Processamento de Linguagens

Trabalho Prático Nº2 - GAWK

Ana Catarina Sousa
a78029

Eduardo Jorge Barbosa
a83344

(28 de Abril de 2019)

Resumo

O (G)AWK é uma linguagem e processador de padrões, que permite programar filtros sobre informação tabular, de maneira muito simples. De forma a demonstrar o poder desta ferramenta, desenvolveu-se processadores de textos pré-anotados com Freeling, cuja implementação em GAWK é discutida neste relatório.

Conteúdo

1 Introdução

Como referido no primeiro trabalho prático, a análise de texto, e consequente manipulação, é uma das tarefas mais comuns na informática. No entanto, o processamento de texto, com recurso a linguagens imperativas e/ou orientadas a objetos, tais como o *C* ou *C++* é um processo moroso, complexo e muito suscetível a erros. Desta forma, almejando oferecer o mesmo poder expressivo que o Perl, mas sem a síntaxe críptica deste, e evitando a complexidade inerente do C, surgem ferramentas como o (G)AWK, que permite desenvolver programas sucintos, que definem um conjunto de procedimentos a realizar quando são encontrados determinados padrões nos ficheiros de input. Assim, com o objectivo de demonstrar o uso do GAWK foram implementados processadores de textos pré-anotados com Freeling, que contam o número de extratos, calculam a lista de personagens de Harry Potter e respetivo número de ocorrências, calculam a lista de ocorrências das diversas categorias gramaticais e criam ficheiros HTML com a lista de cada uma delas e determinam o dicionário implícito no córpora.

1.1 Estrutura do Relatório

O relatório encontra-se dividido nos seguintes capítulos:

1. Introdução;
2. Background;
3. Conceitos básicos;
4. Proposta;
5. Implementação;
6. Conclusão.

No capítulo 2, é feita uma referência aos conceitos por detrás das expressões regulares. No capítulo 3, descrevem-se alguns conceitos fundamentais sobre o GAWK.
No capítulo 4 é analisado o que foi pedido implementar e o foco do grupo. Finalmente, no capítulo 5 é abordada a arquitetura da solução, e a análise necessária para a sua implementação.
O capítulo 6 deixa algumas conclusões sobre o trabalho desenvolvido e trabalho futuro.

2 Background

Informalmente, uma expressão regular pode ser descrita como um pedaço de texto que descreve um padrão. De uma forma mais formal, uma expressão regular permite descrever uma *linguagem regular*, sendo então extremamente

útil para descrever padrões.
Estes conceitos são explicados, de uma forma mais detalhada, no relatório do primeiro trabalho prático, pelo que se omite essa explicação neste relatório.

3 Conceitos básicos de GAWK

De forma simples, o GAWK é um processador de padrões, que permite programar filtros sobre informação tabular, de uma forma simplificada. Por omissão, os ficheiros de input são processados linha a linha e as linhas estão separadas em campos, por um espaço. Sendo uma ferramenta bastante flexível, é possível alterar estas definições. Usando as variáveis pré-definidas, *Record Separator* e *Field Separator*, o GAWK permite definir quais os caracteres de separação.

3.1 Estrutura

Pode-se pensar no GAWK usando a seguinte abstração:

```
padrão   {ação}
```

O elemento padrão especifica um teste que é avaliado em cada *field* e, quando é verdadeiro, a ação correspondente é realizada.

3.2 Padrões

Os padrões podem ser especificados com expressões regulares ou relacionais, podendo fazer *match* com um ou mais *fields*. Existem dois padrões especiais, o **BEGIN** e o **END**.
O *BEGIN* permite especificar ações a serem realizadas antes do processamento do *input*. Por sua vez, as ações do *END* são executadas após o *input* ter sido processado.

3.3 Campos

Como referido, em GAWK, as linhas são divididas em campos pelo valor do separador definido inicialmente. Os valores entre `$1` e `$n` representam os vários campos e $0 representa a totalidade do *record*. Por omissão, um *record* corresponde a uma linha.

4 Proposta

Neste trabalho prático foi proposto o desenvolvimento de um processador de textos pré-anotados com *Freeling*.

4.1 Descrição Informal do Problema

Neste trabalho, é pedido que se analise um conjunto de extratos e se construa um ou mais programas GAWK que processem o corpora Freeling de modo a: contar o número de extratos, calcular a lista de personagens do *Harry Potter* e respetivo número de ocorrências, calcular a lista dos verbos, adjetivos e advérbios PT e criar um ficheiro HTML com cada uma das listas e determinar o dicionário implícito no córpora - lista contendo os lema, pos e palavras dele derivadas.
Dada a natureza do problema proposto, foi decidido dividir a resolução do problema em dois mini-projectos. Separou-se então a análise da lista de personagens do *Harry Potter*, do resto do problema. Para cada mini-projecto, foi criado um programa em GAWK para cada sub-problema. De forma a processar os ficheiros automaticamente foram criados dois *scripts* em BASH.

5 Implementação

Para todos os programas GAWK, o grupo decidiu manter as configurações *default* de do *Record Separator* e do *Field Separator*.

5.1 Harry Potter

Para este problema era apenas pedido que se processasse os nomes próprios presentes nos ficheiros dados. Os ficheiros de *input* seguem um formato de *Freeling*. Analisando os ficheiros, o nome encontra-se na segunda coluna de cada linha e a *pos* encontra-se na quinta coluna. A *pos* `NP` identifica um nome próprio.
De forma a guardar os nomes próprios e o respectivo número de ocorrências, é utilizado um *array* de inteiros, onde os índices correspondem a cada nome próprio presente no ficheiro. Sempre que a *pos* é igual a `NP`, a palavra correspondente é usada como índice e o número de ocorrências é incrementado. No final do processamento é gerada uma pagina HTML com a informação recolhida.

```
BEGIN { RS="\n";  }

# $2 name
# $5 pos
$5 == "NP"  {names[$2]++;}

END { toHTML(); }
```

Parte do program Harry Potter

5.2 Extratos

A análise do extratos divide-se em três sub-problemas:

1. Contar o número total de extratos;
2. Identificar cada palavra de cada categorias gramatical;
3. Gerar o dicionário implícito.

Para cada problema foi criado um programa em GAWK. O resultado de cada programa é unificado com os *scripts* mencionados anteriormente.

5.2.1 Número de extratos

Analisando os ficheiros apresentados concluiu-se que, dentro de cada extrato, todas as linhas possuem um número que identifica a sua posição relativa. Este problema torna-se então num simples exercício de contar quantas vezes aparece o número um. Com objetivo de apresentar mais informação relativa a cada extrato, contam-se também o número de linhas por extrato, apresentado ainda:

1. O extrato com mais linhas;
2. O extrato com menos linhas;
3. O número médio de linhas por extrato.

```
BEGIN { RS="\n";
        max=0; min=2^1024; exctratNr=0;
        Red="\033[0;31m"
        Yeallow="\033[1;33m"
        Gray="\033[1;30m"
        Green="\033[32m"
        NoColour="\033[0m"
      }


$1 == 1 {   if(exctratNr != 0) {
                if(lines[exctratNr]>max)
                    max=exctratNr;
                if(lines[exctratNr]<min)
                    min=exctratNr;
            }
            exctratNr++;
        }

$1 != "" { lines[exctratNr]++; }


END { sum=0; for(nr in lines) sum += nr;
```

Parte do programa do número de extratos

De forma a apresentar a informação é gerado um ficheiro HTML, bem como apresentada no terminal toda a informação.

```
fl2
       [*] Number of extracts: 102
       [*] Extract with the most lines: 99 with 150 lines
       [*] Extract with the least lines: 1 with 45 lines
       [*] Average lines per extract: 51.5
```

Output do terminal

5.2.2 Categorias

De forma a identificar a categoria de cada palavra, inicialmente foi tentado processar o *pos*. No entanto, analisando o *input*, descobriu-se que este atributo não se encontrava presente em todos os ficheiros. Foi então decidido utilizar o atributo *pos-tag* que possui mais informação que o *pos*. A título de exemplo, um nome próprio possui o valor `NP` no atributo *pos*, e o valor `NP00000` no atributo *pos-tag*. Torna-se então claro, que é apenas preciso identifica a primeira letra do atributo *pos-tag*. Para cada categoria é usado um *array* diferente, onde a palavra serve como índice. No final, cada *array* é convertido para uma página HTML.

```
BEGIN { RS="\n";  }

# $4 type
index($4, "V") == 1  {verbs[$2]++;}
index($4, "N") == 1  {nouns[$2]++;}
index($4, "A") == 1  {adjes[$2]++;}
index($4, "R") == 1  {adves[$2]++;}

END {
    toHTML("verbs", "List of verbs!", verbs);
    toHTML("nouns", "List of nouns!", nouns);
    toHTML("adjes", "List of adjectives!", adjes);
    toHTML("adves", "List of adverbs!", adves);
}
```

Parte do programa das categorias gramaticais

5.2.3 Dicionário Implícito

De forma a gerar o dicionário implícito, foi utilizado um *array* multidimensional, onde a primeira chave corresponde a um **lema**, a segunda corresponde a uma **palavra do lema** e o valor corresponde à **pos-tag**.
Tal como no problema anterior, o atributo *pos* não se encontra presente em todos os ficheiros, daí ser utilizado o *pos-tag*. Foi também evidente que algumas palavras se encontram mal identificadas, estando até presente algumas hiperligações. De forma a remediar isto, decidiu-se limitar uma palavra a:

1. Começar com
$$a - zA - Z$$
ou letras acentuadas;
2. Possuir apenas
$$a - zA - Z$$
, letras acentuadas, - ou _.

```
BEGIN { RS="\n"; }

# $4 pos-tag
# $3 lemma
# $2 palavra
$3 ~ /^[a-zA-ZàÀáÁâÂãÃéÉèÈêÊẽẼ
 lemmas[$3][$2]=$4; }

END { toHTML(); }
```

Parte do programa do dicionário

No final, o dicionário gerado é transposto para HTML.

5.3 Scripts

De forma a processar todos os ficheiros automaticamente, e gerar alguns ficheiros HTML que agregam toda a informação foram criados dois *scripts*.

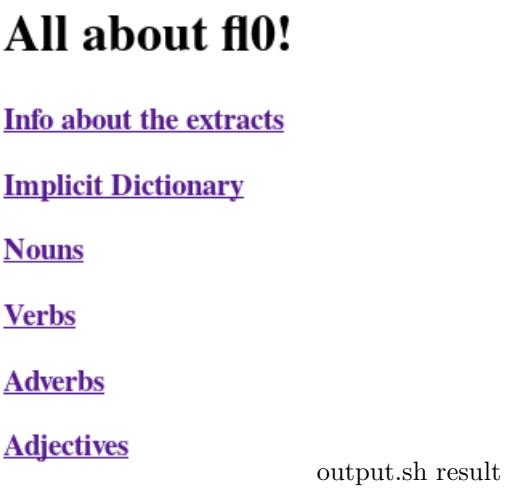
5.4 Output

O *script* `output.sh` é responsavel por processar um ficheiro, executando todos os programas GAWK necessários e gerar um índice em HTML de todos os resultados.

```
gawk -f dic.gawk $1
gawk -f category.gawk $1
gawk -f extractNr.gawk $1


echo "<!DOCTYPE html>" > "$1_index.html";
echo "<HTML>" >> "$1_index.html";
echo "<body>" >> "$1_index.html";
echo "<header><h1> All about $1!</h1></header>" >> "$1_index.html";
echo "<h3><a href=\"$1eNr.html\">Info about the extracts</a></h3>" >> "$1_index.html";
echo "<h3><a href=\"$1dic.html\">Implicit Dictionary</a></h3>" >> "$1_index.html";
echo "<h3><a href=\"$1nouns.html\">Nouns</a></h3>" >> "$1_index.html";
echo "<h3><a href=\"$1verbs.html\">Verbs</a></h3>" >> "$1_index.html";
echo "<h3><a href=\"$1advs.html\">Adverbs</a></h3>" >> "$1_index.html";
echo "<h3><a href=\"$1adjes.html\">Adjectives</a></h3>" >> "$1_index.html";
echo "</body>" >> "$1_index.html";
```

output.sh



output.sh result

5.5 All

O *script* `all.sh` é responsável por invocar o `output.sh` para cada ficheiro de *input* e gerar uma página HTML de índice geral

```
pat="[a-zA-Z]+\..+"


echo "<!DOCTYPE html>" > "index.html";
echo "<HTML>" >> "index.html";
echo "<body>" >> "index.html";
echo "<header><h1> Welcome to the best PL zine around!</h1></header>" >> "index.html";
echo "<h2>Files to checkout:</h2>" >> "index.html";

for file in *; do
    [ -f "$file" ] || continue
    if [[ ! $file =~ $pat ]]; then
        sh output.sh $file
        echo "<h3><a href=\"$file""_index.html\">$file</a></h3>" >> "index.html";
    fi
done


echo "</body>" >> "index.html";
```

all.sh

Welcome to the best PL zine around!

Files to checkout:

fl0

fl1

fl2

Resultado do all.sh

6 Conclusão

Uma vez mais, e à semelhança do que aconteceu no primeiro trabalho, foi possível perceber que a utilização de ferramentas como o GAWK permitem escrever programas de análise de texto de forma muito mais simplificada e com bastante menos código do que linguagens de programação "tradicionais". Consideramos que, de forma global, conseguimos um trabalho positivo, visto que foram implementados todos os pontos pedidos no enunciado, bem como alguns extras. No entanto, é de salientar algumas limitações do GAWK, por exemplo, para casos em que tivéssemos de combinar variáveis com expressões regulares, onde o processo se tornaria demasiado moroso e levantaria problemas de comportamento indefinido.