

Processamento de Linguagens (3o Ano de Mestrado Integrado em
Engenharia Informática)

Trabalho Prático 1

Relatório de Desenvolvimento

Diogo André Teles Fernandes
(A78867)

João Miguel Freitas Palmeira
(A73864)

Luís Manuel Meruje Ferreira
(A78607)

25 de Março de 2018

Resumo

Este trabalho prático tem como principais objetivos aprofundar e aplicar os conhecimentos obtidos durante as aulas teóricas e práticas de Processamento de Linguagens. Tem o intuito de aumentar a nossa capacidade de escrever Expressões Regulares (ER) para descrição de padrões de frases e de, através destas, desenvolver Processadores de Linguagens Regulares que filtrem ou transformem textos. Para executar o projeto, recorreremos ao sistema de produção para filtragem de texto *GAWK*.

Foi-nos atribuído o último dos cinco enunciados disponíveis para este primeiro trabalho prático, mais concretamente, o referente ao Processador de *Thesaurus* 2. Os resultados obtidos estão de acordo com os objetivos previamente definidos e com o que foi pedido no enunciado.

Conteúdo

1	Introdução	3
1.1	Processador de <i>Thesaurus</i> 2	3
2	Análise e Especificação	4
2.1	Descrição informal do problema	4
2.2	Especificação do Requisitos	4
3	Conceção/desenho da Resolução	5
3.1	Algoritmos	5
3.1.1	Determine a lista dos domínios e das relações usadas	5
3.1.2	Mostre os triplos expandidos correspondentes (um triplo por linha)	5
3.1.3	Triplos Expandidos - Versão HTML(Extra)	6
3.1.4	Construa um conjunto de páginas <i>HTML</i> (uma página por cada termo ₁) em que os termos ₂ hiperliguem às correspondentes páginas	7
3.1.5	Calcular o número de ocorrências de uma relação(Extra)	7
4	Codificação e Testes	8
4.1	Alternativas, Decisões e Problemas de Implementação	8
4.2	Testes realizados e Resultados	8
5	Conclusão	14
A	Código do Programa	15

Lista de Figuras

4.1	Exemplo de lista de domínios e relações de um dos ficheiros fornecidos	8
4.2	Exemplo de triplos obtidos a partir de um ficheiro fornecido	9
4.3	Exemplo de páginas <i>HTML</i> geradas para cada termo	10
4.4	Exemplo de página <i>HTML</i> de um termo selecionado	10
4.5	Exemplo do cálculo do número total de relações de um ficheiro	11
4.6	Exemplo de páginas <i>HTML</i> geradas para os triplos do ficheiro <i>profissoes-lojas.mdic</i>	11
4.7	Exemplo de páginas <i>HTML</i> geradas para os triplos do ficheiro <i>profissoes-lojas.mdic</i> com <i>IOFSs</i>	12
4.8	Exemplo de ficheiro <i>CSS</i> gerado para os <i>IOFs</i> do ficheiro <i>mdic</i>	13

Capítulo 1

Introdução

1.1 Processador de *Thesaurus 2*

Área: Processamento de Linguagens

Neste primeiro trabalho prático, foram-nos fornecidos cinco enunciados diferentes e devido ao método de seleção imposto, a alternativa que nos foi destacada foi a última, denominada "Processador de *Thesaurus 2*".

O Processador de *Thesaurus 2* recebe um conjunto de ficheiros *mdic*, cada um deles com diferentes informações sobre termos de diferentes temáticas e com as relações que interligam esses mesmos termos. Coube-nos a nós criar as diferentes funcionalidades deste processador de maneira a que os dados recebidos sejam tratados e que sejam apresentados nas diferentes vertentes que nos foram pedidas. Com esse objetivo em mente, desenvolvemos um processador de texto com o *GAWK* para ler um ou mais ficheiros do tipo fornecido e retirar deles toda a informação pretendida.

Estrutura do Relatório

Este relatório está organizado em cinco capítulos, seguidos de anexos, sendo que nestes últimos é apresentado todo o código desenvolvido. No início do documento existe também um resumo do mesmo.

O capítulo 1 consiste numa introdução em que é feita a descrição dos objetivos propostos pelo enunciado escolhido e onde se encontra a estrutura do relatório. No capítulo 2, é inicialmente feita uma descrição informal do problema proposto, traçando as linhas gerais do mesmo e de seguida são especificados os requisitos concretos que devem ser cumpridos.

De seguida, o capítulo 3 aborda todas as opções tomadas ao longo da realização do trabalho prático, assim como as decisões que lideraram o desenho da solução e da sua implementação.

No quarto capítulo é explicado o porquê de termos optado por abordar cada alínea do modo que abordamos explicando as dificuldades e os problemas que ocorreram para a realização da mesma. Colocamos também ainda alguns testes realizados.

Por fim, o capítulo 5 inclui uma síntese do trabalho realizado com uma análise crítica dos resultados obtidos, possíveis melhorias e ainda trabalho futuro que poderia ser desenvolvido.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

Este enunciado tem como objeto de processamento ficheiros de texto com uma estrutura simples e pré-definida (*.mdic*) que representam nas suas entradas a informação dos triplos (termo1, rel, termo2), além dos domínios dos mesmos e de eventuais classes de relações. Deste ficheiro, o problema principal consiste em extrair a informação necessária capaz de identificar os domínios, relações e classes presentes nestes ficheiros e associar os mesmos aos termos correspondentes, criando assim um *Thesaurus*. Além disto existem comentários nestes ficheiros que devem ser ignorados.

2.2 Especificação do Requisitos

Os requisitos fundamentais deste enunciado são:

- Determinar a lista dos domínios e das relações usadas nos ficheiros introduzidos;
- Mostrar os triplos expandidos correspondentes (um triplo por linha);
- Construir um conjunto de páginas *HTML* (uma página por cada termo1) em que os termos2 hiperliguem às correspondentes páginas.

Além disso, foi aconselhada a adição de funcionalidades extra e consequentemente achamos interessante acrescentar os seguintes requisitos adicionais:

- Calcular o número de ocorrências de cada relação no output produzido pela funcionalidade implementada no requisito número 2;
- Mostrar novamente os triplos expandidos, mas desta vez sendo o output produzido sob a forma de ficheiros *HTML* (e ficheiros *CSS* auxiliares), um por cada domínio analisado.

Capítulo 3

Conceção/desenho da Resolução

3.1 Algoritmos

Para as alíneas principais deste enunciado, declaramos o separador de campo (*FS*) como sendo ":" ou o separador "[:|]". Note-se a exceção do enunciado que criamos para contar o número de ocorrências das relações, no qual usamos o separador ",".

Em todos os enunciados, o *Field Separator* foi declarado dentro do bloco *BEGIN*. No apêndice A encontra-se o código correspondente a cada alínea.

Em todas as instâncias usamos o valor definido por defeito para o separador de registos("\n").

3.1.1 Determine a lista dos domínios e das relações usadas

Para atingir este requisito começamos por definir o separador de campo como sendo ":", de modo a facilitar a resolução do mesmo.

De seguida, seleccionamos as linhas em que o início do registo correspondia a "%dom" através da ER:

```
/^\%dom/
```

O carácter ^ indica que a correspondência com a expressão deve ser feita no início do registo.

De cada vez que se obtém uma correspondência, é imprimido o nome do domínio, que pode ser encontrado no segundo campo (\$2).

No que diz respeito às relações, foi necessário seleccionar as linhas em que o primeiro campo (\$1) da linha correspondia ao campo %THE através da ER:

```
/^\%THE/
```

e após a seleção é realizado um ciclo que à medida que se encontra uma relação, esta é imprimida. Este ciclo *for* é inicializado por um $i=2$, uma vez que as relações se encontram a partir do segundo campo (\$2) e incrementado até o i ser igual ao *Number of Fields* (NF).

Desta forma, garantimos que todas as relações foram encontradas e apresentadas ao utilizador de forma agrupada em listas de relações.

3.1.2 Mostre os triplos expandidos correspondentes (um triplo por linha)

O primeiro passo dado nesta alínea envolveu também a definição do Field Separator como ":".

Num segundo passo definimos as relações inversas, identificadas pela ER:

```
/^\%inv/
```

e começamos por remover os espaços tanto à relação como ao inverso através da função *gensub* e, após isso, é inserido o inverso num *array* na posição identificada pelo nome da relação.

De seguida, temos a secção que trata dos domínios, onde geramos um *print* para cada domínio do ficheiro em questão, de forma a dividir os triplos que iremos gerar por domínios.

Após isto, relativamente aos registos começados por *%THE*, imprime-se o número da lista de relações em relação ao seu domínio (indicando se se trata da 1ª, 2ª,... lista de relações do domínio) e realiza-se um ciclo que irá imprimir cada uma das relações da lista, criar um *array* com essas relações e ainda criar um outro *array* que guarda o valor do *iof* de cada relação, quando esse valor existe.

Esse ciclo *for* inicia-se com um *i=2*, pois a primeira das relações (tal como na alínea anterior) surge no segundo campo (\$2). O *i* é incrementado até ser igual ao *Number of Fields* (NF).

A seguir, realizamos um novo ciclo onde imprimimos os IOF's das relações identificados, recorrendo aos *arrays* que foram construídos no ciclo anterior.

Na parte final desta alínea, tratamos os registos (linhas neste caso) que contêm a informação sobre cada *termo1* e os *termo2* que lhe estão associados, sendo o *matching* feito com a ER `/^[^\%#].*/`.

Apresentamos os triplos obtidos do relacionamento dos termos 1 e 2 com o auxílio de mais um ciclo *for* idêntico aos anteriores. Primeiro verificamos se o campo *i* não está vazio, isto é, se o seu comprimento é maior que zero. Se isso se verificar, é selecionada a relação *i* (*relacao = relacoes[i]*) e fazemos um *split* para o caso de haver mais que uma opção no campo *i*. O ciclo *for* aninhado dentro do anterior, percorre cada uma das opções.

A partir daqui são imprimidos todos os triplos e, caso exista uma relação inversa para a relação associado ao campo *i*, é imprimido também o triplo relativo a essa relação inversa.

3.1.3 Triplos Expandidos - Versão HTML(Extra)

O método de processamento dos ficheiros é baseado no demonstrado na subsecção anterior (3.1.2).

Para cada domínio é criado um ficheiro *HTML* e um ficheiro *CSS*. O ficheiro *CSS* permite fazer com que ao passar por cima de uma relação, seja possível ver a classe que lhe está associada (IOF), caso tenha alguma. Adicionalmente, foi estabelecido o parâmetro **corIOF** que pode ser alterado para mudar a cor com que as classes são mostradas. A utilidade deste parâmetro é explicada em maior detalhe mais à frente.

Sempre que se identifica um novo domínio, definem-se novas variáveis *cssFile* e *htmlFile*, que correspondem aos nomes dos ficheiros para onde iremos escrever. Para além disso, nesta fase imprime-se para o ficheiro *html* as *tags* que inicializam o ficheiro e que importam o ficheiro *CSS* para o *html*.

São usados os operadores de redirecionamento de output ">" e ">>", que *imprimem sobre/apendem a* um ficheiro informação, respetivamente. Estes são introduzidos à frente das chamadas *print* e *printf* para escrever nos ficheiros. Novas chamadas a estas funções foram também introduzidas para a criação do ficheiro *CSS*.

Para evitar várias escritas seguidas para ficheiro, por vezes é agregado texto em variáveis, que são depois imprimidas para os ficheiros. Os nomes destas variáveis são **triplosText** e **headerText**.

Funcionalidade *hover* (passar com rato por cima)

Com recurso a *CSS*, é-nos possível implementar funcionalidades adicionais à nossa página.

Decidimos implementar um mecanismo que, ao passar com o rato por cima de uma relação, mostra o *IOF* dessa relação (para relações em que existe *IOF*), mudando a cor do texto para **corIOF**, para o destacar.

Para produzir esse efeito, são registados todos os *IOF*'s no ficheiro *CSS*.

Os mecanismos de *CSS* usados são o *:hover*, para detetar quando passamos com o rato por cima de uma relação, e o mecanismo *::after* que permite escrever texto depois dos elementos com os quais fazemos *matching*.

Finalmente, para podermos fazer *matching* com relações individuais, todas as relações nos triplos estão declaradas entre *tags* `` que por sua vez se encontram dentro de itens de uma lista `<li class="...">` em que o valor de *class* é o nome da relação.

3.1.4 Construa um conjunto de páginas *HTML* (uma página por cada termo1) em que os termos2 hiperliguem às correspondentes páginas

Nesta alínea começamos por repetir o mesmo passo das anteriores, embora neste caso o *FS* escolhido seja "[:]".

Ainda na cláusula *BEGIN*, geramos o ficheiro *HTML index.html* onde imprimimos o predicado "Índice:" e, posteriormente, iremos associar as diferentes páginas *HTML* de cada termo 1, que contêm os termos 2 que lhe estão associados. É também definida uma string base que será usada para criar as referências para as páginas dos *termo1*. Cada ficheiro tem como nome o nome do termo associado a um sufixo "-associated.html".

É feita a correspondência com as linhas que têm as associações entre termos 1 e 2 através da expressão regular `/^[^\%#].*/`

Através de um *printf* inserimos as referências para os ficheiros dos *termo1*, uma a uma no ficheiro *index.html*. Cada registo identificado por esta expressão regular está associado a um único termo1, e como tal irá levar à criação de 1 ficheiro *html*.

Um ciclo *for* (*for*(*i* = 2; *i* ≤ *NF*; *i*++)) percorre os termos 2 do registo e insere-os no ficheiro *html* do *termo1* desse registo.

3.1.5 Calcular o número de ocorrências de uma relação(Extra)

Quanto a este requisito adicional, iniciamos com um *Field Separator* definido por ",". Após ser definido o *FS*, para todas as linhas do ficheiro será incrementado o *array* conta, com o argumento (\$2), que é o nome da relação usada o número de ocorrências da relação, caso \$2 exista (*length*(\$2) > 0). Quando todos os ficheiros forem percorridos, são imprimidos os nomes das relações e os respetivos números de ocorrências de relações (conta[p]) para todos os p's existentes no *array*.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Relativamente à primeira alínea, a identificação do *FS* foi bastante óbvia e a única parte mais complexa foi a utilização do *gensub* para a remoção dos espaços desnecessários presentes na Lista de Relações. De modo similar, para a terceira alínea, a implementação foi relativamente simples e a decisão do que fazer foi quase imediata uma vez que o Field Separator era fácil de identificar tendo em conta o que era pedido e com base nos *HTMLs* produzidos durante as aulas práticas, resolvemos o enunciado de modo similar.

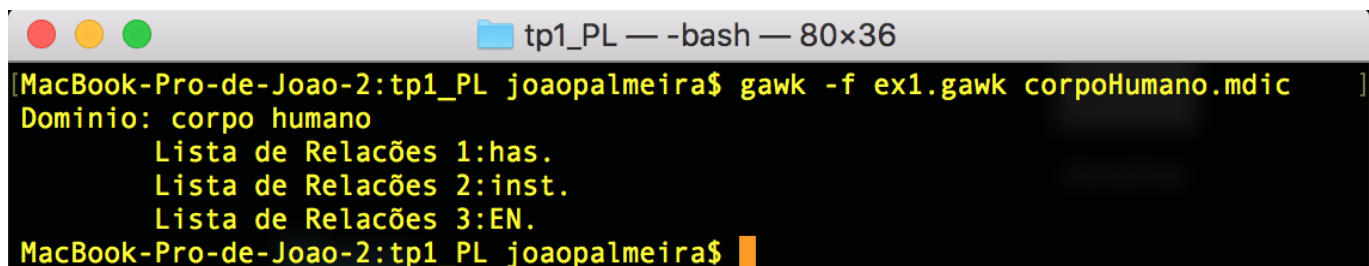
Quanto à segunda alínea a sua resolução foi mais demorada, maioritariamente devido à complexidade dos fatores a ter em conta. Por exemplo, não só a implementação do algoritmo capaz de representar o inverso da relação tenha sido complicada, usamos outras funções, como por exemplo o *split*, para representar as várias opções para uma certa relação.

As alíneas extra foram realizadas segundo sugestão do enunciado em acrescentar mais requisitos. A de contagem de ocorrências de cada relação não causou grandes adversidades. Para além disso, apenas a implementamos pelo possível interesse de saber quais os tipos de relações mais utilizadas.

Por outro lado, a segunda foi bastante mais trabalhosa embora tenhamos usado como base a segunda alínea que nos tinha sido proposta pelo enunciado. Isto deveu-se ao facto de envolver, para além de conhecimentos de *HTML*, conhecimentos de *CSS*, sendo que este último nunca tínhamos usado antes.

4.2 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos (valores introduzidos) e os respetivos resultados obtidos:



```
tp1_PL — -bash — 80x36
[MacBook-Pro-de-Joao-2:tp1_PL joaopalmeira$ gawk -f ex1.gawk corpoHumano.mdic ]
Dominio: corpo humano
Lista de Relações 1:has.
Lista de Relações 2:inst.
Lista de Relações 3:EN.
MacBook-Pro-de-Joao-2:tp1_PL joaopalmeira$
```

Figura 4.1: Exemplo de lista de domínios e relações de um dos ficheiros fornecidos

```

[MacBook-Pro-de-Joao-2:tp1_PL joaopalmeira$ gawk -f ex2.gawk profissoes-lojas.mdi]
C
----- Dominio: profissões -----

[Lista de Relações 1: nt.]

IOF's:
      (nt,iof,profissão)

Triplos:
      (professor,nt, professor do ensino superior politécnico)
      (professor,nt, professor do ensino superior universitário)
      (professor,nt, professor do ensino básico)
      (professor,nt, professor do ensino secundário)
      (músico,nt, acordeonista)
      (músico,nt, afinador)
      (músico,nt, baterista)
      (músico,nt, cantor)
      (músico,nt, clarinetista)
      (músico,nt, concertista)
      (músico,nt, contrabaixista)
      (músico,nt, flautista)
      (músico,nt, gaitreiro)
      (músico,nt, guitarrista)
      (músico,nt, harpista)
      (músico,nt, organista)
      (músico,nt, percussionista)
      (músico,nt, pianista)
      (músico,nt, saxofonista)
      (músico,nt, trombonista)
      (músico,nt, trompetista)
      (músico,nt, violinista)
      (músico,nt, violoncelista)
      (cientista,nt, arqueólogo)
      (cientista,nt, astrofísico)
      (cientista,nt, astrónomo)
      (cientista,nt, biofísico)
      (cientista,nt, biólogo)
      (cientista,nt, botânico)
      (cientista,nt, cartógrafo)
      (cientista,nt, cosmólogo)
      (cientista,nt, egiptólogo)
      (cientista,nt, entomólogo)
      (cientista,nt, estatístico)
      (cientista,nt, etnólogo)
      (cientista,nt, físico)
      (cientista,nt, geneticista)
      (cientista,nt, matemático)

```

Figura 4.2: Exemplo de triplos obtidos a partir de um ficheiro fornecido

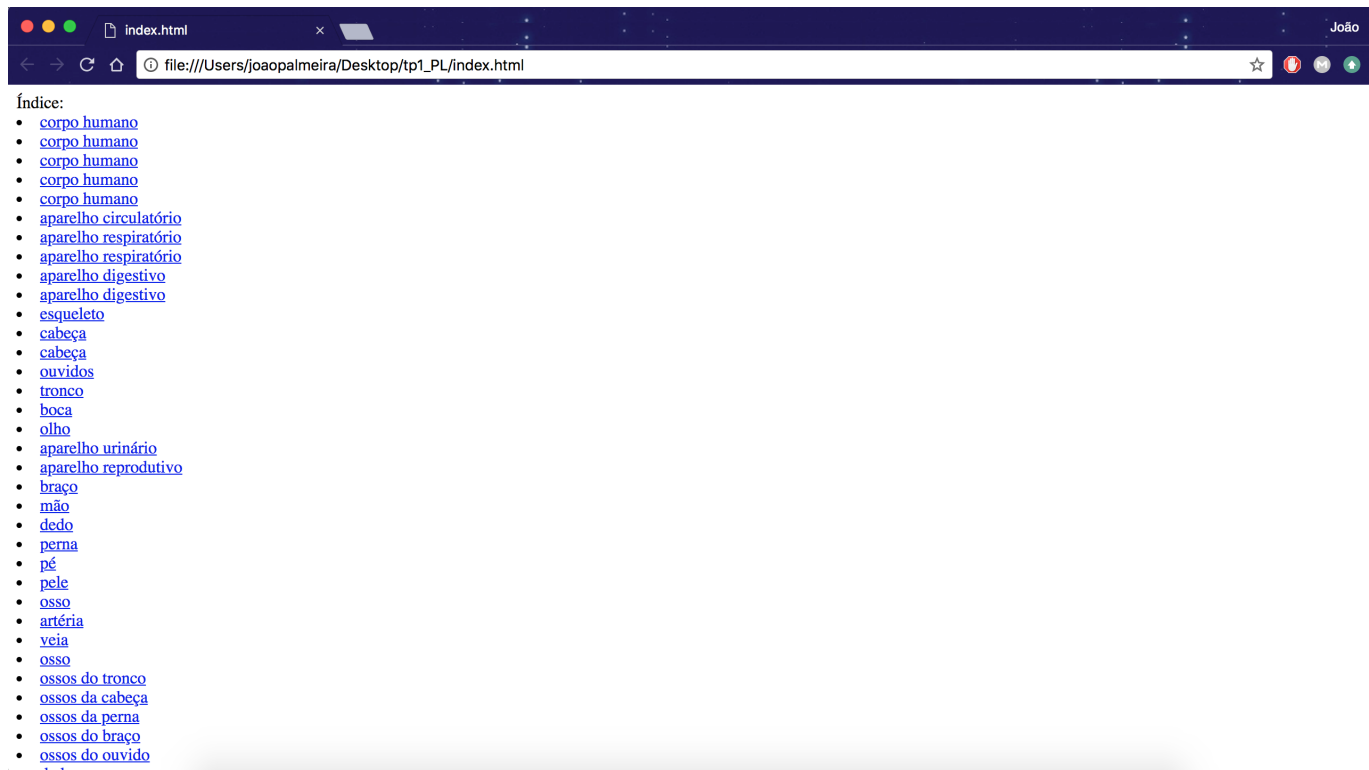


Figura 4.3: Exemplo de páginas *HTML* geradas para cada termo

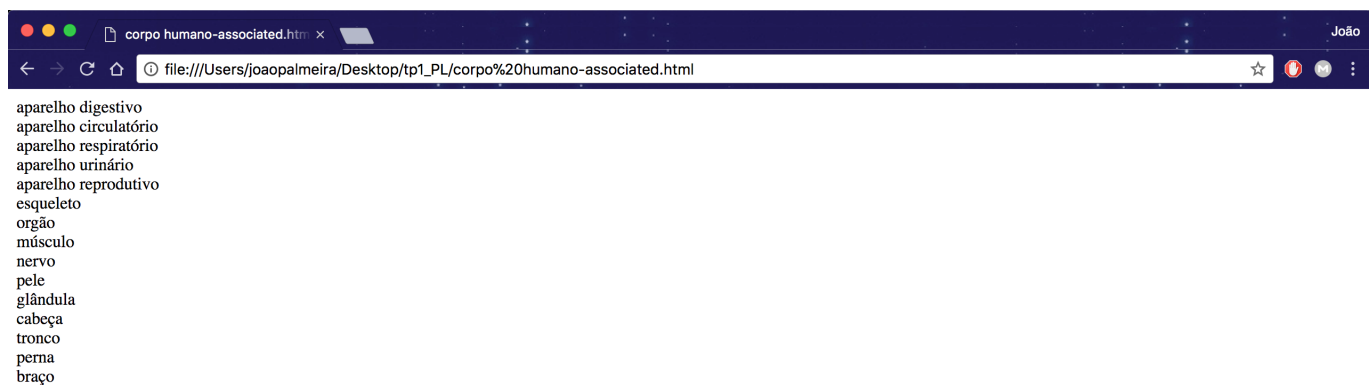


Figura 4.4: Exemplo de página *HTML* de um termo selecionado

```

MacBook-Pro-de-Joao-2:tp1_PL joaopalmeira$ gawk -f ex2.gawk riospt.mdic > rio
MacBook-Pro-de-Joao-2:tp1_PL joaopalmeira$ gawk -f crelacoes.gawk rio
Relacao nascente_de, Número de ocorrencias 22
Relacao desagua_em, Número de ocorrencias 27
Relacao @comprimento, Número de ocorrencias 21
Relacao afluenta_de, Número de ocorrencias 53
Relacao pof, Número de ocorrencias 24
Relacao atravessa, Número de ocorrencias 51
Relacao in, Número de ocorrencias 24
Relacao nasce_em, Número de ocorrencias 23
Relacao foz_de, Número de ocorrencias 27
Relacao iof, Número de ocorrencias 4
Relacao e_atravessado, Número de ocorrencias 51
MacBook-Pro-de-Joao-2:tp1_PL joaopalmeira$

```

Figura 4.5: Exemplo do cálculo do número total de relações de um ficheiro

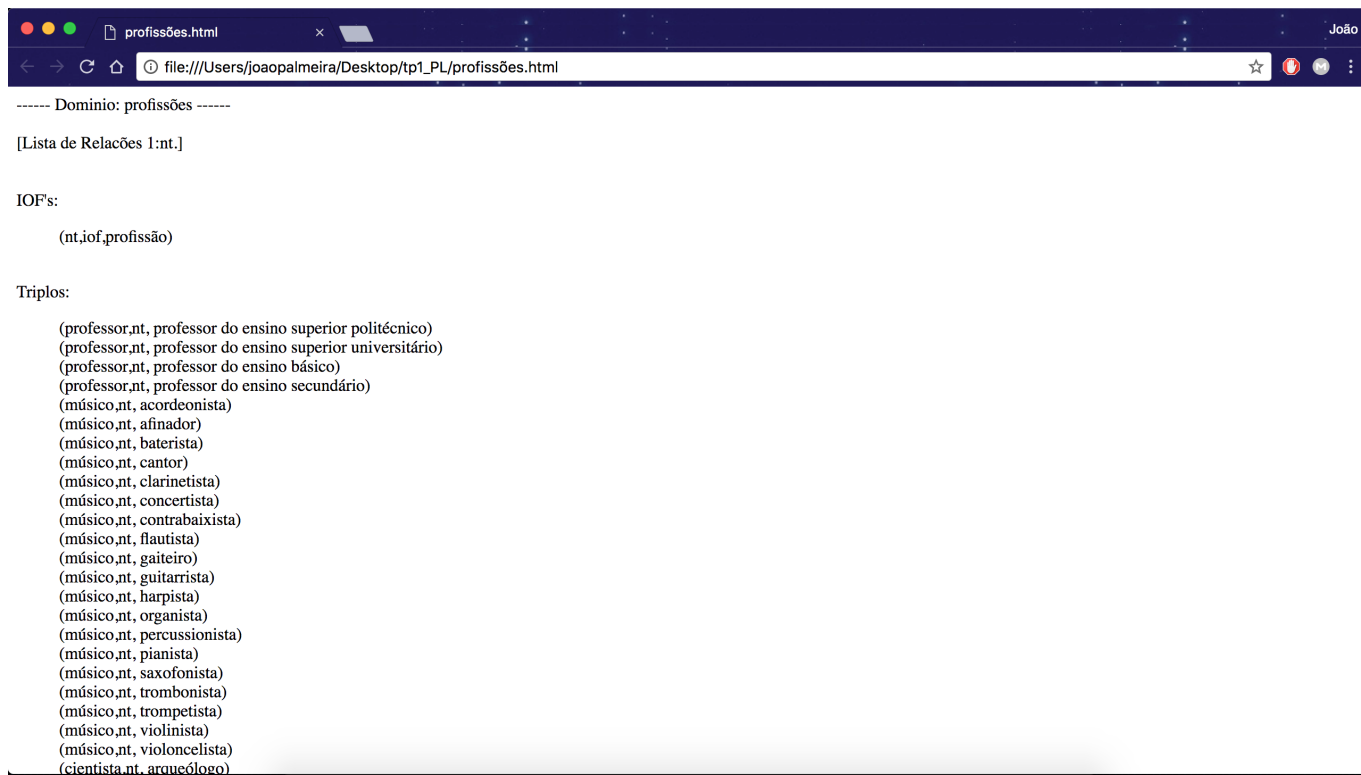


Figura 4.6: Exemplo de páginas *HTML* geradas para os triplos do ficheiro profissoes-lojas.mdic

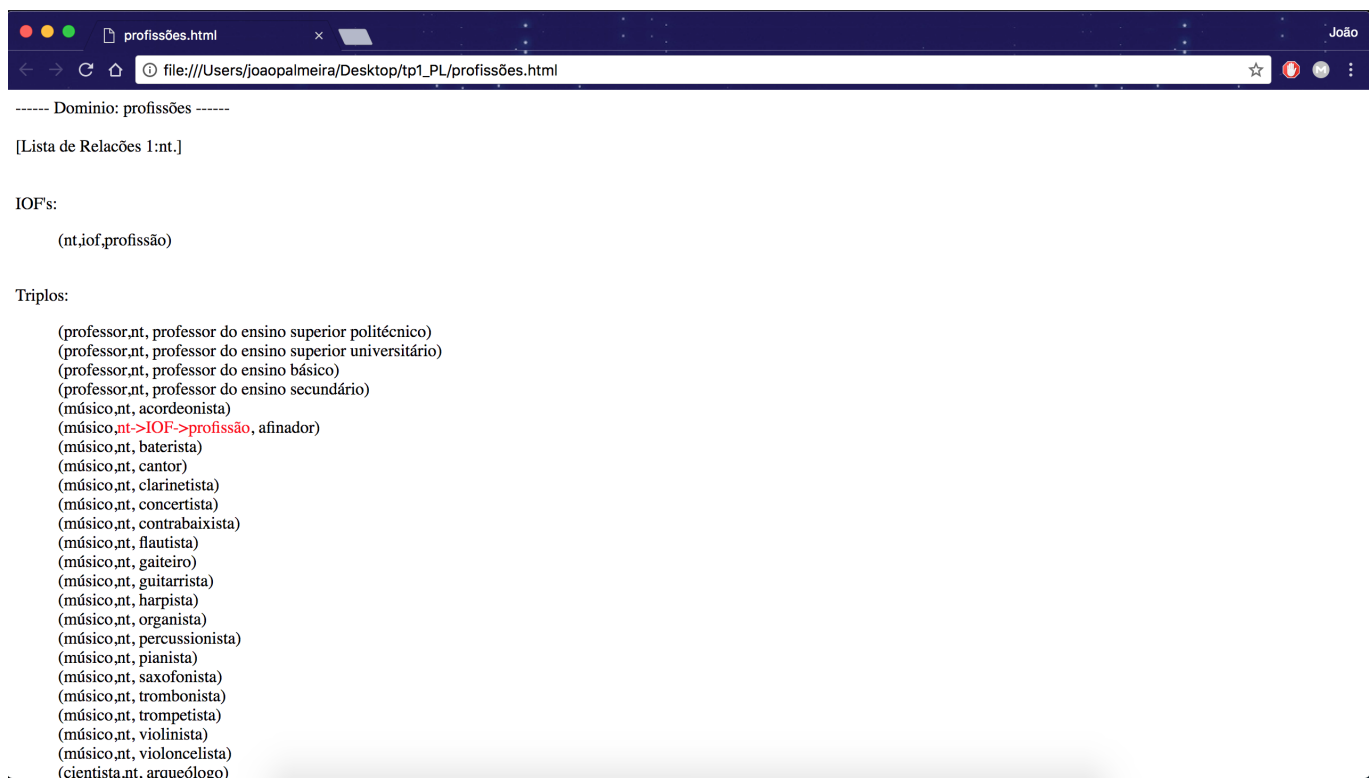
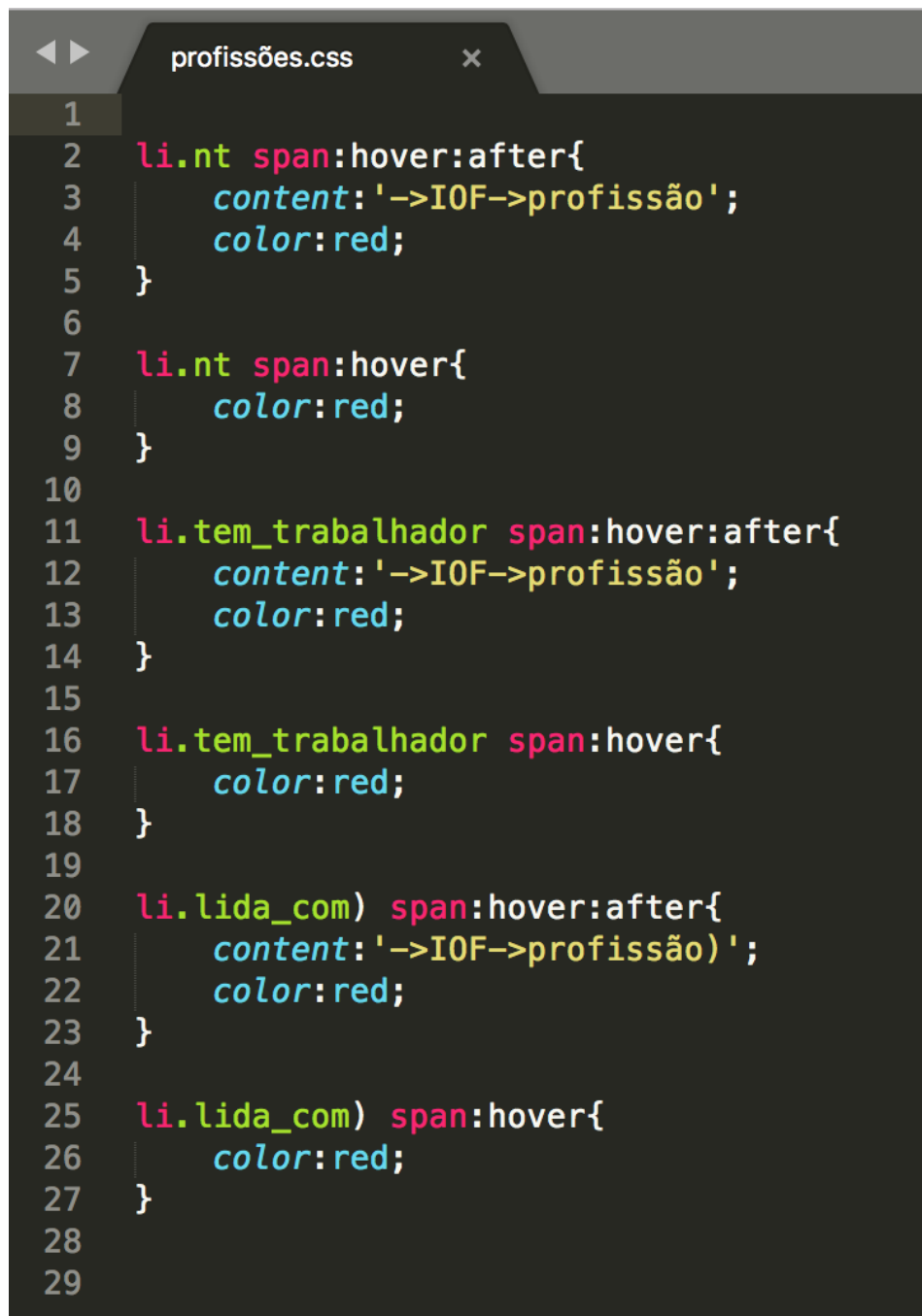


Figura 4.7: Exemplo de páginas *HTML* geradas para os triplos do ficheiro *profissoes-lojas.mdic* com *IOFSs*

A screenshot of a code editor window titled 'profissões.css'. The editor has a dark background with light-colored text. The code is CSS, with selectors and properties color-coded: selectors in pink, property names in green, and values in yellow. The code is organized into five blocks, each with an 'after' and a 'hover' rule. Line numbers 1 through 29 are visible on the left side of the editor.

```
1
2  li.nt span:after{
3      content: '->IOF->profissão';
4      color:red;
5  }
6
7  li.nt span:hover{
8      color:red;
9  }
10
11 li.tem_trabalhador span:after{
12     content: '->IOF->profissão';
13     color:red;
14 }
15
16 li.tem_trabalhador span:hover{
17     color:red;
18 }
19
20 li.lida_com) span:after{
21     content: '->IOF->profissão)';
22     color:red;
23 }
24
25 li.lida_com) span:hover{
26     color:red;
27 }
28
29
```

Figura 4.8: Exemplo de ficheiro *CSS* gerado para os *IOFs* do ficheiro *mdic*

Capítulo 5

Conclusão

Em suma, este projeto permitiu-nos aprofundar os nossos conhecimentos na área do Processamento de Linguagens obtidos durante as aulas, especialmente os da ferramenta de processamento de texto *GAWK* e a elaboração de expressões regulares capazes de representar os elementos que nos interessavam em cada situação.

Julgamos que conseguimos cumprir os objetivos definidos no enunciado e até com um pouco mais de profundidade apesar de termos atravessado algumas dificuldades naturais da aprendizagem e aplicação de conceitos novos.

É de destacar a criação de páginas *HTML* tal como pretendido e achamos que o resultado foi bastante satisfatório.

Quanto ao trabalho futuro, talvez fosse possível expandir ainda mais os objetivos e os requisitos, embora achemos que o que acrescentamos de novo foi algo relevante e de importante contribuição. De qualquer modo, está em aberto a implementação de mais funcionalidades que possam a vir a ser interessantes, por exemplo, ordenando resultados por ordem alfabética.

Apêndice A

Código do Programa

Mostra-se a seguir o código desenvolvido para as várias alíneas e que está explicado na secção de Algoritmos do capítulo 3.

Determine a lista dos domínios e das relações usadas

```
#!/usr/bin/awk -f

BEGIN {FS = ":"}
/^\%dom/ {num= 1;printf ("Dominio: \s \n"),\$2}
/^\%THE/ { printf ("\tLista de Relações \%d:",num);
          for (i=2; i <= NF; i++){
            b =gensub(/<.*\/,"","g",\$i);
            printf("\s",b);
            if(i==NF) printf(".");
            else printf(",");
          }
          printf("\n");
          num++;
}
```

Mostre os triplos expandidos correspondentes (um triplo por linha)

```
#!/usr/bin/gawk -f

BEGIN {FS = ":";
       utf8 = "<meta charset=\\"utf-8\\">";
}
/^\%inv/ { relacao = gensub(/s*(\w+)\s*(<?)\w*/,"\\1","g",\$2);
          inverso = gensub(/s*(\w+)\s*(<?)\w*/,"\\1","g",\$3);
          inv[relacao]=inverso;
}

/^\%dom/ {
  num= 1;
  printf ("----- Dominio: \s -----\\n\\n",\$2)}
```

```

/^%\%THE/ {printf ("Lista de Relações \d: ",num);
    for (i=2; i <= NF; i++){
        relacao = gensub(/\s*(\w+)\s*<?\w*/,"\\1","g",\i);
        relacoes[i] = relacao;
        iof[relacoes[i]] = gensub(/\s*\w+\s*<?(\w*)/,"\\1","g",\i);
        printf("\%s",relacao);
        if(i!=NF) printf(",");
        else print ".]";
    }
    print "\n";

    print "IOF's:"
    for(i = 2; i <= NF; i++){
        if(length(iof[relacoes[i]]) > 0)
            printf("\t(\%s,iof,\%s)\n",relacoes[i],iof[relacoes[i]]);
    }
    printf("\n");
    print "Triplos:"
    num++;
}

/^[^\%#].*/ {

    for(i = 2; i <= NF; i++){
        if(length(\i) > 0){
            relacao = relacoes[i];
            split(\i,opcoes,"|",seps);
            for(j=1; j<=length(opcoes);j++){
                printf("\t(\%s,\%s,\%s)\n",\i,relacao,opcoes[j]);
                if(length(inv[relacao])!=0){
                    printf("\t(\%s,\%s,\%s)\n",opcoes[j],inv[relacoes[i]],\i);
                }
            }
        }
    }
}
}

```

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

```
#!/usr/local/bin/gawk -f
```

```

BEGIN{
    FS = "[:|]";
    utf8 = "<meta charset>=\"utf-8\">";
    print "Índice:<br />" > "index.html"
    referencia = "<li><a href='\"%s-associated.html'>%s</a></li>\n"
}

/^[^\%#].*/ {
    printf(referencia,\$1,\$1)>>"index.html";
}

```

```

        for(i = 2; i <= NF; i++){
            if(length(\$i) > 0){
                printf("%s<br/>\n",\$i)>>\$1"-associated.html";
            }
        }
    }
}

```

Calcular o número de ocorrências de uma relação

```

#!/usr/bin/gawk -f

BEGIN {FS=",";}
    {if (length(\$2)>0) conta[\$2]++;}
END    {for (p in conta) printf ("Relacao %s, Número de ocorrencias %d \n",p,conta[p]);}

```

Construir ficheiro *HTML* e *CSS* com os triplos de cada ficheiro mdic

```

#!/usr/bin/gawk -f

BEGIN {
    FS = ":";
    utf8 = "<meta charset=\"utf-8\">";
    corIOF = "red";
}

/^%\%inv/ { relacao = gensub(/s*(\w+)\s*/,"\\1","g",\$2);
    inverso = gensub(/s*(\w+)\s*/,"\\1","g",\$3);
    inv[relacao]=inverso;
}

/^%\%dom/ {
    #Serve para imprimir últimos triplos, para quando se passam vários ficheiros ao mesmo tempo,
    igual ao END
    if(num > 1){
        endOfDomain();
    }

    num= 1;
    htmlFile = removeSurroundingSpaces(\$2".html");
    cssFile = removeSurroundingSpaces(\$2".css");
    print "<!DOCTYPE html>\n<html>" > htmlFile;
    print "\t<head>\n\t\t<link rel=\"stylesheet\" type=\"text/css\"
href=\"\"cssFile\" \"media=\"screen\" />\n\t</head>" >> htmlFile;
    print "\t<body>" >> htmlFile;
    printf ("\t\t----- Dominio: %s -----<br><br>\n",\$2) >> htmlFile ;
    print "" > cssFile;
}

/^%\%THE/ {
    if(num > 1){

```

```

        triplosText = triplosText"\t\t</ul>"
        print triplosText >> htmlFile;
    }
    headerText = "\t\t[Lista de Relações "num": ";
    for (i=2; i <= NF; i++){
        relacao = gensub(/\s*(\w+)\s*<?\w*/,"\\1","g",\i);
        relacoes[i] = relacao;
        iof[relacoes[i]] = gensub(/\s*\w+\s*<?(\w*)/,"\\1","g",\i);
        if(length(iof[relacoes[i]]) > 0){
            print "li." relacao " span: hover: after{\n\tcontent: '->IOF->' iof[relacoes[i]] ";
            "\n\tcolor: \"corIOF\"; \n} \n" >> cssFile;
            print "li." relacao " span: hover{\n\tcolor: \"corIOF\"; \n} \n" >> cssFile;
        }

        headerText = headerText relacao;
        if(i!=NF) headerText = headerText",";
        else headerText = headerText".";
    }
    headerText=headerText"<br><br><br>\n";

    headerText=headerText"\n\t\tIOF's:\n";
    headerText = headerText"\t\t<ul style=\"list-style-type:none\">\n"
    for(i = 2; i <= NF; i++){
        if(length(iof[relacoes[i]]) > 0)
            headerText = headerText "\t\t\t<li>(\"relacoes[i]\",iof,\"iof[relacoes[i]]\")</li>\n";
    }
    headerText = headerText"\t\t</ul>\n"
    headerText = headerText"\n\t\t<br>Triplos:"
    print headerText >> htmlFile;
    triplosText = "\t\t<ul style=\"list-style-type:none\">\n"
    num++;
}
/^[\^%#].*/ {
    for(i = 2; i <= NF; i++){
        if(length(\i) > 0){
            relacao = relacoes[i];
            split(\i,opcoes,"|",seps);
            for(j=1; j<=length(opcoes);j++){
                triplosText = triplosText"\t\t\t<li class=\"relacao\">
                ("i",<span>"relacao"</span>,"opcoes[j]")</li>\n";
                if(length(inv[relacao])!=0){
                    triplosText = triplosText"\t\t\t<ul style=\"list-style-type:none\">\n";
                    triplosText = triplosText"\t\t\t\t<li>Inversa:
                    ("opcoes[j]","inv[relacoes[i]]","i")</li>\n";
                    triplosText = triplosText"\t\t\t\t</ul>\n\n";
                }
            }
        }
    }
}
END{
    triplosText = triplosText"\t\t</ul>";
    print triplosText >> htmlFile;
    print "\t</body>" >> htmlFile;
}

```

```

    print "</html>" >> htmlFile;
}

function endOfDomain(){
    triplosText = triplosText"\t\t</ul>";
    print triplosText >> htmlFile;
    print "\t</body>" >> htmlFile;
    print "</html>" >> htmlFile;
}

function removeSurroundingSpaces(line, ret){
    ret = gensub(/\s*(.+)\s*/,"\\1","g",line);
    return ret;
}

```