

Processamento de Linguagens
(3º ano de Curso)
Trabalho Prático nº1 - Parte A
Relatório de Desenvolvimento

José Silva
(A74601)

Pedro Cunha
(A73958)

Gonçalo Moreira
(A73591)

15 de Março de 2017

Resumo

Documentação do primeiro trabalho prático da unidade curricular de "Processamento de Linguagens", o principal foco incide sobre a utilização de métodos e ferramentas capazes de solucionar processamento de textos complexos utilizando poucas linhas de código. Por outro lado, em linguagens de programação como C e Pascal, solucionar os mesmos problemas acarreta um maior nível de complexidade. Demonstrando e documentando a solução proposta pelo grupo de trabalho para o problema em concreto, termina-se o relatório com uma análise argumentativa sobre a eficiência dessa mesma solução.

Conteúdo

1	Introdução	2
2	Análise e Especificação	3
2.1	Descrição informal do problema	3
2.2	Especificação dos Requisitos	3
2.2.1	Dados	3
2.2.2	Pedidos	3
3	Concepção/desenho da Resolução	4
3.1	Estruturas de Dados	4
3.2	Algoritmos	4
4	Codificação e Testes	5
4.1	Alternativas, Decisões e Problemas de Implementação	5
4.2	Testes realizados e Resultados	5
4.2.1	O programa é executado com o ficheiro como argumento	6
4.2.2	Obtém-se o seguinte resultado	6
4.2.3	Exemplo da tab que abre quando se clica no local Ferreiros	8
4.2.4	Exemplo da lista que abre quando se clica numa data	9
5	Conclusão	10
A	Código do Programa	11

Capítulo 1

Introdução

O avanço tecnológico dos últimos anos trouxe consigo a inevitabilidade de processar cada vez mais texto. Por parte de grande parte dos utilizadores existe a necessidade frequente de fazer mudanças ou extrair determinadas linhas de grandes quantidades de texto onde certos padrões são bastante evidentes. O uso de expressões regulares, que proporcionam um método eficiente, poderoso e flexível no que toca ao processamento de texto, combinado com as ferramentas que a linguagem AWK (linguagem de programação bastante mais fácil de utilizar que as linguagens mais convencionais) proporciona um método eficiente para solucionar as necessidades descritas acima. Das ferramentas descritas anteriormente destacam-se funções capazes de manipular strings e a utilização de arrays associativos, estruturas de dados bastante úteis e que não são disponibilizadas por todas as linguagens de programação.

Neste primeiro trabalho prático da unidade curricular de "Processamento de Linguagens", através dos meios descritos anteriormente, vai desenvolvido um filtro de texto capaz de realizar o processamento de transações presentes nos extratos mensais disponibilizados pela empresa Via Verde.

Estrutura do Relatório

No capítulo 1 faz-se uma pequena introdução ao problema e às ferramentas utilizadas para a resolução deste. Para além disso, é descrita de uma forma breve a estrutura do relatório.

No capítulo 2 faz-se uma análise breve mas mais detalhada do problema escolhido pelo grupo de trabalho.

No capítulo 3 é descrito de uma forma sumariada como procedemos para solucionar as várias questões propostas pelo enunciado.

No capítulo 4 são apresentados alguns testes e respectivos resultados para comprovar o respectivo funcionamento da solução apresentada.

Finalmente, no capítulo 5 termina-se o relatório com uma síntese do que foi dito, as conclusões e o trabalho futuro.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

É fornecido um ficheiro xml, que corresponde ao extrato mensal emitido pela Via Verde para um dos seus utentes. Pretende-se que se desenvolva um *Processador de Texto* para ler esse mesmo ficheiro e retirar a informação requisitada, apresentada em mais detalhe a baixo, na Especificação dos Requisitos.

2.2 Especificação dos Requisitos

2.2.1 Dados

Como já foi referido, é fornecido um ficheiro xml com informação correspondente ao extrato mensal emitido pela Via Verde para um dos seus utentes. Este ficheiro contém, no início, informação do utente e também especifica o mês de emissão. Depois destes dados, seguem-se todas as transações do cliente. Cada uma destas transações, contém a data, as horas de entrada e saída, o local de entrada e saída, a importância paga, o desconto, o iva, o operador e ainda o tipo de transação (Portagens ou Parques de estacionamento). No final do ficheiro, é apresentado o total gasto no mês de emissão e ainda o valor que diz respeito ao iva.

2.2.2 Pedidos

O utente pode ter efetuado entrada em alguns dos dias do mês a que este extrato se refere. Posto isto, é solicitado que se calcule o número de entradas em cada dia do mês. Considera-se relevante obter a lista de locais de saída pelos quais o utente passou durante todo o mês. Chegando ao fim do mês, uma das informações mais importantes para o utente é, muito provavelmente, o total gasto nesse mesmo mês. É então pedido que se obtenha o total que o utente gastou. Pretende-se que seja calculado quanto desse total é que corresponde ao valor despendido apenas em parques de estacionamento.

Para obter a informação necessária será utilizado o Sistema de Produção GAWK, especificando os padrões de frases que se pretende encontrar com recurso a expressões regulares.

Capítulo 3

Concepção/desenho da Resolução

3.1 Estruturas de Dados

Pensando nos requisitos para este projeto, é fácil de perceber que será necessário guardar informação relativa a alguns tópicos. Por exemplo, será crucial guardar a informação que diz respeito à quantidade de entradas num determinado dia. Para guardar informação deste tipo, serão utilizados arrays associativos. Embora apenas seja requisitado que se liste os diferentes locais de saída, serão utilizados arrays associativos de forma a guardar quantas vezes passou por cada um desses locais. Relativamente a cada dia, serão ainda guardados os totais gastos, as durações totais de viagens e quais são os locais de entrada(em arrays associativos) que, embora não seja um requisito, se considera informação importante. Acrescentando ainda aos extras já apresentados, armazena-se informação relativa ao valor despendido em parques e também em portagens. Informação como o total gasto no mês não precisa de ser guardada em nenhuma estrutura de dados, bastando usar um float para o efeito.

3.2 Algoritmos

No início do programa, serão atribuídos todos os valores necessários a algumas variáveis que serão uteis para o corpo do programa. Por exemplo strings, inteiros e formatos para a função printf. É muito importante a definição de um *Field Separator* adequado para facilitar o processamento do ficheiro.

O corpo do programa será na forma de condição e ação respetiva, como é típico usando GAWK. Sabendo em que campo começa a aparecer a informação do utilizador e em qual acaba, tirar-se-á partido disso para obter toda a informação(Nome, nif, etc) de forma sistemática, com apenas um bloco de ação. Na maioria das condições, estão especificadas expressões regulares que um ou mais 'fields' da linha atual tem que satisfazer. Sabendo que esse 'field' satisfaz a expressão regular, guarda-se informação valiosa para o resultado final do programa. Algumas das entradas são em parques ou não têm data de entrada(ou ambos), nesses casos o campo entrada não está preenchido, existirão variáveis para saber se é para adicionar entrada e saber se é o caso de parque(Adicionando o nome do parque acedendo ao campo saída). Para todo este processo, é necessária uma variável que guarde o último dia de entrada pelo qual se passou. Como se pretende saber o gasto em parques e em portagens separadamente, é fulcral que se guarde o último gasto adicionado ao total numa variável para que, chegando ao campo tipo, se adicione esse mesmo valor ao array associativo que guarda essa informação(no índice do tipo em questao). Uma das ações terá que ser apoiada por uma função cujo objetivo será calcular a duração de uma viagem, recebendo como argumentos a hora de entrada e hora de saída. Esta função partirá do princípio que, no máximo, uma viagem começa num dia e acaba no seguinte.

Chegando à fase final de execução trata-se a informação obtida, de forma a apresentar os resultados em formato apelativo. Por exemplo, usando html com strings definidas no início e não só. Existirá, muito provavelmente, um ciclo para quase todos os arrays associativos de forma a apresentar a informação armazenada no mesmo. Um destes ciclos, deverá juntar a quantidade de entradas num dia e o total gasto no mesmo. Depois de tratar toda a informação, no caso de ser usado html, coloca-se o final no ficheiro fechando o body e o html.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Uma das decisões tomadas logo no início da implementação foi usar html como tipo de ficheiro de 'output'. Este ficheiro utiliza estilos escritos em css e também um script(em javascript) para tornar possível o aparecimento de listas escondidas(clicando em cada dia onde informa quantas entradas existiram, aparece a lista de locais). Também é utilizada a font-awesome para juntar icons que refletem o conteúdo de cada linha apresentada. O objetivo do uso destes recursos é tornar o conteúdo do ficheiro apelativo. Uma alternativa a esta abordagem seria utilizar o standard output para apresentar os resultados, mas considerou-se que o html 'ganhava' em tudo, excepto na complexidade de construção. Tendo apenas esse problema, que não causa muito transtorno, achou-se por bem optar pelo html. Como se pode verificar no bloco begin(linhas 3 a 24), foram criados vários formatos e strings para auxiliar na criação do ficheiro.

Das decisões tomadas, a mais importante foi, talvez, decidir qual *Field Seperator* usar. Depois de observar bem o ficheiro xml, chegou-se à conclusão que seria oportuno utilizar como *Field Seperator* a expressão regular na linha 4 do código: [<>]. Assim, é possível obter o que está dentro de cada elemento xml apenas acedendo à posição 3. Isto facilita muito a obtenção de informação. Com outra abordagem, podia ter sido usado o *Field Seperator* default, mas dessa forma seria necessário um tratamento muito mais complexo das strings em cada posição para retirar dados importantes. Relativamente aos locais de saída, decidiu-se juntar uma referência que pesquisa o nome desse local no google maps e apresenta o resultado numa nova tab do browser utilizado para abrir o html gerado.

Durante a implementação do programa, não houve nenhum problema que mereça grande destaque. Os algoritmos utilizados são relativamente simples e as expressões regulares também. A expressão mais 'complexa' utilizada, foi a que garante que a data de entrada seja no formato correto(removendo ocorrências de null). Esta expressão está presente na linha 57 do código. Para a soma de floats, foi necessário transformar as ',' lidas do ficheiro em '.', usando gsub(linhas 98 e 105). De outra forma, era lida apenas a parte inteira do número.

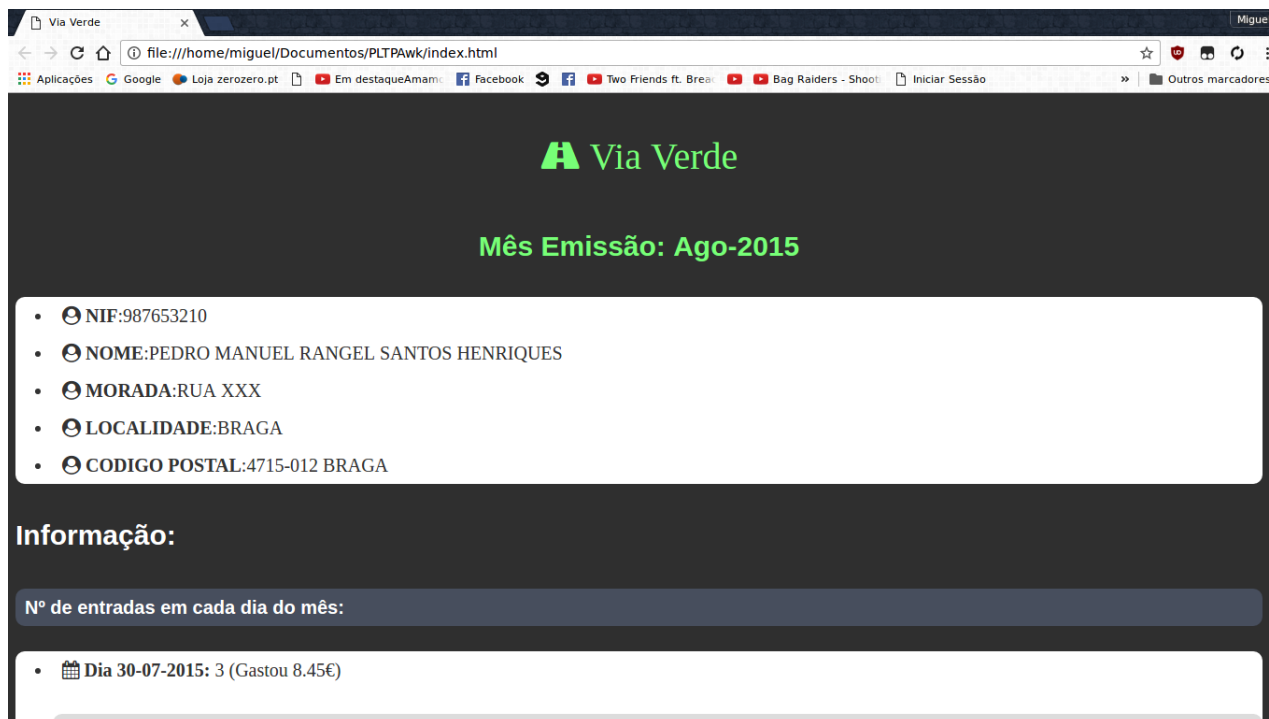
4.2 Testes realizados e Resultados

Mostra-se a seguir o teste feito (o ficheiro recebido pelo programa - viaverde.xml) e os respectivos resultados obtidos(em formato html):

4.2.1 O programa é executado com o ficheiro como argumento

```
[00:20]miguel PLTPAwk $ ./procViaverde viaverde.xml
```

4.2.2 Obtém-se o seguinte resultado



Via Verde x Miguel

file:///home/miguel/Documentos/PLTPAwk/index.html

Aplicações Google Loja zerozero.pt Em destaqueAmam Facebook Two Friends ft. Bre Bag Raiders - Shoot Iniciar Sessão Outros marcadores

Informação:

Nº de entradas em cada dia do mês:

- 📅 **Dia 30-07-2015:** 3 (Gastou 8.45€)
⌚ Tempo total em viagem: 129 minutos
- 📅 **Dia 18-08-2015:** 2 (Gastou 4.00€)
⌚ Tempo total em viagem: 32 minutos
- 📅 **Dia 29-07-2015:** 2 (Gastou 5.20€)
⌚ Tempo total em viagem: 24 minutos
- 📅 **Dia 11-08-2015:** 2 (Gastou 5.20€)
⌚ Tempo total em viagem: 24 minutos

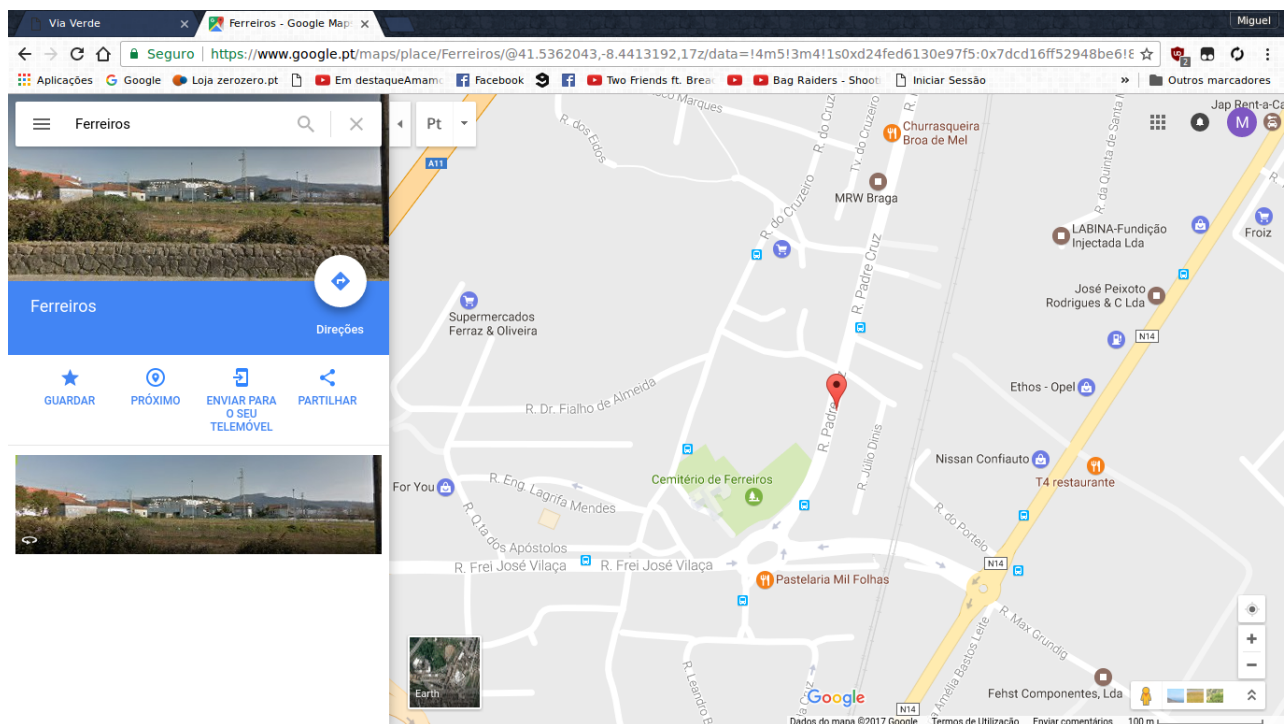
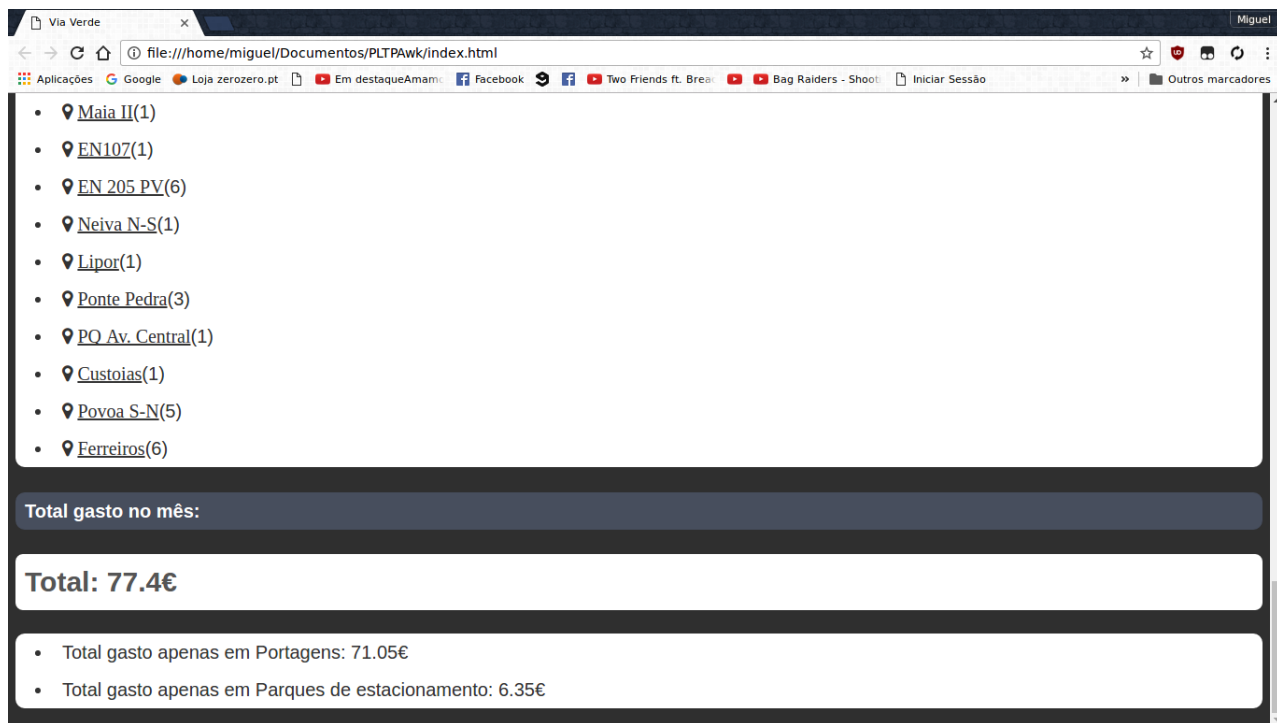
Via Verde x Miguel

file:///home/miguel/Documentos/PLTPAwk/index.html

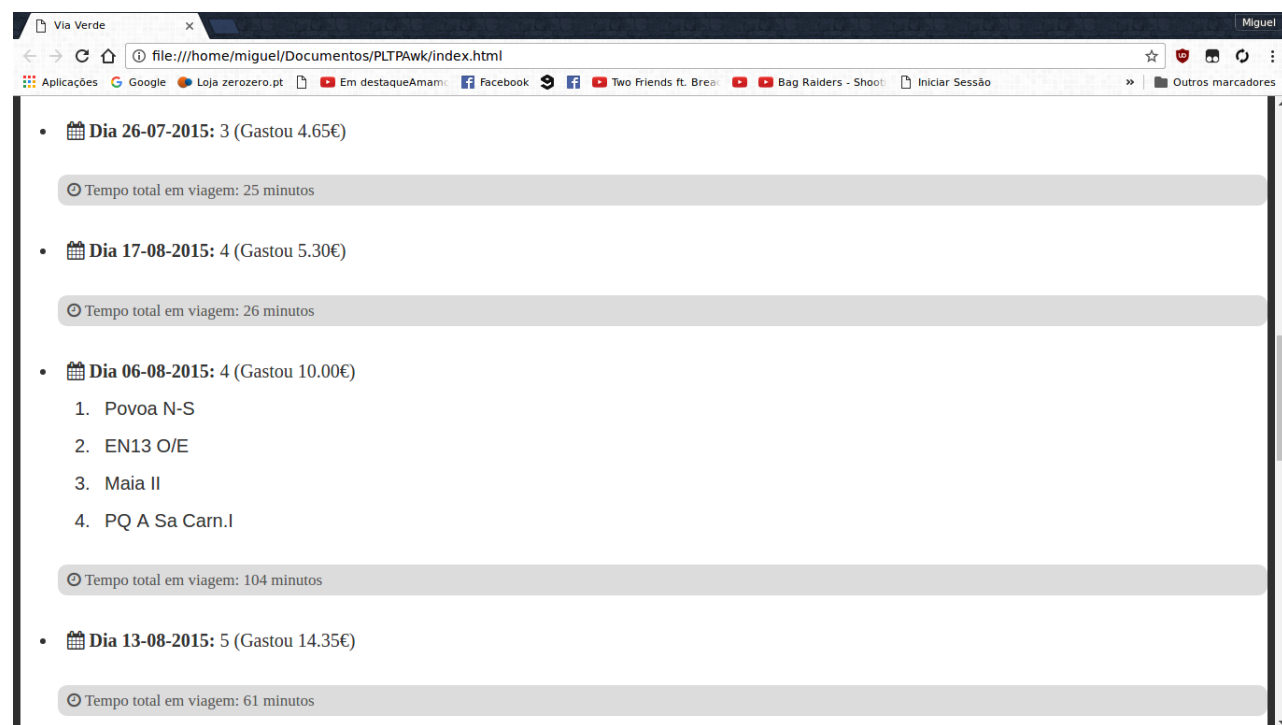
Aplicações Google Loja zerozero.pt Em destaqueAmam Facebook Two Friends ft. Bre Bag Raiders - Shoot Iniciar Sessão Outros marcadores

Locais de saída:

- 📍 [Braga Sul](#)(2)
- 📍 [Maia PV](#)(1)
- 📍 [Neiva S-N](#)(1)
- 📍 [Angeiras N-S](#)(5)
- 📍 [Freixieiro](#)(2)
- 📍 [Valongo](#)(1)
- 📍 [PO A Sa Carn.I](#)(1)
- 📍 [Ermesinde PV](#)(1)
- 📍 [Aeroporto](#)(1)
- 📍 [Maia II](#)(1)
- 📍 [EN107](#)(1)
- 📍 [EN 205 PV](#)(6)
- 📍 [Neiva N-S](#)(1)
- 📍 [Lipor](#)(1)
- 📍 [Ponte Pedra](#)(3)



4.2.4 Exemplo da lista que abre quando se clica numa data



The screenshot shows a web browser window with the address bar displaying `file:///home/miguel/Documents/PLTPAwk/index.html`. The browser's address bar and tabs are visible at the top. The main content area displays a list of travel expenses, each preceded by a calendar icon. The list includes the date, the number of trips, and the total cost in Euros. Below each entry is a grey bar indicating the total travel time in minutes. The list is as follows:

- **Dia 26-07-2015:** 3 (Gastou 4.65€)
Tempo total em viagem: 25 minutos
- **Dia 17-08-2015:** 4 (Gastou 5.30€)
Tempo total em viagem: 26 minutos
- **Dia 06-08-2015:** 4 (Gastou 10.00€)
 - 1. Povia N-S
 - 2. EN13 O/E
 - 3. Maia II
 - 4. PQ A Sa Carn.ITempo total em viagem: 104 minutos
- **Dia 13-08-2015:** 5 (Gastou 14.35€)
Tempo total em viagem: 61 minutos

Capítulo 5

Conclusão

Foi possível, através do uso da linguagem de programação AWK e do uso de ferramentas html, realizar com sucesso os requisitos pedidos neste trabalho prático assim como demonstrar outras funcionalidades importantes e úteis para a recolha e apresentação de informação neste trabalho. Para além dos requisitos necessários, calcular o número de entradas do utente em cada dia do mês, obter a lista de locais de saída utilizados durante o mês retirar o valor total pago no final do mês e o valor total relativo apenas ao estacionamento em parques, foi possível apresentar mais informação detalhada, tais como o total gasto em cada dia, a duração total de viagens e visualização dos locais de saída num mapa para melhor orientação.

Com o uso de expressões regulares GAWK foi possível encontrar a informação específica do ficheiro xml, logo possibilitando retirar essa mesma e guardá-la em arrays associativos para posterior apresentação em formato html. Este foi utilizado como formato de apresentação visto que apresenta a informação necessária num formato conciso e elegante, apesar de ser mais complexo do que usar o standard output. Para melhorar a disposição da informação e simplicidade desta foi utilizado a font-awesome visto que esta permitiria criar icons relativos a cada linha para melhor compreensão e para apelar ao utilizador, permitindo uma maior fluidez na procura dos conteúdos .

O uso das expressões regulares através da linguagem de programação AWK e de ferramentas html permitiu, em termos de processamento de texto e divulgação, gerar código simples e curto, de fácil compreensão e com a capacidade necessária para retirar, armazenar e apresentar informação importante e relativa ao projeto. Com isto dito, existem ainda capacidades extra por explorar para continuar a desenvolver o projeto, tais como, a disponibilização de um trajeto feito pelo utente num determinado dia. Logo apesar de completar os requisitos necessários com as capacidades da linguagem AWK, é possível encontrar novas maneiras de utilizar a informação disposta, de forma a proporcionar maior utilidade ao utentes e gestores.

Apêndice A

Código do Programa

Lista-se a seguir o código do programa que foi desenvolvido.

procViaverde

```
1  #!/usr/bin/gawk -f
2
3  BEGIN {
4      FS = "[<>]";
5      file = "index.html";
6      icon = "<i class='fa fa-calendar'>";
7      fmtEntrad = "<li%s'icon' <b>Dia %s:</b> %d (Gastou %.2f€)</i></li>\n";
8      fmtTotais = "<li>Total gasto apenas em %s: %.2f€</i></li>\n";
9      fmtli = "<li> %s</li>\n";
10     fmth5 = "<h5 class='bg-3'>%s</h5>\n";
11     fmth4 = "<h4 class='bg-2'>%s</h4>\n";
12     fmth2 = "<h2>%s</h2></div>\n";
13     icon = "<i class='fa fa-road'>";
14     ini = "<!DOCTYPE html> <html> <head>\n";
15     tit = "<title> Via Verde </title>";
16     enc = "<meta charset='utf-8'/> </head> <body>";
17     end = "</body> </html>";
18     beg = "<div class='bg-4 title'><h1>'icon' Via Verde </i></h1>";
19     css = "<link rel='stylesheet' href='css/styles.css'>";
20     fta = "<link rel='stylesheet' href='css/fa/css/font-awesome.css'>";
21     ref = "<a target='_blank' href='%s'>%s</a>";
22     maps = "https://www.google.pt/maps/place/";
23     sjs = "<script src='js/script.js'></script>";
24     infoCliente = 0;
25     adicionarEntrada = 0;
26     parque = 0;
27 }
28
29 NR == 1 {
30     print ini > file;
31     print tit > file;
32     print css > file;
33     print fta > file;
34     print sjs > file;
35     print enc > file;
36     print beg > file;
37 }
38
39 $2~/MES_EMISSAO/{
```

```

40     s = "Mês Emissão: " $3;
41     printf(fmth2, s) > file;
42 }
43
44 $2~/NIF/{
45     infoCliente++;
46     print "<ul>" > file;
47 }
48
49 infoCliente {
50     icon = "<i class='fa fa-user-circle'> ";
51     str = icon"<b>"$2"</b>:" $3"</i>";
52     gsub("_"," ", str);
53     printf(fmtli, str) > file;
54
55     if($2~/CODIGO_POSTAL/) {
56         print "</ul>" > file;
57         infoCliente--;
58     }
59 }
60
61 $2~/DATA_ENTRADA/ && $3~/[0-9-]+/ {
62     dias[$3]++;
63     ultDia = $3;
64     adicionarEntrada = 1;
65 }
66
67 $4~/\\SAIDA/ {
68     locaisSaida[$3]++;
69     if(parque){
70         locaisEntrada[ultDia] = temp"--"$3;
71         parque = 0;
72     }
73 }
74
75 $4~/\\ENTRADA/ {
76     if(adicionarEntrada) {
77         temp = locaisEntrada[ultDia];
78         if(temp)
79             locaisEntrada[ultDia] = temp"--"$3;
80         else
81             locaisEntrada[ultDia] = $3;
82         if($3 == "") parque = 1;
83         adicionarEntrada = 0;
84     }
85 }
86
87 $2~/HORA_ENTRADA/ {
88     horaEntrada = $3;
89 }
90
91 $2~/HORA_SAIDA/ {
92     horaSaida = $3;
93     i = duracao(horaEntrada, horaSaida);
94     duracoes[ultDia] += i;
95 }
96
97 $2~/IMPORTANCIA/ {
98     gsub(",",".", $3);

```

```

99     totalGasto += $3;
100     ultimoGasto = $3;
101     totDias[ultDia] += $3;
102 }
103
104 $2~/VALOR_DESCONTO/ {
105     gsub(",", ".", $3);
106     totalGasto -= $3;
107     ultimoGasto -= $3;
108     totDias[ultDia] -= $3;
109 }
110
111 $2~/TIPO/ {
112     totalLocais[$3] += ultimoGasto;
113 }
114
115 END {
116     str = "<h2 class='bg-4'>Informação:</h2>";
117     print str > file;
118
119     str = "Nº de entradas em cada dia do mês:";
120     printf(fmth4, str) > file;
121     print "<ul>" > file;
122     for (i in dias) {
123         click = " onclick='openList(\""i"\")'><a>";
124         printf(fmtEntrad, click, i, dias[i], totDias[i]) > file;
125         print "</a><ol id='\"i\"'>" > file;
126         split(locaisEntrada[i], a, "--");
127         for(j in a)
128             printf(fmtli, a[j]) > file;
129         print "</ol>" > file;
130         icon = "<i class='fa fa-clock-o'> ";
131         aux = "Tempo total em viagem: ";
132         str = icon aux duracoes[i] " minutos</i>";
133         printf(fmth5, str) > file;
134     }
135     print "</ul>" > file;
136
137
138
139     printf(fmth4, "Locais de saída:") > file;
140     print "<ul>" > file;
141     for (i in locaisSaida) {
142         aux = sprintf(ref, maps i, i);
143         icon = "<i class='fa fa-map-marker'> ";
144         str = icon aux " </i>" "("locaisSaida[i]")";
145         printf(fmtli, str) > file;
146     }
147     print "</ul>" > file;
148
149
150     str = "Total gasto no mês:";
151     printf(fmth4, str) > file;
152     str = "Total:" " " totalGasto"€";
153     print "<div class='aux'>" > file;
154     printf(fmth2, str) > file;
155     print "<ul>" > file;
156     for(i in totalLocais)
157         printf(fmtTotais, i, totalLocais[i]) > file;

```

```

158     print "</ul>" > file;
159
160     print end > file;
161 }
162
163 #Args: he - Hora entrada
164 #      hs - Hora saida
165 #Return duracao em minutos
166 function duracao(he, hs) {
167     result = 0;
168     split(he, ae, /[:]/);
169     split(hs, as, /[:]/);
170     if(ae[1]==" " || as[1]==" ")
171         return 0;
172     if(as[1]>=ae[1])
173         aux = (as[1]-ae[1])*60;
174     else
175         aux = (as[1] + (24 - ae[1]))*60;
176     result += aux;
177     result -= ae[2];
178     result += as[2];
179     return result;
180 }

```