Processamento de Linguagens

Trabalho Prático nº1 (Gawk)

Relatório de Desenvolvimento

Cesário Perneta	João Gomes	Tiago Fraga
a73883	a74033	a74092

March 23, 2018

Abstract

Neste relatório será abordado a resolução e verificação do segundo trabalho prático da unidade curricular de Processamento de Linguagens. Será possível encontrar a forma como abordamos o problema proposto bem como uma explicação detalhada do código elaborado.

Foi ainda trabalhado um ponto extra diferente dos que nos foram propostos no enunciado, visto utilizar os mesmos ficheiros e de modo a aprimorar os nossos conhecimentos ao nível desta nova Linguagem.

i

Contents

Chapter 1

Processador de Thesaurus 1

O nosso grupo teve como enunciado o *Processador de Thesaurus 1*, neste enuciado foi nos proposto que fizessemos uma analíse aos ficheiros *.mdic* fornecidos, que continham as entradas de um *Thesaurus*. Depois de efectuada esta analíse a estes tipos de ficheiros, prosseguimos com o desonvolvimento de um programa em *GAWK* que resolvessem os seguintes pontos:

1 - Determinar a lista de domínios e relações usadas.

2 - Mostrar os triplos expandidos correspondentes (um Triplo por linha).

3 - Mostrar a informação contidas nos triplos, agrupadas pelo termo 1 (formato Thesaurus ISO).

Em seguida vamos mostrar como prosseguimos a realização destes pontos.

1.1 Determinar a lista de domínios e relações usados

No primeiro ponto pede-se todos os domínios e as relações presentes nos ficheiros, isto é, temos de filtrar o texto de modo a obter todos os domínios que estão presentes nas *tags %dom* e todas as relações presentes nas *tags %THE*.

```
BEGIN       { FS = ":"; RS = "\n";IGNORECASE = 1;}
```

No final da análise dos ficheiros concluimos que o *Field Separator (FS)* mais adequado era os dois pontos evidenciados em cima. Definimos também o *Record Separator (RS)* igual ao separador de linha.

A instruçao *IGNORECASE = 1* foi colocada para ignorar caso hajam os casos por exemplo : *DOM; dom* assim o *GAWK* consegue assumir que se trata da mesma palavra.

```
$1~/%dom/   { dominios[$2]++;}
```

Com a expressão regular apresentada em cima acedemos só as linhas do ficheiro que contêm a *tag %dom*. A acção evidenciada serve para guardar num array em que o índice é os dominios o numero de vezes que tal domínio ocorre.

```
$1~/%THE/   {
                for(i=2; i<NF+1;i++){

                    if($i~/ ./){
                        split($i,tmp," ");
                        if(tmp[1]~/</){
                            split(tmp[1],aux,"<");
                            relacoes[aux[1]]++;
                        }
                        else if(tmp[1]~/@/){
                            split(tmp[1],aux,"@");
                            relacoes[aux[2]]++;
                        }
                        else{
                            relacoes[tmp[1]]++;
                        }
                    }
                    else if($i!=""){
                        if($i~/</){
                            split($i,tmp,"<");
                            relacoes[tmp[1]]++;
                        }
                        else if($i~/@/){
                            split($i,tmp,"@");
                            relacoes[tmp[2]]++;
                        }
                        else{
                            relacoes[$i]++;
                        }
                    }
                }
            }
```

Depois de termos todos os domínios, podemos retirar todas as relações, como está evidenciado em cima é um processo mais complexo que o anterior visto termos de ter em conta os caracteres especiais, bem como o número variável de campos. O algoritimo consiste em percorrer todos os campos de todas as linhas que contêm a *tag %THE* do ficheiro, e realizar as verificações de espaços, caracteres especiais por exemplo ,etc.

Feito este tratamento da informação usamos o método utilizado nos domínios.

Posto isto realizamos a impressão do conteúdo dos arrays criados.

1.2 Mostrar os Triplos Expandidos Correspondentes (Triplo por linha)

Neste segundo ponto tivemos que gerar triplos de informação consoante as relações de cada ficheiro, portanto para a resolução deste ponto precisamos de guardar informação das relações e informação do texto envolvido nessa relação.

```
BEGIN       { FS = ":"; RS = "\n";IGNORECASE = 1; }
```

Tal como no ponto anterior, o *Field Separator* e *Record Separator* mantêm-se.

```
$1~/%inv/   {
                split($2,segundo," ");
                split($3,terceiro," ");
                inversos[segundo[1]] = terceiro[1];
            }
```

Depois de definirmos o *Field Separator* e o *Record Separator*, começamos por retirar todas as relações inversas, guardando-as num *array*, estas relações estão presentes nas linhas com a *tag %inv*. Antes de as guardarmos realizamos dois *split()*, de modo a tratar os espaços presentes nos ficheiros.

Guardar as relações inversas é importante para a criação dos triplos das relações inversas.

```
$1~/%THE/   {
                if($1~/</){
                    split($1,uni,"<");
                    unico = uni[2];
                }

                for(i=2; i<NF+1;i++){

                    if($i~/ ./ || $i~/.* /){
                        split($i,tmp," ");
                        if(tmp[1]~/</){
                            split(tmp[1],aux,"<");
                            relacoes[i] = aux[1];
                            instancia[i] = aux[2];
                        }
                        else if(tmp[1]~/@/){
                            split(tmp[1],aux,"@");
                            relacoes[i]= aux[2];
                        }
                        else{
                            relacoes[i]=tmp[1];
                        }
                    }
                    else if($i!=""){
                        if($i~/</){
                            split($i,tmp,"<");
                            relacoes[i]=tmp[1];
                            instancia[i]=tmp[2];
                        }
                        else if($i~/@/){
                            split($i,tmp,"@");
                            relacoes[i]=tmp[2]
                        }
                        else{
                            relacoes[i]=$i;
                        }
                    }
                }
            }
```

Para este ponto também precisamos de ir buscar as relações, de modo a conseguirmos formar os triplos, por

exemplo **Ficheiro riospt.mdic - (rio Cávado,nasce_em,serra do Larouco)** , para que tal aconteça temos de comecar por separar o primeiro campo, caso tenha caracteres especiais.

Pôs esta separação, realizamos um processo parecido ao do primeiro ponto, diferenciando-se pelo facto de não indexarmos o *array* por relação, e de não contarmos o número de ocorrências.

```
1!~/%/ &&  $1!~/#/ {
            if(unico!="" && $1!=""){
            print("("$1",iof,"unico")");
            }

            for(i=2; i<NF+1;i++){
                if($i~/|/){
                    split($i,triplos,"|");
                    for(t in triplos){
                        print("("$1","relacoes[i]","triplos[t]")");
                        if(instancia[i] != ""){
                            print("("triplos[t]",iof,"instancia[i]")");
                        }
                        for(inv in inversos){
                            if(relacoes[i]==inv){
                                print ("("triplos[t]","inversos[inv]","$1")");
                            }
                        }
                    }

                }
                else if($i!=""){
                        print("("$1","relacoes[i]","$i")");
                        if(instancia[i] != ""){
                            print("("$i",iof,"instancia[i]")");
                        }
                        for(inv in inversos){
                            if(relacoes[i]==inv){
                                print ("("$i","inversos[inv]","$1")");
                            }
                        }
                    }
            }
        }
```

A fase final da resolução do ponto, consiste em formar os triplos, e a sua respectiva impressão, para tal partirmos a informação caso haja mais que uma designação ou seja caso haja uma ou mais |, a medida que vamos percorrendo os *arrays* criados, vamos imprimindo a informação com a formatação pretendida.

Neste ponto como necessitamos da informação presente nas linhas sem as *tags* começadas % e sem as começadas por # *(Linhas comentadas)*, recorremos a **expressão regular** evidenciada em cima, de modo, a filtrar a informação pretendida.

Evidencio neste ponto a impressão das instanciâs além da impressão dos triplos inversos já referidos, isto é, por exemplo o seguinte triplo **Ficheiro riospt.mdic - (rio Cávado,iof,rio)**.

1.3 Mostrar a informacão contidas nos triplos, agrupadas pelo termo1 (formato Thesaurus ISO)

O último ponto do nosso enunciado, consiste em mostrar a informação anterior, mas agora com a formatação de um *ISO*.

```
BEGIN       { FS = ":"; RS = "\n";IGNORECASE=1; filename = "thesaurus.txt"; }
```

Tal como nos pontos anteriores, temos o mesmo *Field Separator* e *Record Separator*. A única diferença é que agora acrescentamos a váriavel *filename* que possui o nome do ficheiro onde vamos escrever a informação.

```
    $1~/%inv/   {
                split($2,segundo," ");
                while(segundo[1]~/ ./){
                    split(segundo[1],segundo," ");
                }
                split($3,terceiro," ");
                while(terceiro[1]~/ ./){
                    split(terceiro[1],terceiro," ");
                }
                inversos[segundo[1]] = terceiro[1];
            }
```

Tal como no segundo ponto, vamos retirar as relações inversas, só que neste caso devido a um erro detetado, tivemos de alterar o algorítmo, de modo, a remover os espaços atrás da palavra filtrada quer a frente da mesma. De resto a informação é guardada de forma indêntica.

```
$1~/%THE/   {
                if($1~/</){
                    split($1,uni,"<");
                    unico = uni[2];
                }

                for(i=2; i<NF+1;i++){

                    if($i~/ ./ || $i~/.* /){
                        split($i,tmp," ");
                        if(tmp[1]~/</){
                            split(tmp[1],aux,"<");
                            relacoes[i] = aux[1];
                            instancia[i] = aux[2];
                        }
                        else if(tmp[1]~/@/){
                            split(tmp[1],aux,"@");
                            relacoes[i]= aux[2];
                        }
                        else{
                            relacoes[i]=tmp[1];
                        }
```

```
                    }
                    else if($i!=""){
                        if($i~/</){
                            split($i,tmp,"<");
                            relacoes[i]=tmp[1];
                            instancia[i]=tmp[2];
                        }
                        else if($i~/@/){
                            split($i,tmp,"@");
                            relacoes[i]=tmp[2]
                        }
                        else{
                            relacoes[i]=$i;
                        }
                    }
                }
            }
```

Na fase de retirar e guardar as relações, o algorítmo não sofreu alterações em relação ao segundo ponto.

```
$1!~/%/ &&  $1!~/#/ && $1!=""{
    print($1)>filename;

    if(unico!="" && $1!=""){
        print("iof: "unico)> filename;
    }

    for(i=2; i<NF+1;i++){

        if($i~/|/){
            split($i,triplos,"|");
            for(t in triplos){
                print(relacoes[i]": "triplos[t])>filename;
            }
        }
        else if($i!=""){
            print(relacoes[i]": "$i)>filename;
        }
    }
    printf("\n")>filename;

    for(i=2;i<NF+1;i++){
        if($i!=""){
            if(inversos[relacoes[i]]!=""){
                if($i~/|/){
                    split($i,tripletes,"|");
                    for(t in tripletes){
                        print(tripletes[t])>filename;
                        print(inversos[relacoes[i]]": " $1)>filename;
                        if(instancia[i]!=""){
                            print("iof: "instancia[i])>filename;
                        }
                        printf("\n")>filename;
```

```
                    }
                }
                else{
                    print($i)>filename;
                    print(inversos[relacoes[i]]": " $1)>filename;
                    if(instancia[i]!=""){
                        print("iof: "instancia[i])>filename;
                }
            }
          }
        }
    }
}
```

Como evidenciado no algorítmo acima, a **expressão regular** vai filtrar todas as linhas que não comecem com % e # tal como anteriormente, só que desta vez filtamos só as linhas que não são vazias.

Neste ponto o todas as impressões são feitas para o ficheiro guardado na variável inicial. Como o formato de impressão é diferente tivemos que alterar o algoritmo anterior, de modo, a imprimimos o os triplos iniciais e só depois as relações inversas criamos dois ciclos que percorrem todos os campos, das linhas filtradas.

1.4 Construir um conjunto de páginas HTML (uma página por cada termo1) em que os termos2 hiperliguem às correspondentes páginas

Este ponto pertence ao enunciado 5, foi o ponto extra que decidimos implementar, visto estar relacionado com o nosso enunciado, uma vez que trabalhamos sobre os mesmos tipos de ficheiros.

Neste ponto o objectivo é construir páginas *HTML*, com informação presente nos ficheiros.

```
BEGIN       { FS = ":"; RS = "\n";IGNORECASE=1; filename = "thesaurus.html"; print("<html>")>filena
```

Começamos por definir uma variável com o tipo de ficheiro, e duas impressões para o respectivo ficheiro criado com os cabeçalhos presentes no *HTML*.

Tal como nos pontos anteriores, guardamos a informação relativa as relações inversas, repetindo o algoritmo feito anteriormente.

Seguindo o mesmo raciocínio anteriormente exposto, retiramos todas as relações existentes em cada ficheiro.

```
$1!~/%/ &&  $1!~/#/ && $1!=""{
        auxfile = $1".html";
        print("<html>")>auxfile;
        print("\t<body>")>auxfile;

        print("<a href=\""$1".html\">"$1"</a><br>")>filename;

        if(unico!="" && $1!=""){
            print("<p>"$1"</p>")>auxfile;
            print("\t\t<ul>")>auxfile;
```

```
            print("<li>iof: "unico"</li>")>auxfile;
        }

        for(i=2; i<NF+1;i++){

            if($i~/|/){
                split($i,triplos,"|");
                for(t in triplos){
                    print("<li>"relacoes[i]": "triplos[t]"</li>")>auxfile;
                }
            }
            else if($i!=""){
                    print("<li>"relacoes[i]": "$i"</li>")>auxfile;
                }
            }

            print("\t\t</ul>")>auxfile;
            print("<p> INVERSOS </p>")>auxfile;
            print("<p></p>")>auxfile;

            for(i=2;i<NF+1;i++){
                if($i!=""){
                    if(inversos[relacoes[i]]!=""){
                        if($i~/|/){
                            split($i,tripletes,"|");
                            for(t in tripletes){
                                print("<p>"tripletes[t]"</p>")>auxfile;
                                print("\t\t<ul>")>auxfile;
                                print("<li>"inversos[relacoes[i]]": " $1"</li>")>auxfile;
                                if(instancia[i]!=""){
                                    print("<li>iof: "instancia[i]"</li>")>auxfile;
                                }
                                print("\t\t</ul>")>auxfile;
                            }
                        }
                        else{
                            print("<p>"$i"</p>")>auxfile;
                            print("\t\t<ul>")>auxfile;
                            print("<li>"inversos[relacoes[i]]": " $1"</li>")>auxfile;
                            if(instancia[i]!=""){
                                print("<li>iof: "instancia[i]"</li>")>auxfile;
                            }
                            print("\t\t</ul>")>auxfile;
                        }
                    }
                }
            }
            print("\t</body>")>auxfile;
            print("</html>")>auxfile;
    }
```

Esta acção contêm todas as impressões de informação necessárias para gerar as páginas *HTML*, tal como aconteceu no ponto anterior percorremos os *arrays* criados a medida que percorremos as linhas filtradas pela **expressão regular** acima evidenciada.

Ao percorremos estas linhas vamos imprimindo a informação para o ficheiro, respeitando as normas de criação de uma página páginas *HTML*.

```
END {print("\t</body>")>filename;print("</html>")>filename;}
```

Por último terminamos a criação das páginas com a impressão das *tags HTML* para termirnar uma página.

Chapter 2

Conclusão

O uso do sistema de produção GAWK torna a implementação de um filtro de texto capaz de fazer o reconhecimento de padrões e proceder à sua transformação, um exercício de complexidade inferior àquela que o enunciado sugere. Em particular, tal como é descrito neste documento, é possível obter as mais diversas informações sobre o conteúdo de ficheiros, realizando com elas computações de grande utilidade para o utilizador/programador. Mais, é possível produzir conhecimento a partir de informação que se antevia desconexa (e.g., listagem total dos rios e serras portuguesas, saber onde nascem e desaguam e que cidades atravessam). Os resultados produzidos pelos filtros implementados impressionam pela sua precisão e utilidade, sobretudo quando se tem em conta o pequeno tamanho e complexidade dos scripts que os originaram. Os filtros implementados respondem aos requisitos impostos pelos enunciados (e não só), pelo que este grupo de trabalho se considera sobremaneira satisfeito. Como trabalho futuro, é evidente que esta ferramenta tem capacidade para produzir filtros diferentes de forma a captar informação alternativa á que foi capturada neste trabalho, portanto, mais funcionalidades podem ser implementadas.