* **conta**, como podemos observar no código acima, criado ficheiros *.html* numa pasta com o nome **Ano** e com a seguinte sintaxe:

freq<**ano**>.html

E, dentro de cada um destes ficheiros, é preenchida uma tabela com objetivo desejado: a quantidade de processos de emigração registados naquele ano, por concelho. Isto é conseguido através do último ciclo *for* que se encontra no código demonstrado.

4.3 Resultados obtidos

Demonstrado o raciocínio utilizado para a resolução deste exercício, vejamos agora os resultados do mesmo. Começamos por obter uma página HTML, de seu nome **ex2.html**, com todos os anos presentes no ficheiro dado, estando este guardado na pasta **results**.



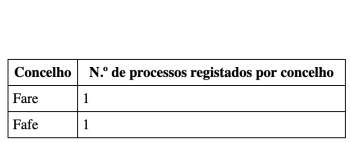
Processos por ano e os seu concelhos

Anos

- Ano 1910
- Ano 1912
- Ano 1917
- Ano 1920
- Ano 1921
- Ano 1922
- Ano 1923
- Ano 1924
- Ano 1925
- Ano 1926
- Ano 1927
- Ano 1928
- Ano 1929
- Ano 1930
- Ano 1931
- Ano 1932
- Ano 1933
- Ano 1934
- Ano 1935
- Ano 1936
- Ano 1937
- Ano 1938
- Ano 1941
- Ano 1943
- Ano 1944
- Ano 1945
- Ano 1962

Figura 3: Página HTML com todos os anos presentes no ficheiro *input*

Agora, clicando num destes anos, somos redirecionados para outra página HTML que contém a quantidade de processos desse ano e respetivos concelhos.



Processos do Ano 1930

Concelho	N.º de processos registados por concelho
Fare	1
Fafe	1

Figura 4: Número de processos registados no ano de 1930 e respetivos concelhos

5 Exercício 3

5.1 Enunciado e Descrição do Problema

A principal entidade presente num processo de emigração é o utente. Esta entidade possui várias caraterísticas que são retratadas no ficheiro recebido. Todavia, na sua grande maioria, estas fazem referência a nomes quer do próprio utente quer dos seus parentes. Neste sentido, o terceiro exercício deste trabalho prático propõe o estudo da ocorrência de nomes próprios no ficheiro.

Como foi evidenciado no Capítulo 2 deste relatório, a informação está organizada por campos e, para solucionar este problema, os campos que usamos foram os seguintes:

- Nome
- NomePai
- NomeMae
- Conjuge-Nome

É de destacar que nem todos os utentes possuem estes campos e, nessas circunstâncias, os campos em faltam encontram-se por preencher ou seja, vazios.

Por fim, a definição de nome próprio pode influenciar a estratégia para resolver o problema. Assim, decidimos que todas as palavras de um nome que se iniciem por letra maiúscula constituem um nome próprio. Os seguintes exemplos demonstram a nossa interpretação:

Nome Completo	**Nomes Próprios**
Maria do Carmo	[Maria,Carmo]
João Teixeira Santos	[João,Teixeira,Santos]

Tabela 1: Exemplo dos nomes próprios de um nome

Esta abordagem complica bastante o problema, visto que o filtro GAWK não tem nenhuma função que indique se uma dada palavra se inicia por maiúscula e o *Field Separator* usado apenas nos permite chegar ao nome completo.

5.2 Estratégia de Implementação

Como nos exercícios anteriores, escolhemos utilizar **HTML** para apresentar o resultado da nossa soluçãom no entanto, a sua implementação não será abordada nesta secção. O HTML gerado contém uma tabela com o nome próprio e o respetivo número de ocorrências.

Tal como foi referido na secção anterior, o *Field Separator* que decidimos usar foi o **;**. Embora este acarrete complicações na partição de um nome completo, permite-nos aceder aos campos que pretendemos. Os campos relevantes que usamos foram nome, nomepai, nomemae e conjuge que ocupam, respetivamente, as posições 2,7,9 e 11 em cada linha.

Uma vez que nem todos os campos estão preenchidos, se os usássemos iríamos ter valores sem conteúdo e, a nosso ver, não faz sentido algum ter essa informação. Para resolver este conflito, estabelecemos uma expressão regular capaz de caraterizar um nome completo e verificamos se o campo que queremos comparar a respeita.

[A-Z][a-z]*([a-zA-Z][a-z]*)*

O processo acima fica descrito em GAWK do seguinte modo:

```
NR > 2 && ($2 ~/[A-Z][a-z]*( [a-zA-Z][a-z]*)*/)
```

As duas primeiras linhas só contêm informação sobre a estrutura e obrigam-nos a usar $NR > 2$ na expressão acima para as descartar. Neste ponto, já conseguimos aceder a um nome completo e o passo que se sucede é a separação em várias palavras. Para isto, utilizamos a função *split* com o limitador *space*.

```
NR > 2 && ($2 ~/[A-Z][a-z]*( [a-zA-Z][a-z]*)*/) {split($2,data," ");
```

De seguida, é necessário adicionar as novas palavras a um dicionário, no entanto, estas ainda possuem exceções que, segundo a nossa definição para nomes próprios, devem ser retiradas. Uma vez que o GAWK não fornece uma função para verificar se um palavra começa por letra maiúscula foi necessário improvisar. Na cláusula **BEGIN**, criamos um dicionário em que a chave corresponde a um carácter e o valor associado é o valor na tabela ASCII. Após a construção deste dicionário, para cada uma das palavras que pretendemos adicionar, calculamos a usa primeira letra e verificamos o valor associado à mesma no dicionário anteriormente criado. Caso o valor seja inferior a 97, este é adicionado a um dicionário para os resultados.

```
NR > 2 && ($2 ~/[A-Z][a-z]*( [a-zA-Z][a-z]*)*/) {split($2,data," ");
for(i in data){ if (ascii[substr(data[i],1,1)] < 97) {conta[data[i]]++;} } }
```

Este procedimento é efetuado para os 4 campos de contêm informações relevantes.

Por fim, toda a informação guardada no dicionário com os resultados é gravada no ficheiro HTML.

5.3 Resultados obtidos

A imagem abaixo retrata alguns dos resultados obtidos quando utilizamos o ficheiro recebido:

Nome Próprio	Nº de ocorrências
Braz	2
Claudino	2
Nogueira	5
Olinda	2
Queirós	1
Albina	2
Soledade	1
Soares	2

Figura 5: Resultados obtidos.

Analisando os resultados obtidos, foi possível notar que os mesmos apresentação uma solução muito satisfatória e versátil. No entanto, há casos em que o filtro não é capaz de detetar uma ocorrência. No ficheiro recebido existe um nome completo em que o nome Maria está escrito a começar em letra minúscula e não é possível detetar a ocorrência. Como este caso, todos os que se encontrem na mesma situação passarão ao lado.

6 Exercício 4

6.1 Enunciado e Descrição do Problema

À semelhança do exercício 3, este assenta novamente em nomes para produzir o resultado final. No entanto, ao invés de produzir estatísticas sobre os mesmos, o objetivo é interliga-los de modo a constituir uma espécie de árvore genealógica. Para isto, usaremos novamente os campos utilizados no exercício anterior juntamente com o mesmo *Field Separator*.

6.2 Estratégia de Implementação

O primeiro desafio para este problema, passa por decidir como é que vamos guardar a informação e como é que a vamos relacionar, de modo a atingir o objetivo esperado. Uma vez que, por cada linha, vamos utilizar 4 campos, decidimos que iríamos utilizar 4 dicionários, 1 para guardar o nome dos utentes e 3 para guardar as relações Utente-Pai,Utente-Mãe e Utente-Cônjuge.

```
nomes[Utente]; pai[Utente] = nome_pai;
mae[Utente] = nome_mãe; conjuge[Utente] = nome_conjuge;
```

Cada linha processada detêm a informação toda necessária para fazer a ligações necessárias para um utente e estas encontram-se nas seguintes posições:

Nome do Campo	Posição do Campo
Nome	2
NomePai	7
NomeMae	9
Conjuge	11

À semelhança do exercício 3, os campos utilizados podem estar vazios e, para impedir erros, precisamos de utilizar expressões regulares para ler os valores. Dando uso às expressões regulares e aos dicionários armazenamos toda a informação necessária. A implementação acima descrita fica traduzida em GAWK do seguinte modo:

```
NR > 2 && ($2 ~/[A-Z][a-z]*( [a-zA-Z][a-z]*)*/) {nomes[$2]++;}
NR > 2 && ($7 ~/[A-Z][a-z]*( [a-zA-Z][a-z]*)*/) {pai[$2] = $7;}
NR > 2 && ($9 ~/[A-Z][a-z]*( [a-zA-Z][a-z]*)*/) {mae[$2] = $9;}
NR > 2 && ($11 ~/[A-Z][a-z]*( [a-zA-Z][a-z]*)*/) {conjuge[$2] = $11;}
```

Por fim, para obter o grafo final, basta percorrer as chaves no dicionário dos nomes e, caso haja uma chave idêntica nos outros 3 dicionários, é escrita para o ficheiro final uma ligação entre a chave e a valor correspondente.

6.3 Resultados obtidos

O exemplo abaixo mostra um dos grafo gerados:

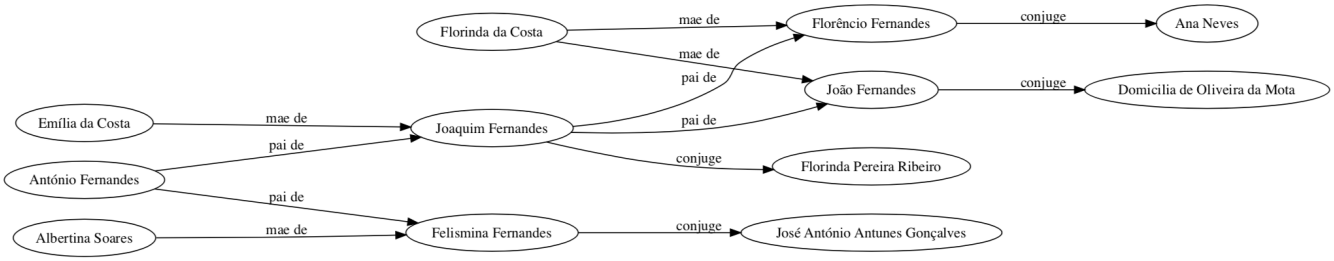


Figura 6: Um do grafos gerados.

7 Conclusão e Análise Crítica

Finalizado este segundo trabalho prático da Unidade Curricular de Processamento de Linguagens, podemos fazer um balanço do mesmo.

Como era de esperar, a realização deste trabalho revelou-se bastante enriquecedora e útil, na medida em que pudemos consolidar conhecimentos adquiridos nas aulas desta UC. Para além do aprofundamento dos nossos conhecimentos acerca de Expressões Regulares e filtragem de texto *GAWK*, pudemos ainda explorar as nossas capacidades no que toca ao uso de *HTML*, tornando este projeto ainda mais gratificante e desafiador.

Em suma, estamos satisfeitos com o nosso desempenho na elaboração deste trabalho, considerando que todas as nossas respostas foram bem estruturadas, ao ponto de estarem visualmente apelativas e explicadas de forma sucinta e simples.