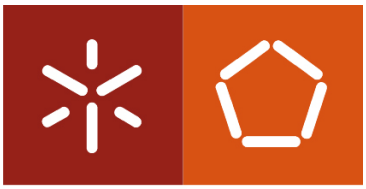

Escola de Engenharia

Universidade do Minho

Processamento de Linguagens

Segundo Trabalho Prático

GAWK

A78485 - Diogo Araújo, A78957 - Diogo Nogueira

29 de Abril de 2019

Resumo

Este segundo trabalho prático surge no âmbito da Unidade Curricular de Processamento de Linguagens e destina-se essencialmente à utilização da linguagem de programação **GAWK**, destinada a processar texto e extrair do mesmo as informações desejadas. Através dele, pretende-se escrever funções que depois permitam filtrar/-transformar todo um ficheiro de texto `.csv` que tem como divisão e organização por colunas e linhas.

Conteúdo

Capítulo 1

Processador de Processos de Formação

Neste capítulo irá ser apresentado, de um modo geral, aquilo que é o trabalho proposto em si e o problema que se pretende ver resolvido.

1.1 Enunciado

O ficheiro para os Processos de Formação consiste num conjunto de processos que se encontram parametrizadas por um conjunto de colunas tal como Estado, Código, Título, Descrição mas também colunas que interligam com Leis e Diplomas. Além desta base, existe também o texto correspondente a cada uma dessas. Este conjunto de informação é então universal a todos os Processos de Formação e é sobre este padrão que se pretende manipular o ficheiro `.csv` que agrupa toda esta informação por filtrar.

1.2 Descrição do Problema

Como já foi referido numa fase inicial, todo este projeto baseia-se na interpretação, filtragem e posterior transformação de texto, proveniente de um ficheiro que aglomera todos os Processos de Formação. Por ser um ficheiro que enumera aquilo que define um Processo, é perfeitamente normal existirem informações que depois se tornam irrelevantes ao olhos de quem pretende manipular este conteúdo. Daí surgir a necessidade de interpretar primeiro. Observar aquilo que realmente importa e pensar também em termos de GAWK, nos diversos padrões de frases e estruturação do ficheiro, tentando já facilitar a maneira como o algoritmo pode ser escrito e pensado. Assim conseguimos criar ações semânticas para serem executadas após ser analisados os padrões criados para os diversos propósitos. Ao fazer-se isto, consegue-se eliminar a informação desnecessária e responder a todas as alíneas do projeto.

Perante esta descrição e nunca esquecendo os requisitos principais do processo de filtragem de informação, irá ser realizada uma análise do problema com explicações pormenorizadas sobre como pensamos no código GAWK utilizando todo o potencial do mesmo para responder às várias questões pedidas sobre o projeto.

Capítulo 2

Decisões e Implementação

2.1 Análise do Problema

O tipo de ficheiro que se pretende manipular é caracterizado por um conjunto de informação de vários processos que assentam sobre uma base universal daquilo que distingue o conteúdo entre cada um deles. A análise de todo o problema enunciado passa precisamente por perceber aquilo que delimita cada registo/campo de modo a que se possa ir lendo todos os processos, filtrando apenas a informação que realmente é precisa.

Sabe-se que o GAWK possui de raiz um conjunto de *Built-in Variables*, que servem para definir aquilo que separa cada campo e também cada registo dos vários processos contidos neste tipo de ficheiros (entre outras finalidades). Com o uso destas variáveis consegue-se ler processo a processo, de modo a manipular os vários campos que cada um deles contém. Tomando um processo como referência, nota-se desde logo esta igualdade na disposição dos parâmetros. É precisamente sobre este padrão que vai ser feita a limpeza inicial do ficheiro, removendo linhas extras vazias, processos cujos campos estejam todos a nulo e ainda possíveis caracteres desnecessários para o processamento em si.

Veja-se então um extrato daquilo que é considerado uma processo. Note-se que este extrato apenas apresenta pequenas partes referente a alguns dos campos, uma vez que apenas se pretende aferir como pode ser feita a remoção do conteúdo desnecessário. (As cores identificam os vários campos)

```
;750.10.001;Seleção e seriação (...);"Candidatos selecionados ou (...)";
"Lei 45/2013#
Lei 45/2012# (...)";DL 50/98 alterado pelos DL 70-A/2000 (...)#;PC;S ;"DGES#
AP#";"IES#
AP#";"Comunicar#
Assessorar#;
"150.20.001#
150.20.101#
(...)";(...);Elevada;N;;;;;;;;;;;;;;;;

;750.10.001.01;Seleção e seriação para ingresso (...);;;;;;;
```

Analisando os dois registos acima transcritos, facilmente se percebe que os vários campos do processo estão a ser separados pelo caractér **;** e que os registos em si se encontram distanciados por uma mudança de linha. Esta observação permite definir de imediato as variáveis **FS (*Field Separator*)** e **RS(*Record Separator*)** do nosso ficheiro não processado.

1. O primeiro campo de cada ação de formação é sempre vazio;
2. Alguns campos encontram-se delimitados pelas aspas (que começam/terminam assim que o **FS** é acionado);
3. Existe a presença do caracter # em alguns campos do registo - caracter que serve de separador entre as várias enumerações presentes nesse campo;
4. Existe a possibilidade de um campo se encontrar vazio;

Foi sobre esta base de pensamento que as expressões regulares para a filtragem/limpeza de informação do ficheiro inicial foram desenvolvidas. Na subsecção dedicada à Implementação do Código GAWK irá ser debatida toda a temática da resolução dos requisitos e de que modo esta análise inicial contribui para mitigar tudo isso. Vão ser também abordados os detalhes que entretanto foram detetatos no ficheiro e as mudanças que essa descoberta implicou em termos de processamento do mesmo.

2.2 Implementação Código GAWK

Ficou claro que todo o processo de reconstrução do ficheiro de dados inicial depende da maneira como esse mesmo ficheiro está estruturado. Nesta fase, pretende-se organizar e ao mesmo tempo descartar informação para que depois a resposta às restantes alíneas e o desenho do grafo final sejam mais simples, poupando-se também no processamento extra de informação.

2.2.1 Limpeza do Ficheiro de Dados

A limpeza do ficheiro de dados é o passo principal de todo o projeto, já que é sobre este ficheiro préprocessado que se pretende que as próximas alíneas sejam resolvidas. Perante a análise anteriormente exercida, consegue-se produzir quase a totalidade da limpeza em si, existindo no entanto pormenores que complicam a resposta aos vários requisitos precisando de uma solução que os contorne.

À medida que se foi observando o ficheiro de dados (tanto através do *Visual Studio Code* como do *Excel*), percebeu-se que o penúltimo campo de alguns registos poderiam possuir na sua constituição o caracter que está a servir de ***Field Separator***. Isto gera uma confusão no *parse* do ficheiro, existindo momentos em que são criados novos campos no registo quando na verdade esses novos campos fazem parte todos do mesmo *field*.

Foi com esta análise extra, que se consegui uma limpeza total do ficheiro, adaptada às futuras necessidades e que vai ao encontro das decisões estabelecidas pelo grupo.

Em termos de *source-code*, a nossa especificação GAWK começa com a definição das variáveis ***Field Separator*** e ***Record Separator***, declaradas na seção BEGIN.

```
BEGIN {
    RS="\n;" # Definir o que separa os vários registos.
    FS=";" # Definir o que separa os vários campos.
}
```

É sobre estas definições de valores que toda a limpeza do ficheiro deve ser feita. Dado que a primeira linha do nosso ficheiro `.csv` corresponde ao nome dos vários campos, faz-se logo uma remoção, usando para isso uma condição que obriga o interpretador do GAWK a ignorar essa linha.

```
NR==1 {next}
```

Ignorada a primeira linha, começa o processamento dos registos em si. Como se sabe a quantidade exata de registos presentes no ficheiro, executa-se a mesma condição para todos eles, terminando a leitura do ficheiro quando for alcançado o último. Ao estabelecer-se esta condição está-se já a remover todos os registos que apresentam todos os seus campos a nulos (dado que isto acontece imediatamente a seguir ao último registo preenchido).

```
NR<=30 {
    i=1;
```

Pensando no novo ficheiro `.csv` que deve ser criado, foram tomadas certas decisões que tiveram de ser implementadas dentro dessa condição.

Note-se que toda a especificação AWK está a ser progressivamente anexada consoante as decisões que foram tomadas para a limpeza do ficheiro.

1. Dado que se sabe que todos os registos apresentam o 'Estado' vazio, esse campo toma o valor 'NIL' para todos eles. Para isso, imprime-se o valor 'NILL' sempre que se começa a leitura de um novo registo. Esta ação permite que o novo ficheiro possa ter como ***Field Separator*** esse mesmo valor. Com esta decisão vai-se facilitar em muito a resposta às alíneas propostas, uma vez que o campo 'Estado' é irrelevante para a interpretação da ação de formação, podendo ser descartado como *field*;

    ```
    # Sempre que o primeiro campo está vazio, colocamos como NIL.
    # Sabemos que o primeiro campo é sempre vazio.

    if($1~"") print("NILL");
    ```

2. Remove-se a ocorrências das aspas que vão surgindo ao longo de todo o texto, poupando linhas brancas que muitas das vezes estão a ser ocupadas apenas por este caracter;

    ```
    # Eliminar ocorrência do caractér " (aspas).
    # Conveniente para o trabalho em si.

    gsub("\"","")
    ```

3. Com a análise do ficheiro de dados via *Excel*, percebeu-se que em alguns campos o texto continha bastantes linhas em branco. No decorrer de algumas pesquisas, chegou-se à conclusão que estas linhas eram representadas sob a forma de **\r\n**, e que por isso a substituição tinha de ser feita para todas as ocorrências deste caracter;

```
# Substituir linhas em branco extras por apenas um espaço.

gsub("\r\n"," ")
```

Importante declarar que por vezes, existiam mudanças de linhas normais pelos textos. No entanto, ao fazer-se esta remoção/substituição, apenas se tornou o texto mais compacto, não eliminando qualquer informação importante.

4. Como se tomou a decisão de que o campo 'Estado' acabaria por não ser tratado como um *field*, teve de se ignorar este campo sempre que o conteúdo do registo estava a ser manipulado (decisão refletida no ciclo *while*);

```
# Só contamos 26 fields. O Estado não está a ser tratado como field.

while(i<=26){
```

5. O penúltimo campo 'Justificação DF' apresenta muitas das vezes dentro do seu texto o caracter que define o ***Field Separator***. Tínhamos visto mais acima que isto era um problema que deveria ser resolvido. A solução passou por ir acumulando os vários novos campos criados a partir do campo 'Justificação DF' incial. Assim, sabíamos que todos esses campos que desnecessariamente estavam a existir, eram incluídos apenas no inicial;

```
# Sempre que estamos no penúltimo field.
# E sempre que esse registo não seja o número 17.

if(i==25 && NR!=17){

    if($25=="" && $26==""){
        print("NOVOCAMPO - VAZIO");
    }
    else{
        flag=0;
        printf("NOVOCAMPO - ");

        while(flag==0){
            if($i!="") printf($i);
            else {flag=1; print("\nNOVOCAMPO - VAZIO");}
            i++;
        }
    } i++;
}
```

6. A implementação abordada anteriormente funciona porque se sabe que o campo 'Notas' (que segue o campo 'Justificação DF' e finaliza o registo), é sempre vazio. No entanto, isto não é assim tão linear. O registo número 17 apresenta o campo 'Notas' preenchido, mas em contrapartida o seu campo 'Justificação DF' não contém o caractér **;**. Isto facilita muito as coisas porque assim pode-se aplicar a mesma resolução anterior, mas desta vez apenas para o campo 'Notas' desse tal registo.

```
# Sempre que estamos no último field.
# E sempre que esse registo seja o número 17.
# Sabemos que este registo contém o último field preenchido.

if(i==26 && NR==17){

    flag=0;
    printf("NOVOCAMPO - ");

    while(flag==0){
        if($i!="") printf($i);
        else {flag=1; printf("\n"); break;}
        i++;
    }
}
```

7. Para os restantes *fields* apenas se filtra o texto em si, não existindo qualquer exceção ou preocupação em termos de processamento de conteúdo.

```
        # Caso não seja nenhum destes fields.

        else{
            # Sempre que o campo é vazio, colocamos como VAZIO.
            if($i=="") print("NOVOCAMPO - VAZIO");
            # Caso contrário, imprimimos o campo em si.
            else printf("NOVOCAMPO - %s\n", $i);
        } i++;
    }
}
```

Com a filtragem de informação e preparação de dados consegui-se pré-processar todo o ficheiro, adaptando-o às necessidades exigidas nas alíneas seguintes e respeitando sempre o requisito inicial. O gerar deste novo ficheiro passa pela execução de um comando que guarde num outro ficheiro (de nome arbitário) as alterações definidas na implementação GAWK.

```
awk -f cleanFile formacao.csv > ficheiroLimpo.csv
```

2.2.2 Processamento do Ficheiro Limpo

As alíneas que utilizam a limpeza do ficheiro, requerem uma resolução baseada nesse novo ficheiro, ao mesmo tempo que permitem desenvolver a capacidade de manipular a informação de um `.csv`. Para cada um das alíneas, foi criado um ficheiro diferente, de modo a processar individualmente cada um dos pedidos.

Considerando que o ficheiro limpo possui novos separadores de campos e registos, foi preciso redefinir as variáveis ***Field Separator*** e ***Record Separator***.

Note-se que esta secção BEGIN é então universal a ambos os processamentos.

```
BEGIN {
    RS="NILL" # Definir o que separa os vários registos.
    FS="NOVOCAMPO - " # Definir o que separa os vários campos.
}
```

1. **Processamento 1 - Mostrar para cada uma das ações de formação, o seu Código, Título, Descrição e Notas.**

   Atendendo a que o novo ficheiro possui logo na sua primeira linha o valor 'NILL', esta precisa de ser ignorada, caso contrário o processamento do ficheiro é feito de maneira errada, não se conseguindo gerar o resultado pretendido.

   ```
   NR==1 {next}
   ```

   Para processar o conjunto de todos os registos, o procedimento é simples. Apenas é necessário imprimir para um novo ficheiro os *values* dos *fields* que se quer obter. Esta impressão é feita para todos os registos e apenas para os campos que não se enontrem vazios.

   ```
   NR<=30 {

       i=1;

       # Cada registo tem agora 27 fields.
       # Dado que estamos agora a trabalhar com o ficheiro .csv limpo.

       while(i<=27){

           if(i==2 && $i!="") printf("Ação de formação de código número: %s", $i);
           if(i==3 && $i!="") printf("Título: %s", $i);
           if(i==4 && $i!="") printf("Descrição: %s", $i);
           if(i==27 && $i~"VAZIO") printf("Não existem notas a exibir.\n\n");
           if(i==27 && $i!~"VAZIO") printf("Notas: %s.\n\n", $i);

           i++;
       }
   }
   ```

2. **Processamento 2 - Identificar os diferentes tipos de processo, calculando o número de processos por cada tipo.**

   A resposta a este requisito exigiu outro tipo de pensamento, dado que não só era preciso iterar os vários registos (verificando o tipo de processo associado a cada) como ir contando a quantidade de registos associados aos diferentes tipos de processo.

   Antes mesmo de se criar a secção BEGIN, declararam-se duas mini funções que fizeram todo o trabalho de contagem de registos por cada tipo de processo. A função `contaTipoProcessos` recebe o array que contém todos os tipos de processos existintes no ficheiro, contando o número de processos associados a um dado tipo que é passado como parâmetro.

```
function contaTipoProcessos(array, indice){

    contador=0;

    for(i in array){
        if(array[i]==array[indice]) contador++;
    }

    return contador;
}
```

   A função `existeTipoProcesso` verifica se um certo tipo de processo (obtido pelo array dos tipos e respetivo índice) existe no novo array.

```
function existeTipoProcesso(novoArray, array, indice){

    for(x in novoArray){
        if(novoArray[x]==array[indice]) return 1;
    }

    return 0;
}
```

   Estas duas funções são de extrema importância para o funcionamento de toda a especificação GAWK. Analisando de seguida esta especificação consegue-se perceber melhor a funcionalidade oferecida por cada uma destas funções e os parâmetros que cada uma delas está realmente a receber/interpretar.

   Igualmente ao que acontece no primeiro processamento, é preciso ignorar a primeira linha do ficheiro limpo.

```
NR==1 {next}
```

Quanto aos registos, vai-se acumulando num array todos os valores do campo 'Tipo de Processo' - array este que será lido na secção END que finaliza a especificação AWK. Apenas se guardam os valores que não sejam vazios, dado que estes não podem sequer ser incluidos na contagem.

```
NR<=30 {

    i=1; j;

    # Colocamos num array o tipo de processo de cada registo.

    while(i<=27){

        if(i==10 && $10!~"VAZIO") array[j++]=$10;
        i++;
    }
}
```

Dado que este array guardou o tipo de processo de cada um dos registos e que estes podem partilhar tipos entre si, acabaram por se criar repetições em alguns índices. Foi por essa razão que não se imprimiu para o ficheiro enquanto o *parse* estava a ser feito, existindo a necessidade de criar um novo array que eliminasse todas as repetições do array inicial.

Com esta necessidade surge a criação das duas funções a princípio mencionadas. A metodologia do trabalho conjunto de ambas passa por iterar o array anterior, colocando num novo array os vários tipos de processo que ainda não existam nesse novo. Se isto se verificar, ao inserir no novo array, faz-se também a contagem dos registos que se associam esse tipo de processo, imprimindo finalmente para o ficheiro.

```
END{

    k; contador;

    for(indice in array){

        if (!existeTipoProcesso(novoArray, array, indice)){
            novoArray[k]=array[indice];
            contador=contaTipoProcessos(array, indice);

            printf("Processo do tipo %s", novoArray[k]);
            printf("Número de processos associados: %i\n\n", contador);

            k++;
        }
    }
}
```

O comando para execução destes dois requisitos é similar para todo o projeto. A diferença é que aqui já se pode guardar o resultado para um ficheiro .txt facilitando assim a leitura dos resultados.

```
awk -f processFile1 ficheiroLimpo.csv > resultado1.txt
awk -f processFile2 ficheiroLimpo.csv > resultado2.txt
```

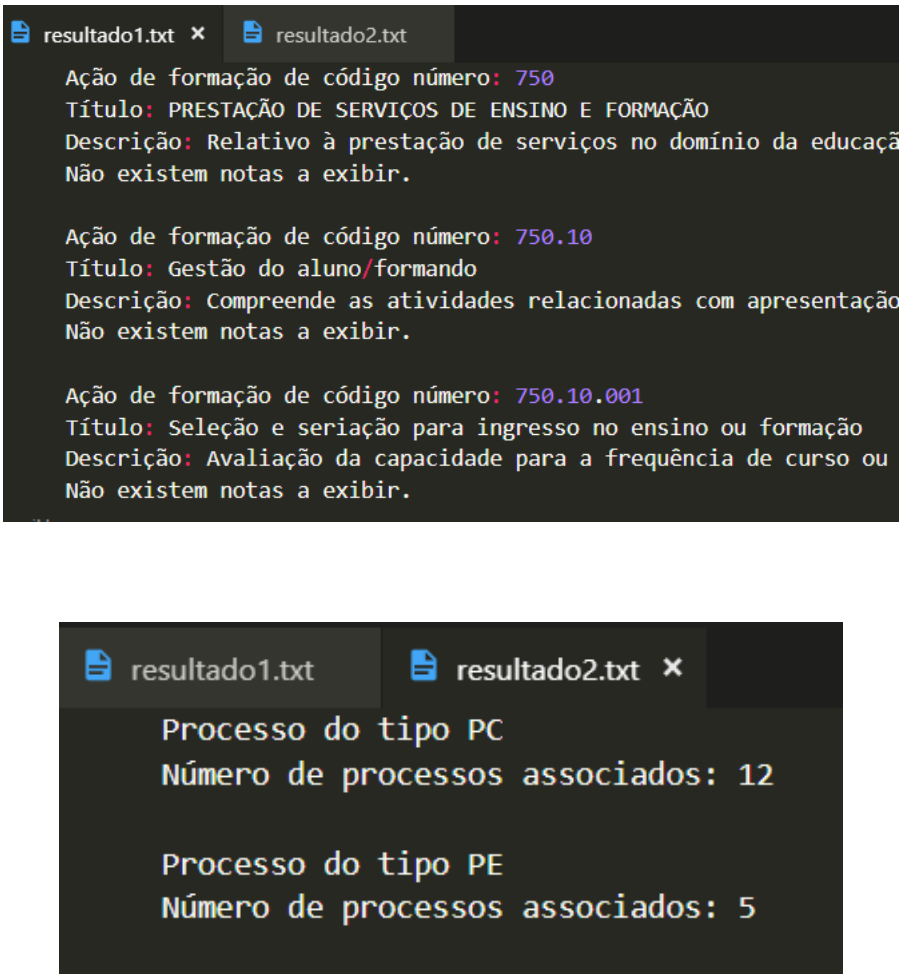

Figura 2.1: O *output* parcial gerado através da execução dos dois comandos acima.

2.2.3 Desenho do grafo em DOT

O desenho do grafo tornou-se das etapas mais simples de todo o projeto. Conhecendo-se a maneira como o código do grafo deve ser criado, foi apenas necessário filtrar a informação relevante, imprimindo-a para o grafo em si.

Apesar de ser uma etapa mais descomplicada, influenciou diretamente nas decisões tomadas para o limpeza do ficheiro fornecido. Relembre-se que esse ficheiro continha em vários segmentos o caracter # e que esse caracter não foi eliminado considerando que acabava por servir de separador entre as enumerações presentes em alguns dos *fields*. Esse caracter foi importante para o desenho do grafo dado que permitiu distinguir os diferentes diplomas presentes no campo em si.

Nesta especificação, a variáveis iniciais mantêm-se as mesmas. Ignora-se do mesmo modo o primeiro registo mas aproveita-se essa condição para imprimir o início da criação do *source-code* do grafo.

```
BEGIN {
    RS="NILL" # Definir o que separa os vários registos.
    FS="NOVOCAMPO - " # Definir o que separa os vários campos.
}

NR==1 {
    print "digraph grafo{" > "grafo.dot"

}
```

Assim, faz-se a substituição do carácter # por uma mudança de linha de modo a apresentar todos os diplomas por cada nodo de um modo mais percetível. Eliminam-se também as mudanças de linhas seguidas de múltiplos espaços que estavam a afetar a criação dos nodos do grafo (tornando alguns deles mais extensos apenas com linhas em branco). Com esta limpeza extra feita, imprime-se para o ficheiro do grafo os dois tipos de diplomas de cada ação de formação (caso existam).

```
NR<=30 {

    # Substituir o delimitador # por uma mudança de linha.
    # Assim imprime-se no nodo sem este carácter desnecessário.
    gsub("#", "\n")

    # Eliminar ocorrências de linhas em branco/espaços
    # desnecessários que possam existir.
    gsub("\n[ ]*\n", "")

    if($8!~"VAZIO" && $2!="")
        print "\""$2"\"" "->" "\""$8"\"" ";" > "grafo.dot"

    if($9!~"VAZIO" && $2!="")
        print "\""$2"\"" "->" "\""$9"\"" ";" > "grafo.dot"
}
```

Quando todos os registos forem iterados e os nodos completamente escritos no ficheiro *.dot*, encerra-se o grafo em si.

```
END{
    print "}\n" > "grafo.dot"
}
```

Esta especificação GAWK funciona perante aquilo que se pretende visualizar em termos de grafo, reproduzindo assim as relações que cada processo tem aos vários diplomas do tipo REF e Complementar.

```
awk -f processDOT ficheiroLimpo.csv
```

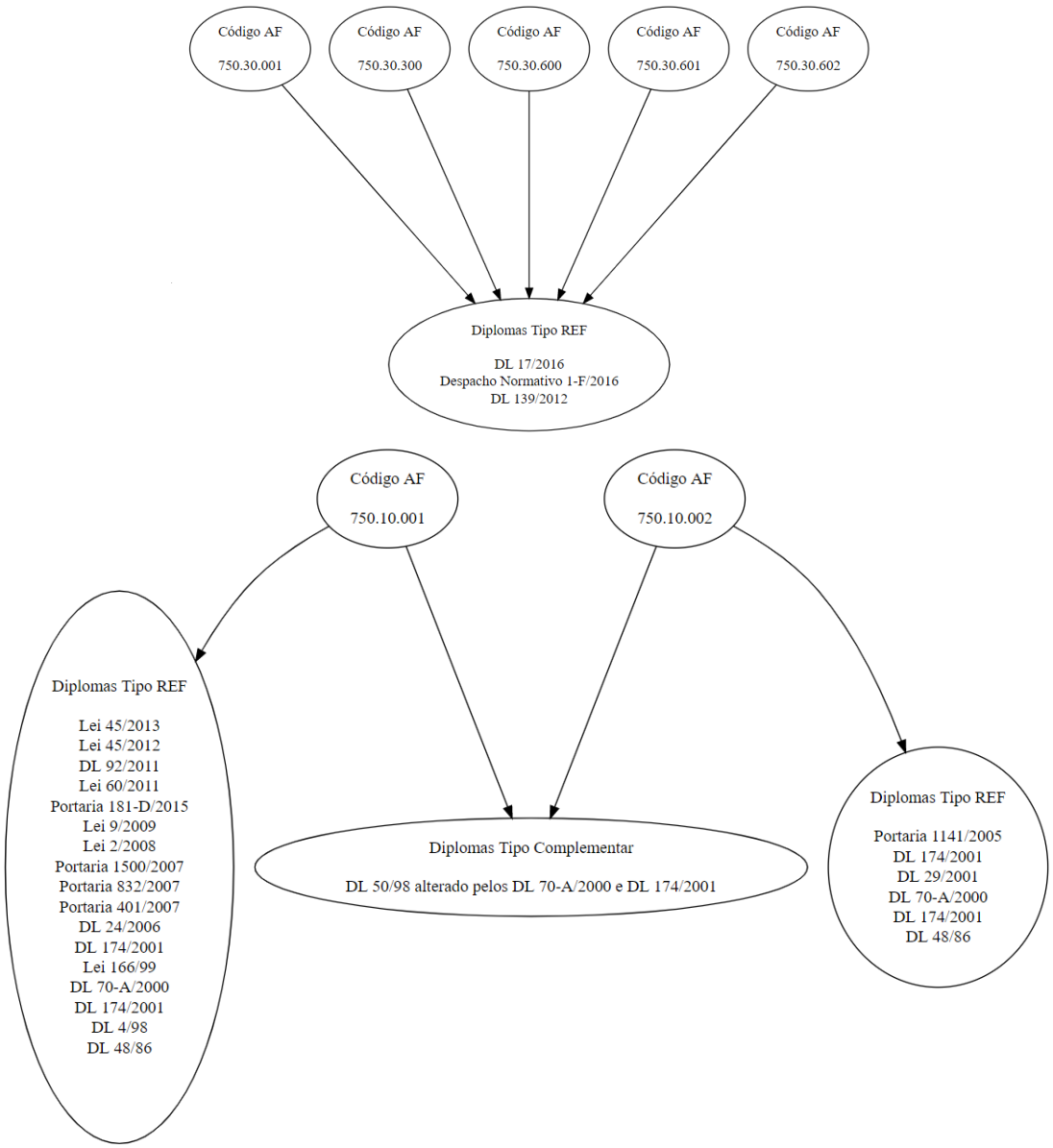

Figura 2.2: Demonstração gráfica do grafo DOT com as ligações entre os Processos e os seus Diplomas

Capítulo 3

Considerações Finais e Trabalho Futuro

A ideia por detrás da realização de todo o projeto foi, desde o início, solucionar os requisitos de uma forma sequencial. Após se conseguir fazer o *parse* de todo o ficheiro `.csv`, filtrando através da linguagem **GAWK**, todos os campos necessários não só para a limpeza do ficheiro mas também para acumular e mostrar os dados efetivos que seriam necessários para responder ao resto das alíneas pedidas.

A linguagem **GAWK** é imensamente poderosa para tipos de ficheiros que se baseiem em linhas e colunas bem delineadas, como forma estrutural do mesmo. Com isso foi possível existir uma limpeza do ficheiro de forma eficaz sendo que foi considerado inúmeras funcionalidades da linguagem, tal como o ***FS*** (*Field Separator*) e o ***RS*** (*Record Separator*) que foram fundamentais para todo o processo de filtração, processamento e extração dos dados necessários para a limpeza.

Ao extrair os dados necessários para um ficheiro limpo teve de existir um cuidado maior e um estudo sobre a capacidade que a linguagem tinha de forma a conseguir ter acesso a algoritmos/métodos predefinidos mas também a criação de funções e ciclos ***for*** para conseguir desta maneira responder corretamente às questões das alíneas seguintes.

Como trabalho futuro, tem de existir um estudo para uma melhor gestão e pensamento sobre como interligar as várias fases do trabalho em **GAWK** de forma a não ser preciso a criação de vários ficheiros de filtração e processamento do texto e tentar fazer tudo o mais compacto possível.