Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 2

Relatório de Desenvolvimento

José Pereira
(a82880@alunos.uminho.pt)

Ricardo Petronilho
(a81744@alunos.uminho.pt)

9 de Novembro de 2019

Resumo

O projeto elaborado na Unidade Curricular de Processamento de Linguagens do Mestrado integrado em Engenharia Informática da Universidade do Minho, tem como principal objetivo o processamento de informação contida em ficheiros do tipo Córpora, utilizando para isso a ferramenta *awk*.

Conteúdo

Capítulo 1

Introdução

Na unidade curricular de Processamento de Linguagens, do Mestrado integrado em Engenharia Informática da Universidade do Minho, foi nos proposta a elaboração do exerício 5, que se trata da elaboração de vários scripts que através da aplicação *awk*, processa ficheiros, sendo o principal objetivo filtrar as informações mais importantes.

Deste modo, os ficheiros analisados contém extratos com as respetivas palavras e associado a estas informação de anotação frásica e morfossintática.

Para a elaboração destes scripts, foi necessária uma análise dos vários ficheiros disponibilizados pela equipa docente, que contêm os extratos.

A nível de trabalho adicional, tentou-se em todos os problemas gerar uma página *HTML*, com um formato mais apelativo para leitura, do que no terminal, sendo que estes processos serão especificados mais adiante.

Capítulo 2

Análise e Especificação

2.1 Contextualização do problema

O objetivo central deste trabalho prático é focado no processamento da informação de grandes ficheiros.

Neste caso, tal como já foi referenciado anteriormente o ficheiro para processamento contém extratos e informação de anotação frásica e morfossintática associada a cada palavra contida nestes.

Deste modo, utilizando a aplicação *awk*, definiu-se um script capaz de contar o número de extratos contidos num ficheiro, calcular o número de ocorrências de nomes próprios, criar uma página *HTML* com as listas dos verbos, substantivos, adjetivos e advérbios contidos no ficheiro e por fim criar uma página *HTML* com o dicionário implicíto no córpora.

Capítulo 3

Concepção/desenho da Resolução

3.1 API

Com uma análise geral aos vários problemas, entendeu-se de imediato que existem várias funções que serão utilizadas nos diferentes problemas.

Deste modo decidiu-se criar um ficheiro denominado *api.awk* que inclui as funções da API, das quais: **beginHTML()**, que cria o ficheiro *index.html* e a estrutura inicial desse mesmo ficheiro, **endHTML()**, que faz o inverso, ou seja, estrutura o fim do ficheiro *HTML*, **link()**, gera o código necessária *HTML*, para um *link* e por fim, **createHTMLdir()**, remove caso exista e cria a diretoria **html/**, para guardar os ficheiros necessaários para as páginas *HTML*.

3.2 Problema 1

O primeiro problema pede para contar o número de extratos existentes nos ficheiros fornecidos. Desta froma foi utilizado o padrão **BEGIN** para inicialmente especificar qual o **separador de registo**, neste caso é uma linha em branco ou seja: RS = "\n\n".

No fim imprime-se dentro do padrão **END** o número de extratos percorridos utilizando a variável **NR**.

3.3 Problema 2

Este problema pede para listar os nomes próprios e o número de ocorrências dos mesmos.

Inicialmente definiu-se dentro do padrão **BEGIN** que o **separador de registo** corresponde á mudança de linha, não interessa que seja a sepração entre extratos uma vez que para o contexto deste problema o conteúdo útil é cada palavra e a sua caracterização, assim RS = "\n" e que o **separador de campos** corresponde um espaço, isto é FS = " ".

Definiu-se uma **exrpessão regular** para apenas processar nomes próprios ao longo da iteração do ficheiro, sendo a mesma: $4 ~ /NP.*/, o campo **$4** correspondente à **TAG** da palavra.

Existe um array denominado **counter** que aramazena para cada nome o número de ocorrências atuais do mesmo, desta forma quando se deteta um nome próprio é incrementado o número de ocorrências do mesmo: **counter[tolower($2)]++**, o campo **$2** corresponde ao nome próprio.

É utilizada a função **tolower()** para garantir que não se conta mais que uma vez a mesma palavra, por exemplo as palavras: **"sim"** e **"Sim"** são a mesma no entanto caso não se usa-se a função **tolower()** seriam detetadas como diferentes.

No fim dentro do padrão **END** impreme-se os resultados obtidos numa página **HTML**. Todos os ficheiros **.html** são armazenados numa pasta **html/** sendo a mesma criada através da função **createHTMLdir()**. Para imprimir o **header** e **footer** de cada página foram criadas as funções **beginHTML()** e **endHTML()** respetivamente.

Finalmente falta imprimir a lista dos nomes próprios e o respetivo número de ocorrências à frente, para tal é percorrido o array **counter** com o critério de iteração necessário, sendo o mesmo especificado da seguinte forma: **PROCINFO["sorted_in"] = "comparator"**, sendo a função **comparator()** a que estabelece o critério de ordenação, que neste caso é **descendente** do número de ocorrências.

3.4 Problema 3

O objetivo deste problema é calcular a lista de verbos, substantivos, adjetivos e advérbios e criar uma página *HTML* com esta informação.

Deste modo, começou-se por definir os campos RS = "\n" e FS = " " no padrão **BEGIN**, pois, as palavras no extrato estão separadas por um "\n" e a informação de cada palavra por um espaço.

De seguida, determinou-se o campo necessário para a identificação do tipo de palavra (verbo, substativo, adjetivo e advérbio), que neste caso é o campo quatro ($4), ou seja, *pos-tag*.

Assim, detetou-se que as tags começam por uma determinada letra, que identificam o tipo de palavra (V=verbo, N=substantivo, A=adjetivo e R=advérbio), ou seja, os padrões usados são:

- verbos: $4~/V.*/
- substantivos: $4~/N.*/
- adjetivos: $4~/A.*/
- advérbios: $4~/R.*/

De seguida, definiu-se o array de duas dimensões denominado por *words*, que é responsável por guardar cada tipo de palavra e o respetivo número de ocorrências, isto é, as linhas correspondem ao tipo de palavra (verbo, substantivo, adjetivo e advérbio), e as colunas correspondem às palavras desse tipo, guardando o número de ocorrências das mesmas.

"verbos"	"ser" 1170	"ter" 1106	... ...
"substantivos"	"harry" 1430	"que" 1369	... ...
"adjetivos"	"grande" 114	"enorme" 96	... ...
"adverbios"	"que" 1353	"mais" 337	... ...

Tabela representativa do array *words*.

Atente-se que o motivo de se armazenar o lema e não a palavra é para que, por exemplo as palavras **"dissestе"** e **"disse"** sejam contabilizadas como o mesmo verbo **"dizer"**.

Deste modo, aos padrões referidos anteriormente, associa-se o array do respetivo tipo de palavra, ou seja, por exemplo, tal como se pode verificar no código abaixo, para o padrão que captura verbos, vai-se ao array que está na linha correspondente a "verbos"e à coluna correspondente ao lema da palavra encontrada ($3) e incrementa-se uma unidade, que no final será usada para fins estatísticos.

A razão pela qual se usa o *tolower*, tal como já foi referido anteriormente, é para não haver repetições da mesma palavra, nos casos onde a diferença é na letra inicial ser maíuscula ou minúscula.

```
$4~/V.*/{
    words["verbos"][tolower($3)]++
}
```

No padrão **END**, começa-se por chamar a função **createHTMLdir()**, que é responsável por criar a diretoria, onde são guardados todos os ficheiros necessários para a página *HTML*, tal como já foi referênciado no problema anterior.

De seguida, chama-se a função principal deste script, denominada por **listing()**, que é responsável por listar os diferentes tipos de palavras em diferentes páginas, onde, as palavras são ordenadas pelo seu número de ocorrências. Importante referir, que logo, no ínicio desta função, é chamada para cada ficheiro a função **header()**, responsável por criar os ficheiros **.html** para cada lista e estruturar o código *HTML* inicial.

Como a informação foi guardada num array bi-dimensional, torna-se fácil o processo de listar as palavras por tipos em diferentes ficheiros, pois itera-se o mesmo, por linha, escrevendo os dados para o ficheiro específico do tipo de palavra (tipoDePalavra.html), onde o nome do mesmo corresponde à linha do array. Ainda nesta função, faz-se a análise estatística, contando-se o número de ocorrências de palavras de cada tipo sendo estes valores guardados em variáveis para depois serem apresentados.

Importante referir, que as palavras nos ficheiros são ordenadas pelo seu número de ocorrências, isto consegue-se, tal como já foi referenciado anteriormente, com a adição da linha **PROCINFO[“sorted_in“] = “comparator“** e definição da função **comparator** que é utilizada para o critério de ordenação.

Por fim, são chamadas funções responsáveis pela estrutuação dos ficheiros *HTML*, das quais, **meta()**, que é responsável por criar a página *index.html*, com *links* a redirecionar para as páginas com as listas e a apresentação dos dados estatísticos

3.5 Problema 4

O propósito deste programa é determinar e apresentar o dicionário implícito nos ficheiros fornecidos.

Incialmente definiu-se dentro do padrão BEGIN os valores de RS = "\n" e FS = " ' uma vez que cada linha é considreada um registo e os campos dentro da linha são sperados por um espaço.

De seguida definiu-se as 8 **expressões regulares** relvantes para determinar o dicionário:

- verbos: $4~/V.*/
- nomes: $4~/N.*/
- adjetivos: $4~/A.*/
- advérbios: $4~/R.*/
- preposições: $4~/S.*/
- determinantes: $4~/D.*/
- conjunções: $4~/C.*/
- pronomes: $4~/P.*/

Sendo que o campo **$4** corresponde à palavra detetada. Note-se que não se processou por exemplo os sinais de pontuação apesar de nos ficherios fornecidos serem descritos, uma vez que não é um elemento relevante para a descrição num dicionário.

Posterioremente, definiu-se uma matriz denominada por **dic**, nome alusivo a **dicionário**, que para cada **lema** e **palavra** aramazena a respetiva **TAG** que descreve efetivamente a palavra. A seguinte figura representa gráficamete a informação da matriz **dic**.

"ser"	"era" VMII1S0	"seja" VMSP1S0	"sou" VMIP1S0	... ...
"ter"	"tendo" VMG0000	"terei" VMIF1S0	"tivera" VMIM1S0	... ...
"..."	"..." ...	"..." ...	"..." ...	... ...

Tabela representativa da matriz **dic**.

Segundo a informação da matriz em cima, por exemplo, a palavra **"era"** tem a **TAG=VMII1S0**, por isso é um verbo **(VM)** no modo indicativo **(I)** no pretérito imperfeito **(I)** na 1ª **(1)** pessoa no singular **(S)**. No momento de armazenamento tanto do lema como das palavras, utilizou-se a função **tolower()** para não contabilizar palavras repetidas.

Dentro do padrão **END**, evoca-se a função **createHTMLdir()**, criando a diretoria, onde são guardados todos os ficheiros necessários para a página *HTML*.

A variável **PROCINFO["sorted_in"] = "comparator"** garante que a iteração sobre a matriz **dic** é feita por ordem alfabética.

Para a apresentação do dicionário utilizou-se a convenção que todos os dicionários físicos ou virtuais usam: existe uma página *HTML* para cada letra do alfabeto fonético internacional (de A a Z) assim, por exemplo os **lemas**: **"comer"** e **"começar"** estão dentro da página *HTML* **C**. Contudo as palavras derivadas desses mesmos lemas estão dentro das respetivas páginas *HTML* de cada lema ou seja, por exemplo as palavras **"comendo"** e **"comeu"** estão dentro da página *HTML* **"comer"**. A implementação deste requesito foi facilitada uma vez que **cada linha da matriz contém apenas palavras do mesmo lema**, assim para aceder, por exemplo a todas as palavras derivadas do lema **"começar"** basta evocar: **dic["começar"]**.

Observação:
Uma imprecisão detetada ao analisar os ficheiros fornecidos foi que certas palavras são associadas ao lema incorreto, por exemplo a palavra **"é"** por vezes vem associada ao lema **"é"**, o correto é ser associada ao lema **"ser"**, e também devia ser considerada um verbo, no entanto por vezes é considrada um nome próprio.

Figura 3.1: Palavra **"é"** associada a um lema incorreto.

Esta imprecisão compromete a implementação referida anteriormente para a apresentação do dicionário uma vez que **não deveria existir um lema começado com uma letra com acentos**. Para corrigir esta imprecisão e incluir estas palavras no dicionário, sempre que um lema começar com letra com acentos, é adicionado à respetiva página *HTML* sem acento. No exemplo anterior a palavra **"é"** será acrescentada à página *HTML* **E**. Para implementar esta correção foram definidas 5 condições que quando detetam vogais com acentos, indica a respetiva letra sem acento.

```
# garante que por exemplo a palavra "à" ou "á" fica na página HTML "A"
if (     c~/Á/ || c~/À/ || c~/Â/ || c~/Ã/) c = "A"
else if (c~/É/ || c~/È/ || c~/Ê/ )         c = "E"
else if (c~/Í/ || c~/Ì/ || c~/Î/ )         c = "I"
else if (c~/Ó/ || c~/Ò/ || c~/Ô/ || c~/Õ/) c = "O"
else if (c~/Ú/ || c~/Ù/ || c~/Û/ )         c = "U"
```

Figura 3.2: 5 condições para implementar a correção.

Corrigido o problema, finalmente foi implementado o **formato de apresentação** de cada palavra contida na página do respetivo lema: é apresentada a palavra, seguida da TAG e da **descrição extensa** da mesma. A seguinte figura mostra a apresentação da página do lema **"ser"**.

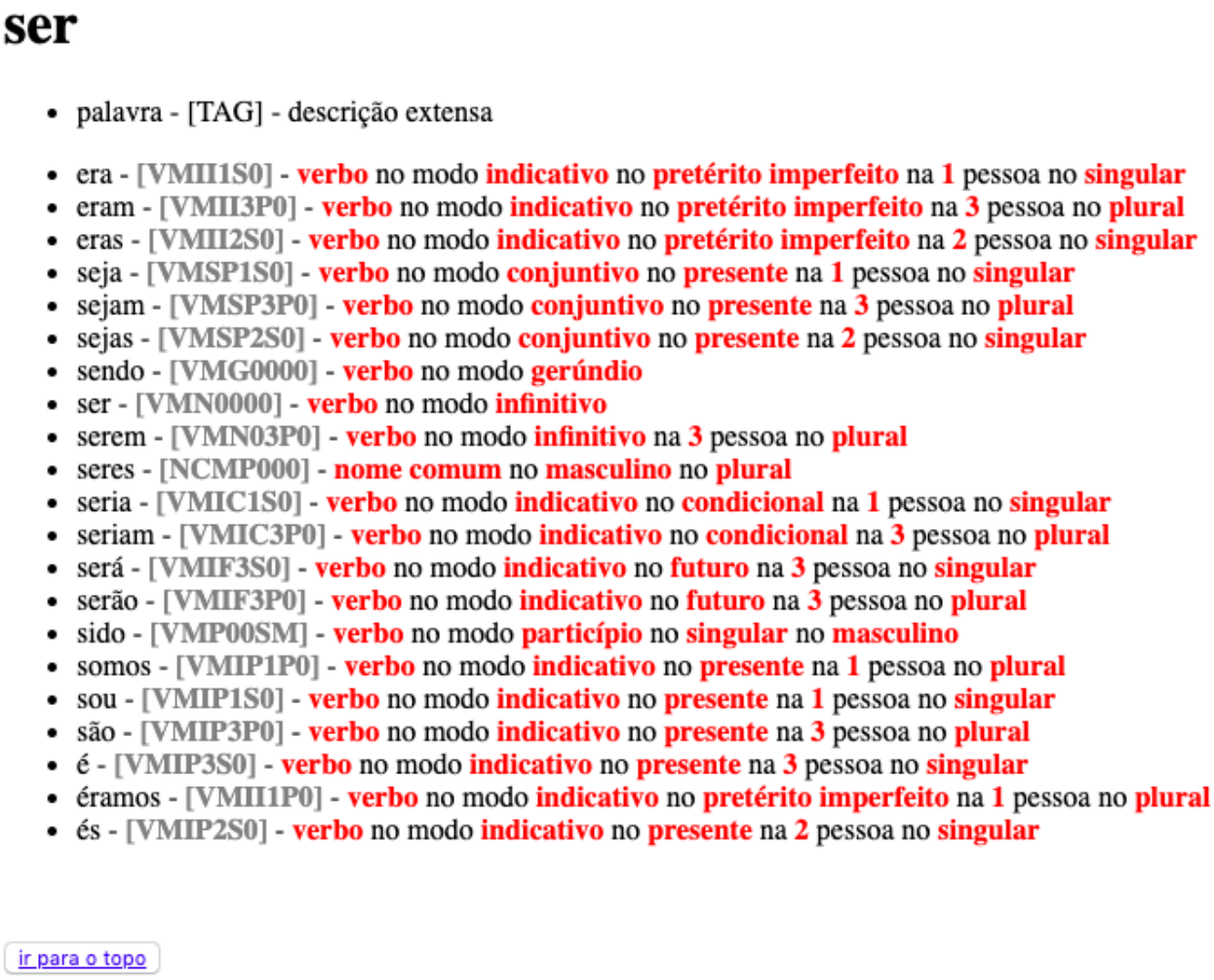

Figura 3.3: página *HTML* do lema **"ser"**.

Note-se que **apenas foi implementada a descrição extensa completa de cada TAG para os verbos e nomes**, para os restantes elementos apenas se indica o tipo de elemento, isto é por exemplo para o adjetivo **"mortal"** apenas se indica que é um adjetivo, em vez de informar completamente, por extenso, que é um adjetivo qualificativo para os dois géneros (masculino e feminino) no singular. A seguinte figura evidencia esta descrição incompleta propositada.

Figura 3.4: descrição incompleta propositada.

Atente-se que no entanto a **TAG é sempre apresentada** uma vez que é um requisito obrigatório do problema 4. O motivo de não ter sido desenvolvido a descrição extensa para todos os elementos é que apenas querimos evendicar uma prova de conceito visto que é um extra.

Capítulo 4

Utilização, Codificação e Testes dos Scritps

O grupo definiu uma *Makefile*, que corre os quatro scripts elaborados para cada problema com a utilização do **awk**. Visto que se trata de scritps processados pelo *gawk*, não foi necessário definir a instalação da aplicação.

Assim para executar cada problema, basta fazer *make* **(p1,p2,p3,p4)** para cada respetivo problema, sendo que o ficheiro utilizado como padrão na *Makefile* é o *harrypotter2*, caso se queira proceder ao processamento de outro ficheiro faz-se gawk -f (p1,p2,p3,p4) ficheiroInput.

Os testes realizados para verficação da resolução de cada problema foram por observações dos resultados em páginas *HTML* e também através de exemplos mais simples gerados pela equipa.

Capítulo 5

Conclusão

Em suma, os objetivos inicialmente propostos foram cumpridos, dos quais, a contagem do número de extratos, nomes próprios, listagem dos verbos, substantivos, adjetivos e advérbios e por fim a elaboração do dicionário implicíto no córpora.

A análise exaustiva dos ficheiros para encontrar os padrões necessários para os campos *FS* e *RS*, foi importante para se conseguir capturar a informação necessária.

A nível de trabalho adicional, no problema dois, fez-se a geração do output para uma página *HTML* o problema três, procedeu-se à contagem das ocorrências de cada palavra, ordenando-se as mesmas pelo critério do número de ocorrências, e fez-se o cálculo de alguns dados estatísticos. Por fim, e não menos importante, o problema quatro apresentou o dicionário implícito dos ficheiros fornecidos tal como a lista de palavras derivadas de cada lema, apresentando as mesmas em *HTML* com formato de apresentação equivalente ao dicionário comum. *HTML*

Apêndice A

Código do Programa

Lista-se a seguir o CÓDIGO do API criada pelo grupo:

```
function beginHTML(t){
return "<html><head><meta charset='UTF-8'/></head><body><h1>" t "</h1>"
}


function endHTML(){
return "</body></html>"
}


function listItem(text){
return "<li>" text "</li><br>"
}


function link(href, text){
return "<li><a href=" href ">" text "</a></li>"
}


function createHTMLdir() {
system("rm -f -r html/")
system("mkdir html/")
}
```

Lista-se a seguir o CÓDIGO do programa do problema 1:

```
#!/usr/local/bin/gawk -f

BEGIN {
RS = "\n\n"
}

END {
print "Número de extratos => " NR
}
```

Lista-se a seguir o CÓDIGO do programa do problema 2:

```
#!/usr/local/bin/gawk -f

@include "api.awk"

BEGIN {
RS = "\n"
FS = " "
}


$4~/NP.*/ {
counter[$2]++
}


END {
createHTMLdir()
print beginHTML("Contador de Ocorrências de Nomes Próprios") > "html/index.html"
print "<ul>" > "html/index.html"

PROCINFO["sorted_in"] = "comparator"
for(nome in counter) {
c = counter[nome]
print listItem(nome " -> " c) > "html/index.html"
}

print "</ul>" > "html/index.html"
print endHTML() > "html/index.html"
}


function comparator(i1, v1, i2, v2)
{
    if (v1 == v2) return 0;
    else if (v1 > v2) return -1;
    else return 1;
}
```

Lista-se a seguir o CÓDIGO do programa do problema 3:

```
#!/usr/local/bin

@include "api.awk"

BEGIN{
RS = "\n"
FS = " "
}

$4~/V.*/{ # VERBO
words["verbos"][tolower($3)]++
}

$4~/N.*/{ # SUBSTANTIVO
words["substantivos"][tolower($3)]++
}

$4~/A.*/{ # ADJETIVO
words["adjetivos"][tolower($3)]++
}

$4~/R.*/{ # ADVÉRBIO
words["adverbios"][tolower($3)]++
}

END{
createHTMLdir() # remove e cria a pasta html/
listing()
meta()
footer()
}

# usado para ordenar o array pelo número de ocorrências de cada palavra
function comparator(i1, v1, i2, v2){
    if (v1 == v2) return 0;
    else if (v1 > v2) return -1;
    else return 1;
}

# lista as palavras de cada tipo em cada ficheiro
function listing(){
header("Verbos", "verbos.html")
header("Substantivos", "substantivos.html")
header("Adjetivos", "adjetivos.html")
header("Advérbios", "adverbios.html")
```

```
for(tipo in words){
PROCINFO["sorted_in"] = "comparator"
for(palavra in words[tipo]){
if(tipo~"verbos"){
tiposVerbos++
verbos = verbos + words[tipo][palavra]
}
if(tipo~"substantivos"){
tiposSubstantivos++
substantivos = substantivos + words[tipo][palavra]
}
if(tipo~"adjetivos"){
tiposAdjetivos++
adjetivos = adjetivos + words[tipo][palavra]
}
if(tipo~"adverbios"){
tiposAdverbios++
adverbios = adverbios + words[tipo][palavra]
}

print "<li>" > "html/" tipo ".html"
print palavra " -----> " words[tipo][palavra] > "html/" tipo ".html"
print "</li>" > "html/" tipo ".html"
}
}
}

# cria e estrutura o index.html
function meta(){
print beginHTML("Listas") > "html/index.html"
linkListas()
estatistica()
}

# faz um header especial para os ficheiros que vão ter as palavras
function header(title, filename){
print specialbeginHTML("Listas", "index.html", filename1,title) > "html/" filename
print "<h3 id="filename ">" title " -----> Nº de ocorrências</h2>" > "html/" filename
print "<ul>" > "html/" filename
}

# finaliza todos os ficheiros
function footer(){
finally("index.html",0)
finally("verbos.html",1)
finally("substantivos.html",1)
finally("adjetivos.html",1)
finally("adverbios.html",1)
```

```
}

# finaliza um ficheiro
function finally(filename, flag){
print "</ul>" > "html/" filename
if(flag == 1)print "<button type="button"><a href=#" filename ">IR PARA O TOPO!!!</a></button><br
print endHTML() > "html/" filename
}

function specialbeginHTML(t1,filename1, filename2, t2){
return "<html><head><meta charset='UTF-8'/></head><body><h1><a href=" filename1 ">" t1 "</a><a hr
}

function estatistica(){
print "<br><h2>Estatística:</h2>" > "html/index.html"
print "<ul><h4>" > "html/index.html"
print "<li>Número de tipos de verbos = " tiposVerbos "</li>"  > "html/index.html"
print "<li>Número de tipos de substantivos = " tiposSubstantivos "</li>"  > "html/index.html"
print "<li>Número de tipos de adjetivos = " tiposAdjetivos "</li>"  > "html/index.html"
print "<li>Número de tipos de advérbios = " tiposAdverbios "</li>"  > "html/index.html"
print "<li>Total de tipo de palavras = " tiposVerbos + tiposSubstantivos + tiposAdjetivos + tipos
print "<br>" > "html/index.html"
print "<li>Número de ocorrências de verbos = " verbos "</li>"  > "html/index.html"
print "<li>Número de ocorrências de substantivos = " substantivos "</li>"  > "html/index.html"
print "<li>Número de ocorrências de adjetivos = " adjetivos "</li>"  > "html/index.html"
print "<li>Número de ocorrências de advérbios = " adverbios "</li>"  > "html/index.html"
print "<li>Total de palavras = " verbos + substantivos + adjetivos + adverbios "</li>"  > "html/i

print "</h4></ul>" > "html/index.html"
}

function linkListas(){
print "<ul><h3>" > "html/index.html"
print link("verbos.html", "Verbos") > "html/index.html"
print link("substantivos.html", "Substantivos") > "html/index.html"
print link("adjetivos.html", "Adjetivos") > "html/index.html"
print link("adverbios.html", "Advérbios") > "html/index.html"
print "</h3></ul>" > "html/index.html"
}
```

Lista-se a seguir o CÓDIGO do programa do problema 4:

```
#!/usr/local/bin/gawk -f

@include "api.awk"


BEGIN {
RS = "\n"
FS = " "
}

#  VERBO     NOME     ADJETIVO    ADVERBIO   PREPOSIÇÂO  DETERMINANTE  CONJUNÇÂO    PRONOME
$4~/V.*/ || $4~/N.*/ || $4~/A.*/ || $4~/R.*/ || $4~/S.*/ || $4~/D.*/ || $4~/C.*/ || $4~/P.*/ {

#      LEMA         WORD       TAG
dic[tolower($3)][tolower($2)] = $4
}


END {
createHTMLdir()
print beginHTML("DICIONÁRIO") > "html/index.html"

# percorre a matriz com ordenação alfabética
PROCINFO["sorted_in"] = "comparator"

# imprimir o link no index.html para cada letra
# e imprimir o header em cada página {A,B,C, ...}
for(i=65; i<=90; i++) {
href = sprintf("_%c_.html", i)
text = sprintf("%c", i)
print link(href, text) > "html/index.html"
print beginHTML(text) > "html/" href

# para guardar o topo da página
print "<p id=top></p>" > "html/" href
}

for (lema in dic) {

split(lema, str, "")
c = toupper(str[1])

# garante que por exemplo a palavra "à" ou "á" fica na página HTML "A"
if (     c~/Á/ || c~/À/ || c~/Â/ || c~/Ã/) c = "A"
else if (c~/É/ || c~/È/ || c~/Ê/ )         c = "E"
else if (c~/Í/ || c~/Ì/ || c~/Î/ )         c = "I"
```

```
else if (c~/Ó/ || c~/Ò/ || c~/Ô/ || c~/Õ/) c = "O"
else if (c~/Ú/ || c~/Ù/ || c~/Û/ )         c = "U"

# imprime o link para o lema no index.html
print link(lema ".html", lema) > "html/_" c "_.html"

# imprimir a página de cada lema
printLemaHTML()
    }

    # imprimir o footer em cada página {A,B,C, ...}
    # e imprimir o botão (ir para o topo)
    for(i=65; i<=90; i++) {
href = sprintf("_%c_.html", i)
print "<br><br><button type=button><a href=#" top ">ir para o topo</a></button><br><br>" > "html/
print endHTML() > "html/" href
}

printEstatistica()
print endHTML() > "html/index.html"
}


function printEstatistica(){
print "<br><h2>Estatística:</h2>" > "html/index.html"
print "<ul><h4>" > "html/index.html"
print "<li>nº nomes -> " n_nomes "<br></li>" > "html/index.html"
print "<li>nº adjetivos -> " n_adjetivos "<br></li>" > "html/index.html"
print "<li>nº adverbios -> " n_adverbios "<br></li>" > "html/index.html"
print "<li>nº preposicoes -> " n_preposicoes "<br></li>" > "html/index.html"
print "<li>nº determinantes -> " n_determinantes "<br></li>" > "html/index.html"
print "<li>nº conjuncoes -> " n_conjuncoes "<br></li>" > "html/index.html"
print "<li>nº pronomes -> " n_pronomes "<br></li>" > "html/index.html"
total = n_verbos + n_nomes + n_adjetivos + n_adverbios + n_preposicoes + n_determinantes + n_con
print "<li>total de palavras -> " total "<br><br></li>" > "html/index.html"
print "</h4></ul>" > "html/index.html"
}


function printLemaHTML() {
path = "html/" lema ".html" # caminho para a página do lema

print beginHTML(lema) > path
print "<ul><li>palavra - [TAG] - descrição extensa</li></ul>" > path

# para guardar o topo da página
print "<p id=top></p>" > "html/" href
```

```
print "<ul>" > path
for(word in dic[lema]) {
tag = dic[lema][word]
split(tag, str, "")

print "<li>" word " - " > path

# imprimir a TAG
print "<b style=color:gray;>[" tag "]</b> - " > path

if (tag~/V.*/) {
printVerbo()
n_verbos++
}
else if (tag~/N.*/) {
printNome()
n_nomes++
}
else if (tag~/A.*/) {
printAdjetivo()
n_adjetivos++
}
else if (tag~/R.*/) {
printAdverbio()
n_adverbios++
}
else if (tag~/S.*/) {
printPreposicao()
n_preposicoes++
}
else if (tag~/D.*/) {
printDeterminante()
n_determinantes++
}
else if (tag~/C.*/) {
printConjuncao()
n_conjuncoes++
}
else if (tag~/P.*/) {
printPronome()
n_pronomes++
}
print "</li>" > path
}
print "</ul>" > path

# imprime o botão para ir para o TOPO
print "<br><br><button type=button><a href=#" top ">ir para o topo</a></button><br><br>" > path
```

```
print endHTML() > path
}


function printPronome() {
print "<b style=color:red;>pronome</b>" > path
}


function printConjuncao() {
print "<b style=color:red;>conjuncão</b>" > path
}


function printDeterminante() {
print "<b style=color:red;>determinante</b>" > path
}


function printPreposicao() {
print "<b style=color:red;>preposição</b>" > path
}


function printAdverbio() {
print "<b style=color:red;>advérbio</b>" > path
}


function printAdjetivo() {

# 1 A -> nome
# ------------------------
# 2 t -> tipo
# 3 gen -> género
# 4 num -> cardinalidade

print "<b style=color:red;>adjetivo</b>" > path
}


function printNome() {

# 1 N -> nome
# ------------------------
# 2 t -> tipo
# 3 gen -> género
```

```
# 4 num -> cardinalidade

print "<b style=color:red;>nome</b>" > path

t = str[2]; # tipo -> tem sempre
if (t != "0") {
print "<b style=color:red;>" > path
if (t == "P") print " próprio" > path
else if (t == "C") print " comum" > path
else print "___AINDA_SEM_TIPO___" > path
print "</b>" > path
}

printGenero(str[3])
printCardinalidade(str[4])
}


function printVerbo() {

# 1 V -> verbo
# 2 M -> main
# ----------------------
# 3 m -> modo
# 4 t -> tipo
# 5 p -> pessoa
# 6 num -> cardinalidade
# 7 gen -> género

print "<b style=color:red;>verbo</b>" > path

m = str[3] # modo -> tem sempre
if (m != "0") {
print " no modo <b style=color:red;>" > path
if (m == "I") print "indicativo" > path
else if (m == "S") print "conjuntivo" > path
else if (m == "G") print "gerúndio" > path
else if (m == "N") print "infinitivo" > path
else if (m == "P") print "particípio" > path
else print "___AINDA_SEM_MODO___" > path
print "</b>" > path
}

t = str[4]; # tipo -> pode não não ter (ex: gerúndio ou infinitivo)
if (t != "0") {
print " no <b style=color:red;>" > path
if (t == "P") print "presente" > path
else if (t == "S") print "pretérito perfeito" > path
```

```
else if (t == "I") print "pretérito imperfeito" > path
else if (t == "M") print "pretérito mais que perfeito" > path
else if (t == "C") print "condicional" > path
else if (t == "F") print "futuro" > path
else print "___AINDA_SEM_TEMPO___" > path
print "</b>" > path
}

p = str[5] # pessoa -> pode não ter
if (p != "0") print " na <b style=color:red;>" p "</b> pessoa" > path

printCardinalidade(str[6])
printGenero(str[7])
}


function printCardinalidade(num) {

if (num != "0") {
print " no <b style=color:red;>" > path
if (num == "S") print "singular" > path
else print "plural" > path
print "</b>" > path
}
}


function printGenero(gen) {
if (gen != 0) {
print " no" > path
print "<b style=color:red;>" > path
if (gen == "F") print "feminino" > path
else print "masculino" > path
print "</b>" > path
}
}


function comparator(i1, v1, i2, v2) { # ordena por ordem alfabética

word1 = i1
word2 = i2

if (word1 == word2) return 0;
    else if (word1 > word2) return 1;
    else return -1;
}
```