

Universidade do Minho - Escola de Engenharia

Mestrado Integrado em Engenharia Informática

Processamento de Linguagens e Compiladores

2017/2018 - 2º semestre

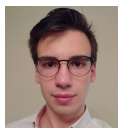
Trabalho Prático nº 1 - GAWK

Autores :

Diana Costa (A78985)



Marcos Pereira (A79116)



Vitor Castro (A77870)



Braga, 25 de Março de 2018

Resumo

Perante a proposta de realizar um exercício sobre expressões regulares e processadores de linguagens regulares utilizando GAWK, houve um impasse inicial, devido à necessidade de uma boa estruturação dos problemas. Tudo isto requis a escolha acertada de expressões regulares e uma análise aprofundada de extratos de texto (*datasets*), de forma a que a resolução destes fosse a mais clara e simples possível. O grupo acabou por realizar mais um exercício do enunciado, juntamente com um extra que considerou pertinente, sobre previsão de palavras, com recurso a HTML.

Depois de algum tempo e trabalho, o resultado encontrado foi satisfatório, e os objetivos e respostas às questões do enunciado proposto cumpridos.

Conteúdo

| | | |
|----------|---|-----------|
| 1 | Introdução | 3 |
| 2 | Preliminares | 4 |
| 3 | Processador de CETEMPúblico | 5 |
| 3.1 | Análise dos extratos | 5 |
| 3.2 | Filtro de Texto - Sistema de Produção GAWK | 6 |
| 3.2.1 | Contar o número de Extratos, Parágrafos e Frases | 6 |
| 3.2.2 | Extrair a lista das <i>multi-word-expressions</i> e respetivo número de ocorrências | 7 |
| 3.2.3 | Calcule a lista dos verbos PT: (Lema, para palavras com pos=V) e respetivo número de ocorrências | 8 |
| 3.2.4 | Determinar o dicionário implícito no corpora - calcule a lista das palavras associando-lhes os possíveis (lema,pos) | 9 |
| 3.3 | Análise de Resultados | 11 |
| 4 | Processador de textos preanotados com <i>Freeling</i> | 13 |
| 4.1 | Análise dos extratos | 13 |
| 4.2 | Filtro de Texto - Sistema de Produção GAWK | 14 |
| 4.2.1 | Contar número de extratos | 14 |
| 4.2.2 | Calcular a lista dos personagens do Harry Potter (nomes próprios) e respetivo número de ocorrências | 15 |
| 4.2.3 | Calcular a lista dos verbos, substantivos, adjetivos e advérbios PT; e criar um ficheiro com cada uma destas listas. | 17 |
| 4.2.4 | Determinar o dicionário implícito no corpora - lista contendo os lema, pos e palavras dele derivadas | 18 |
| 4.3 | Análise de Resultados | 20 |
| 5 | Exercício extra - Previsão de palavras | 25 |
| 5.1 | Análise de Resultados | 27 |
| 6 | Conclusões e Sugestões | 28 |

1 Introdução

O objetivo deste trabalho prende-se com o aumento da experiência em ambiente *Linux*, em Expressões Regulares e utilização do *GAWK* para filtragem de texto. De entre os enunciados disponibilizados o grupo ficou com a tarefa de desenvolver o número 1, - **Processador de CE-TEMPúblico**. Não satisfeitos apenas com a resolução deste ponto, decidiu-se desenvolver também o enunciado 2 - **Processador de textos preanotados com Freeling**, bem como realizar um ponto extra, e elaborar um programa de previsão de palavras.

Ao nível dos enunciados número 1 e 2, era requerido, no geral, que se procedesse à contagem de extratos, parágrafos ou frases, se extraíssem listas com palavras com determinadas classes e se indicasse a frequência das mesmas, e se determinassem os dicionários implícitos. A grande diferença entre estes dois exercícios era apenas o formato dos extratos de texto, que fazia com que o conteúdo das colunas e/ou linhas variasse.

Já no ponto extra, foi implementado um programa de previsão de palavras, testado num dos extratos disponibilizados pela equipa docente, e, no fim, elaborado em HTML um gráfico circular descritivo da situação e que serviria de teste do exercício.

Em suma, a Secção 2 descreve os preliminares necessários ao projeto, enquanto que a Secção 3 descreve a resolução do primeiro enunciado do trabalho, desde a análise dos extratos(3.1), soluções(3.2.1 a 3.2.4), e análise consequente de resultados(3.3). Na Secção(4) a estrutura é exatamente a mesma, mas para o segundo enunciado. Por fim, encontra-se o exercício extra, em (5) e uma análise de resultados, na Secção(5.1). O relatório termina com uma breve conclusão na Secção 6, onde é feito um balanço do trabalho realizado, tendo em conta as dificuldades ao longo do desenvolvimento do mesmo.

2 Preliminares

Para o desenvolvimento deste exercício prático não foi necessário o estudo de conteúdos adicionais aos lecionados nas aulas da unidade curricular.

A leitura e compreensão deste relatório são facilitadas através de conhecimentos de programação em AWK e de expressões regulares, necessárias à resolução dos tópicos requisitados.

3 Processador de CETEMPúblico

3.1 Análise dos extratos

Para este exercício, a anotação frásica é feita através das seguintes tags XML:

- **<t>** - títulos;
- **<p>** - parágrafos;
- **<s>** - frases;
- **<mwe>** - multi-word-expressions.

A anotação morfossintática é extensa mas, da totalidade da informação que se pode extrair, importa especialmente a denotada por:

- **1ª coluna:** Palavra - a palavra ou pontuação original, presente no texto;
- **2ª coluna:** Secção
- **3ª coluna:** Semestre
- **4ª coluna:** Lema - base da palavra/pontuação
- **5ª coluna:** pos
- **6ª coluna:** tempoVerbal-modo
- **7ª coluna:** num-pessoa
- **8ª coluna:** Género

O **FS** a utilizar será o separador por defeito ([\t]+), que usa como separador qualquer combinação de um ou mais espaços ou tabs.

Para efeitos de simplicidade, assumimos que o formato dos dados recebidos está correto e não precisa de ser verificado.

```
1 <ext n=1668 sec=pol sem=92a>
2 <p par=ext1668-pol-92a-1>
3 <s>
4 <mwe pos=ADV>
5 Ontem pol 92a ontem ADV 0 0 0 ADVL> 0 0 0
6 de pol 92a de PRP 0 0 0 N< 0 0 0
7 manhã pol 92a manhã N 0 S F P< 0 0 0
8 </mwe>
9 , pol 92a , PU 0 0 0 PONT 0 0 0
10 reuniram pol 92a reunir V PS/MQP_IND 3P 0 FMV 0 0 0
11 na pol 92a em+o PRP+DET_artd 0 S F <ADVL+>N 0 0 0
12 sede pol 92a sede N 0 S F P< 0 0 0
13 do pol 92a de+o PRP+DET_artd 0 S M N<+>N 0 0 0
14 MDP pol 92a MDP PROP 0 S M P< 0 0 0
15 , pol 92a , PU 0 0 0 PONT 0 0 0
16 na pol 92a em+o PRP+DET_artd 0 S F <ADVL+>N 0 0 0
17 Damaia pol 92a Damaia PROP 0 S F P< 0 0 0
18 , pol 92a , PU 0 0 0 PONT 0 0 0
19 as pol 92a o DET_artd 0 P F >N 0 0 0
20 comissões pol 92a comissão N 0 P F <ACC 0 0 0
21 de pol 92a de PRP 0 0 0 N< 0 0 0
22 redacção pol 92a redacção N 0 S F P< 0 0 0
23 e pol 92a e KC 0 0 0 CD 0 0 0
24 de pol 92a de PRP 0 0 0 N< 0 0 0
25 organização pol 92a organização N 0 S F P< 0 0 0
26 do pol 92a de+o PRP+DET_artd 0 S M N<+>N 0 0 0
27 movimento pol 92a movimento N 0 S M P< 0 0 0
28 . pol 92a . PU 0 0 0 PONT 0 0 0
29 </s>
```

Figura 1: Extrato do enunciado 1

3.2 Filtro de Texto - Sistema de Produção GAWK

3.2.1 Contar o número de Extratos, Parágrafos e Frases

```
# Contar número de extratos, parágrafos, e frases.

BEGIN {
    # FS=" "
    # By default, FS is set to a single space character, which awk interprets to mean "one or more spaces or tabs."
    ext=0
    p=0
    s=0
}

/<\ext>/ {
    ext++
}

/<\p>/ {
    p++
}

/<\s>/ {
    s++
}

END {
    print "Extratos (<ext>): " ext
    print "Paragrafos (<p>): " p
    print "Frases (<s>): " s
}
```

Figura 2: Código ponto 1

Para este ponto, entre os blocos *BEGIN* e *END* foram usados três blocos, com um padrão de pesquisa cada um, para incrementar os contadores.

- `/<\ext>/` encontra qualquer linha que contenha `</ext>`
- `/<\p>/` encontra qualquer linha que contenha `</p>`
- `/<\s>/` encontra qualquer linha que contenha `</s>`

Sendo os contadores de cada um destes elementos incrementados quando encontramos a tag que indica o fim do elemento, e sabendo que cada um destes elementos tem de ser fechado em algum ponto e apenas uma vez, acabamos por conseguir contar o número de ocorrências de cada um de uma maneira simples.

3.2.2 Extrair a lista das *multi-word-expressions* e respetivo número de ocorrências

```
BEGIN {
  # FS=" "
  # By default, FS is set to a single space character, which awk interprets to mean "one or more spaces or tabs."

  capture=0 # Equal to 1 when we should be capturing a multi word expression

  justfinished=0 # Equal to 1 when we just finished capturing a multi word expression,
  # and should increment its counter

  mwe="" # Will hold the captured multi word expression
}

# Catch all </mwe> and end capture
/(</mwe>)/ {
  capture=0
  justfinished=1
}

# Capture is on, add current word to mwe
capture == 1 {
  mwe = mwe " " tolower($1)
}

# Just finished catching a multi word expression (reached </mwe>),
# increment the counter and reset mwe
justfinished == 1 {
  justfinished=0
  array[mwe]++
  mwe=""
}

# Catch all <mwe [...]> tags and turn on capture flag
# This is the last block so we only start capturing on
# the next record
/(<mwe.*>)/ {
  capture=1
}

END {
  for (i in array) {
    print i "," array[i]
  }
}
```

Figura 3: Código ponto 2

No bloco *BEGIN*, começa-se por inicializar algumas variáveis:

- **capture** é inicializada a zero, sendo uma flag que é ativada (com valor igual a 1) quando estamos se passa por uma multi word expression, e por consequência deve-se guardar a palavra da linha atual.
- **justfinished** é inicializada a zero, sendo também uma flag que é ativada apenas durante uma linha, quando se chega ao fim de uma multi word expression. Esta flag indica que se deve guardar a multi word expression que se acumulou até ao momento.
- **mwe** é inicializada com uma string vazia, sendo esta variável onde será acumulada cada multi word expression, à medida que esta é atravessada.

De seguida, entre os blocos *BEGIN* e *END* tem-se quatro blocos, sendo cada um deles bastante simples:

O primeiro bloco deteta o fim de um bloco de multi word expression (*</mwe>*), e quando é ativado desliga a flag *capture* e liga a flag *justfinished*.

O segundo bloco é ativado quando a flag *capture* está ligada, e concatena a string *mwe* acumulada até ao momento com a palavra da linha atual. O uso de *tolower()* assegura que não são mantidos dois contadores para a mesma multi word expression com capitalização diferente.

O terceiro bloco é ativado quando a flag *justfinished* está ligada, e devendo esta flag apenas ter efeito uma vez, desliga-a prontamente. De seguida, incrementa o valor do índice com o valor de *mwe* (a multi word expression capturada) em *array*. Esta lógica acaba por ser bastante simples uma vez que o *awk* inicializa automaticamente o valor do contador em *array[mwe]* a zero, caso este não tenha sido inicializado anteriormente.

O quarto bloco é ativado quando se encontra o início de um bloco de multi word expression, ligando a flag *capture*. Este bloco é colocado no final para que apenas se comece a capturar a multi word expression na linha seguinte, ignorando a linha que contém *<mwe ...>*.

No bloco *END* procede-se à impressão do array de multi word expressions, imprimindo a multi word expression (índice) e o seu contador (*array[índice]*) separados por uma vírgula, uma multi word expression por linha.

No fim, para apresentar os resultados de uma maneira atrativa, o script pode ser invocado com o comando seguinte:

```
awk -f e1p2.awk micro.txt | sort -k 2 -t ",-g | column -t -s ",
```

3.2.3 Calcule a lista dos verbos PT: (Lema, para palavras com pos=V) e respetivo número de ocorrências

```
BEGIN {
  # FS=" "
  # By default, FS is set to a single space character, which awk interprets to mean "one or more spaces or tabs."
}

# Catch words with part of speech == V
$5 == "V" {
  lema = tolower($4) # lema is $4 (make sure it's lowercase)
  array[lema]++      # increment count (defaults to 0 when uninitialized)
}

END {
  for (i in array) {
    print i "," array[i]
  }
}
```

Figura 4: Código ponto 3

Para este subexercício, usou-se um bloco entre o *BEGIN* e o *END* que é ativado quando a linha tem o valor de *pos* (quinta coluna) igual a *V*.

Quando o bloco é ativado, obtém-se o lema na quarta coluna usando *tolower()* para evitar contadores diferentes para o mesmo verbo com capitalização diferente, e incrementa-se o valor de *array[lema]* (que se não tiver sido inicializado anteriormente, é inicializado a zero automaticamente pelo *awk*).

Por fim, no bloco *END*, imprimem-se os verbos e do seu respetivo contador, um por linha.

Para apresentar os resultados de uma maneira atrativa, sugere-se invocar o script com o comando seguinte:

```
awk -f e1p3.awk micro.txt | sort -k 2 -t ",-g | column -t -s ",
```

3.2.4 Determinar o dicionário implícito no corpora - calcule a lista das palavras associando-lhes os possíveis (lema,pos)

```
BEGIN {
    # FS=" "
    # By default, FS is set to a single space character, which awk interprets to
    # mean "one or more spaces or tabs."

    # Arrays used:
    # array[palavra] = string of (lema,pos),(lema,pos),[...] found associated with
    # word
    # arrayaux[palavra lema pos] = 1 if combination has been found before
}

# Catch lines with lema and pos columns
# pos is $5, so check for at least 5 columns
NF >= 5 {

    palavra = tolower($1)
    lema = tolower($4)
    pos = $5

    # Avoid repeating entries
    if (!((palavra lema pos) in arrayaux)) {

        # Mark this combination as found to avoid repetitions
        arrayaux[palavra lema pos] = 1

        # Initialize dictionary entry for this word
        if (!(palavra in array)) {
            array[palavra] = ""
        } else {
            # Add comma
            array[palavra] = array[palavra] ","
        }

        # Esta lista de parts of speech por extenso não é exaustiva
        if (pos == "V") {
            pos = "Verbo"
        } else if (pos == "N") {
            pos = "Nome"
        } else if (pos == "ADJ") {
            pos = "Adjetivo"
        } else if (pos == "ADV") {
            pos = "Advérbio"
        } else if (pos == "DET_artd") {
            pos = "Determinante artigo definido"
        } else if (pos == "DET_arti") {
            pos = "Determinante artigo indefinido"
        } else if (pos == "NUM_year_date_card") {
            pos = "Número cardinal ano/data"
        } else if (pos == "NUM_card") {
            pos = "Número cardinal"
        }

        # Commas between the various (lema,pos) are added above
        array[palavra] = array[palavra] "(" lema "," pos ")"
    }
}

END {
    for (i in array) {
        print i ":" array[i]
    }
}
```

Figura 5: Código ponto 4

Neste ponto são usados dois arrays:

- `array[palavra]` é uma string de vários (lema,pos) que foram associados à palavra do índice, separados por vírgulas.
- `arrayaux[palavra lema pos]` é um array de flags que tomam o valor 1 quando a combinação de *palavra lema pos* já foi encontrada anteriormente, de maneira a evitar repetições no dicionário.

O único bloco entre *BEGIN* e *END* é ativado para todas as linhas que tenham pelo menos 5 campos, sendo o quinto campo referente ao *point of speech* (*pos*).

Para evitar repetições, a palavra e o lema são capturados com `tolower()`. O campo *pos* está sempre em maiúsculas e por isso não necessita deste tratamento.

De seguida, se a combinação atual de *palavra lema pos* ainda não foi encontrada (flag em `arrayaux` não foi ligada), ativa-se essa mesma flag e adiciona-se essa combinação ao dicionário. Inicializa-se, depois, a entrada em `array` dessa palavra como uma string vazia (ou acrescenta-se uma vírgula se não estiver vazia), e concatena-se a essa string a combinação (*lema,pos*).

Foi tomado o cuidado adicional de "traduzir" algumas das parts of speech mais frequentes, passando por exemplo *V* para *Verbo*, ou *ADJ* para *Adjetivo*.

Por fim, no bloco *END*, é imprimido `array`, com a palavra e a string das várias combinações de (*lema,pos*) separadas por uma vírgula, com uma palavra por linha.

Para apresentar os resultados de uma maneira atrativa, o script pode ser invocado com o comando seguinte:

```
awk -f elp4.awk micro.txt | sort | column -t -s ":"
```

3.3 Análise de Resultados

```
$ awk -f e1p1.awk micro.txt
Extratos (<ext>): 1295
Paragrafos (<p>): 3029
Frases (<s>): 6701
```

Figura 6: Resultado do ponto 1, enunciado 1

| | |
|----------------|-----|
| por outro lado | 23 |
| bem como | 24 |
| já não | 27 |
| já que | 27 |
| apesar | 28 |
| a partir | 29 |
| de acordo | 32 |
| pelo menos | 37 |
| por exemplo | 38 |
| toda a | 40 |
| apesar de | 41 |
| mais de | 41 |
| todo o | 42 |
| o que | 47 |
| todas as | 47 |
| quanto | 49 |
| no entanto | 58 |
| todos os | 72 |
| do que | 73 |
| cerca de | 92 |
| por cento | 172 |

Figura 7: Resultado do ponto 2, enunciado 1

| | |
|------------|------|
| chegar | 86 |
| deixar | 92 |
| conseguir | 96 |
| apresentar | 97 |
| encontrar | 98 |
| começar | 100 |
| ficar | 125 |
| vir | 125 |
| querer | 134 |
| ver | 155 |
| passar | 156 |
| dizer | 161 |
| dever | 174 |
| dar | 192 |
| haver | 196 |
| ir | 327 |
| poder | 344 |
| fazer | 348 |
| estar | 618 |
| ter | 985 |
| ser | 2745 |

Figura 8: Resultado do ponto 3, enunciado 1

```

zhenxue:(yan=zhenxue,PROP)
ziklon:(ziklon=b,PROP)
zimbabwe:(zimbabwe,PROP)
zimbabweano:(zimbabweano,Adjetivo)
zinco:(zinco,Nome)
zindzi:(zindzi,PROP)
zingaro:(carlos=zingaro,PROP)
ziuganov:(guennadi=ziuganov,PROP)
zixaxa:(roberto=zixaxa,PROP),(zixaxa,PROP)
zob:(zob,PROP)
zobel:(zobel,PROP)
zoben:(zoben,PROP)
zohra:(zohra=ouaziz,PROP)
zona:(zona,Nome),(zona=económica=exclusiva,PROP)
zonal:(zonal,Adjetivo)
zonas:(zona,Nome)
zoologia:(manual=de=zoologia=fantástica,PROP)
zorro:(zorro=e=os=três=mosqueteiros,PROP)
zubero:(martin=lópez=zubero,PROP),(zubero,PROP)
zulus:(zulu,Nome)
zurique:(zurique,PROP)

```

Figura 9: Resultado do ponto 4, enunciado 1

4 Processador de textos preanotados com *Freeling*

Este tema visa a análise de ficheiros sobre os *corpora*, que agrupam grandes quantidades de texto, e aos quais se adicionam informação de anotação frásica e morfossintática (lema, categoria gramatical,...). Para o presente exercício, foram fornecidos ao grupo cinco *datasets* em formato *Freeling* (explicado à frente no relatório), "fl0.txt", "fl1.txt", "fl2.txt", "harrypotter1.txt" e "harrypotter2.txt", sendo que os três primeiros possuem informação relativa a notícias, e dois últimos referente aos tão conhecidos livros de *Harry Potter*.

4.1 Análise dos extratos

Para compreensão total do exercício pedido, houve a necessidade de uma boa análise dos extratos, inclusive a noção de lema, *part of speech*, entre outros, informação esta que estava presente em cada coluna do ficheiro. Assim, e exceptuando alguns casos particulares, apresenta-se de seguida um excerto do texto-fonte, como representação ilustrativa e suporte para a estratégia adotada.

| | | | | | | | | | |
|----|-----------|-----------|---------|------|---|----|--------------------------------------|----|----|
| 1 | Mas | mas | CC | CC | pos=conjunction type=coordinating | -- | (S:0(coord:1)) | -- | -- |
| 2 | Mickelson | mickelson | NP00000 | NP | pos=noun type=proper | -- | (sn:2(grup-nom-ms:2(w-ms:2))) | -- | -- |
| 3 | , | , | Fc | Fc | pos=punctuation type=comma | -- | -- | -- | -- |
| 4 | jogando | jogar | VMB0000 | VMB | pos=verb type=main mood=gerund | -- | (ger:4) | -- | -- |
| 5 | atrás | atrás | RG | RG | pos=adverb type=general | -- | (sadv:5) | -- | -- |
| 6 | , | , | Fc | Fc | pos=punctuation type=comma | -- | -- | -- | -- |
| 7 | respondeu | responder | VMIS350 | VMIS | pos=verb type=main mood=indicative tense=past person=3 num=singular | -- | (grup-verb:7(verb:7)) | -- | -- |
| 8 | de | de | SP | SP | pos=adposition type=preposition | -- | (sp-de:8) | -- | -- |
| 9 | a | a | DA0FS0 | DA | pos=determiner type=article gen=feminine num=singular | -- | (sn:11(espec-fs:9(j-fs:9))) | -- | -- |
| 10 | melhor | melhor | AQ0CS00 | AQ | pos=adjective type=qualificative gen=common num=singular | -- | (grup-nom-fs:11(s-a-fs:10(a-fs:10))) | -- | -- |
| 11 | forma | forma | NCFS000 | NC | pos=noun type=common gen=feminine num=singular | -- | (grup-nom-fs:11(n-fs:11)))) | -- | -- |
| 12 | , | , | Fc | Fc | pos=punctuation type=comma | -- | -- | -- | -- |
| 13 | com | com | SP | SP | pos=adposition type=preposition | -- | (prep:13) | -- | -- |
| 14 | , | , | Fc | Fc | pos=punctuation type=other | -- | (F-no-c:14) | -- | -- |
| 15 | Birdies | Birdies | NCFP000 | NC | pos=noun type=common gen=feminine num=plural | -- | (sn:15(grup-nom-fp:15(n-fp:15))) | -- | -- |
| 16 | , | , | Fz | Fz | pos=punctuation type=other | -- | (F-no-c:16) | -- | -- |
| 17 | em | em | SP | SP | pos=adposition type=preposition | -- | (prep:17) | -- | -- |
| 18 | os | o | DA0MP0 | DA | pos=determiner type=article gen=masculine num=plural | -- | (espec-mp:18(j-mp:18)) | -- | -- |

num Lema, isto é, forma canónica da palavra da coluna anterior
pos-tag pos("part of speech")
pos-tag Features, que indicam as classes gramaticais e respetivas subclasses
Árvore

Palavra como aparece na notícia

Figura 10: Análise sumária de um extrato

Assim sendo, verifica-se uma constante de colunas "temáticas", separadas por espaços na mesma linha, que são comuns a quase todos os ficheiros fornecidos pela equipa docente. Exemplificando, se for necessário saber a classe gramatical da palavra "Mickelson", saber-se-á que, na coluna seis, estarão indicadas as opções "nome" e "próprio". O grupo reparou que, como é mostrado em baixo, poderão aparecer casos em que são omitidas colunas, o que resultará numa análise mais pormenorizada dos extratos em questão.

| | | | | | | | | | |
|----|---------------|-----------|---------|----|----|----|--|----|----|
| 1 | De | de | SP | -- | -- | -- | (S:0(sp-de:1)) | -- | -- |
| 2 | este | este | DD0MS0 | -- | -- | -- | (sn:3(espec-ms:2(dem-ms:2))) | -- | -- |
| 3 | processo | processo | NCMS000 | -- | -- | -- | (grup-nom-ms:3(n-ms:3))) | -- | -- |
| 4 | não | não | RN | -- | -- | -- | (neg:4) | -- | -- |
| 5 | estão | estar | VMIP3P0 | -- | -- | -- | (grup-verb:5(verb:5)) | -- | -- |
| 6 | ausentes | ausente | AQ0CP00 | -- | -- | -- | (sn:7(grup-nom-mp:7(s-a-mp:6(a-mp:6))) | -- | -- |
| 7 | mecanismos | mecanismo | NCMP000 | -- | -- | -- | (grup-nom-mp:7(n-mp:7))) | -- | -- |
| 8 | de | de | SP | -- | -- | -- | (sp-de:8) | -- | -- |
| 9 | coopta | coopta | NCFS000 | -- | -- | -- | (sn:9(grup-nom-fs:9(n-fs:9))) | -- | -- |
| 10 | , | , | Fc | -- | -- | -- | -- | -- | -- |
| 11 | de | de | SP | -- | -- | -- | (sp-de:11) | -- | -- |
| 12 | resto | resto | NCMS000 | -- | -- | -- | (sn:12(grup-nom-ms:12(n-ms:12))) | -- | -- |
| 13 | eventualmente | eventual | RG | -- | -- | -- | (sadv:13) | -- | -- |
| 14 | antecipados | antecipar | VMP00PM | -- | -- | -- | (s-adj:15(s-a-mp:15(s-a-mp:14(parti-mp:14))) | -- | -- |
| 15 | , | , | Fc | -- | -- | -- | -- | -- | -- |
| 16 | desejados | desejar | VMP00PM | -- | -- | -- | (s-a-mp:16(parti-mp:16))) | -- | -- |

Figura 11: Análise sumária de um extrato - Exceção

4.2 Filtro de Texto - Sistema de Produção GAWK

4.2.1 Contar número de extratos

Para este tópico, era requerido apenas que fossem contados os números de extratos dos vários ficheiros. Deste modo, o grupo começou por definir que o *Field Separator* seria o caracter "espaço" (uma ou mais ocorrências deste caracter), o que não seria necessário, já que é o que está definido por omissão. De seguida, e ainda na secção *BEGIN*, inicializou-se uma variável "conta" a zero, que serviria de contador de extratos. Para o corpo do programa, considerou-se por bem declarar o *Number of Fields* superior a zero, para não serem processadas as linhas em branco. Neste contexto, as diferenças registadas por um programa com este filtro não são relevantes, mas se forem considerados *datasets* de grandes volumes de dados, isto revela-se de grande utilidade. Por fim, na secção *END*, imprime-se para o ecrã o conteúdo da variável "conta", ação que está também definida por omissão.

Segue-se o código do programa, juntamente com as ações do terminal - comando e output - para o ficheiro de input "fl2.txt" e "harrypotter1.txt".

```
BEGIN{ FS = " "; conta = 0 }  
NF>0{ conta++ }  
END{ print conta }
```

Figura 12: Programa contaN.awk

```
$ gawk -f contaN.awk harrypotter1  
66234
```

Figura 13: Comando e output para o ficheiro harrypotter1.txt

```
$ gawk -f contaN.awk fl2.txt  
5343
```

Figura 14: Comando e output para o ficheiro fl2.txt

4.2.2 Calcular a lista dos personagens do Harry Potter (nomes próprios) e respetivo número de ocorrências

Neste exercício, era requerido que fosse "calculada" a lista de personagens do Harry Potter, indicando o seu nome próprio e a frequência (número de ocorrências). Na verdade, o que o enunciado pede é que, para cada linha, os extratos que tiverem na sexta coluna a indicação de *noun* e *proper* (como na imagem seguinte), devem ser extraídos e guardados numa lista.

| | | | | | |
|----|---------------|---------------|---------|----|-----------------------------------|
| 26 | e | e | CC | CC | pos=conjunction type=coordinating |
| 27 | Sergio_Garcia | sergio_garcia | NP00000 | NP | pos=noun type=proper |
| 28 | , | , | Fc | Fc | pos=punctuation type=comma |
| 29 | a | a | SP | SP | pos=adposition type=preposition |
| 30 | duas | 2 | Z | Z | pos=number |

Figura 15: Exemplo de um registo com os atributos *noun* e *proper*, identificativo de um nome próprio

Repara-se aqui que existe um ficheiro, nomeadamente, o "fl0.txt" em que não se apresentam as informações de classe e subclasse na coluna seis, o que levou a que o grupo reformulasse o problema ao nível da expressão regular.

Segue-se o código implementado pelo grupo, juntamente com uma breve explicação, comando e output produzido.

```
BEGIN{  
  
/noun.*proper/{nomes[$3]++;}  
  
END{  
    for(ind in nomes){  
        print ind, nomes[ind]  
    }  
}
```

Figura 16: Programa nomesProp.awk

Antes de mais, deixou-te a secção *BEGIN* vazia, uma vez que não seria necessário declarar nenhuma variável inicial, nem alterar parâmetros como o *Field Separator*. No corpo do programa, apenas foi necessário definir o padrão de pesquisa por nome, e quando fossem encontrados nomes próprios, guardar-se-iam numa estrutura de dados "nomes" o lema da palavra original. Compara-se a expressão regular atual com a anterior do grupo, definida por

`$6 noun.*proper``nomes[$3]++;`

que não funcionava no ficheiro "fl0.txt", já que o conteúdo da coluna 6 não é o típico dos outros ficheiros.

Por fim, na secção *END*, bastou um ciclo que iterasse pelo *array* e imprimisse todos os elementos da lista já construída com o resultado requerido, juntamente com o número indicativo da frequência com que cada nome aparecia no ficheiro em análise.

Segue-se o comando usado pelo grupo, assim como os diversos outputs produzidos. É de salientar que, ao invocar `sort -r -n`, a lista será ordenada por ordem numérica e decrescente de frequência.


```
$ gawk -f nomesProp.awk harrypotter1.txt | sort -r -n
679 :: harry
196 :: hagrid
189 :: ron
120 :: dudley
102 :: vernon
70 :: mc_gonagall
69 :: dumbledore
63 :: hermione
55 :: petA°nia
55 :: malfoy
51 :: gryffindor
51 :: dursley
49 :: neville
49 :: hogwarts
46 :: snape
46 :: dursleys
41 :: wood
38 :: slytherin
34 :: quidditch
33 :: potter
28 :: percy
28 :: harry_potter
23 :: muggles
23 :: filch
22 :: peeves
22 :: ollivander
20 :: gringotts
19 :: quirrell
19 :: fred
17 :: hermione_granger
15 :: goyle
14 :: george
14 :: crabbe
13 :: scabbers
13 :: privet_drive
```

Figura 17: Comando e output para o ficheiro harrypotter1.txt

```
$ gawk -f nomesProp.awk harrypotter2.txt | sort -r -n
568 :: harry
267 :: ron
91 :: hermione
74 :: dobby
71 :: fred
70 :: lockhart
56 :: george
54 :: hagrid
50 :: mrs._weasley
41 :: hogwarts
40 :: malfoy
38 :: vernon
38 :: mr._weasley
38 :: harry_potter
34 :: gryffindor
33 :: muggles
32 :: nick
29 :: dudley
28 :: dursleys
27 :: petA°nia
25 :: ginny
24 :: gilderoy_lockhart
22 :: weasley
22 :: snape
22 :: percy
22 :: mc_gonagall
22 :: filch
21 :: quidditch
21 :: colin
20 :: wood
19 :: mr._malfoy
19 :: hedwig
18 :: dumbledore
17 :: slytherin
15 :: sprout
15 :: peeves
14 :: murta
```

Figura 18: Comando e output para o ficheiro harrypotter2.txt

4.2.3 Calcular a lista dos verbos, substantivos, adjetivos e advérbios PT; e criar um ficheiro com cada uma destas listas.

Para este ponto, era exigida que fosse calculada uma lista de verbos, seguida de uma outra com os substantivos, adjetivos e, por fim, advérbios de um determinado extrato de texto. No fim disto, o output seria redirecionado para um ficheiro.

Desta forma, o que era realmente pedido, era que, em cada linha, se verificasse a classe gramatical e se extraísse o lema das respetivas palavras. Para os verbos, procurou-se por *"verb"*, para os substantivos por *"noun"*, *"adjective"* para os adjetivos e *"adverb"* para os advérbios.

| | | | | | |
|----|----------------|--------------|---------|-----|--|
| 26 | depois | depois | RG | RG | pos=adverb type=general |
| 27 | de | de | SP | SP | pos=adposition type=preposition |
| 28 | ambos | ambos | PI0MP00 | PI | pos=pronoun type=indefinite gen=masc num=plural |
| 29 | terem | ter | VMN03P0 | VMN | pos=verb type=main mood=infinitive person=3 num=plural |
| 30 | completado | completar | VMP00SM | VMP | pos=verb type=main mood=pastparticiple num=singular gen=masc |
| 31 | as | o | DA0FP0 | DA | pos=determiner type=article gen=feminine num=plural |
| 32 | quatro | 4 | Z | Z | pos=number |
| 33 | voltas | volta | NCFP00 | NC | pos=noun type=common gen=feminine num=plural |
| 34 | regulamentares | regulamentar | AQ0CP00 | AQ | pos=adjective type=qualificative gen=common num=plural |

Figura 19: Exemplo de ocorrências de *"verb"*, *"noun"*, *"adjective"*, *"adverb"*.

Começando pela secção *BEGIN*, mais uma vez, foi deixada em branco, uma vez que as ações por *default* serviriam o propósito do exercício. No corpo do programa, optou-se por definir os padrões de procura na forma:

```
/=verb/verbos[$3]++;
```

que, ao encontrar um verbo, guarda o lema da palavra(coluna três) no array. Uma versão inicial do grupo, definia o padrão como

```
/verb/verbos[$3]++;
```

O que estava incorreto, uma vez que ia buscar ocorrências da palavra *verb* na coluna da "Árvore", e não à das classes e subclasses gramaticais, onde vem um sinal de igual antes de definir, por exemplo, um verbo.

Continuando com a explicação do programa, na secção *END* apenas se imprimiu o conteúdo dos quatro arrays resultantes, sendo que posteriormente, no terminal, redireciona-se este conteúdo para um ficheiro *"out.txt"*.

Segue-se o programa e o comando executado no final para dois ficheiros. O output do programa encontra-se na secção 4.3

```
BEGIN{

    /\=verb/{verbos[$3]++;}
    /\=noun/{nomes[$3]++;}
    /\=adjective/{adjectivos[$3]++;}
    /\=adverb/{adverbios[$3]++;}

END{

    print("-----$VERBOS-----")
    for(ind in verbos){print ind}
    print("\n")
    print("-----$NOMES-----")
    for(ind in nomes){print ind}
    print("\n")
    print("-----$ADJECTIVOS-----")
    for(ind in adjectivos){print ind}
    print("\n")
    print("-----$ADVERBIOS-----")
    for(ind in adverbios){print ind}

}
```

Figura 20: Programa lists.awk

```
$ gawk -f lists.awk fl2.txt > out.txt
```

Figura 21: Comando sobre o ficheiro fl2.txt, com redireccionamento do output para out.txt

4.2.4 Determinar o dicionário implícito no corpora - lista contendo os lema, pos e palavras dele derivadas

Este último exercício requeria que fosse feito um dicionário, isto é, uma lista com a palavra, lema e pos da primeira. O grupo decidiu organizar o dicionário pela palavra, em primeiro lugar, tal como aparecia no extrato, no formato:

palavra : (lema , pos)

Antes de passar à explicação do programa, é pertinente clarificar um problema que surgiu aquando da realização deste exercício: para o caso do, por exemplo, ficheiro "fl0.txt", onde não existe a coluna "pos", o grupo decidiu que não fazia sentido, num contexto real, um dicionário sem esta caracterização de uma palavra. Assim sendo, optou-se por ignorar este extrato totalmente, extraindo apenas dicionários de extratos que possuíssem condições para tal.

Segue-se o código do programa "dic.awk", já anotado para melhor compreensão, seguido de uma explicação, breve pelo facto de já ter sido elaborado um dicionário no problema da secção 3.

```
BEGIN{}
NF>0{
    palavra = tolower($2)
    lema = tolower($3)
    pos = $5

    # Avoid repeating entries
    if(!((palavra lema pos) in arrayaux)){

        # Mark this combination as found to avoid repetitions
        arrayaux[palavra lema pos] = 1

        # Initialize dictionary entry for this word
        if (!(palavra in array)) {
            array[palavra] = ""
        } else {

            # Add comma
            array[palavra] = array[palavra] ","

        }

        array[palavra] = array[palavra] "(" lema "," pos ")"
    }
}
END{
    for(ind in array){
        print ind " : " array[ind]
    }
}
```

Figura 22: Programa dic.awk

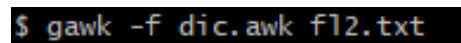
Como é visível e costume nos programas anteriores, deixa-se a secção *BEGIN* em branco, pelas opções por *default* se enquadrarem nos resultados que o grupo pretende. De seguida, no

corpo do programa, começa-se por obter a palavra, o lema e a pos usando o *tolower()*, para evitar contadores diferentes para a mesma palavra, por exemplo. Como a pos já se encontra com letra maiúscula, este tratamento não é necessário. De seguida, usam-se dois arrays:

- `array[palavra]`, string de vários (lema,pos) que foram associados à palavra do índice, separados por vírgulas
- `arrayaux[palavra lema pos]`, array de flags que tomam o valor 1 caso a combinação *palavra lema pos* já foi encontrada anteriormente, de maneira a evitar repetições no dicionário

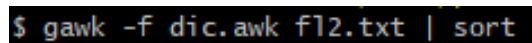
Com isto, na condição (*if*) verifica-se se uma combinação *palavra lema pos* ainda não se encontra no dicionário, através de "arrayaux", e, nesse caso, marca-se essa combinação como encontrada e inicializa-se o dicionário para a mesma. De seguida, adiciona-se ao "array", por palavra, a sua pos e lema. É sabido que, se já houver no dicionário uma ocorrência de uma palavra, todo este trabalho não será efetuado, evitando repetições. Concluindo, no final do programa apenas se imprime para o ecrã a palavra, juntamente com o lema e a pos, percorrendo o "array".

Apresenta-se em baixo o comando utilizado, e na próxima secção o output do programa.



```
$ gawk -f dic.awk fl2.txt
```

Figura 23: Comando para o ficheiro fl2.txt



```
$ gawk -f dic.awk fl2.txt | sort
```

Figura 24: Comando para o ficheiro fl2.txt, para produção de um dicionário ordenado

4.3 Análise de Resultados

```
$ gawk -f contaN.awk harrypotter1
66234
```

Figura 25: Comando e output para o ficheiro harrypotter1.txt - Exercício 1

```
$ gawk -f contaN.awk fl2.txt
5343
```

Figura 26: Comando e output para o ficheiro fl2.txt Exercício 1

```
$ gawk -f nomesProp.awk harrypotter1.txt | sort -r -n
679 :: harry
196 :: hagrid
189 :: ron
120 :: dudley
102 :: vernon
70 :: mc_gonagall
69 :: dumbledore
63 :: hermione
55 :: petA^nia
55 :: malfoy
51 :: gryffindor
51 :: dursley
49 :: neville
49 :: hogwarts
46 :: snape
46 :: dursleys
41 :: wood
38 :: slytherin
34 :: quidditch
33 :: potter
28 :: percy
28 :: harry_potter
23 :: muggles
23 :: filch
22 :: peeves
22 :: ollivander
20 :: gringotts
19 :: quirrell
19 :: fred
17 :: hermione_granger
15 :: goyle
14 :: george
14 :: crabbe
13 :: scabbers
13 :: privet_drive
```

Figura 27: Comando e output para o ficheiro harrypotter1.txt - Exercício 2

```

$ gawk -f nomesProp.awk harrypotter2.txt | sort -r -n
568 :: harry
267 :: ron
91 :: hermione
74 :: dobby
71 :: fred
70 :: lockhart
56 :: george
54 :: hagrid
50 :: mrs._weasley
41 :: hogwarts
40 :: malfoy
38 :: vernon
38 :: mr._weasley
38 :: harry_potter
34 :: gryffindor
33 :: muggles
32 :: nick
29 :: dudley
28 :: dursleys
27 :: petunia
25 :: ginny
24 :: gilderoy_lockhart
22 :: weasley
22 :: snape
22 :: percy
22 :: mc_gonagall
22 :: filch
21 :: quidditch
21 :: colin
20 :: wood
19 :: mr._malfoy
19 :: hedwig
18 :: dumbledore
17 :: slytherin
15 :: sprout
15 :: peeves
14 :: murta

```

Figura 28: Comando e output para o ficheiro harrypotter2.txt - Exercício 2

```

-----$VERBOS-----
defender
empobrecer
estremecer
determinar
sublinhar
acrescentar
desvalorizar
desalojar
trazer
descartar
assinalar
jogar
interceptar
regressar
pensar
melhorar
prescrever
transmitir
encontrar
considerar
cruzar
beneficiar
dar
acusar
ganhar
pelar
marcar

```

(...)

Figura 29: Output produzido - Exercício 3

```

-----$NOMES-----
doente
henrique_granadeiro
chuva
ano
direito
florida
shots
utilizaÃ§Ã£o
mensagem
play
regulamento
especificidade
bruto
estaÃ§Ã£o
negÃ³cio
cena
zona
espanhol
factor
rentabilidade
titular
semana
proposta (...)

```

Figura 30: Output produzido - continuação - Exercício 3

```

-----$ADJECTIVOS-----
médico
dunar
comum
maior
eventual
sério
consentâneo
total
cigano
real
parlamentar
português
mau
múltiplo
econômico
oceânico
clássico
americano
norte-americano
magro
central (...)

```

Figura 31: Output produzido - continuação - Exercício 3

```

-----$ADVERBIOS-----
de_acordo
tambãem
simplesmente
eventual
total
antes
muito
mais
hoje
ao_vivo
gratuito
especificamente
quase
nunca
praticamente
atãe
nãeo_sãa
desta_vez
pelo_menos
abaixo
como (...)

```

Figura 32: Output produzido - continuação - Exercício 3

```

interior : (interior,NP),(interior,NC)
vir : (vir,VMN)
encerramento : (encerramento,NC)
vigor : (vigor,NC)
ver : (ver,VMN)
exames : (exame,NC)
cinematográfica : (cinematográfico,AQ)
permitiu : (permitir,VMi)
real : (real,AQ)
empresãrio : (empresãrio,NC)
sido : (ser,VMP)
principais : (principal,AQ)
tinha : (ter,VMi)
todos : (todo,DI)
por : (por,SP),(por,NP)
causados : (causar,VMP)
podem : (poder,VMi)
vez : (vez,NC)
estivesse : (estar,VMS)
devastada : (devastar,VMP)
doc : (doc,NC)
negou : (negar,VMi)
essas : (esse,DD)
sustenta : (sustentar,VMi)
tudo : (tudo,PI)
iniciado : (iniciar,VMP) (...)

```

Figura 33: Output para o ficheiro fl2.txt - Exercício 4


```

avaliados : (avaliar,VMP)
avanãsa : (avanãsar,VMI) (...)
avanãsar : (avanãsar,VMN)
avipg : (avipg,NP)
azã,fama : (azã,fama,NC)
baila : (baila,NC)
baixa : (baixa,NC)
balanãço : (balanãço,NC)
banqueiro : (banqueiro,NC)
baseada : (basear,VMP)
baseado : (basear,VMP)
bastante : (bastante,RG)
batendo : (bater,VMG)
bater : (bater,VMN)
bbc : (bbc,NP)
beira_da_estrada : (beira_da_estrada,NP)
bem : (bem,RG)
bem_como : (bem_como,RG)
beneficiado : (beneficiar,VMP)
bes_ricardo_salgado : (bes_ricardo_salgado,NP)
bienal : (bienal,NC)
birdies : (birdies,NC)
bola : (bola,NC)
bradley_wiggins : (bradley_wiggins,NP)
bradley_wiggins : (bradley_wiggins,NP)
branqueamento : (branqueamento,NC)
brasil : (brasil,NP)
brasileira : (brasileiro,AQ) (...)
brasileiros : (brasileiro,AQ)

```

Figura 34: Output ordenado para o ficheiro fl2.txt - Exercício 4

5 Exercício extra - Previsão de palavras

Neste exercício extra foram usados os *datasets* do exercício da Secção 4. Por este motivo, não é aqui necessário proceder à explicação dos ficheiros de texto a serem processados.

O objetivo é simular uma utilidade como a presente nas aplicações dos teclados de telemóveis (ex: *SwiftKey*), que identifica qual a próxima palavra mais provável tendo em conta a escrita no momento. Aqui os *datasets* são usados como base de conhecimento.

```
# Example:
# gawk -f exextra.awk -v word='governo' f10.txt > filename.html
# instead of governo you use any word you wanna search

BEGIN {
  # FS=" "
  # By default, FS is set to a single space character, which awk interprets to mean "one or more spaces or tabs."
  exists=0
  word
  saveProx=0
  saved=0
  numSugestion=0
  sugestedWord
  putComma=0

  print "<!DOCTYPE html>"
  print "<html lang='pt-PT'>"
  print "<body>"

  print "<h1>" word "</h1>"

  print "<div id='piechart'></div>"

  print "<script type='text/javascript' src='https://www.gstatic.com/charts/loader.js'></script>"

  print "<script type='text/javascript'>"
  print "// Load google charts"
  print "google.charts.load('current', {'packages':['corechart']});"
  print "google.charts.setOnLoadCallback(drawChart);"

  print "// Draw the chart and set the chart values"
  print "function drawChart() {"
  print "var data = google.visualization.arrayToDataTable(["
}
```

Figura 35: Secção BEGIN do programa

No bloco BEGIN procedeu-se à inicialização de algumas variáveis:

- **exists** guarda o número de vezes que a palavra a procurar foi encontrada. No caso seria a palavra acabada de escrever no teclado.
- **word** é a suposta palavra que o utilizador acaba de escrever no teclado.
- **saveProx** é uma flag que, após ser identificada a palavra a procurar, fica a 1, permitindo registar a palavra seguinte para entrar na contagem de probabilidades.
- **saved** é o número de palavras guardadas para entrar nas probabilidades.
- **numSugestion** permite saber, no fim, qual o número de vezes que a palavra mais frequente apareceu, para debug.
- **sugestedWord** é a palavra escolhida para ser a seguinte.
- **putComma** é inicializado a 0 para que, na primeira vez que é corrido o algoritmo, não seja posta nenhuma ",", na variável *data* no HTML.

```

$2~/^[a-zA-Z]+[']?[a-zA-Z]*$/ {

    # word after the one which we are looking for
    if(saveProx==1) {
        previsao[$2]++;
        saveProx=0;
        saved++;
    }

    # check if its the word we are looking for
    if(word==$2) {
        exists++;
        saveProx=1;
    }

}

```

Figura 36: Secção de processamento - corpo intermédio do programa

A ER filtra palavras, não captando pontuação. Filtra também quaisquer palavras que contêmham " '(pelica), uma vez que estes provocam estragos em páginas HTML.

É, então, verificado no segundo **if** se a palavra a procurar é a que está a ser avaliada. No primeiro **if** é verificado se aquela é uma palavra a guardar para possível sugestão, guardando-a em **previsao** caso seja.

```

END {
    for (i in previsao) {
        #print previsao[i] ":" i;
        if (previsao[i] > numSugestion) {
            numSugestion=previsao[i];
            sugestedWord=i;
        }

        # Print , after the first argument
        if (putComma) print ",\n"
        else print "['Prevision', 'Word possibilities'], "

        # Print the arguments for html graph
        print "[" i " ", " previsao[i] "]"

        putComma=1
    }

    # Print chart closing and options. Also html closing elements.
    print "]);"

    print "// Optional; add a title and set the width and height of the chart"
    print "var options = {'title':'Palavra seguinte: " sugestedWord "' , 'width':880, 'height':640};"

    print "// Display the chart inside the <div> element with id='piechart'"
    print "var chart = new google.visualization.PieChart(document.getElementById('piechart'));"
    print "chart.draw(data, options);"
    print ";"
    print "</script>"

    print "</body>"
    print "</html>"

    # Debugging
    # print "Encontrou " exists " " word
    # print "#Previsoes: " saved
    # print "Sugestao: " sugestedWord
}

```

Figura 37: Secção END do programa

Nesta parte, dentro do **for**, o primeiro **if** serve para obter a palavra com maior frequência dentro das possíveis próximas. Na página HTML vai ser apresentada claramente esta palavra, como se poderá ver na imagem relativa aos resultados.

No segundo **if**, na primeira vez o `putComma` é 0, pois não se quer introduzir uma "," antes de qualquer elemento da tabela.

O *print* seguinte serve para imprimir no *array* escrito no HTML.

5.1 Análise de Resultados

Após feita a chamada do ficheiro `awk`, o output é enviado para um ficheiro HTML que, posteriormente, é aberto no navegador.

```
$ gawk -f exextra.awk -v word='tinha' fl2.txt > fl2.html
```

Figura 38: Invocação no terminal

tinha

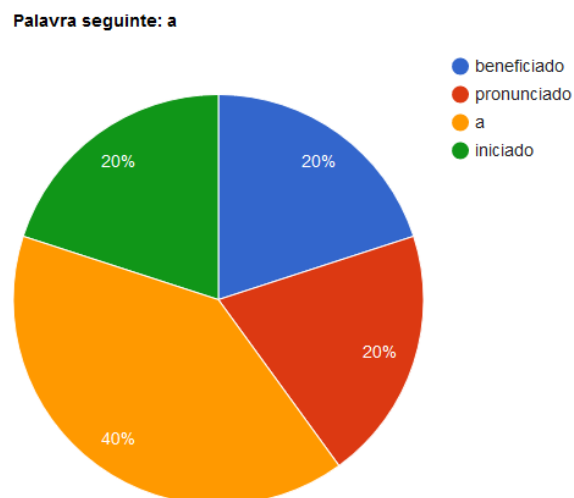


Figura 39: Resultado do ponto extra, usando o ficheiro `fl2.txt`, para a palavra "tinha"

Em alguns casos, podem aparecer caracteres desformatados, sendo isto perfeitamente normal, tendo em conta o ficheiro de onde é feito o *parsing*. De facto, numa situação real, um telemóvel guardaria toda a escrita de um leitor em formato correto.

6 Conclusões e Sugestões

A resolução deste primeiro exercício prático foi bastante importante e enriquecedora, pois permitiu aos membros do grupo perceber e interiorizar melhor os conceitos abordados nas aulas práticas da Unidade Curricular de Processamento de Linguagens. Deste modo, foram adquiridos conhecimentos mais sólidos acerca de programação em *awk* e da utilidade que a ferramenta *GAWK* pode ter numa utilização real. De facto, foi possível perceber que a exigência de programar algo equivalente ao desenvolvido, quer em Java, quer C (linguagens de maior domínio do grupo), seria muito maior. Assim, o grupo ficou satisfeito por ter desenvolvido este trabalho.

Em ambos os enunciados, número 1 e 2, a maior dificuldade residiu, em primeira instância, na análise dos *datasets*, para ajudar a prever casos de exceção, tornando os programas mais robustos. A realização da maioria dos exercícios foi simples, com exceção de alguns mais extensos, que requeriram um algoritmo mais elaborado e dispenderam de mais tempo por parte dos elementos do grupo.

Já no exercício extra, o algoritmo, embora extenso, seria semelhante aos já elaborados nos exercícios dos enunciados anteriores, sendo que o desafio consistiu na elaboração do gráfico circular em HTML, algo que os elementos do grupo nunca haviam realizado. Ainda assim, e através da correta divisão de tarefas e revisão por toda a equipa, foi conseguida uma solução eficaz para este e os restantes problemas.

Em suma, é feita uma apreciação positiva relativamente ao trabalho realizado, visto que a implementação de todas as funcionalidades propostas foram conseguidas com sucesso. O grupo conseguiu tirar partido dos conhecimentos adquiridos neste projeto, sentido-se capaz de, num contexto futuro, aplicar os conceitos subjacentes de forma eficaz.