Processamento de Linguagens e Compiladores (3º ano de Curso)

Trabalho Prático nº 1

Relatório de Desenvolvimento

Grupo nr. 15

Ruben Silva (a94633)

Carlos Costa (a94543)

12 de novembro de 2022

Resumo

Neste relatório encontra-se a resolução do Trabalho Prático 1 mais precisamente os exercícios 1 e 2 do respetivo enunciado de 2022 para a unidade curricular Processamento de Linguagens e Compiladores.
O objetivo deste trabalho é a utilização de expressões regulares (ER) para transformação de texto bem como processos de filtragem entre outros.

Conteúdo

Capítulo 1

Processador de Pessoas listadas nos Róis de Confessados

1.1 Análise do Problema 1

Neste problema recebemos 1 ficheiro que continha algumas informações sobre uns processos de pessoas listadas nos róis de confessados. Estas estavam claramente divididas pela expressão "::". O objetivo deste problema era obter:

- A frequência de processos por ano (primeiro elemento da data);
- A frequência de nomes próprios (o primeiro em cada nome) e apelidos (o ultimo em cada nome) por séculos;
- A frequência dos vários tipos de relação: irmão, sobrinho, etc.
- Obter um ficheiro JSON com os primeiros 20 registos desse ficheiro.

Na seguinte figura podemos ver um excerto desse ficheiro:

```
579::1904-05-21::Abel Marques Reis::Jose Joaquim Marques Reis::Bernardina Dantas::::
578::1901-11-12::Abel Martins Pereira::Serafim Martins Pereira::Emilia Goncalves::::
572::1883-02-01::Abel Pedro Pereira Freitas::Joao Freitas Oliveira::Cecilia Rosa Pereira::::
578::1900-08-30::Abel Silva Carvalho::Constantino Silva Rego::Margarida Rosa Carvalho::::
575::1894-04-30::Abelardo Jose Cerqueira Araujo::Jose Maria Araujo::Leopoldina Cerqueira Ribeiro Araujo::::
575::1894-04-30::Abelardo Jose Cerqueira Araujo::Jose Maria Araujo::Leopoldina Cerqueira Ribeiro Araujo::::
```

Figura 1.1: Excerto do ficheiro processos.txt

Cada coluna tinha respetivamente as seguintes informações:

- Coluna 1/2/3: Pasta::Data::Nome;
- Coluna 4/5: Pai/Mãe;
- Coluna 6: Observação.

1.2 Resolução do Problema 1

Para a resolução deste problema recorremos às seguintes soluções sequenciadas:

i) Criar um dicionário e alocar cada processo com as respetivas informações;

ii) Processar cada frequência individualmente;

iii) Criar o JSON;

1.2.1 Alocar no Dicionário

Esta parte do problema foi resolvido com a função chamada filtrar_info
Esta função inicialmente percorre o "file descriptor"do "processos.txt"que foi entregue por parâmetros e dá split ao código por linhas. Após isso é gerado uma lista que será percorrida novamente e levará outro split por cada vez que a expressão "::"aparecer e, claramente, se a linha não estiver vazia.
Após isso, é inicializada um dicionário que contém todas as informações necessárias para a realização deste problema.

```python
#Função para Filtrar Toda a informação e colocar-la num dicionário
def filtrar_info(fd):
    data_processos = []

    # Seperar por Linhas
    lista_newlines = fd.split("\n")

    # Seperar cada Linha em uma array com as informações
    for line in lista_newlines:
        if len(line):
            processos = line.split("::")

        # Dicionario default para colocar as informações
            dict_processo = {
                "Pasta": "",
                "Data": "",
                "Nome": "????",
                "Pai": "????",
                "Mae": "????",
                "Observacao": ""
            }
```

Figura 1.2: Excerto da função filtrar_info(fd) PARTE 1

Como visto na figura anterior, cada informação do nosso problema, agora numa lista criada pelo split, e chamada "paramatro", será processada pelas Expressões Regulares (ER):

- ER Data: `"[0-9]4-[0-9]1,2-[0-9]1,2"`;

- ER Numero da Pasta: `"[0-9]1,4"`;
- ER que verifica se existe palavras com letras maiúsculas: `"([A-Z]1[a-z]* ?)"`;
- ER que verifica se é uma observação: `"[^A-Z]1\."` .

Inicialmente, passa pelas ER da data e do nr da pasta que são as mais fáceis de identificar. Após isso, queremos encontrar a existência de letras maiúsculas nessa informação, se existir, então queremos encontrar uma letra maiúscula seguida de um ponto final, se isto acontecer, claramente é uma observação.
A alocação do Nome, Pai, Mãe é feita ordenadamente consoante o seu preenchimento no dicionário, pois eles aparecem de forma ordenada no ficheiro processos. Para conseguirmos isso, simplesmente vamos identificando se já foi ou não alocado. Como ultima alocação, se informação não se identificar com nenhumas das ER e ser igual a (), então essa informação não existe e convém marcar-la de algum modo.

Após esta verificação em cada umas ER anteriores, é alocado devidamente no dicionário e posteriormente metido numa lista para return

```
# Filtrar o Processo:
for parametro in processos:
    # Recolher a data
    if (re.match("[0-9]{4}-[0-9]{1,2}-[0-9]{1,2}", parametro)):
        dict_processo["Data"] = parametro
        continue
    # Recolher nr da Pasta
    if (re.match("[0-9]{1,4}", parametro)):
        dict_processo["Pasta"] = parametro
        continue
    # Recolher Informações que possuem Nomes Proprios (Começam por Letra maiscula)
    if (re.match("([A-Z]{1}[a-z]* ?)", parametro)):

        if (re.search("[^A-Z]{1}\.", parametro)):
            dict_processo["Observacao"] = parametro
            continue
        # Alocção ordenada de parametros ordenados:
        if (dict_processo["Nome"] == "????"):
            dict_processo["Nome"] = parametro
            continue
        if (dict_processo["Pai"] == "????"):
            dict_processo["Pai"] = parametro
            continue
        if (dict_processo["Mae"] == "????"):
            dict_processo["Mae"] = parametro
            continue
    # Informação Inexistente
    if (re.match("", parametro)):
        for (p1, p2) in dict_processo.items():
            if dict_processo[p1] == "" or dict_processo[p1] == "????":
                dict_processo[p1] = "!!NO INFO!!"
                break
```

Figura 1.3: Excerto da função filtrar_info(fd) PARTE 2

1.2.2 Cálculo das Frequências

Esta parte do problema foi resolvido com 3 funções distintas: calcular_freq_processos, calcular_freq_nomes_sec, calcular_freq_relacao.

calcular_freq_processos(data)

Nesta função era necessário calcular a frequência de processos por ano.
A nossa solução para este problema foi a criação de um dicionário auxiliar e uma pesquisa na nossa estrutura de dados que irá alocar nesse dicionário todos os processos de cada ano especifico.
Para a recolha do ano, usamos uma expressão regular que simplesmente pega no único valor com 4 dígitos (ano).
Após isto, a nossa nova data foi tabelada de forma ordenada.
A fórmula matemática para esta frequência foi:

$$\text{Frequência} = \frac{\text{Processos de um Certo Ano}}{\text{Todos os Processos}}$$

```python
def calcular_freq_processos(data):
    dict_anos = {}
    total=0
    #Obter todos os processos por ano
    for processo in data:
        year = re.match("[0-9]{4}", processo["Data"]).group()
        if year not in dict_anos:
            dict_anos[year]=0
        total+=1
        dict_anos[year]+=1

    #Tabelar
    print(dict_anos)
    for (nome,counter) in sorted(dict_anos.items()):
        n = counter / total
        print("O ano tem "+str(nome) + " tem freq. de processos de:" + str(n))

    pass
```

Figura 1.4: Função calc_freq_processos(data)

calcular_freq_nomes_sec(data)

Nesta função era necessário calcular a frequência de nomes por século.
Esta funciona foi de longe a mais complexa porque tivemos de filtrar os nomes que constavam nas observações A nossa solução para este problema foi a criação de dois dicionários auxiliares (nomes e apelidos) e uma pesquisa na nossa estrutura de dados que irá alocar nesses dicionários todos os nomes e apelidos de cada século.
Para a recolha do século, usamos uma expressão regular que simplesmente pega no único valor com 4 dígitos (ano) e aplicamos-lhe a seguinte formula matemática:

$$\text{Século} = \frac{(\text{Ano} - 1)}{100} + 1$$

```python
# Falta descobrir o que está no "obs"
def calcular_freq_nomes_sec(data):
    #Variaveis Iniciais
    seculos = {}
    lista = ["Nome", "Pai", "Mae"]
    total_nomes_sec = {}
    total_apelidos_sec = {}

    #Ciclar toda a data e recolher toda a informação necessário para o exercício
    for processo in data:
        # Recolher o Ano e Fazer o Dicionario com o Seculo
        year = re.match("[0-9]{4}", processo["Data"]).group()
        year = int((int(year)-1) / 100)+1
        if (year not in seculos):
            seculos[year] = {"Primeiro_Nome": {}, "Apelido_Nome": {}}
            total_apelidos_sec[year] = 0
            total_nomes_sec[year] = 0
```

Figura 1.5: Excerto da Função calc_freq_nomes_sec(data) PARTE 1

Numa segunda parte desta função, tivemos de recolher os nomes e os apelidos de cada processo. Para isto, percorremos os processos e filtrávamos o pretendido com ER. Inicialmente, ao percorrer os nossos dados, usamos uma ER para recolhemos logo o primeiro nome: (ER Numero da Pasta: `"[0-9]1,4"`).
Para recolhermos o apelido, tivemos bastante mais trabalho, pois muitos nomes, não eram uniformes, por exemplo:

```
:Diogo Teixeira Machado::Ana Pereira Faria (ou Ana Joaquina Faria)::Antonio Maximo Teixeira Machado,Irmao.
```

Figura 1.6: Exemplo de um Processo com o nome irregular

Para este caso especifico, simplesmente criamos duas ER que encontrava o parêntese "(", ")"e a partir dai extraiamos os dois possíveis apelidos.
Tivemos outro caso muito irregular também, em que continha uma virgula, por exemplo: "Maria Vale, Solteira". Para este caso também criamos uma ER que recolhia a palavra antes da vírgula.

```
# Recolher Primerio e Ultimo Nome de Pai........
for l in lista:
    if (processo[l] != "!!NO INFO!!"):
        primeiro_nome = re.search("^[A-Za-z]+", processo[l]).group()
        primeiro_nomes_lista.append(primeiro_nome)
```

Figura 1.7: Excerto da Função calc_freq_nomes_sec(data) PARTE 2.1

```
# Este if serve para os casos por exemplo "Ana Lopes Coelho (ou Ana Fernandes Lopes)"
if (re.search("\)", processo[l])):
    # Voltar a mandar o primeiro nome pra lista porque o Apelido é a unica coisa que difere
    primeiro_nomes_lista.append(primeiro_nome)

    # O apelido que está dentro de parentises
    apelido_nome = re.search(
        "[A-Za-z]*\)", processo[l]).group()[:-1]
    apelidos_lista.append(apelido_nome)
    # O apelido que está fora de parentises
    apelido_nome = re.search(
        "[A-Za-z]* \(", processo[l]).group()[:-2]
    apelidos_lista.append(apelido_nome)
else:
    aux=processo[l]
    #Em casos como: Maria Vale, Solteira
    if (re.search(",", processo[l])):
        aux=processo[l].split(",")[0]
    apelido_nome = re.search("[A-Za-z]+$", aux).group()
    apelidos_lista.append(apelido_nome)
```

Figura 1.8: Excerto da Função calc_freq_nomes_sec(data) PARTE 2.2

Neste excerto da função, filtramos o que estava na observação. Para isso criamos inicialmente uma ER que encontra um nome seguido de vírgula (o padrão que encontramos para este filtro) (ER = `"(([A-Z]1[a-z]1, ?)+)"` Após encontrarmos esse nome, aplicávamos as ER normais para obter o nome e o apelido.

```
# Recolher Primerio e Ultimo Nome das observaçoes.
if (processo["Observacao"] != "!!NO INFO!!"):
    obs_lista = re.findall(
        "(([A-Z]{1}[a-z]{1,} ?)+\,)", processo["Observacao"])
    #(nome,junk) pois a função findall está a criar um tuplo e nós queremos o primeiro membro
    for (nome, junk) in obs_lista:
        primeiro_nome = re.search("^[A-Za-z]+", nome[:-1]).group()
        primeiro_nomes_lista.append(primeiro_nome)
        apelido_nome = re.search("[A-Za-z]+$", nome[:-1]).group()
        apelidos_lista.append(apelido_nome)
```

Figura 1.9: Excerto da Função calc_freq_nomes_sec(data) PARTE 3

No final destas filtragens, simplesmente metemos dentro dos dicionário inicialmente criados e aplicamos a seguinte formula matemática a cada século:

$$\texttt{Frequência} = \frac{\texttt{Número de Certo Nome desse Século}}{\texttt{Todos os Nomes desse Século}}$$

Por fim, esses valores foram tabelados no output.

```
    # Somar os Apelidos e Nomes de cada Seculo
    for n in primeiro_nomes_lista:
        if n not in seculos[year]["Primeiro_Nome"]:
            seculos[year]["Primeiro_Nome"][n] = 0
        total_nomes_sec[year] += 1
        seculos[year]["Primeiro_Nome"][n] += 1
    for n in apelidos_lista:
        if n not in seculos[year]["Apelido_Nome"]:
            seculos[year]["Apelido_Nome"][n] = 0
        total_apelidos_sec[year] += 1
        seculos[year]["Apelido_Nome"][n] += 1

# Tabelar a Resposta:
for (sec, data) in sorted(seculos.items()):
    print("Seculo: "+str(sec))
    print("\tNomes:")
    for (nome, counter) in data["Primeiro_Nome"].items():
        final = counter / total_nomes_sec[sec]
        print("\t\t"+str(nome)+" tem uma freq: "+str(final))
    print("\tApelidos:")
    for (nome, counter) in data["Apelido_Nome"].items():
        final = counter / total_nomes_sec[sec]
        print("\t\t"+str(nome)+" tem uma freq: "+str(final))
```

Figura 1.10: Excerto da Função calc_freq_nomes_sec(data) PARTE 4

calcular_freq_relacao(data)
Nesta função era necessário calcular a frequência de relações.
A nossa solução para este problema foi a criação de um dicionário que guardará o nr de cada relação existente. No inicio da função, simplesmente percorremos os nossos dados e vamos somando ao dicionário sempre que existe mãe ou pai.

```
def calcular_freq_relacao(data):
    #Criação de Variaveis
    total_relacoes =0
    dict_relacoes = {
        "Pai": 0,
        "Mae": 0
    }

    #Procurar as relações e somar las
    for processo in data:
        # Ver se o Pai ou Mae existe (podem ser desconhecido)
        if (processo["Pai"] != "!!NO INFO!!"):
            dict_relacoes["Pai"]+=1
            total_relacoes+=1

        if (processo["Mae"] != "!!NO INFO!!"):
            total_relacoes+=1
            dict_relacoes["Mae"]+=1
```

Figura 1.11: Excerto da Função calc_freq_relacao(data) PARTE 1

Nesta parte da função, temos de encontrar as relações que estão presentes da observação e com para esse objetivo conseguimos encontrar um padrão após a virgula e a partir dai chegar à ER = "`"[^A-Z] (([A-Z]1[a-z]+)+)"`. Mas contudo, tivemos 2 casos que não se encaixavam neste padrão, e como tal, tivemos de os excluir.

```
#Relações que estão na observação
relacoes = re.findall("[^A-Z]\,(([A-Z]{1}[a-z ]+)+)\.",processo["Observacao"])
if(relacoes):
    relacoes = relacoes[0][0]

    #Tivemos de Adicionar estas 2 Excluisoes, pois eles sairam fora do nosso ER, Somente estes 2 casos!
    if not(re.match("Jeronimo Silva Coelho", relacoes) or re.match("Frei", relacoes)):
        if (relacoes not in dict_relacoes):
            dict_relacoes[relacoes] =0
        dict_relacoes[relacoes]+=1
        total_relacoes+=1
```

Figura 1.12: Excerto da Função calc_freq_relacao(data) PARTE 2

No fim de tudo, tabelamos o resultado

```
#Criar a Tabela:
print("Frequencias de Relacoes: ")
for (rel,counter) in dict_relacoes.items():
    freq = counter/total_relacoes
    print("\tA relação "+str(rel)+" tem de freq: " +str(freq))
```

Figura 1.13: Excerto da Função calc_freq_relacao(data) PARTE 2

1.2.3 Criar o JSON

Esta parte do problema, foi a mais simples de toda, pois já tínhamos os processos todos tratados devidamente. Simplesmente criamos uma função que "junta"o nosso dicionário a uma lista básica que é normalmente usada em ficheiros JSON.

```
#Criar o JSON
def criar_json(data,n):
    lista=[]
    counter=0
    for d in data:
        lista.append(d)
        if(counter > n-2):
            break
        counter+=1
    return lista
```

Figura 1.14: Função que cria o JSON

1.3 Resultados do Problema 1

1.3.1 Resultado da Frequência de Processos por Ano

Aqui temos um excerto, devido ao output ser extenso, do resultado da Frequência de Processos por Ano.

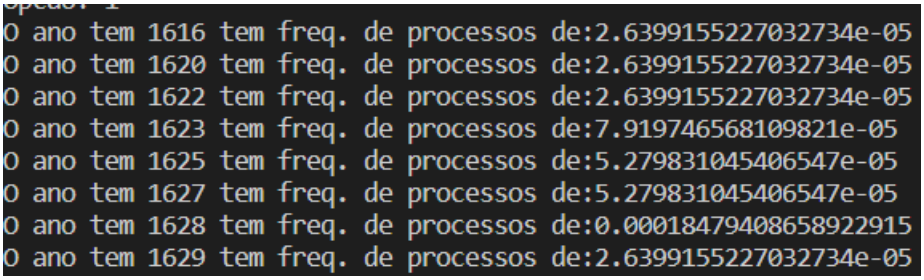

Figura 1.15: Excerto do Resulto da freq. de Processos por Ano

1.3.2 Resultado da Frequência de Nomes e Apelidos por Século

Aqui temos um excerto, devido ao output ser extenso, do resultado Frequência de Nomes e Apelidos por Século.

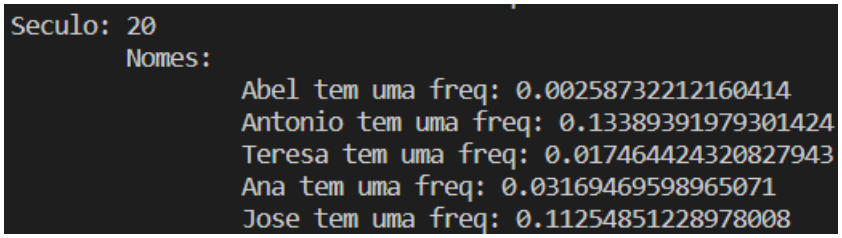

Figura 1.16: Excerto do Resulto da freq. Nomes e Apelidos por Século

1.3.3 Resultado da Frequência de Relações

```
Frequencias de Relacoes:
        A relação Pai tem de freq: 0.4157348898842443
        A relação Mae tem de freq: 0.4138841128864455
        A relação Tio Paterno tem de freq: 0.01446672434374076
        A relação Irmao tem de freq: 0.10870303242692717
        A relação Tio Materno tem de freq: 0.007720696943480118
        A relação Sobrinho Materno tem de freq: 0.008191604700316494
        A relação Sobrinho Paterno tem de freq: 0.009746695432194759
        A relação Filho tem de freq: 0.0019493390864389517
        A relação Sobrinhos Paternos tem de freq: 0.00031758895228499776
        A relação Sobrinho Neto Paterno tem de freq: 0.0007008859636634434
        A relação Primo tem de freq: 0.0040738996637937645
        A relação Primo Paterno tem de freq: 0.0011498910341353366
        A relação Irmaos tem de freq: 0.0062203629275130595
        A relação Tio Avo Paterno tem de freq: 0.00048185910001861726
        A relação Sobrinho Bisneto Paterno tem de freq: 3.2854029546723 91e-05
        A relação Irmao Materno tem de freq: 0.0004051996977429282
        A relação Irmaos Paternos tem de freq: 0.00020807552046258473
        A relação Sobrinhos Maternos tem de freq: 0.00041615104092516947
        A relação Irmao Paterno tem de freq: 0.0030882787773920473
        A relação Sobrinho Neto Materno tem de freq: 0.0007556426795746499
        A relação Primo Materno tem de freq: 0.0006680319341167194
        A relação Neto Materno tem de freq: 0.00016427014773361952
        A relação Tio Avo Materno tem de freq: 0.00027378357955603255
        A relação Neto Paterno tem de freq: 5.475671591120651e-05
        A relação Avo Materno tem de freq: 0.00010951343182241303
        A relação Primos tem de freq: 7.665940227568912e-05
        A relação Filhos tem de freq: 9.856208864017172e-05
        A relação Tios Paternos tem de freq: 4.380537272896521e-05
        A relação Tio Bisavo Paterno tem de freq: 4.380537272896521e-05
        A relação Sobrinhos Netos Paternos tem de freq: 2.1902686364482605e-05
        A relação Parente tem de freq: 3.285402954672391e-05
        A relação Irmaos Maternos tem de freq: 2.1902686364482605e-05
        A relação Avo Paterno tem de freq: 5.475671591120651e-05
        A relação Sobrinho Bisneto Materno tem de freq: 2.1902686364482605e-05
        A relação Primos Maternos tem de freq: 1.0951343182241303e-05
        A relação Tio Avo tem de freq: 2.1902686364482605e-05
        A relação Tio tem de freq: 2.1902686364482605e-05
        A relação Sobrinhos Netos Maternos tem de freq: 1.0951343182241303e-05
```

Figura 1.17: Resulto de freq. de Relações

1.4 Main

Por fim, achamos necessário abrir o ficheiro "processos.txt"e criar um menu para escolhermos qual das frequências imprimir na consola. Após o processamento de tudo, é executada a função que irá gerar o JSON com os primeiro 20 processos.

```
#Função para abrir o processos
with open('processos.txt') as file:
    #Abrir o file descriptor
    fd = file.read()
    #Colocar os processos na estrutura de dados
    processos_filtrados = filtrar_info(fd)

    #Criar um Menu e escolher a opção
    x = int(input("Digite qual exercicio:\n1-> Frequencia de
    if (x == 1):
        calcular_freq_processos(processos_filtrados)
    elif(x == 2):
        calcular_freq_nomes_sec(processos_filtrados)
    elif(x == 3):
        calcular_freq_relacao(processos_filtrados)

    #Criar o ficheiro JSON
    with open('processos.json', 'w') as file_output:
        dados_json= criar_json(processos_filtrados,20)
        file_output.write(str(dados_json))
```

Figura 1.18: Código da main

Na imagem abaixo encontra-se o menu do nosso código.

```
Digite qual exercicio:
1-> Frequencia de Processos por Ano
2-> Frequencia de Nomes Proprios e Apelidos por Seculo
3-> Frequencia dos Tipos de Relação
Opcao:
```

Figura 1.19: Menu do Problema

Capítulo 2

Processador de Registos de Exames Médicos Desportivos

2.1 Análise do Problema 2

Neste problema recebemos um "dataset"gerado no âmbito do registo de exames médicos desportivos. O objetivo deste problema era obter:

- Datas extremas dos registos no dataset;
- Distribuição por modalidade em cada ano e no total;
- Distribuição por idade e género (para a idade, considera apenas 2 escalões: ¡ 35 anos e ¿= 35);
- Distribuição por morada;
- Percentagem de aptos e não aptos por ano.

E num segundo parâmetro, era requerido a listagem de todos os dados usados para cada uma dos indicadores gerados.
Na seguinte figura podemos ver um excerto desse ficheiro:

```
_id,index,dataEMD,nome/primeiro,nome/último,idade,género,morada,modalidade,clube,email,federado,resultado
6045074cd77860ac9483d34e,0,2020-02-25,Delgado,Gay,28,F,Gloucester,BTT,ACRroriz,delgado.gay@acrroriz.biz,true,true
6045074ca6adebd591b5d239,1,2019-07-31,Foreman,Prince,34,M,Forestburg,Ciclismo,ACDRcrespos,foreman.prince@acdrcrespos.org,false,true
6045074c221e2fdf430e9ef0,2,2021-01-06,Cheryl,Berger,21,M,Umapine,Basquetebol,Vitoria,cheryl.berger@vitoria.biz,false,true
6045074c529cbdce549d3923,3,2020-11-19,Graves,Goff,29,F,Babb,Andebol,AVCfamalicão,graves.goff@avcfamalicão.co.uk,false,false
6045074c3319a0f9e79aad87,4,2019-09-01,Mckay,Bolton,29,F,Chilton,Futebol,ACDRcrespos,mckay.bolton@acdrcrespos.me,false,false
6045074c222607e7520ffd24,5,2019-10-07,Marla,Kelley,22,M,Clarence,Atletismo,AmigosMontanha,marla.kelley@amigosmontanha.tv,false,false
```

Figura 2.1: Excerto do ficheiro emd.csv

2.2 Resolução do Problema 2

Para a resolução deste problema recorremos às seguintes soluções sequenciadas:

i) Criar um dicionário e alocar cada processo com as respetivas informações;

ii) Calcular cada indicador individualmente;

iii) Desenhar a tabela de cada indicador individualmente em html

iv) Gerar a lista de dados usados para cada indicador individualmente em html

v) Gerar todos os ficheiros HTML com todos os dados devidamente processados

2.2.1 Alocar no Dicionário

Esta parte do problema foi resolvido com a função chamada filtrar_csv
Esta função inicialmente percorre o "file descriptor"do "processos.txt"que foi entregue por parâmetros e dá split ao código por linhas. Após isso é gerado uma lista que será percorrida novamente e levará outro split por cada vez que a expressão ","aparecer.
Após isso, é inicializada um dicionário que contém todas as informações necessárias para a realização deste problema.

```
# Função Inicial para filtrar toda a informação para um dicionário
def filtrar_csv(fd):
    lista_newlines = fd.split("\n")
    data = []
    counter = 0
    # Seperar cada Linha em uma array com as informações
    for line in lista_newlines:
        # Dicionario para colocar a info do CSV
        dict_processo = {
            "id": "",
            "index": "",
            "data": "",
            "nome": "",
            "idade": "",
            "genero": "",
            "morada": "",
            "modalidade": "",
            "club": "",
            "email": "",
            "federado": "",
            "resultado": ""
        }
```

Figura 2.2: Excerto da função filtrar_csv(fd) PARTE 1

Como visto na figura anterior, cada informação do nosso problema, agora numa lista criada pelo split, e chamada "row", será processada pelas Expressões Regulares (ER):

- ER Id: `"[0-9a-z]24"`;

- ER Index: `"[0-9]1,2"`;
- ER Data: `"[0-9]4-[0-9]1,2-[0-9]1,2"`;
- ER Nome—Morada—Modalidade—Club: `"[A-Za-z]+"`;
- ER Idade: `"[0-9]2"`;
- ER Género: `"[MF]1"`;
- ER Email: `"[a-zA-Z.]+@[a-zA-Z.]+"`;
- ER Numero da Pasta: `"[0-9]1,4"`;
- ER Federado—Resultado: `"[true|false]"`;

Cada Coluna da linha (membro da lista) é processado diretamente pela respetiva coluna (pois o dataset é perfeitamente ordenado). Fazemos a verificação com as ERs para evitar dados anormais.

```
if(line and counter > 0):
    # Colocar a Informação do dicionario apos uma pequena verificação
    row = line.split(",")
    if (re.match("[0-9a-z]{24}", row[0])):
        dict_processo["id"] = row[0]
    if (re.match("[0-9]{1,2}", row[1])):
        dict_processo["index"] = row[1]
    if (re.match("[0-9]{4}-[0-9]{1,2}-[0-9]{1,2}", row[2])):
        dict_processo["data"] = row[2]
    if (re.match("[A-Za-z]+", row[3])):
        dict_processo["nome"] = row[3]
    if (re.match("[A-Za-z]+", row[4])):
        dict_processo["nome"] += " "+row[4]
    if (re.match("[0-9]{2}", row[5])):
        dict_processo["idade"] = row[5]
    if (re.match("[MF]{1}", row[6])):
        dict_processo["genero"] = row[6]
    if (re.match("[A-Za-z]+", row[7])):
        dict_processo["morada"] = row[7]
    if (re.match("[A-Za-z]+", row[8])):
        dict_processo["modalidade"] = row[8]
    if (re.match("[A-Za-z]+", row[9])):
        dict_processo["club"] = row[9]
    if (re.match("[a-zA-Z.]+@[a-zA-Z.]+", row[10])):
        dict_processo["email"] = row[10]
    if (re.match("true|false", row[11])):
        dict_processo["federado"] = row[11]
    if (re.match("true|false", row[12])):
        dict_processo["resultado"] = row[12]
if (dict_processo["id"] != ""):
    data.append(dict_processo)
counter += 1
```

Figura 2.3: Excerto da função filtrar_csv(fd) PARTE 2

2.2.2 Cálculo dos Indicadores

Esta parte do problema foi resolvido com 5 funções distintas: datas_extremas, distribuicao_modalidade_ano, distribuicao_modalidade_ano,distribuicao_morada,aptos.

datas_extremas(data)

Nesta função era necessário simplesmente pegar no ano de cada processo e colocar-los numa lista para ordenar-la por ordem crescente (sort).
Após isso, pegamos no primeiro membro dessa lista (menor data) e o último membro dessa lista (maior data).
No final damos return a um tuplo com as datas extremas.

```python
# Função que devolve as datas mais extremas do nosso CSV
def datas_extremas(data):
    # Obter todas as datas numa lista e ordenar por ano -> mes -> dia
    aux_list = []
    for d in data:
        n = d["data"].split("-")
        aux_list.append(n)
        aux_list = sorted(aux_list, key=lambda i: (i[0], i[1], i[2]))

    # Passar os extremos para strings
    menor_data = aux_list.pop(0)
    menor_data_str = ""
    for i in menor_data:
        menor_data_str += i+"-"

    maior_data = aux_list.pop(-1)
    maior_data_str = ""
    for i in maior_data:
        maior_data_str += i+"-"

    return (menor_data_str[:-1], maior_data_str[:-1])
```

Figura 2.4: Função datas_extremas(data)

distribuicao_modalidade _ano(data)

Nesta função era necessário pegar no ano de cada processo e criar um dicionário com as chaves são os anos e os valores são as modalidades que estas últimas, possuem contadores.

```
# Função que devolve um dicionário com cada modalidade por ano
def distribuicao_modalidade_ano(data):
    dict_distribuicao = {}
    for d in data:
        #obter o ano da data
        year = re.match("[0-9]{4}", d["data"]).group()
        #Obter a distribuição por modalidade
        if year not in dict_distribuicao:
            dict_distribuicao[year] = {}
        if d["modalidade"] not in dict_distribuicao[year]:
            dict_distribuicao[year][d["modalidade"]] = 0
        dict_distribuicao[year][d["modalidade"]] += 1

    return dict_distribuicao
```

Figura 2.5: Função distribuicao_modalidade _ano(data)

distribuicao_modalidade _ano(data)

Nesta função era necessário pegar nas idades de cada processo e criar um dicionário com as chaves são é a idade e os valores são o sexo e estas últimas, possuem contadores.
Numa segunda parte, ao mesmo tempo que é filtrado a idade, também adicionamos os escalões.

```
# Função que devolve um dicionário com idades, generos e os escalões
def distribuicao_genero_idade(data):
    dict_distribuicao = {}
    for d in data:
        #obter a distribuição por genero e idade
        if d["idade"] not in dict_distribuicao:
            dict_distribuicao[d["idade"]] = {}
            dict_distribuicao[d["idade"]]["M"] = 0
            dict_distribuicao[d["idade"]]["F"] = 0
        dict_distribuicao[d["idade"]][d["genero"]] += 1
        # Obter os escaloes
        if (int(d["idade"]) < 35):
            dict_distribuicao[d["idade"]]["Escaloes"] = 1
        else:
            dict_distribuicao[d["idade"]]["Escaloes"] = 2
    return dict_distribuicao
```

Figura 2.6: Função distribuicao_genero _idade(data)

distribuicao_morada(data)

Nesta função era necessário pegar nas moradas de cada processo e criar um dicionário com as chaves são é a morada e os valores os contadores.

```
# Função que devolve um dicionário com as moradas
def distribuicao_morada(data):
    dict_distribuicao = {}
    for d in data:
        #Obter a distribuição por modalidade
        if d["morada"] not in dict_distribuicao:
            dict_distribuicao[d["morada"]] = 0
        dict_distribuicao[d["morada"]] += 1
    return dict_distribuicao
```

Figura 2.7: Função distribuicao_morada(data)

aptos(data)

Nesta função, numa primeira parte era necessário pegar no ano de cada processo e criar um dicionário com as chaves são é o resultado e os valores "true"ou "false"e estas últimas, possuem contadores.
Numa segunda parte, é criado a percentagem com este dicionário.

```
# Função para calcular os atletas que estão aptos ou não
def aptos(data):
    dict_distribuicao = {}
    for d in data:
        #Obter o ano da Data
        year = re.match("[0-9]{4}", d["data"]).group()
        #Obter a distribuição por modalidade
        if year not in dict_distribuicao:
            dict_distribuicao[year] = {}
        if d["resultado"] not in dict_distribuicao[year]:
            dict_distribuicao[year][d["resultado"]] = 0
        dict_distribuicao[year][d["resultado"]] += 1

    # Calcular o Total para a percentagem:
    for year in dict_distribuicao:
        counter = 0
        for (key, value) in dict_distribuicao[year].items():
            counter += value
        if("true" in dict_distribuicao[year] and "false" in dict_distribuicao[year]):
            dict_distribuicao[year]["true"] = str(
                dict_distribuicao[year]["true"]/counter)+"  %"
            dict_distribuicao[year]["false"] = str(
                dict_distribuicao[year]["false"]/counter)+" %"
    return dict_distribuicao
```

Figura 2.8: Função aptos(data)

2.2.3 Tabela dos indicadores

Esta parte do problema foi resolvido com 4 funções distintas: draw_table _hmtl, draw_table_html_morada, datas_extremas_html, indicadores

draw_table_html(data)

Esta função é a responsável por desenhar as tabelas dos indicadores mais gerais (género e idade, modalidade, aptos). Esta função recebe alguns parâmetros para poder escrever o HTML (estes estão descritos no comentário da função).
Resumidamente, é criado uma tabela simples com os dados do dicionário e as tags do html são alocadas em strings criando assim um string gigante que será o suficiente para gerar o nosso html.
Essa string é depois substituída na string principal do html onde fica a tag ≪body¿".

```
# Função para escrever tabelas recebendo:
# data -> Dicionario com os parametros
# str1 -> string que escreverá o html
# nome_tabela -> String que dá o nome ao "h1" do html
# p_header -> String que dá o titulo à primeira coluna das tabelas
# anchor -> String com o path para o html
def draw_table_html(data, str1, nome_tabela, p_header, anchor):
    used_key = []
    str_header = '<body><h1><a href='+anchor+">"+nome_tabela + \
        '</a></h1><table><tr><th style="text-align: center">'+p_header+'</th>'
    str_table = ""
    for (year, var) in sorted(data.items()):
        str_table += '<tr><td style="text-align: center">'+str(year)+"</td>"
        for (key, value) in var.items():
            if key not in used_key:
                used_key.append(key)
                str_header += '<th style="text-align: center">' + \
                    str(key)+"</th>"
            str_table += '<td style="text-align: center">'+str(value)+"</td>"
        str_table += "</tr>"
    str_header += "</tr>"
    str_table += "</table>"
    str_final = re.sub("<body>", str_header+str_table, str1)

    return str_final
```

Figura 2.9: Função draw_table _hmtl(data, str1, nome_tabela, p_header, anchor)

draw_table_html_morada(data)

Esta função é basicamente igual à anterior, mas como o dicionário gerado pela função relativa à distribuição da morada tinha uma estrutura ligeiramente diferente, esta função teve de ser adaptada.

```
# Função "igual" à draw_table_html
# Esta teve de ser diferente pois é um dicionario bem mais simples
def draw_table_html_morada(data, str1, nome_tabela):
    str_header = '<body><h1><a href="morada.html">'+nome_tabela + \
        '</a></h1><table><tr><th style="text-align: center">Moradas</th><th style="text-align: center">Numero de Habitantes</th>'
    str_table = ''
    for (morada, value) in sorted(data.items()):
        str_table += '<tr><td style="text-align: center">'+str(morada)+'</td>'
        str_table += '<td style="text-align: center">'+str(value)+"</td></tr>"
    str_header += "</tr>"
    str_table += "</table>"
    str_final = re.sub("<body>", str_header+str_table, str1)
    return str_final
```

Figura 2.10: Função draw_table _html(data, str1, nome_tabela)

datas_extremas_html_morada(data)

Esta função simplesmente cria uma pequena tabela com as 2 datas do tuplo.

```
# Função para escrever os extremos no html
def data_extremas_html(datas, str1):
    str_header = '<body><h1><a href="extremos.html">Datas Extremas</a></h1><table><tr><th style="text-align: center">Datas</th>'
    str_table = '<tr><td style="text-align: center">'+str(datas[0])+'</td>'
    str_table += '<tr><td style="text-align: center">'+str(datas[1])+'</td>'
    str_header += "</tr>"
    str_table += "</table>"
    str_final = re.sub("<body>", str_header+str_table, str1)
    return str_final
```

Figura 2.11: Função datas_extremas _html(data, str1)

indicadores(data,str1)

Esta função recebe a string principal para gerada o html e data.
Ela irá executar cada uma das de calcular as distribuições, percentagens... e irá adicionar à string principal cada uma das tabelas que irão ser obtidas também pela execução das respetivas funções de criação delas.
No fim, retornará a string principal para gerar o ficheiro html.

```
# Função com os indicadores estatisticos
def indicadores(data, str1):
    datas_extremas_result = datas_extremas(data)
    distribuicao_modalidade_ano_result = distribuicao_modalidade_ano(data)
    distribuicao_genero_idade_result = distribuicao_genero_idade(data)
    distribuicao_morada_result = distribuicao_morada(data)
    aptos_result = aptos(data)

    # Desenhar as Tabelas no index.html
    str1 = draw_table_html_morada(
        distribuicao_morada_result, str1, "Distribuicao de Moradas")
    str1 = draw_table_html(
        aptos_result, str1, "Percentagem de Aptos", "Anos", "aptos.html")
    str1 = draw_table_html(distribuicao_genero_idade_result,
                           str1, "Distribuicao de Genero e Idades", "Idades", "idade_genero.html")
    str1 = draw_table_html(distribuicao_modalidade_ano_result,
                           str1, "Distribuicao de Modalidades por Ano", "Anos", "modalidade.html")
    str1 = data_extremas_html(datas_extremas_result, str1)
    return str1
```

Figura 2.12: Função indicadores(data, str1)

2.2.4 Lista dos Indicadores

Esta parte do problema foi resolvido com 5 funções distintas: lista_final _modalidade, lista_final_genero_idade, lista_final_extremos, lista_final_morada, lista_final_aptos

lista_final_modalidade(data,str1)

Esta função é a responsável por obter a lista ordenada das modalidades por ano. Nesta função é criada a lista em html sendo que os dados são ordenados pelo ano da modalidade e depois cada modalidade em si, obtendo assim todos os dados de cada processo ordenadamente. Foi seguido a mesma lógica de "draw _table _html". Uma única string para escrever todo o ficheiro html.

```
# Função para escrever a lista da modalidade
# data -> Dicionario com os parametros
# str1 -> string que escreverá o html
def lista_final_modalidade(data, str1):
    str_header = "<body>"
    str_list = ""
    dist_mod = distribuicao_modalidade_ano(data)
    for (ano, vars) in sorted(dist_mod.items()):
        used_desp = []
        str_list += "<h1>"+ano+"</h1>"
        for (desp, counter) in vars.items():
            for d in data:
                y = re.match("[0-9]{4}", d["data"]).group()
                if d["modalidade"] == desp and y == ano:
                    if desp not in used_desp:
                        str_list += "<h2>"+desp+"</h2>"
                        used_desp.append(desp)
                    for (key, value) in d.items():
                        if(key != "id" and key != "index" and key != "resultado"):
                            str_list += "<p>"+key+" -> "+value+"</p>"
                    str_list += "<p>.</p>"
    str_final = str_header+str_list
    str_final = re.sub("<body>", str_final, str1)
    return str_final
```

Figura 2.13: Função lista_final _modalidade(data, str1)

As outras função são exatamente iguais a esta, só com o diferencial da ordenação dos dados.

2.2.5 Gerar HTMLs/Main

Numa primeira parte da main, criamos o "file descriptor"dos ficheiro csv. Numa segunda parte, abrimos um ficheiro "index.html"onde será executado praticamente todo o código e criada a string para escrever o ficheiro html principal. Após isto tudo, é criado todos as listas em html e em ficheiro separados seguindo a mesma lógica da string única para escrever tudo.

```
# Filtrar o csv para um dicionário
processos_filtrados = []
with open("emd.csv") as file:
    fd = file.read()
    processos_filtrados = filtrar_csv(fd)

# Geramos Sempre todos os html do nada
# Esta parte do código é responsavel pela criação de todos os html e a sua informação
with open("index.html", "w") as file:
    string_index = indicadores(processos_filtrados, string_iniciar_html)
    file.write(string_index)

    # Abrir o html dos indicadores
    with open("extremos.html", "w") as file2:
        str_aux = lista_final_extremos(
            processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
    with open("modalidade.html", "w") as file2:
        str_aux = lista_final_modalidade(
            processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
    with open("idade_genero.html", "w") as file2:
        str_aux = lista_final_genero_idade(
            processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
    with open("morada.html", "w") as file2:
        str_aux = lista_final_morada(processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
    with open("aptos.html", "w") as file2:
        str_aux = lista_final_aptos(processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
```

Figura 2.14: Função gerar_html (Main)

2.3 Resultados do Problema 2

2.3.1 Indicadores

Aqui temos o ficheiro "index.html".

Datas Extremas

Datas
2019-01-12
2021-03-02

Distribuicao de Modalidades por Ano

Anos	Ciclismo	Futebol	Atletismo	Patinagem	Triatlo	Dança	Andebol	Karaté	Equitação	Basquetebol	Badminton	BTT	Esgrima	Parapente	Orientação
2019	11	13	9	9	9	11	11	9	7	9	12	12	5	10	8
2020	16	9	5	8	12	7	12	14	10	11	9	6	10	4	5
2021	3	1	2	1	1	2	1	1	1	1	1	1	1		

Distribuicao de Genero e Idades

Idades	M	F	Escaloes
21	8	9	1
22	10	14	1
23	9	11	1
24	13	6	1
25	6	6	1
26	11	10	1
27	11	13	1
28	13	11	1
29	8	13	1
30	3	14	1
31	11	13	1
32	12	12	1
33	10	10	1

Figura 2.15: Função index.html

2.3.2 Exemplo de Listas

Aqui temos um exemplo de uma lista ao clicarmos no título do indicador em questão.

2019

Ciclismo

data -> 2019-07-31

nome -> Foreman Prince

idade -> 34

genero -> M

morada -> Forestburg

modalidade -> Ciclismo

club -> ACDRcrespos

email -> foreman.prince@acdrcrespos.org

federado -> false

.

data -> 2019-03-12

nome -> Deirdre Higgins

idade -> 31

genero -> M

morada -> Clayville

modalidade -> Ciclismo

club -> SCBraga

Figura 2.16: Função modalidade.html

Capítulo 3

Conclusão

A realização deste trabalho prático possibilitou a consolidação da matéria lecionada sobre Expressões Regulares e processamento de texto, e também desenvolver mais conhecimentos sobre Python.
Em especifico este projeto permitiu-nos aprimorar a escrita de ER para resolução de problemas e filtragem de texto, bem como a a utilização do módulo 're' e as suas funções.

Apêndice A

Código do Programa

A.0.1 Código do Exercício 1

```
import re

#Função para Filtrar Toda a informação e colocar-la num dicionário
def filtrar_info(fd):
    data_processos = []

    # Seperar por Linhas
    lista_newlines = fd.split("\n")

    # Seperar cada Linha em uma array com as informações
    for line in lista_newlines:
        if len(line):
            processos = line.split("::")

        # Dicionario default para colocar as informações
            dict_processo = {
                "Pasta": "",
                "Data": "",
                "Nome": "????",
                "Pai": "????",
                "Mae": "????",
                "Observacao": ""
            }


            # Filtrar o Processo:
            for parametro in processos:
                # Recolher a data
                if (re.match("[0-9]{4}-[0-9]{1,2}-[0-9]{1,2}", parametro)):
                    dict_processo["Data"] = parametro
                    continue
                # Recolher nr da Pasta
                if (re.match("[0-9]{1,4}", parametro)):
```

```
                    dict_processo["Pasta"] = parametro
                    continue
                # Recolher Informações que possuem Nomes Proprios (Começam por Letra maiscula)
                if (re.match("([A-Z]{1}[a-z]* ?)", parametro)):

                    if (re.search("[^A-Z]{1}\.", parametro)):
                        dict_processo["Observacao"] = parametro
                        continue
                    # Alocção ordenada de parametros ordenados:
                    if (dict_processo["Nome"] == "????"):
                        dict_processo["Nome"] = parametro
                        continue
                    if (dict_processo["Pai"] == "????"):
                        dict_processo["Pai"] = parametro
                        continue
                    if (dict_processo["Mae"] == "????"):
                        dict_processo["Mae"] = parametro
                        continue
                # Informação Inexistente
                if (re.match("", parametro)):
                    for (p1, p2) in dict_processo.items():
                        if dict_processo[p1] == "" or dict_processo[p1] == "????":
                            dict_processo[p1] = "!!NO INFO!!"
                            break

            data_processos.append(dict_processo)
    return data_processos


def calcular_freq_processos(data):
    dict_anos = {}
    total=0
    #Obter todos os processos por ano
    for processo in data:
        year = re.match("[0-9]{4}", processo["Data"]).group()
        if year not in dict_anos:
            dict_anos[year]=0
        total+=1
        dict_anos[year]+=1

    #Tabelar
    for (nome,counter) in sorted(dict_anos.items()):
        n = counter / total
        print("O ano tem "+str(nome) + " tem freq. de processos de:" + str(n))

    pass
```

```
# Falta descobrir o que está no "obs"
def calcular_freq_nomes_sec(data):
    #Variaveis Iniciais
    seculos = {}
    lista = ["Nome", "Pai", "Mae"]
    total_nomes_sec = {}
    total_apelidos_sec = {}

    #Ciclar toda a data e recolher toda a informação necessário para o exercício
    for processo in data:
        # Recolher o Ano e Fazer o Dicionario com o Seculo
        year = re.match("[0-9]{4}", processo["Data"]).group()
        year = int((int(year)-1) / 100)+1
        if (year not in seculos):
            seculos[year] = {"Primeiro_Nome": {}, "Apelido_Nome": {}}
            total_apelidos_sec[year] = 0
            total_nomes_sec[year] = 0

        # Recolher o Primeiro e o Ultimo Nome
        primeiro_nomes_lista = []
        apelidos_lista = []

        # Recolher Primerio e Ultimo Nome de Pai........
        for l in lista:
            if (processo[l] != "!!NO INFO!!"):
                primeiro_nome = re.search("^[A-Za-z]+", processo[l]).group()
                primeiro_nomes_lista.append(primeiro_nome)
                # Este if serve para os casos por exemplo "Ana Lopes Coelho (ou Ana Fernandes Lo
                if (re.search("\)", processo[l])):
                    # Voltar a mandar o primeiro nome pra lista porque o Apelido é a unica coisa
                    primeiro_nomes_lista.append(primeiro_nome)

                    # O apelido que está dentro de parentises
                    apelido_nome = re.search(
                        "[A-Za-z]*\)", processo[l]).group()[:-1]
                    apelidos_lista.append(apelido_nome)
                    # O apelido que está fora de parentises
                    apelido_nome = re.search(
                        "[A-Za-z]* \(", processo[l]).group()[:-2]
                    apelidos_lista.append(apelido_nome)
                else:
                    aux=processo[l]
                    #Em casos como: Maria Vale, Solteira
                    if (re.search(",", processo[l])):
                        aux=processo[l].split(",")[0]
                    apelido_nome = re.search("[A-Za-z]+$", aux).group()
                    apelidos_lista.append(apelido_nome)
```

```
        # Recolher Primerio e Ultimo Nome das observaçoes.
        if (processo["Observacao"] != "!!NO INFO!!"):
            obs_lista = re.findall(
                "(([A-Z]{1}[a-z]{1,} ?)+\,)", processo["Observacao"])
            #(nome,junk) pois a função findall está a criar um tuplo e nós queremos o primeiro me
            for (nome, junk) in obs_lista:
                primeiro_nome = re.search("^[A-Za-z]+", nome[:-1]).group()
                primeiro_nomes_lista.append(primeiro_nome)
                apelido_nome = re.search("[A-Za-z]+$", nome[:-1]).group()
                apelidos_lista.append(apelido_nome)

        # Somar os Apelidos e Nomes de cada Seculo
        for n in primeiro_nomes_lista:
            if n not in seculos[year]["Primeiro_Nome"]:
                seculos[year]["Primeiro_Nome"][n] = 0
            total_nomes_sec[year] += 1
            seculos[year]["Primeiro_Nome"][n] += 1
        for n in apelidos_lista:
            if n not in seculos[year]["Apelido_Nome"]:
                seculos[year]["Apelido_Nome"][n] = 0
            total_apelidos_sec[year] += 1
            seculos[year]["Apelido_Nome"][n] += 1

    # Tabelar a Resposta:
    for (sec, data) in sorted(seculos.items()):
        print("Seculo: "+str(sec))
        print("\tNomes:")
        for (nome, counter) in data["Primeiro_Nome"].items():
            final = counter / total_nomes_sec[sec]
            print("\t\t"+str(nome)+" tem uma freq: "+str(final))
        print("\tApelidos:")
        for (nome, counter) in data["Apelido_Nome"].items():
            final = counter / total_nomes_sec[sec]
            print("\t\t"+str(nome)+" tem uma freq: "+str(final))
    return 0


def calcular_freq_relacao(data):
    #Criação de Variaveis
    total_relacoes =0
    dict_relacoes = {
        "Pai": 0,
        "Mae": 0
    }

    #Procurar as relações e somar las
    for processo in data:
        # Ver se o Pai ou Mae existe (podem ser desconhecido)
```

```
        if (processo["Pai"] != "!!NO INFO!!"):
            dict_relacoes["Pai"]+=1
            total_relacoes+=1

        if (processo["Mae"] != "!!NO INFO!!"):
            total_relacoes+=1
            dict_relacoes["Mae"]+=1

        #Relações que estão na observação
        relacoes = re.findall("[^A-Z]\,(([A-Z]{1}[a-z ]+)+)\.",processo["Observacao"])
        if(relacoes):
            relacoes = relacoes[0][0]

            #Tivemos de Adicionar estas 2 Excluisoes, pois eles sairam fora do nosso ER, Somente
            if not(re.match("Jeronimo Silva Coelho", relacoes) or re.match("Frei", relacoes)):
                if (relacoes not in dict_relacoes):
                    dict_relacoes[relacoes] =0
                dict_relacoes[relacoes]+=1
                total_relacoes+=1

    #Criar a Tabela:
    print("Frequencias de Relacoes: ")
    for (rel,counter) in dict_relacoes.items():
        freq = counter/total_relacoes
        print("\tA relação "+str(rel)+" tem de freq: " +str(freq))


    return 0


#Criar o JSON
def criar_json(data,n):
    lista=[]
    counter=0
    for d in data:
        lista.append(d)
        if(counter > n-2):
            break
        counter+=1
    return lista

#Função para abrir o processos
with open('processos.txt') as file:
    #Abrir o file descriptor
    fd = file.read()
    #Colocar os processos na estrutura de dados
    processos_filtrados = filtrar_info(fd)
```

```
    #Criar um Menu e escolher a opção
    x = int(input("Digite qual exercicio:\n1-> Frequencia de Processos por Ano\n2-> Frequencia de
    if (x == 1):
        calcular_freq_processos(processos_filtrados)
    elif(x == 2):
        calcular_freq_nomes_sec(processos_filtrados)
    elif(x == 3):
        calcular_freq_relacao(processos_filtrados)

    #Criar o ficheiro JSON
    with open('processos.json', 'w') as file_output:
        dados_json= criar_json(processos_filtrados,20)
        file_output.write(str(dados_json))
```

A.0.2 Código do Exercício 2

```
import re

string_iniciar_html = '<!DOCTYPE html><html lang="en"><head><meta charset="UTF-8"><meta http-equi


# Função Inicial para filtrar toda a informação para um dicionário
def filtrar_csv(fd):
    lista_newlines = fd.split("\n")
    data = []
    counter = 0
    # Seperar cada Linha em uma array com as informações
    for line in lista_newlines:
        # Dicionario para colocar a info do CSV
        dict_processo = {
            "id": "",
            "index": "",
            "data": "",
            "nome": "",
            "idade": "",
            "genero": "",
            "morada": "",
            "modalidade": "",
            "club": "",
            "email": "",
            "federado": "",
            "resultado": ""
        }
        if(line and counter > 0):
            # Colocar a Informação do dicionario apos uma pequena verificação
            row = line.split(",")
            if (re.match("[0-9a-z]{24}", row[0])):
                dict_processo["id"] = row[0]
```

```
            if (re.match("[0-9]{1,2}", row[1])):
                dict_processo["index"] = row[1]
            if (re.match("[0-9]{4}-[0-9]{1,2}-[0-9]{1,2}", row[2])):
                dict_processo["data"] = row[2]
            if (re.match("[A-Za-z]+", row[3])):
                dict_processo["nome"] = row[3]
            if (re.match("[A-Za-z]+", row[4])):
                dict_processo["nome"] += " "+row[4]
            if (re.match("[0-9]{2}", row[5])):
                dict_processo["idade"] = row[5]
            if (re.match("[MF]{1}", row[6])):
                dict_processo["genero"] = row[6]
            if (re.match("[A-Za-z]+", row[7])):
                dict_processo["morada"] = row[7]
            if (re.match("[A-Za-z]+", row[8])):
                dict_processo["modalidade"] = row[8]
            if (re.match("[A-Za-z]+", row[9])):
                dict_processo["club"] = row[9]
            if (re.match("[a-zA-Z.]+@[a-zA-Z.]+", row[10])):
                dict_processo["email"] = row[10]
            if (re.match("true|false", row[11])):
                dict_processo["federado"] = row[11]
            if (re.match("true|false", row[12])):
                dict_processo["resultado"] = row[12]
        if (dict_processo["id"] != ""):
            data.append(dict_processo)
        counter += 1
    return data


# Função que devolve as datas mais extremas do nosso CSV
def datas_extremas(data):
    # Obter todas as datas numa lista e ordenar por ano -> mes -> dia
    aux_list = []
    for d in data:
        n = d["data"].split("-")
        aux_list.append(n)
        aux_list = sorted(aux_list, key=lambda i: (i[0], i[1], i[2]))

    # Passar os extremos para strings
    menor_data = aux_list.pop(0)
    menor_data_str = ""
    for i in menor_data:
        menor_data_str += i+"-"

    maior_data = aux_list.pop(-1)
    maior_data_str = ""
    for i in maior_data:
```

```
        maior_data_str += i+"-"

    return (menor_data_str[:-1], maior_data_str[:-1])


# Função que devolve um dicionário com cada modalidade por ano
def distribuicao_modalidade_ano(data):
    dict_distribuicao = {}
    for d in data:
        #obter o ano da data
        year = re.match("[0-9]{4}", d["data"]).group()
        #Obter a distribuição por modalidade
        if year not in dict_distribuicao:
            dict_distribuicao[year] = {}
        if d["modalidade"] not in dict_distribuicao[year]:
            dict_distribuicao[year][d["modalidade"]] = 0
        dict_distribuicao[year][d["modalidade"]] += 1

    return dict_distribuicao


# Função que devolve um dicionário com idades, generos e os escalões
def distribuicao_genero_idade(data):
    dict_distribuicao = {}
    for d in data:
        #obter a distribuição por genero e idade
        if d["idade"] not in dict_distribuicao:
            dict_distribuicao[d["idade"]] = {}
            dict_distribuicao[d["idade"]]["M"] = 0
            dict_distribuicao[d["idade"]]["F"] = 0
        dict_distribuicao[d["idade"]][d["genero"]] += 1
        # Obter os escaloes
        if (int(d["idade"]) < 35):
            dict_distribuicao[d["idade"]]["Escaloes"] = 1
        else:
            dict_distribuicao[d["idade"]]["Escaloes"] = 2
    return dict_distribuicao


# Função que devolve um dicionário com as moradas
def distribuicao_morada(data):
    dict_distribuicao = {}
    for d in data:
        #Obter a distribuição por modalidade
        if d["morada"] not in dict_distribuicao:
            dict_distribuicao[d["morada"]] = 0
        dict_distribuicao[d["morada"]] += 1
    return dict_distribuicao
```

```
# Função para calcular os atletas que estão aptos ou não
def aptos(data):
    dict_distribuicao = {}
    for d in data:
        #Obter o ano da Data
        year = re.match("[0-9]{4}", d["data"]).group()
        #Obter a distribuição por modalidade
        if year not in dict_distribuicao:
            dict_distribuicao[year] = {}
        if d["resultado"] not in dict_distribuicao[year]:
            dict_distribuicao[year][d["resultado"]] = 0
        dict_distribuicao[year][d["resultado"]] += 1

    # Calcular o Total para a percentagem:
    for year in dict_distribuicao:
        counter = 0
        for (key, value) in dict_distribuicao[year].items():
            counter += value
        if("true" in dict_distribuicao[year] and "false" in dict_distribuicao[year]):
            dict_distribuicao[year]["true"] = str(
                dict_distribuicao[year]["true"]/counter)+"  %"
            dict_distribuicao[year]["false"] = str(
                dict_distribuicao[year]["false"]/counter)+" %"
    return dict_distribuicao


# Função para escrever tabelas recebendo:
# data -> Dicionario com os parametros
# str1 -> string que escreverá o html
# nome_tabela -> String que dá o nome ao "h1" do html
# p_header -> String que dá o titulo à primeira coluna das tabelas
# anchor -> String com o path para o html
def draw_table_html(data, str1, nome_tabela, p_header, anchor):
    used_key = []
    str_header = '<body><h1><a href='+anchor+">"+nome_tabela + \
        '</a></h1><table><tr><th style="text-align: center">'+p_header+'</th>'
    str_table = ""
    for (year, var) in sorted(data.items()):
        str_table += '<tr><td style="text-align: center">'+str(year)+"</td>"
        for (key, value) in var.items():
            if key not in used_key:
                used_key.append(key)
                str_header += '<th style="text-align: center">' + \
                    str(key)+"</th>"
            str_table += '<td style="text-align: center">'+str(value)+"</td>"
        str_table += "</tr>"
```

```
    str_header += "</tr>"
    str_table += "</table>"
    str_final = re.sub("<body>", str_header+str_table, str1)

    return str_final

# Função "igual" à draw_table_html
# Esta teve de ser diferente pois é um dicionario bem mais simples
def draw_table_html_morada(data, str1, nome_tabela):
    str_header = '<body><h1><a href="morada.html">'+nome_tabela + \
        '</a></h1><table><tr><th style="text-align: center">Moradas</th><th style="text-align: ce
    str_table = ''
    for (morada, value) in sorted(data.items()):
        str_table += '<tr><td style="text-align: center">'+str(morada)+'</td>'
        str_table += '<td style="text-align: center">'+str(value)+"</td></tr>"
    str_header += "</tr>"
    str_table += "</table>"
    str_final = re.sub("<body>", str_header+str_table, str1)
    return str_final


# Função para escrever os extremos no html
def data_extremas_html(datas, str1):
    str_header = '<body><h1><a href="extremos.html">Datas Extremas</a></h1><table><tr><th style="
    str_table = '<tr><td style="text-align: center">'+str(datas[0])+'</td>'
    str_table += '<tr><td style="text-align: center">'+str(datas[1])+'</td>'
    str_header += "</tr>"
    str_table += "</table>"
    str_final = re.sub("<body>", str_header+str_table, str1)
    return str_final


# Função com os indicadores estatisticos
def indicadores(data, str1):
    datas_extremas_result = datas_extremas(data)
    distribuicao_modalidade_ano_result = distribuicao_modalidade_ano(data)
    distribuicao_genero_idade_result = distribuicao_genero_idade(data)
    distribuicao_morada_result = distribuicao_morada(data)
    aptos_result = aptos(data)

    # Desenhar as Tabelas no index.html
    str1 = draw_table_html_morada(
        distribuicao_morada_result, str1, "Distribuicao de Moradas")
    str1 = draw_table_html(
        aptos_result, str1, "Percentagem de Aptos", "Anos", "aptos.html")
    str1 = draw_table_html(distribuicao_genero_idade_result,
                           str1, "Distribuicao de Genero e Idades", "Idades", "idade_genero.html"
    str1 = draw_table_html(distribuicao_modalidade_ano_result,
```

```
                            str1, "Distribuicao de Modalidades por Ano", "Anos", "modalidade.html"
    str1 = data_extremas_html(datas_extremas_result, str1)
    return str1


# Função para escrever a lista da modalidade
# data -> Dicionario com os parametros
# str1 -> string que escreverá o html
def lista_final_modalidade(data, str1):
    str_header = "<body>"
    str_list = ""
    dist_mod = distribuicao_modalidade_ano(data)
    for (ano, vars) in sorted(dist_mod.items()):
        used_desp = []
        str_list += "<h1>"+ano+"</h1>"
        for (desp, counter) in vars.items():
            for d in data:
                y = re.match("[0-9]{4}", d["data"]).group()
                if d["modalidade"] == desp and y == ano:
                    if desp not in used_desp:
                        str_list += "<h2>"+desp+"</h2>"
                        used_desp.append(desp)
                    for (key, value) in d.items():
                        if(key != "id" and key != "index" and key != "resultado"):
                            str_list += "<p>"+key+" -> "+value+"</p>"
                    str_list += "<p>.</p>"
    str_final = str_header+str_list
    str_final = re.sub("<body>", str_final, str1)
    return str_final

# Função para escrever a lista do genero/idade
def lista_final_genero_idade(data, str1):
    used_idade = []
    str_header = "<body>"
    str_list = ""
    dist_genero = distribuicao_genero_idade(data)
    for (idade, vars) in sorted(dist_genero.items()):
        for d in data:
            if d["idade"] == idade:
                if idade not in used_idade:
                    str_list += "<h2>"+idade+" Anos </h2>"
                    used_idade.append(idade)
                for (key, value) in d.items():
                    if(key != "id" and key != "index" and key != "resultado"):
                        str_list += "<p>"+key+" -> "+value+"</p>"
                str_list += "<p>.</p>"
    str_final = str_header+str_list
    str_final = re.sub("<body>", str_final, str1)
```

```
    return str_final

# Função para escrever a lista das datas extremas
def lista_final_extremos(data, str1):
    str_header = "<body>"
    str_list = ""
    extremos = datas_extremas(data)
    for d in data:
        if (d["data"] == extremos[0] or d["data"] == extremos[1]):
            str_list += "<h1>"+ d["data"] + "</h1>"
            for (key, value) in d.items():
                if(key != "id" and key != "index" and key != "resultado"):
                    str_list += "<p>"+key+" -> "+value+"</p>"
            str_list += "<p>.</p>"
    str_final = str_header+str_list
    str_final = re.sub("<body>", str_final, str1)
    return str_final

# Função para escrever a lista dos aptos
def lista_final_aptos(data, str1):
    str_header = "<body>"
    str_list = ""
    for d in data:
        for (key, value) in d.items():
            if(key != "id" and key != "index" and key != "resultado"):
                str_list += "<p>"+key+" -> "+value+"</p>"
        str_list += "<p>.</p>"
    str_final = str_header+str_list
    str_final = re.sub("<body>", str_final, str1)
    return str_final

# Função para escrever a lista das moradas
def lista_final_morada(data, str1):
    used_name = []
    str_header = "<body>"
    str_list = ""
    dist_moradas = distribuicao_morada(data)
    for (morada, counter) in sorted(dist_moradas.items()):
        for d in data:
            if d["morada"] == morada:
                if morada not in used_name:
                    str_list += "<h2>"+morada+"</h2>"
                    used_name.append(morada)
                for (key, value) in d.items():
                    if(key != "id" and key != "index" and key != "resultado"):
                        str_list += "<p>"+key+" -> "+value+"</p>"
                str_list += "<p>.</p>"
    str_final = str_header+str_list
```

```
    str_final = re.sub("<body>", str_final, str1)
    return str_final


# Filtrar o csv para um dicionário
processos_filtrados = []
with open("emd.csv") as file:
    fd = file.read()
    processos_filtrados = filtrar_csv(fd)

# Geramos Sempre todos os html do nada
# Esta parte do código é responsavel pela criação de todos os html e a sua informação
with open("index.html", "w") as file:
    string_index = indicadores(processos_filtrados, string_iniciar_html)
    file.write(string_index)

    # Abrir o html dos indicadores
    with open("extremos.html", "w") as file2:
        str_aux = lista_final_extremos(
            processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
    with open("modalidade.html", "w") as file2:
        str_aux = lista_final_modalidade(
            processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
    with open("idade_genero.html", "w") as file2:
        str_aux = lista_final_genero_idade(
            processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
    with open("morada.html", "w") as file2:
        str_aux = lista_final_morada(processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
    with open("aptos.html", "w") as file2:
        str_aux = lista_final_aptos(processos_filtrados, string_iniciar_html)
        file2.write(str_aux)
```