Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 1

Relatório de Desenvolvimento

Tiago Barata	Duarte Oliveira	Simão Oliveira
(a81195)	(a85517)	(a57041)

24 de abril de 2021

Resumo

O presente trabalho de processamento de linguagens visa o enunciado 5 em que depois de extrair a informação útil de um *dataset* foi necessário organizar e apresentar a informação através de páginas *html*.

Conteúdo

Capítulo 1

Introdução

No âmbito unidade curricular Processamento de Linguagens, ano letivo 2020/2021, foi-nos proposto desenvolver o enunciado **Machine Learning: datasets de treino**, onde o objetivo é a extração de vários elementos informativos de datasets. Nomeadamente, do dataset de treino disponibilizado pela Google para treino docente TensorFlow: *'train.txt'*.

Este relatório inicia-se com um breve enquadramento acerca do enunciado, seguindo-se a metodologia abordada pelo grupo de forma a obter os resultados pretendidos. Posteriormente, é feita uma a exposição da arquitetura da solução, sendo descritas as diversas estruturas criadas. Por fim, são apresentadas algumas conclusões sobre a elaboração do caso de estudo abordado. O objetivo principal deste relatório é assim expor o trabalho desenvolvido e explicar as razões das decisões tomadas ao longo do projeto.

Supervisor: Pedro Rangel Henriques, José Carlos Ramalho, Pedro Moura

1.1 Machine Learning: datasets de treino

Área: Processamento de Linguagens

Enquadramento: filtrar informação através de expressões regulares

Contexto: no âmbito do primeiro trabalho prático desta UC

Problema: analisar um *dataset* e retirar as informações importantes

Objetivo: dividir por categorias os resultados obtidos

Resultados ou Contributos: construção de um pequeno website em HTML

Estrutura do documento análise e especificação, concepção e desenho, codificação e testes

Estrutura do Relatório

No capítulo 2 faz-se uma análise detalhada do problema proposto de modo a poder-se especificar as entradas, resultados e formas de transformação.
No capítulo 3 será explicado o modo de resolução dos problemas, as estruturas de dados usados e os algoritmos criados para se resolver o problema proposto.
No capítulo 4 faz-se uma análise mais prática, relativamente ao código usado e o que motivou o grupo a usar certas soluções.
Por fim, no capítulo 5 termina-se o relatório com uma síntese do que foi dito.

Capítulo 2

Análise e Especificação

2.1 Descrição informal do problema

O problema proposto foi a filtragem e manipulação de dados de um *dataset* da Google, no âmbito do estudo de algoritmos de Machine Learning que necessitam de ser treinados com um dataset. Assim, e de forma mais sucinta, o objetivo é extrair do dataset *train.txt* a informação útil para depois ser tratada e mostrada ao utilizador através da apresentação dos resultados em páginas HTML.

2.2 Especificação dos Requisitos

2.2.1 Dados

O excerto seguinte contém informação relativa a um extrato do ficheiro fornecido pela equipa docente train.txt.

```
B-GENRE science
I-GENRE fiction
I-GENRE films
O directed
O by
B-DIRECTOR steven
I-DIRECTOR spielberg
...
O move
O that
B-ACTOR frank
I-ACTOR sinatra
O was
O in
...
```

Em relação a estes dados foram tomadas em consideração as diferentes assunções:

- Há duas palavras-chave:
  - **Categoria**
    Refere-se a qualquer conjunto de caracteres em letras maiúsculas delimitado por "**B-**"ou por "**I-**",

e um carácter de espaço. As categorias são caracterizadas por **elementos** a ela associados pelo seu prolongamento:

  * Há uma "*flag*" **B** - que denota o início de uma categoria.
  * Há uma "*flag*" **I** que denota a continuação de uma categoria.

- **Elemento**
  Constituído por um ou mais conjuntos de caracteres (caso haja prolongamento), conjuntos esses que ocorrem após o espaço que delimita a categoria.

  * Um elemento é sempre introduzido pela palavra associada ao delimitador **B** de uma determinada linha do *dataSet*. O número de palavras que completam e que trazem o restante significado ao elemento vão ser dados pelo número de "*flags*" **I** e o conjunto de caracteres associados a cada uma dessas "*flags*".

- **Não há** repetição de categorias.
- **Pode** haver repetição de elementos.
- A nível de dados cada categoria vai conter **um conjunto de elementos**.

2.2.2 Pedidos

Ao nível dos pedidos que foram feitos o objetivo será identificar numa primeira fase se a linha consultada se refere ao inicio de uma categoria, ou se acaba por ser referente à continuação de uma categoria, dado que qualquer outra entrada será ignorada. Assim, posteriormente, a extração destes dados pode então ser agrupada, sendo possível criar vários tipos de relações entre os vários dados das categorias.

2.2.3 Relações

Como referido anteriormente, após a extração dos dados relevantes de cada categoria do *dataset*, houve a necessidade de criar várias relações entre os dados. Fez-se a sua agregação por categorias, sendo possível também visualizar em quantas entradas/linhas do ficheiro aparece uma determinada categoria, verificar a linha de ocorrência de cada entrada na categoria, bem como quantas entradas existem no *dataset*.
Como extra, o grupo optou por fazer uma dupla análise, e apresentar ambos os resultados ao utilizador: primeiramente uma análise que contemplava repetidos, e posteriormente uma análise que não contempla repetidos, concatenando toda a informação, por cada repetido.

Capítulo 3

Concepção/desenho da Resolução

Descreve-se nesta secção do documento o raciocínio e a ideia por trás de todo o programa desenvolvido, explicando de que forma e como se estruturaram e especificaram os dados relativos a este problema. De forma a organizar este problema foi necessário organizar toda a informação relativa ao *dataset* de forma coerente e correta em estruturas de dados, passando antes por algoritmos que possibilitam essa mesma estruturação.

3.1 Estruturas de Dados

Assim, dados os objetivos que inferimos, pela nossa interpretação, desenvolver neste projeto, definimos quatro estruturas de dados para aglomerar informação, todas com a mesma natureza.

- Lista ***lista***

Apesar de não ser uma estrutura totalmente fixa no nosso projeto, desenvolvemos uma lista para se poder trabalhar com o *dataSet* fornecido. Para isso abriu-se (*open()*) e leu-se (*read()*) o ficheiro associado ao mesmo e fez-se *split* linha a linha ($re.split(r'\backslash n', open(file).read())$). Ficou-se com uma lista em que cada elemento corresponde a uma linha do ficheiro.

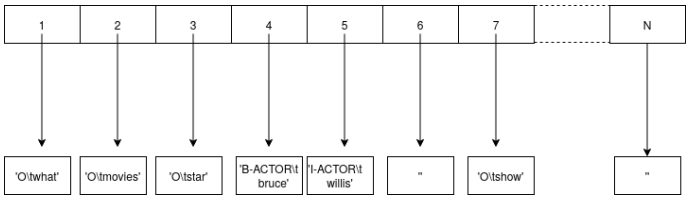


Figura 3.1: Estrutura figurativa da lista *lista*

As restantes estruturas são todas similares. Na linguagem de programação *Python* estas definem-se como dicionários (um *data type* bastante flexível e genérico mas bastante simplista nesta linguagem) que basicamente possuem uma coleção de itens do tipo *Key*, *Value*. Cada um destes dicionários tem uma utilidade em específico para este enunciado.

- Dicionário ***dic***

Este *dataType* tem como intuito guardar como *Key* a *string* referente a cada categoria que for encontrada no *dataSet* e como *value* a lista de *strings* referente a cada elemento correspondente a essa categoria em questão. Exemplifica-se de seguida esta estrutura.

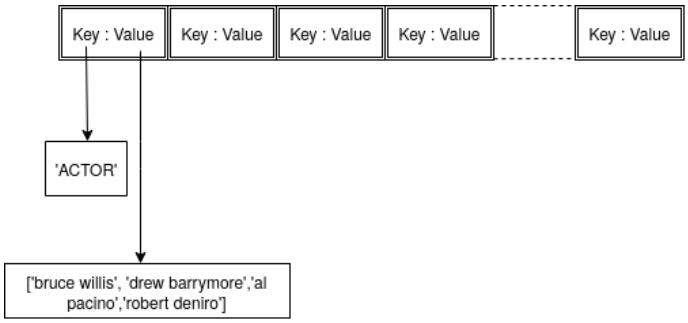


Figura 3.2: Estrutura figurativa do dicionário *dic*

- Dicionário ***info***

Este *dataType* tem como intuito guardar como *Key* a *string* referente a cada categoria que for encontrada no *dataSet* e como *value* a lista das linhas onde se iniciam as ocorrências de cada um dos elementos dessa categoria em questão. Exemplifica-se de seguida esta estrutura.

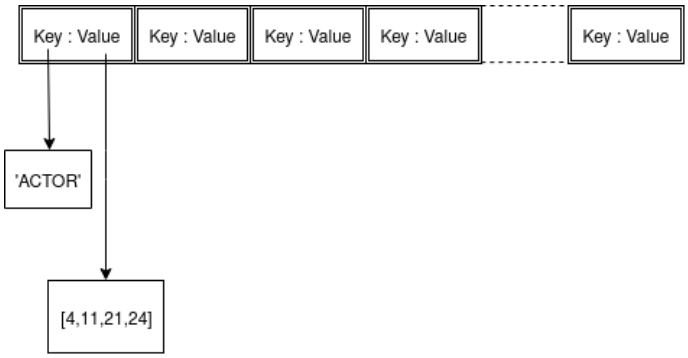


Figura 3.3: Estrutura figurativa do dicionário *info*

Entenda-se portanto, que estes últimos dicionários são paralelos, ou seja, estão organizados de forma a que cada elemento pertencente ao dicionário *dic* tenha o seu valor correspondente no dicionário *info* de forma a que a informação esteja disposta de uma forma mais simples. A nível de eficiência perde-se um pouco devido a estarem a ser criadas duas estruturas, quando bastaria associar a cada *key* do dicionário uma lista de *tuplos (elemento, linha da ocorrência)* mas para a elegibilidade do código e com o intuito de talvez implementar algum tipo de *extra* associado apenas a este parâmetro acabou-se por manter ambas as estruturas conforme referido.

- Dicionário ***semRepetidos***

De forma a trazer alguma profundidade intlectual e criativa a este projeto, decidimo-nos a criar mais um dicionário. Aproveitando a flexibilidade que o *Python* nos oferece, desenvolvemos uma estrutura um pouco peculiar. Esta contém como *key* o nome da categoria e como *value* um outro dicionário. Este vai ter como *key* o elemento e como *value* uma lista com todas as ocorrências do elemento no *dataSet*.

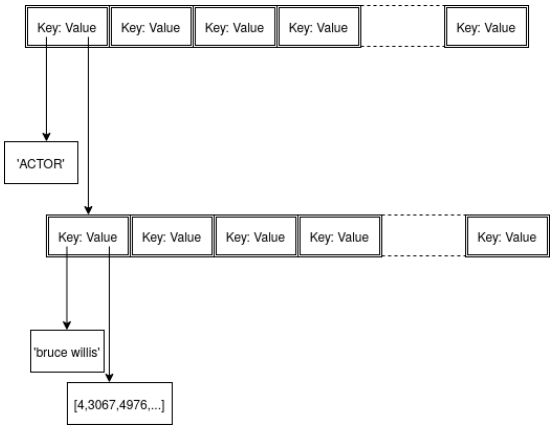


Figura 3.4: Estrutura figurativa do dicionário *semRepetidos*

3.2 Algoritmos

3.2.1 Ficheiro *main.py*

A nível de *back-end* foram desenvolvidos 3 algoritmos de alguma relevância para a organização das estruturas que nos darão o resultado deste problema.

- omega
- alpha
- beta

Omega

Esta função itera sobre a estrutura *lista* e tomando partido da biblioteca *'re'* utiliza a função *re.compile* e *re.search*, verificando a presença de duas expressões regulares nas linhas a serem iteradas:

```
keyB = re.compile(r'B-([^ ]+)\s(.+)')
keyI = re.compile(r'I-([^ ]+)\s(.+)')
```

Caso haja *match* com a ***keyB***, e dada a natureza da função *search* guarda-se em duas variáveis o *group(1)* e o *group(2)* derivados dos grupos que compõem a expressão regular da **keyB**:

```
r'([^ ]+)\s'
r'(.+)'
```

Estas duas variáveis terão em si guardadas a categoria e o elemento (ou parte do elemento) da linha iterada. Após isso, cria-se um novo item categoria: [elemento] no dicionário *dic* caso a *key* não exista ou adiciona-se o elemento à lista correspondente ao *value* da categoria à qual o elemento pertence. Caso haja *match* com a ***keyI*** mais uma vez guarda -se em duas variáveis o *group(1)* e o *group(2)* derivados dos segunites grupos que compõem a expressão regular da **keyI**:

```
r'([^ ]+)\s'
r'(.+)'
```

Alpha

Esta função segue a lógica do algoritmo *omega*, recebendo a lista *lista* mas contando apenas com a expressão regular compilada graças à biblioteca *re*:

```
keyB = re.compile(r'B-([^ ]+)\s(.+)')
```

Na nossa interpretação sentimos que era coerente que as linhas correspondentes à ocorrência de um elemento no *dataset* fossem aquelas nas quais os elementos eram introduzidos, ou seja, naquelas que fossem definidas por B. Assim, adiciona-se a categoria:[ocorrência] ao dicionário *info* caso a *key* não existisse, ou adiciona-se à lista representativa do *value* o valor da linha da ocorrência do elemento, na categoria respetiva. O facto de ser usado o mesmo processo iterativo que na função *omega* deixa claro que não há problemas de incompatibilidade entre os dicionários *info* e *dic*. Retorna o dicionário *info*.
Trazendo um pouco a vertente teórica associada a esta Unidade Curricular, apresenta-se de seguida os **Autómatos Finitos** associados à **KeyB** e à **KeyI**, respetivamente:

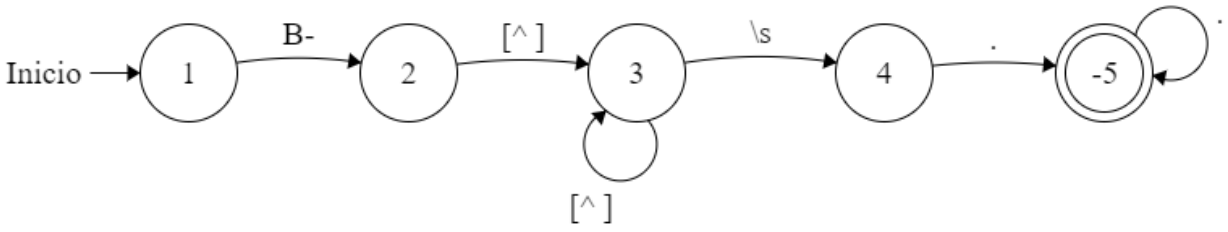


Figura 3.5: Autómato Finito associado à **KeyB**

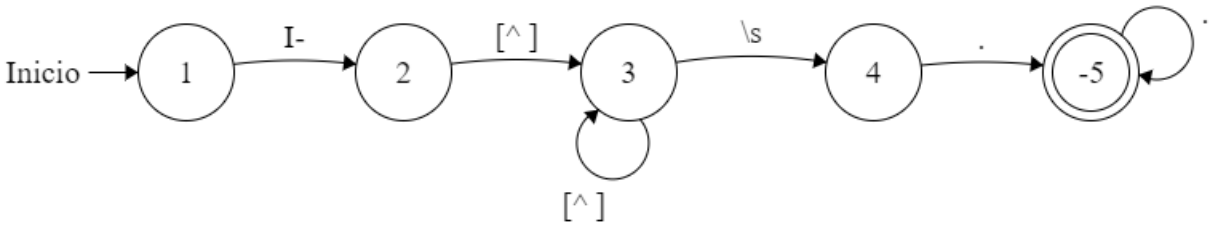


Figura 3.6: Autómato Finito associado à **KeyI**

Beta

Para se obterem os elementos não repetidos com as respetivas ocorrências, desenvolveu-se *Beta*. Esta vai receber os dicionários *dic* e *info*, convenientemente povoados, e como é certo o paralelismo entre ambos, para cada categoria do dicionário vai ser criado um dicionário. Para povoar-se este dicionário, faz-se *zip* dos *values* da categoria nos dicionários *info* e *dic*. Ou seja, fica-se, para cada categoria, com uma lista de *tuplos* (elemento,ocorrência).

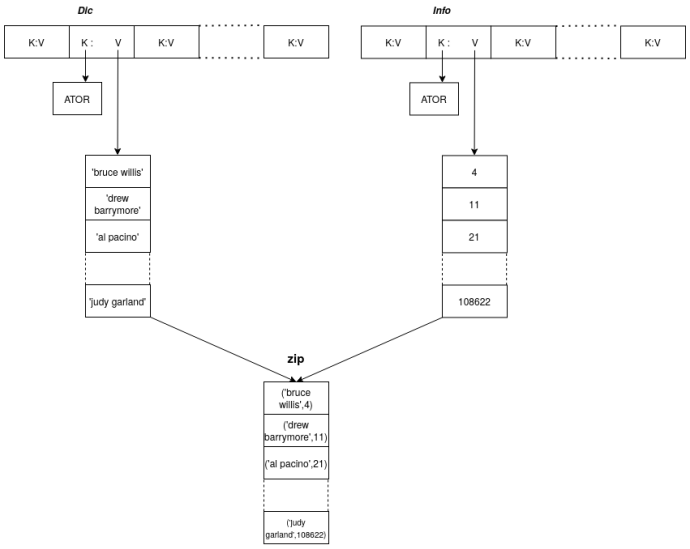


Figura 3.7: Figuração de parte do processo algorítmico associado a *beta*

Finalmente (e mais uma vez com uma lógica muito similar às funções anteriormente descritas) verificou-se a existência ou não de cada elemento no dicionário dedicado à sua respetiva categoria, e consoante a sua presença ou não, adicionou-se esse elemento como *key* e fez-se *append* da sua ocorrência (na lista que é tomada como *value* para esse elemento) ou criou-se um novo item elemento:[ocorrência], atualizando o dicionário. Finalmente é de notar que para além do dicionário semRepetidos, esta função também retorna uma lista (*naoRepetidos*). Esta foi desenvolvida numa parte mais tardia do nosso projeto e teve como intuito auxiliar a indicação do número de elementos não repetidos em cada categoria no desenvolvimento do website que nos foi requerido.

main

A função onde se chamam todas as funções anteriormente descritas mais as que serão especificadas no resto deste documento. Para além de alguma interação com um possível utilizador sentimos que é relevante referimos que de modo a trazer alguma robustez ao programa adicionamos algumas particularidades de pouca relevância para a Unidade Curricular mas que denotam algum esforço e apreciação pelo projeto que desenvolvemos. Em primeiro lugar, e considerando que poderia haver o uso com diferentes *dataSet* consideramos e implementamos a remoção dos ficheiros temporários de se gerar diferentes páginas HTML. Para isso fizemos uso do package '*os*'. Em segundo lugar e de modo a trazer um aspeto mais *clean* ao programa utilizamos a biblioteca *webbrowser* que nos permitiu abrir as páginas *web* diretamente no browser predefinido do utilizador.

3.2.2 Ficheiro *html.py*

A nível de *front-end*, foram desenvolvidas funções que permitissem escrever em ficheiros, tanto o HTML, como o CSS, bem como a respetiva criação de diretorias.

- makeDirs
- htmlStyle
- htmlHome
- htmlRepetidos
- htmlSemRepetidos

MakeDirs

Esta funcão permite a criação das diretorias necessárias para criar e manter organizado o código HTML. Começa por se criar uma pasta com o nome 'HTML' na diretoria atual. De seguida cria duas subpastas da pasta refererida anteriormente, tendo os nomes 'categorias' e 'styles'. Para a criação e gestão, tira-se partido do módulo *'OS'* do *Python*, onde se verifica inicialmente se as pastas existem. Se as pastas existirem, e um programa tentar criar uma pasta, é gerado um erro. Optou-se então por fazer este controlo de erros, criando as pastas apenas se não existirem.

HtmlStyle

Para obtermos uma página HTML apelativa ao utilizador foi necessário usar CSS para embelezar os dados a serem mostrados. Esta, apesar de ser a função mais extensa do código, é também a mais simples. É responsável por criar 2 ficheiros CSS: um responsável por embelezar a página principal e as páginas de informação dos repetidos, e outro para embelezar as páginas de ficheiros únicos.

HtmlHome, htmlRepetidos, e htmlSemRepetidos

Estas três funções são iguais se olharmos ao resultado final de cada uma delas: produzir ficheiros HTML. Podiam muito bem ser feitas em apenas uma função, mas a separação torna mais simples a leitura e interpretação do código.
A função 'htmlHome' é responsável por criar a página principal, com os dados gerais de cada categoria e todas as suas hiperligações às páginas que irão existir.
Já as funções 'htmlRepetidos' e 'htmlSemRepetidos' usam as estruturas de dados existentes, para criar tabelas, onde são apresentados os elementos em cada categoria, e a linha onde foi capturado no *dataset*.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Inicialmente criou-se um único ficheiro *Python* que cobria a resolução do problema. Posteriormente, visto que grande parte da resolução do problema era exportar para ficheiros HTML, optou-se por separar a resolução em 2 ficheiros. O primeiro que visa analisar o *dataset* fornecido, fazer o tratamento da sua informação e descartar toda a informação não necessário e o segundo que, após receber certas estruturas de dados como parâmetros, gerar o(s) respetivo(s) ficheiro(s) HTML, organizados nas diretorias indicadas.
Relativamente às diretorias, optou-se por criar uma diretoria chamada 'HTML', onde se colocou o ficheiro referente à página inicial e 2 sub-diretorias, '*styles*' e 'categorias', onde se organizam os ficheiros HTML referentes a cada uma dessas diretorias.
Outra decisão tomada pelo grupo após visualização do HTML num *web browser* foi tornar as páginas mais apelativas visualmente. Após algum trabalho de pesquisa, optou-se por implementar dois ficheiros *styles*, baseados em CSS para melhorar essa apresentação.
Para evitar erros do *Python* na leitura de ficheiros ou de diretorias, implementou-se também controlo de erros nesse sentido, não só a criação das diretorias e ficheiros necessários, mas também a remoção de outros ficheiros já existentes nas diretorias previamente criadas. Para tal, usou-se o módulo *'OS'*.

4.2 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos com o *dataset train.txt* e os respectivos resultados obtidos:

Primeiramente, mostram-se em seguida a número de elementos por cada categoria:

```
ACTOR: 3220
YEAR: 2858
TITLE: 2376
GENRE: 4354
DIRECTOR: 1720
SONG: 245
PLOT: 1927
REVIEW: 221
CHARACTER: 385
RATING: 2007
RATINGS_AVERAGE: 1869
```

```
TRAILER: 113
```

De seguida, dado o tratamento de dados, apresentam-se o número de elementos únicos por categoria:

```
ACTOR: 1338
YEAR: 204
TITLE: 1615
GENRE: 313
DIRECTOR: 948
SONG: 173
PLOT: 1219
REVIEW: 102
CHARACTER: 257
RATING: 48
RATINGS_AVERAGE: 167
TRAILER: 19
```

Por fim, apresentam-se algumas capturas de ecrã da informação no website criado:

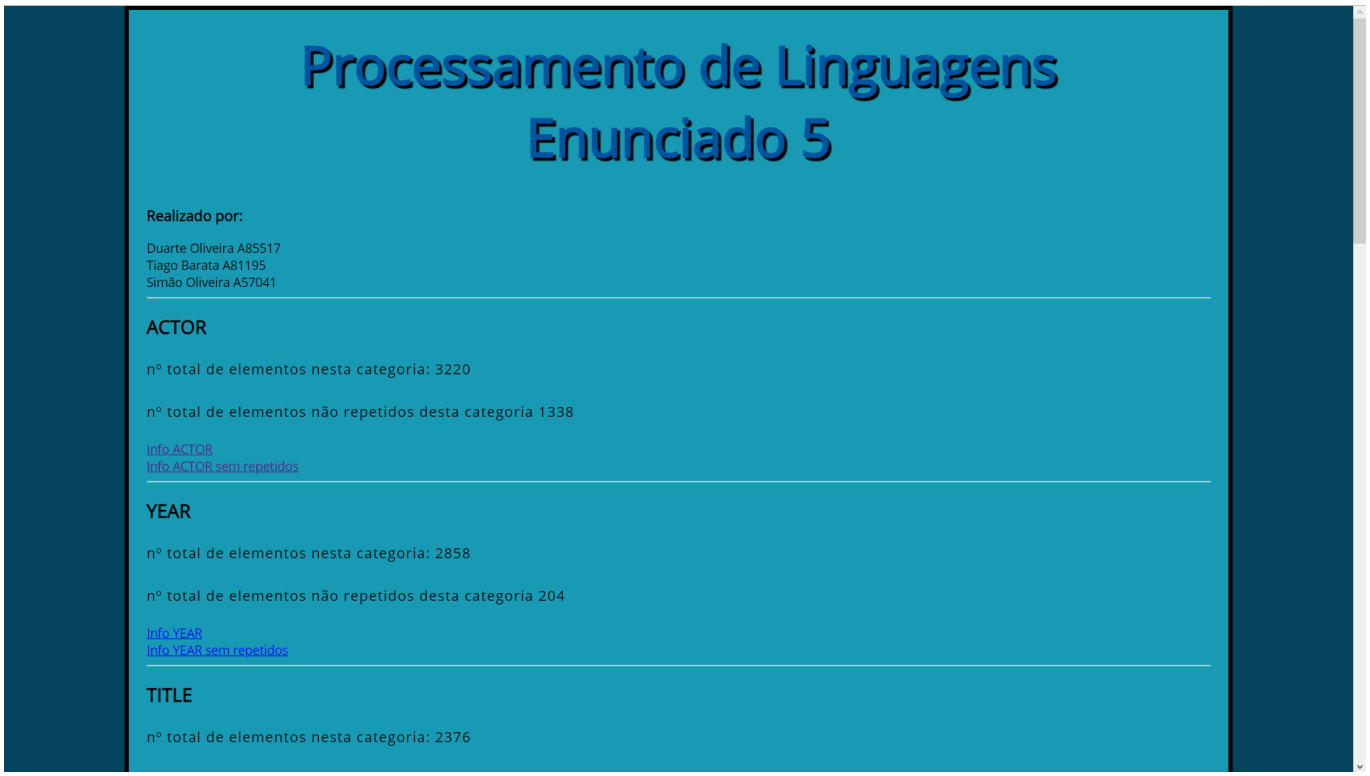
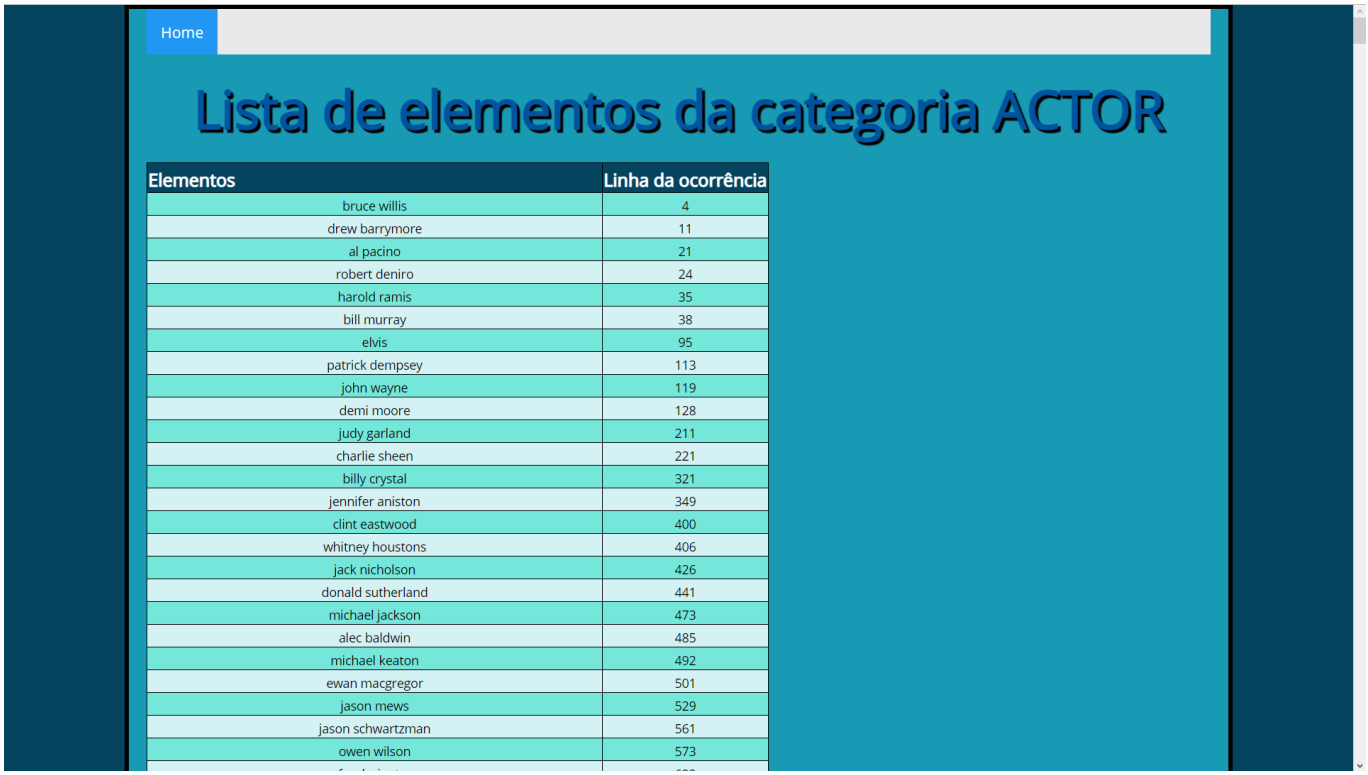
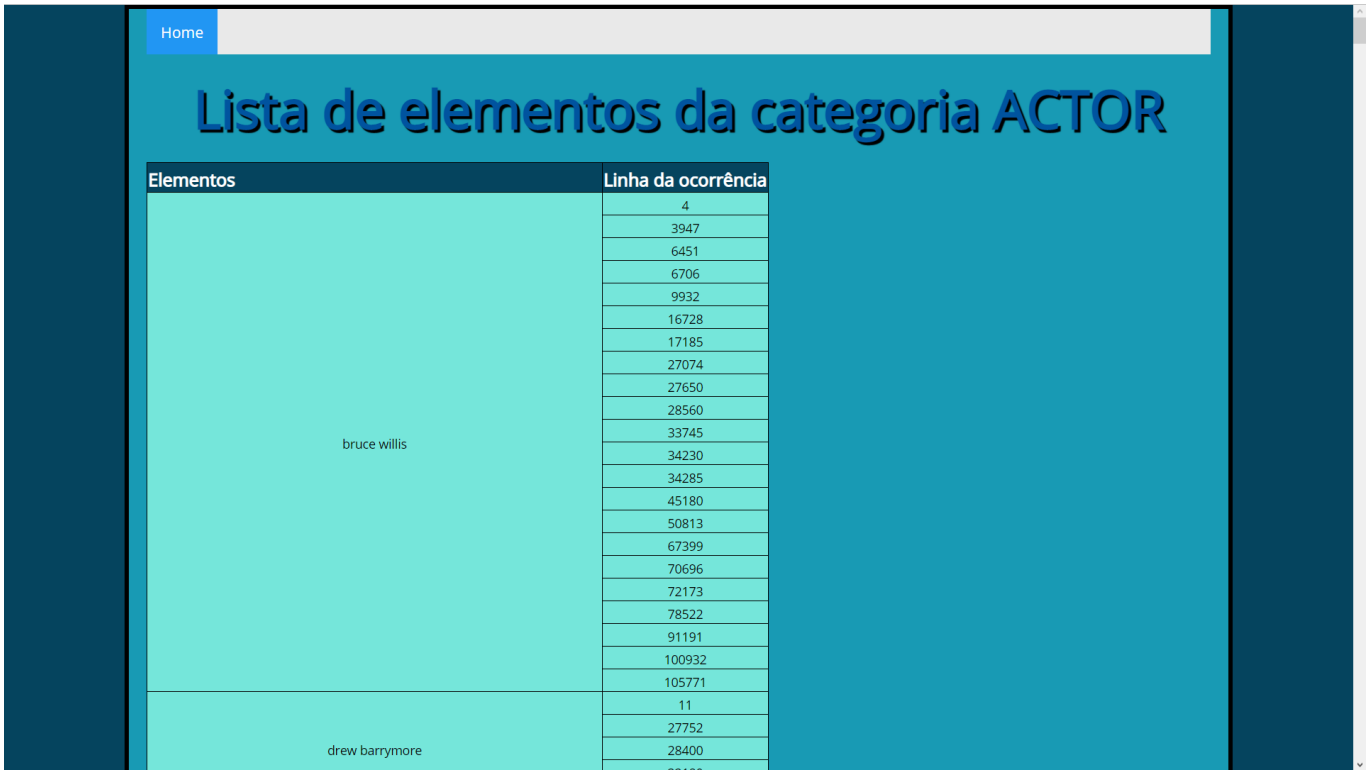


Figura 4.1: Página inicial



Elementos	Linha da ocorrência
bruce willis	4
drew barrymore	11
al pacino	21
robert deniro	24
harold ramis	35
bill murray	38
elvis	95
patrick dempsey	113
john wayne	119
demi moore	128
judy garland	211
charlie sheen	221
billy crystal	321
jennifer aniston	349
clint eastwood	400
whitney houstons	406
jack nicholson	426
donald sutherland	441
michael jackson	473
alec baldwin	485
michael keaton	492
ewan macgregor	501
jason mews	529
jason schwartzman	561
owen wilson	573

Figura 4.2: Categoria ator com repetições (exemplo)



Elementos	Linha da ocorrência
bruce willis	4
	3947
	6451
	6706
	9932
	16728
	17185
	27074
	27650
	28560
	33745
	34230
	34285
	45180
	50813
	67399
	70696
	72173
	78522
	91191
	100932
	105771
drew barrymore	11
	27752
	28400

Figura 4.3: Categoria ator sem repetições (exemplo)

Capítulo 5

Conclusão

Após a realização do projeto o grupo considera que deve ser feita uma análise critica dos resultados obtidos. Assim, para o *dataset* disponibilizado pela equipa docente, o grupo considera que os objetivos requeridos no enunciado foram atingidos e ainda foram exploradas algumas funcionalidades extra. Consideramos que esta nova abordagem às expressões regulares através da utilização de *python*, que hoje em dia é uma linguagem cada vez mais em uso, e relativamente acessível a nível de aprendizagem. As temáticas abordadas nas aulas permitiram então a criação de expressões regulares que juntamente com as funções já existentes no modulo ER nos permitiram satisfazer todos os objetivos. Também ao nível da apresentação de resultados em *html* foram apresentados novos desafios, pois este tipo de abordagem é novidade para alguns elementos do grupo. De salientar que várias abordagens foram pensadas como anteriormente descrito e que após várias alterações se atingiram todos os objetivos propostos. Em suma, consideramos que este projeto nos permitiu verificar como o uso de expressões regulares pode ser poderoso a nível de um caso de estudo real e permitiu-nos adquirir novas valências que poderão ser bastante proveitosas quer a nível académico, como no nosso futuro profissional.

Apêndice A

Código do Programa

Lista-se a seguir o código do ficheiro principal para análise do dataset e respetivo tratamento de dados.

```
import re
import webbrowser
import html
import os



def omega(lista):
  keyB = re.compile(r'B-([^ ]+)\s(.+)')
  keyI = re.compile(r'I-([^ ]+)\s(.+)')

  dic= {}
  typo = []
  for line in lista:

    if res:= keyB.search(line):
      category = res.group(1)
      catInfo = res.group(2)
      if category in dic:
        typo = dic[category]
      typo.append(catInfo)
      dic.update({category:typo})
      typo = []

    elif res:=keyI.search(line):
      category = res.group(1)
      catInfo = res.group(2)
      typo = dic[category]
      aux = typo[-1] + " " + catInfo
      del typo[-1]
      typo.append(aux)
      dic.update({category:typo})
      typo = []
    else:
      pass

  return dic


def alpha(lista):
  keyB = re.compile(r'B-([^ ]+)\s(.+)')
```

```
42    info = {}
43    typo = []
44    n = 1
45    for lines in lista:
46      if res:=keyB.search(lines):
47        category = res.group(1)
48        if category in info:
49          typo = info[category]
50        typo.append(n)
51        info.update({category:typo})
52        typo = []
53      n+=1
54    return info
55
56
57  def beta (dic, info):
58    semRepetidos = {}
59    naoRepetidos = []
60    for categoria in dic:
61      elemDic = {}
62      for elemento, ocorrencia in zip(dic[categoria], info[categoria]):
63        if elemento in elemDic:
64          elemDic[elemento].append(ocorrencia)
65        else:
66          elemDic.update({elemento:[ocorrencia]})
67      semRepetidos.update({categoria:elemDic})
68
69    for elem in semRepetidos:
70      a=0
71      for lista in semRepetidos[elem]:
72        a+=1
73      naoRepetidos.append(a)
74    return semRepetidos, naoRepetidos
75
76
77  def main():
78    html.makeDirs()
79
80    directory = 'HTML/categorias'
81
82    if len(os.listdir(directory))>0:
83      print("Possui ficheiros (possivelmente) tempor rios em mem ria. Gostaria de os
            remover antes de prosseguir?(Y/N)")
84      n=input().upper()
85
86      if n=='Y':
87        for f in os.listdir(directory):
88          os.remove(os.path.join(directory, f))
89        print("Ficheiros eliminados")
90      else:
91        print("N o  ser  removido qualquer ficheiro.")
92
93    print("Por favor, insira o nome do ficheiro a tratar:")
94    file = input()
95
96    if(os.path.isfile(file)):
97      lista = re.split(r'\n',open(file).read())
98    else:
99      print("O ficheiro  n  o  existe")
```

```
100     return
101
102   dic = omega(lista)
103   info = alpha(lista)
104   semRepetidos, naoRepetidos = beta(dic,info)
105
106   html.htmlStyle()
107   html.htmlHome(dic,naoRepetidos)
108   html.htmlRepetidos(dic,info)
109   html.htmlSemRepetidos(dic,semRepetidos)
110
111   webbrowser.open('file://' + os.path.realpath('HTML/website.html'), new=2)  # open in new
          tab
112
113
114 main()
```

Listing A.1: main.py

Lista-se a seguir o código do ficheiro que faz a exportação dos dados e respetiva formatação para HTML.

```
1  import os
2
3
4  def makeDirs():
5      dir0 = './HTML'
6      dir1 = './HTML/categorias'
7      dir2 = './HTML/styles'
8      if (os.path.exists(dir0) == False):
9          os.mkdir(dir0)
10     if (os.path.exists(dir1) == False):
11         os.mkdir(dir1)
12     if (os.path.exists(dir2) == False):
13         os.mkdir(dir2)
14
15
16 def htmlStyle():
17     directory = 'HTML/styles/style1.css'
18     directory2 = 'HTML/styles/style2.css'
19
20     if(os.path.isfile(directory) and os.path.isfile(directory2)):
21         return
22
23     htmlstyle = open(directory,"w")
24     htmlstyle.write(r'''html {
25     font-size: 14px;
26     font-family: 'Open Sans', sans-serif;
27 }
28
29
30 h1 {
31     font-size: 60px;
32     text-align: center;
33 }
34
35 p, li {
36     font-size: 16px;
37     line-height: 2;
38     letter-spacing: 1px;
39 }
```

```
40
41 tr:nth-child(even) {
42     background-color: #75E6DA;
43 }
44
45 tr:nth-child(odd) {
46     background-color: #D4F1F4;
47 }
48
49 table, th, td{
50     border: 1px solid black;
51     border-collapse: collapse;
52     padding-top: 5px;
53     text-align: center;
54 }
55
56 th{
57     font-size: 20px;
58     text-align: left;
59     height: 50%;
60     background-color: #05445E;
61     color: white;
62 }
63
64 html {
65     background-color: #05445E;
66 }
67
68 body {
69     width: 1200px;
70     margin: 0 auto;
71     background-color: #189AB4;
72     padding: 0 20px 20px 20px;
73     border: 5px solid black;
74
75 }
76
77 h1 {
78     margin: 0;
79     padding: 20px 0;
80     color: #00539F;
81     text-shadow: 3px 3px 1px black;
82 }
83
84 /* Add a black background color to the top navigation bar */
85 .topnav {
86     overflow: hidden;
87     background-color: #e9e9e9;
88 }
89
90 /* Style the links inside the navigation bar */
91 .topnav a {
92     float: left;
93     display: block;
94     color: black;
95     text-align: center;
96     padding: 14px 16px;
97     text-decoration: none;
98     font-size: 17px;
```

```
}

/* Change the color of links on hover */
.topnav a:hover {
    background-color: #ddd;
    color: black;
}

/* Style the "active" element to highlight the current page */
.topnav a.active {
    background-color: #2196F3;
    color: white;
}

/* Style the search box inside the navigation bar */
.topnav input[type=text] {
    float: right;
    padding: 6px;
    border: none;
    margin-top: 8px;
    margin-right: 16px;
    font-size: 17px;
}

/* When the screen is less than 600px wide, stack the links and the search field
    vertically instead of horizontally */
@media screen and (max-width: 600px) {
    .topnav a, .topnav input[type=text] {
        float: none;
        display: block;
        text-align: left;
        width: 100%;
        margin: 0;
        padding: 14px;
    }
    .topnav input[type=text] {
        border: 1px solid #ccc;
    }
}  ''')

    htmlstyle = open(directory2,"w")
    htmlstyle.write(r'''html {
    font-size: 14px;
    font-family: 'Open Sans', sans-serif;
}


h1 {
    font-size: 60px;
    text-align: center;
}

p, li {
    font-size: 16px;
    line-height: 2;
    letter-spacing: 1px;
}

table, th, td{
```

```
157     border: 1px solid black;
158     border-collapse: collapse;
159     padding-top: 5px;
160     text-align: center;
161     background-color: #75E6DA;
162 }
163 
164 th{
165     font-size: 20px;
166     text-align: left;
167     height: 50%;
168     background-color: #05445E;
169     color: white;
170 }
171 
172 html {
173     background-color: #05445E;
174 }
175 
176 body {
177     width: 1200px;
178     margin: 0 auto;
179     background-color: #189AB4;
180     padding: 0 20px 20px 20px;
181     border: 5px solid black;
182 
183 }
184 
185 h1 {
186     margin: 0;
187     padding: 20px 0;
188     color: #00539F;
189     text-shadow: 3px 3px 1px black;
190 }
191 
192 /* Add a black background color to the top navigation bar */
193 .topnav {
194     overflow: hidden;
195     background-color: #e9e9e9;
196 }
197 
198 /* Style the links inside the navigation bar */
199 .topnav a {
200     float: left;
201     display: block;
202     color: black;
203     text-align: center;
204     padding: 14px 16px;
205     text-decoration: none;
206     font-size: 17px;
207 }
208 
209 /* Change the color of links on hover */
210 .topnav a:hover {
211     background-color: #ddd;
212     color: black;
213 }
214 
215 /* Style the "active" element to highlight the current page */
```

```
.topnav a.active {
    background-color: #2196F3;
    color: white;
}

/* Style the search box inside the navigation bar */
.topnav input[type=text] {
    float: right;
    padding: 6px;
    border: none;
    margin-top: 8px;
    margin-right: 16px;
    font-size: 17px;
}

/* When the screen is less than 600px wide, stack the links and the search field
    vertically instead of horizontally */
@media screen and (max-width: 600px) {
    .topnav a, .topnav input[type=text] {
        float: none;
        display: block;
        text-align: left;
        width: 100%;
        margin: 0;
        padding: 14px;
    }
    .topnav input[type=text] {
        border: 1px solid #ccc;
    }
}  ''')


def htmlHome(dic,naoRepetidos):

    htmlfirst = open("HTML/website.html","w")

    htmlfirst.write("""<!DOCTYPE html>
<html>
    <head>
        <title>Enunciado 5</title>
        <meta charset="UTF-8"/>
        <link href="http://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet"
            type="text/css">
        <link href="styles/style1.css" rel="stylesheet" type="text/css">
    </head>
    <body>
        <h1>
            <b>Processamento de Linguagens</b>
            <br>
            Enunciado 5
        </h1>
        <p><h3><b> Realizado por: </b></h3></p>

            Duarte Oliveira A85517
            <br>
            Tiago Barata A81195
            <br>
            Sim&atildeo Oliveira A57041
        <hr>""")
```

```
273     x = 0
274     for elem in dic:
275         htmlfirst.write(
276             rf"""
277         <h2>{elem}</h2>
278             <p>n&ordm total de elementos nesta categoria: {len(dic[elem])} </p>
279             <p>n&ordm total de elementos n&atildeo repetidos desta categoria {naoRepetidos
                [x]} </p>""")
280
281         htmlfirst.write(
282         rf"""
283             <a href="categorias/{elem}Info.html"> Info {elem}</a>
284             <br>
285             <a href="categorias/{elem}InfoSRepeticao.html"> Info {elem} sem repetidos </a>
286             <hr>""")
287         x+=1
288
289     htmlfirst.write(r"""
290     </body>
291 </html>""")
292
293
294 def htmlRepetidos(dic, repetidos):
295     for elem in dic:
296         htmln = open(rf'HTML/categorias/{elem}Info.html',"w")
297         htmln.write(rf"""
298 <!DOCTYPE html>
299 <html>
300     <head>
301         <link href="http://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet"
            type="text/css">
302         <link href="../styles/style1.css" rel="stylesheet" type="text/css">
303         <title>Informa&ccedil&atildeo {elem} </title>""")
304         htmln.write(r"""
305         <meta charset="UTF-8"/>
306     </head>
307     <body>
308         <div class="topnav">
309         <a class="active" href="../website.html"> Home </a>
310         </div>""")
311
312         htmln.write(rf"""
313         <h1>Lista de elementos da categoria {elem}</h1>
314         <table>
315             <tr>
316                 <th>Elementos</th>
317                 <th>Linha da ocorr&ecircncia</th>
318             </tr>""")
319
320         for listado, nrs in zip(dic[elem],repetidos[elem]):
321             htmln.write(rf"""
322             <tr>
323                 <td>{listado}</td>
324                 <td>{nrs}</td>
325             </tr>""")
326
327         htmln.write(rf"""
328         </table>
329     </body>
```

```
</html>""")


def htmlSemRepetidos(dic, semRepetidos):
    for elem in dic:
        htmlsemsepetidos = open(rf"HTML/categorias/{elem}InfoSRepeticao.html","w")
        htmlsemsepetidos.write(rf"""
<!DOCTYPE html>
<html>
    <head>
        <link href="http://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet"
            type="text/css">
        <link href="../styles/style2.css" rel="stylesheet" type="text/css">
        <title>Informa&ccedil &atildeo sem repetidos de {elem} </title>""")

        htmlsemsepetidos.write(r"""
        <meta charset="UTF-8"/>
    </head>
    <body>
        <div class="topnav">
            <a class="active" href="../website.html"> Home </a>
        </div>""")

        htmlsemsepetidos.write(rf"""
        <h1>Lista de elementos da categoria {elem}</h1>
        <table>
            <tr>
               <th>Elementos</th>
               <th>Linha da ocorr&ecircncia</th>
            </tr>""")

        for categoria in semRepetidos[elem]:
            tam = len(semRepetidos[elem][categoria])+1
            htmlsemsepetidos.write(rf"""
            <td rowspan={tam}>{categoria}</td>""")

            for ocorrencia in semRepetidos[elem][categoria]:
                htmlsemsepetidos.write(rf"""
            <tr>
                <td>{ocorrencia}</td>
            </tr>""")

        htmlsemsepetidos.write(rf"""
        </table>
    </body>
</html>""")
```

Listing A.2: html.py