Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 1

Relatório de Desenvolvimento

Alexandre Lopes Mandim da Silva
A73674

Luís Miguel da Cunha Lima
A74260

Hugo Alves Carvalho
A74219

15 de Março de 2017

Resumo

O presente projeto apresentado neste relatório foi desenvolvido no âmbito da Unidade Curricular de Processamento de Linguagens e tem como principal objetivo o desenvolvimento de um Processador de Texto com o sistema de produçao GAWK para ler um extrato mensal da Via Verde.

Ao longo deste relatório iremos explicar todo o processo de desenvolvimento e todas as decisões tomadas para a realização do trabalho.

Conteúdo

Capítulo 1

Introdução

No âmbito da unidade curricular de Processamento de Linguagens do 3º ano do Mestrado Integrado em Engenharia Informática foi proposta a realização de um trabalho prático com o objetivo de aumentar a experiência do uso do ambiente Linux e de ferramentas de apoio à programação; aumentar a capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases; desenvolver sistemática e automaticamente Processadores de Linguagens Regulares, que filtrem ou transformem textos e utilizar o sistema de produção para produção de filtragem de texto GAWK.

Neste contexto, o grupo escolheu o tema processador de transações de Via Verde, onde desenvolveu um Processador de Texto com GAWK capaz de ler e extrair informação de um extrato mensal da Via Verde.

Em primeiro lugar, fizemos uma análise do modo como estava construído o ficheiro de texto com os dados. Posteriormente, e com base nesta análise, implementamos as soluções necessárias para responder ao que foi pedido no enunciado do trabalho.

Neste relatório, será explicado o raciocínio que tivemos durante a execução do trabalho. Para isso, serão especificados os padrões de frases pretendemos encontrar no texto-fonte, através das duas respetivas expressões regulares, as estruturas de dados utilizadas e ainda a especificação do filtro de texto necessário para o reconhecimento dos padrões identificados, através do uso do GAWK.

Capítulo 2

Caso de Estudo: Via Verde

Para a realização deste trabalho prático o grupo decidiu abordar o caso da via verde.

Para podermos criar programas em GAWK e conseguirmos extrair a informação que pretendemos de qualquer tipo de ficheiro, inicialmente, é necessário uma observação e compreensão de como a informação está estruturada no ficheiro. Posto isto, essa foi a primeira abordagem do nosso grupo, observamos e estudamos o ficheiro XML que contem a informação e percebemos que inicialmente o ficheiro tinha informações sobre o cliente (nome, NIF, morada, etc.) e o respetivo extrato (data de emissão, ID do extrato, etc.), em seguida, as respetivas transações que o cliente fez, bem como todas as informações associadas a cada transação, finalmente, no fim do ficheiro existe informação sobre o montante que deve ser pago pelo cliente.

Agora que entendemos como o ficheiro estava organizado fica muito mais fácil tirar partido do GAWK e de expressões regulares para obtermos a informação necessária.

Neste caso específico, é-nos pedido, pelo enunciado, a criação de quatro implementações GAWK, que dessem solução ao que é pedido em cada alínea. O grupo decidiu também realizar mais algumas implementações para dar soluções a alguns problemas criados por nós.

Em seguida são apresentadas essas soluções e como foram implementadas.

2.1 Número de entradas de cada dia do mês

Nesta alínea é pedido que calculemos o número de entradas em cada dia dos meses.

Como temos dois meses, apresentamos para cada dia de cada respetivo mês o numero de entradas. Uma abordagem possível seria colocarmos o RS (Record Separator) com "<DATA_ENTRADA>" e o FS (Field Separator) como "[<>]", deste modo o campo que pretendíamos (o campo com a data de entrada) seria logo o primeiro (à exceção do primeiro registo que tem os dados do cliente).

No entanto, o grupo decidiu optar por outra abordagem, o FS ficou definido como foi indicado anteriormente, no entanto, definimos o RS como "<TRANSACCAO>". A razão de dividirmos os registos por transação foi porque achamos que esta solução era mais global (facilmente obteríamos a data, mas qualquer outro campo relacionado com uma transação) e mais organizada. A razão desta solução ser viável é o facto de que cada transação tem o mesmo número de campos e todos "no mesmo lugar", ou seja, a data é sempre o 3º campo, por exemplo. Posto isto, a data de entrada é sempre o registo $3 e facilmente obtemos a data de todos os registos. Assim sendo, o BEGIN do programa GAWK é o seguinte:

```
BEGIN {RS="<TRANSACCAO>";FS="[<>]"}
```

Agora queremos percorrer todos os registos e guardar os dias e mês de quando existiram transações. Desta forma, a única condição que temos no nosso programa GAWK é "NR > 1" isto deve-se ao facto de, como já foi mencionado, o 1º registo não é uma transação, mas sim informações sobre o extrato e o cliente, assim sendo só nos interessam do 2º registo em diante.

A ação realizada quando NR>1 é a seguinte:

```
NR>1    {split($3,data,"[-./]");
        resultado[data[2]][data[1]]++;
        }
```

O campo $3 tem a data, como queremos apenas o dia e o mês fazemos um split. A informação vai ser guardada numa matriz (mês X dia). Sempre que um dia num determinado mês é encontrado incrementa em um nessa posição na matriz. Desta forma, no final do ficheiro vamos ter uma matriz com as linhas a representar o mês e as colunas o dia e na matriz vai conter o numero de vezes que essa data foi encontrada no ficheiro.

Quando chegamos ao fim do ficheiro, só nos resta percorrer a matriz e apresentar ao utilizador a informação recolhida.

```
END{
    for(mes in resultado)
        for(dia in resultado[mes])
            print "Na data " dia "/" mes " existem "
            resultado[mes][dia] " entradas.";
}
```

2.2 Lista de locais de saída

Nesta segunda alínea é pedido para escrever a lista de todos os locais de saída da Via Verde dos vários meses.

O RS e o FS continuaram idênticos à alínea antecedente devido às mesmas razões esclarecidas previamente.

Queremos percorrer todos os registos (excetuando o primeiro pois não se trata de uma transação) e guardar todos os locais de saída. Assim, como na alínea anterior a única condição do nosso programa é "NR>1". Sempre que verificamos a condição anterior é verificada a ação seguinte:

```
NR>1    {
        resultado[$23];
        }
```

É guardado todos os campos $23, que correspondem aos valores dos vários locais de saída, numa *hash table*, sendo que só acrescenta esse valor se ainda não existir na *hash*.

Por fim, com todos os locais guardados mostramos ao utilizador a lista pretendida:

```
END{
    for(lugar in resultado) print lugar;
}
```

2.3 Total gasto no mês

Nesta alínea é pedido que calculemos o valor total gasto no mês.

Para podermos calcular o valor total gasto em cada transacção precisamos de ter em consideração a importância, o valor do desconto e a taxa do IVA. Para isso, e tal como já foi justificado anteriormente neste relatório, o grupo optou por uma abordagem com RS definido como "<TRANSACCAO>" e FS com "[<>]", uma vez que analisando o ficheiro de texto correspondente ao extrato mensal, facilmente conseguimos identificar quais os campos correspondentes a estes dados.

```
BEGIN {RS="<TRANSACCAO>";FS="[<>]"}
```

Analisando o ficheiro XML, podemos verificar que a importância corresponde sempre ao campo nº 27, o valor do desconto corresponde ao campo nº31 e a taxa de IVA ao campo nº 35.

Assim, uma vez que para realizar o cálculo do valor de cada transação é preciso utilizar a fórmula valor total = valor sem iva + valor sem iva * taxa iva, elaboramos um algoritmo para calcular este mesmo valor.

Como o ficheiro de texto contém os valores decimais separados por uma virgula, foi necessário fazer um split destes valores: no primeiro campo fica a parte inteira que somará com a parte decimal (valor do segundo campo * 0.01).

Seguidamente, o objetivo passa por percorrer todos os registos e fazer o somatório de todos os valores finais. Desta forma, e tal como já referido anteriormente neste relatório, utilizamos a condição GAWK "NR>1". Com o auxílio do array data, conseguimos realizar este cálculo da seguinte forma:

```
NR>1    {
        split($27,valorInicial,",");e=valorInicial[1];c=valorInicial[2]*0.01;
        split($31,valorInicial,",");de=valorInicial[1];dc=valorInicial[2]*0.01;
        valorSemIva = (e+c)-(de-dc);
        valorFinal = valorSemIva + valorSemIva*$35*0.01;

        split($3,data,"[-./]");
        resultado[data[2]] += valorFinal;
        }
```

O campo $3 tem a data, como queremos apenas o mês, basta usar um split e somar o valor final de cada registo à segunda posição do array.

Por último, quando chegamos ao fim do ficheiro, só nos resta percorrer o array e apresentar ao utilizador a informação recolhida.

```
END{
    for(mes in resultado)
        print "No mes " mes " gastamos " resultado[mes] " €.";
}
```

2.4 Total gasto em parques

Agora pretendemos calcular o valor total das transações que o seu tipo é portagens. Para esse efeito, mantemos o RS (por transação) e o FS igual ao das alíneas anteriores. Nesta alínea decidimos colocar a flag 'IGNORECASE' a um pois podia aparecer "Parque de estacionamento" ou "Parque de Estacionamento" e para evitar esse tipo de problemas decidimos colocar o GAWK case-insensitive.

```
BEGIN {RS="<TRANSACCAO>";FS="[<>]";IGNORECASE=1;total=0}
```

Uma vez que temos o BEGIN bem definido é hora de criar as condições às quais os registos do ficheiro vão ser sujeitos. Assim sendo, apenas temos uma condição: NR>1 && $43 ~ "Parque". Esta condição diz que apenas queremos analisar as transações (NR>1) e apenas as transações que são do tipo parque. Como o campo 43 é o campo que diz o tipo da transação basta pedir que este campo contenha (~) a string "Parque'.

Sempre que a condição explicada no paragrafo anterior é verificada, fazemos um split ao campo que contem o valor monetário e o desconto e calculamos o valor dessa transação (já explicado na alínea anterior) e acrescentamos a uma variável "total".

```
NR>1 && $43~"Parque"    {
    split($27,valorInicial,",");e=valorInicial[1];c=valo
    rInicial[2]*0.01;
    split($31,valorInicial,",");de=valorInicial[1];dc=va
    lorInicial[2]*0.01;
    valorSemIva = (e+c)-(de-dc);
    total += valorSemIva + valorSemIva*$35*0.01}
```

Por fim, com o total calculado mostramos ao utilizador esse valor:

```
END{print "No mes gastou " total " €."}
```

2.5 Dinheiro que cada operador ganhou

Após a realização das quatro alíneas pedidas no enunciado, o grupo decidiu realizar mais duas alíneas: X e Y.

Nesta alínea queremos determinar o dinheiro que cada operador ganhou.

O RS e o FS continuaram os mesmo das alíneas anteriores devido às mesmas razões explicadas anteriormente.

A condição para cada registo é que o registo tem que se tratar de uma transação, ou seja, 'NR>1'.

Em cada registo vamos obter o valor total da transação e vamos guardar numa *hash table* em que a key é o nome de cada operador e o respeito valor corresponde ao total cobrado por esse operador:

```
BEGIN {RS="<TRANSACCAO>";FS="[<>]"}
NR>1    {
        split($27,valorInicial,",");e=valorInicial[1];c=
        valorInicial[2]*0.01;
        split($31,valorInicial,",");de=valorInicial[1];d
        c=valorInicial[2]*0.01;
        valorSemIva = (e+c)-(de-dc);

        operadores[$39] += valorSemIva;
        }
```

Finalmente é so iterar sobre a hash table e mostrar os valores finais.

```
END{
    for(operador in operadores)
        print "O operador " operador " ganhou "
        operadores[operador] " €.";
}
```

2.6 Saída mais frequente

Neste caso queremos determinar a saída que tem mais registos, ou seja, por onde o utilizador saiu mais vezes.

Para resolvermos este problema seguimos a seguinte abordagem: guardamos numa *hash table* todas as saídas e respetivo número de vezes que foram saídas e apenas no fim de processar o ficheiro todo é que iteramos sobre a *hash table* e guardamos numa variável a saída com mais registos. Em caso de empate no número de saídas é mostrada a saída que se encontrar primeiro na tabela de *hash*.

```
BEGIN {RS="<TRANSACCAO>";FS="[<>]";max=0;}
NR>1    {
        saidas[$23]++;
        }
END{
    for(saida in saidas)
        print "Saida: " saida " -> " saidas[saida];
        if(saidas[saida] > max){
            resultado = saida;
            max = saidas[saida];
        }
        print "A saida mais frequente é " resultado;
}
```

Capítulo 3

Conclusão

Terminada a realização deste projeto e, de uma forma global, o balanço do trabalho desenvolvido é bastante positivo. As funcionalidades propostas pelo enunciado foram implementadas com sucesso, juntamente com a adição de algumas funcionalidades extra.

A execução deste projeto foi fundamental para consolidar a matéria lecionada quer nas aulas praticas como nas teóricas, uma vez que as técnicas de utilização de expressões regulares aí aprendidas facilitou a implementação do problema.

Foi ainda possível ao grupo, praticar com o Sistema de produção GAWK bem como identificar padrões de frase, de forma a resolver o problema da melhor forma, e as estruturas de dados que necessitamos para armazenar a informação.

De uma forma geral, os resultados produzidos foram bastante satisfatórios e enriquecedores para todos os elementos do grupo pois permitiu consolidar conhecimentos adquiridos bem como identificar algumas lacunas que temos de corrigir futuramente.