Escola de Engenharia
Universidade do Minho

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
Mestrado Integrado em Engenharia Informática
Processamento de Linguagens

GAWK

Trabalho prático nº2

Enunciado 4

Marco Gonçalves
A75480

Ricardo Canela
A74568

Braga, 28 de Abril de 2019

Conteúdo

1. Introdução

Foi proposto como trabalho prático da unidade curricular de Processamento de Linguagem, a utilização de *GAWK* para tratamento de informação. Este trabalho prático tem como principais objectivos: Com esta proposta o professor disponibilizou 5 enunciados distintos em que cada grupo escolheria um deles baseado na seguinte formula : (menor nº de aluno do grupo %5)+1, que nos nosso caso (74568%5)+1 = 4. No exercício 4 é nos pedido para recebendo o ficheiro *CETEMPublicoAnotado2018-p1.txt*,(O *CETEMPúblico* é um corpus de aproximadamente 180 milhões de palavras em português europeu) e tratar a informação de forma a efetuar a análize do ficheiro e responder ao requesitos do enunciado:

- 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 descrição de padrões de frases;
- Desenvolver, a partir de ERs, sistemática e automaticamente *Processadores de Linguagens Regulares*, que filtrem ou transformem textos.
- Utilizar o sistema de produçao para filtragem de texto *GWAK*.

Como complemento do problema, os resultados do projeto são apresentados numa estrutura HTML.

2. Processador de CETEMPublico

2.1 Enunciado 2.4

O formato CETEMPúblico, é um ficheiro dividido por linhas e nessas linhas contém tags em XML que adicionam informação sobre anotação frásica para além dessas linhas contem linhas com a informação detalhada representados palavras. Pretende-se, assim escrever um sistema de produção para filtragem de texto, GAWK.

- Conta o número de Extratos, Parágrafos e Frases;
- Extrai a lista das muti-word-expressions e respetivo número de ocrrências;
- Calcular a lista dos verbos PT: (Lema, para palavras com pos=V) e resprtivo número de ocorrências;
- Determina o dicionário implícito no córpora - calcule a lista das palavras associando-lhes os possíveis (lema, pos).

2.2 Descrição do problema

O nosso ficheiro esta separado por linhas e existem 2 tipos de linhas as linhas para dar inico ou fim a conjuntos (tipo xml). Neste trabalho temos que contar os Extratos que começam por <*ext* ***informação desnecessária***> e acabam por <*/ext*>, contar os parágrafos que começam por <*p* ***informação desnecessária***> e acabam por <*/p*> e contar frases que começão por <*s*> e terminam quando aparece uma linha com <*/s*>. Temos ainda que extrair a lista das *multi-word-expression* e os respectivos números de ocorrências. Outra das finalidades deste trabalho é calcular a lista de verbos e respectivas ocorrências, bem como criar um dicionário de palavras onde lhes associo o (lema, pos).

2.3 Resolução do problema

Para a resolução deste problema apenas precisamos de um programa. Neste programa começamos no **BEGIN** por defenir o *Field Separator* (**FS**) como "\t", pois as linhas que contem mais de 1 *Number of Filds* (**NF**) tem o campos separados por *tabs* e inicializamos a mweF = 0 que serve para sabermos se estamos dentro de uma multi-word-expression, inicaliazamos ainda o p que representa o numero de paragrafos, o s que representa o numero de frazes, e o ext que representa o numero de extratos. Após o BEGIN decidimos dividir as linhas por NF<5 e NF>5 isto porque supostamente apenas existem linhas com 1 campo e linhas com 9 campos e como quando tem mais de um campo precisamos de utilizar como o maior campo o $5.

```
NF<5{
        aux = first($0)
        if(aux == "<mwe" || aux == "<mwe>"){
                mweF = 1
                mweW = ""
        }
        if(aux == "<p" || aux == "<p>") p++
        if(aux == "<s>") s++
        if(aux == "<ext" || aux == "<ext>") ext++
        if(aux == "</mwe>"){
                mweF = 0;
                if(!mweList[mweW]){
                        mweList[mweW] = 1;
                } else {
                        mweList[mweW]++;
                }
        }
}
```

Figura 2.1: NF<5

Durante o processamento das linhas com NF<5, sabemos que estamos perante uma linha que contem informação sobre se começa ou acaba algum tipo de conjunto e como temos conjuntos que contém informação desnecessária separada por um espaço decidimos criar uma função *first* que utiliza a função split para separar uma string e ao receber o retornando assim o primeiro elemento da lista recebido pelo split.

```
function first(s){
        split(s,a," ",b)
        return a[1]
}
```

Figura 2.2: First

Tendo este primeiro elemento este pode ser igual a «mwe", «mwe> o que faz com que tenhamos que por a mweF=1 , para quando processar-mos os NF>5 saibamos que estamos dentro de uma multi-word-epression, e ainda inicializamos a mweW=. Caso seja igual a «p"ou igual a «p>"incrementamos o p, caso seja «s>"incrementamos o s, caso sejas «ext"ou «ext>"incrementamos o ext e caso seja «/mwe>"que significa que acabou a multi-word-expression pomos a mweF=0 e adicionamos a mweW atual á nossa lista de expressões mweList, caso esta ja exista na nossa lista incrementamos o numero de apariçoes.

```
NF>5 {
        if($5 =="V") {
                if(!verbos[$4]){
                        verbos[$4] = 1;
                } else {
                        verbos[$4]++;
                }
        }
        if($5 != "" && !dicionario[$1] && $5 != "PU" && $5 != "NUM
_card") {
                dicionario[$1] = $4 ", " $5 "."
        }
        if(mweF == 1){

                mweW = concat(mweW,$1)
        }
}
```

Figura 2.3: NF>5

Durante o tratamento do NF>5 estamos perante uma linha que contem uma palavra e informação á cerca da mesma. O 1º campo contem a palavra, o 4º contem o lema, e o 5º contem a pos(part of speach). Neste processamento fazemos 3 coisas, que sao elas adicionar a palavra á lista de verbos caso o $5 seja igual a "V"ou incrementá-la caso já la esteja presente, adicionamos a palavra ao nosso dicionario caso o $5 seja difrente de , "PU"ou "NUM_card", e caso a mweF esteja igual a 1 igualamos a mweW á nossa função concat definida por nos que recebe 2 stings e devolve apenas uma que é o resultados da sua junção e neste caso recebe a antiga mweW e o $1.

```
function concat(a,b){
        return a " " b
}
```

Figura 2.4: concat

```
END {
        #Indice
        print indice() > "indice.html"
        #Estatísticas a)
        print begin() > "data/stats.html"
        print title("Estatísticas") > "data/stats.html"
        print "<li>Parágrafos: " p "</li>\n" > "data/stats.html"
        print "<li>Frases: " s "</li>\n" > "data/stats.html"
        print "<li>Extratos: " ext "</li>\n" > "data/stats.html"
        print end() > "data/stats.html"
        #MWE b)
        print begin() > "data/mwe.html"
        print title("Multi-word-expressions") > "data/mwe.html"
        for(k in mweList){
                print "<li>" k " : " mweList[k] "</li>" > "data/mw
e.html"
        }
        print end() > "data/mwe.html"
        #Verbos c)
        print begin() > "data/verbos.html"
        print title("Verbos") > "data/verbos.html"
        for(k in verbos){
                print "<li>" k " : " verbos[k] "</li>" > "data/ver
bos.html"
        }
        print end() > "data/verbos.html"
        #Dicionario d)
        print title("Dicionário") > "data/dicionario.html"
        print begin() > "data/dicionario.html"
        for(k in dicionario){
                if(dicionario[k] != "") print "<li>" k ": " dicion
ario[k] "</li>" > "data/dicionario.html"
        }
}
```

Figura 2.5: Criação de html

```
function link(u,t){
        return  "<li><a href='" u "'>" t "</a></li>\n"
}
function title(u){
        return "<h1>" u "</h1>\n"
}
function begin(){
        return "<html>\n\t<head>\n\t\t<meta charset=\"UTF-8\"/></h
ead>\n\t<body>\n\t\t<ul>\n<link rel=\"stylesheet\" href=\"https://
stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css\"
 integrity=\"sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOh
cWr7x9JvoRxT2MZw1T\" crossorigin=\"anonymous\">"
}
function end(){
        return "\n\t\t</ul>\n\t</body>\n</html>"
}
function indice(){
        return begin() "\n" title("Indíce") link("data/stats.html"
,"Estatísticas") link("data/mwe.html","Multi-word-expressions") li
nk("data/verbos.html","Lista de Verbos") link("data/dicionario.htm
l","Dicionário") end()
}
```

Figura 2.6: Funções auxiliares aos prints

Por fim no **END** geramos todos o html recorrendo a prints em ficheiros da informação até aqui guardada.

2.3.1 Alínea a)

Figura 2.7: Estatísticas

2.3.2 Alínea b)

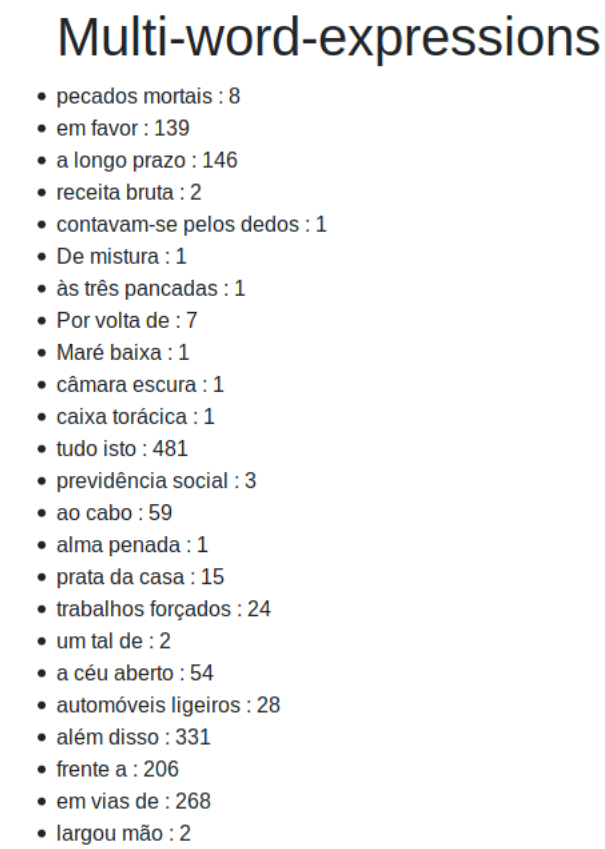

Figura 2.8: Pequena amostra de Multi-word-expressions

2.3.3 Alínea c)

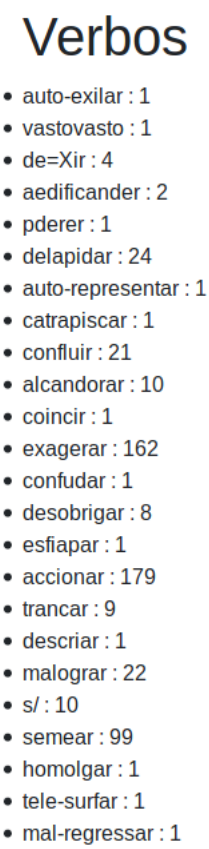

Figura 2.9: Pequena amostra dos Verbos

2.3.4 Alínea d)

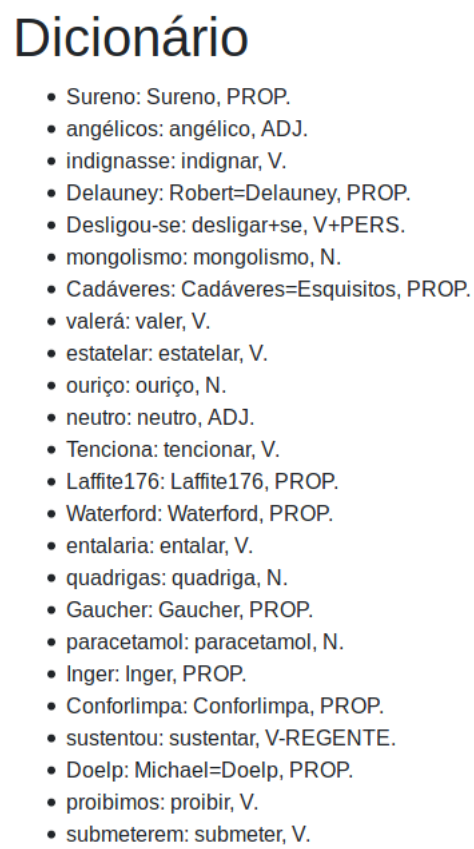

Figura 2.10: Pequena amostra do Dicionário

3. Estrutura HTML

A escrita dos ficheiros HTML é começada pelo programa *esquema1* que coloca no ficheiro *index.html* quatro ancoras. A primeira reporta para o ficheiro *stats.html*, que contem a informação, para responder á alinea a), respetivamente:

- Parágrafos: numero de paragrafos;
- Frases: numero de frases;
- Extratos: numero de extratos;

Da mesma maneira é necessário fazer a referencia para o ficheiro *mwe.html* que contem todas as multi-world-expressions e o respetivo numero de ocorrências. As próximas duas ancoras dizem respeito aos ficheiros que contêm a lista de todos os vérbos e o respetivo numero de ocorrências. Por ultimo uma ligação ao ficheiro *dicionario.html* que contém a lista de palavras e os possíveis (lema,pos).

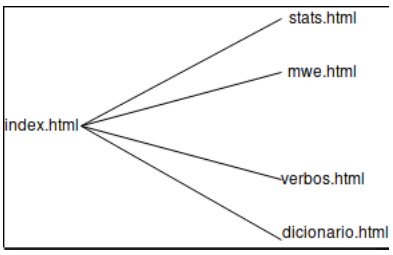

Figura 3.1: Árvore html

4. Desfecho

A realização de um projeto prático numa unidade curricular é sempre benéfica, neste projeto desenvolveram-se expressões regulares para identificar e alterar o texto, ajudando a consolidar toda a matéria lecionada, tão ou mais importante, dá-nos uma melhor perceção do impacto que o que estudamos tem em sistemas mais completos, construindo-se assim um sisstema de produção para filtragem de texto GAWK,para além da experiência adquirida no uso do ambiente Linux. hapterConclusão