Processamento de Linguagens (3º ano de Curso)

Trabalho Prático 1

Relatório de Desenvolvimento

Paulo Silva Sousa
(a89465)

João Figueiredo Martins Peixe dos Santos
(a89520)

Luis Filipe Cruz Sobral
(a89474)

11 de maio de 2021

Resumo

Neste trabalho iremos desenvolver Processadores de Linguagens Regulares, atavés da escrita de Expressões Regulares (ER), que permitam processar um ficheiro de texto. O objetivo deste trabalho será, assim, desenvolver Processadores de Linguagens Regulares que permitam responder aos vários desafios colocados.

Índice

Capítulo 1

Introdução

Supervisor: Pedro Rangel Henriques

1.1 Processador de Pessoas listadas nos Róis de Confessados

Área: Processamento de Linguagens

Enquadramento do tema proposto

Contexto do tema que é abordado ao longo do documento

Problema o problema que se quer resolver e o objetivo do projeto relatado

Objetivo do relatório

Resultados ou Contributos – pontos a evidenciar

Estrutura do documento o que é abordado em cada capitulo.

1.2 Estrutura do Relatório

Neste relatório iremos apresentar todos os passos para a realização do projeto, desde a apresentação e descrição do problema, referindo o desenho e concepção da resolução e acabando com a codificação e os testes realizados para verificar a solução. Assim, iremos referir as estruturas de dados que utilizamos para armazenar a informação e as expressões regulares utilizadas para filtrar a informação necessária do texto.

Capítulo 2

Análise e especificação

2.1 Enunciado

Os Róis de Confessados são arquivos do arcebispado existentes no ADB que contém o registo de rapazes que pretendiam seguir a vida clerical e se candidatavam aos seminários, tendo-os sido possível obter uma versão digital desse arquivo. Foi-nos proposto a construção de um ou vários programas em Python que permitam processar o ficheiro de texto 'processos.xml' e permitam verifar ou calcular as seguintes informações:

a) Calcular o número de processos por ano; apresente a listagem por ordem cronológica e indique o intervalo de datas em que há registos bem como o número de séculos analisados.

b) Calcular a frequência de nomes próprios (primeiro nome) e apelidos (último nome) global e mostre os 5 mais frequentes em cada século.

c) Calcular o número de candidatos (nome principal de cada processo) que têm parentes (irmão, tio, ou primo) eclesiásticos; diga qual o tipo de parentesco mais frequente.

d) Verifcar se o mesmo pai ou a mesma mãe têm mais do que um filho candidato.

e) Utilizando a linguagem de desenho de grafos DOT desenhe todas as árvores genealógicas (com base nos triplos <filho, pai, mãe>) dos candidatos referentes a um ano dado pelo utilizador.

2.2 Descrição informal do problema

Observando o ficheiro 'processos.xml' percebemos que é fulcral desenvolver uma expressão regular que permita filtrar e obter a informação necessária para cada um dos desafios propostos. Assim para cada processo indentificamos as seguintes informações:

ID Identificador do processo

Pasta Identificador da pasta

Data Data de criação do processo

Nome Designação associada ao id

Pai Designação do pai

Mãe Designação da mãe

Observacao Informação adicional (opcional)

```
<processo id="23478">
  <pasta>1061</pasta>
  <data>1783-03-10</data>
  <nome>Joao Veloso Rebelo</nome>
  <pai>Tome Veloso Rebelo</pai>
  <mae>Ana Maria Rosario</mae>
  <obs>Andre Costa Ferreira,Tio Materno. Proc.6890. Manuel Veloso Tavares,Primo Paterno. Assistente no Bispado do Rio
    de Janeiro-Brasil.</obs>
</processo>
```

Figura 2.1: Exemplo de Candidatura

2.2.1 Alínea a)

Nesta alinea, é essencial escrever uma ER que permita ler um id de processo e a data associada a este. Temos de ter em conta que poderâo existir processos repetidos e que precisamos de uma estrutura que permita ordenar e armazenar o número de processo por ano. Temos, também, de ter em atenção que será necessário identificar o intervalo de datas em que existem registos disponiveis.

2.2.2 Alínea b)

Nesta alinea, além se ser necessário utilizar uma ER para ler o id do processo para evitar ler processos repetidos, é preciso que esta capture o ano em que o processo foi efetuado e os primeiro e último nomes de cada candidato. Temos de ter em consideração que temos de ter uma estrutura de dados que nos permita guardar o número de vezes que ocorre cada nome próprio e cada último nome em cada século.

2.2.3 Alínea c)

Nesta alinea, será preciso escrever uma ER que permita ler o id de um processo e as observações associadas. Iremos precisar de uma estrutura que permita guardar o número de candidatos para cada grau de parentesco, tendo em conta que poderão existir processos repetidos e que cada candidato poderá ter mais do que 1 parente do mesmo tipo.

2.2.4 Alínea d)

Nesta alinea é necessário criar uma expressão regular que para cada candidato capture o seu id de processo, bem como o nome do pai e da mãe, de modo a compararmos com o nome introduzido pelo utilizador do programa e verificar se este ocorre mais do que uma vez.

2.2.5 Alínea e)

Nesta alinea, teremos de construir uma ER que permita identificar para cada processo, o seu id,a sua data,o nome correspondente,o pai,a mãe e as observações. Precisaremos também de identificar nas observações o nome dos irmãos, caso existam. Temos de ter em conta que existem processos repetidos e que precisaremos de atribuir um id a cada nodo para que seja possivel distingir candidatos com o mesmo nome.

Capítulo 3

Desenho e concepção da resolução

3.1 Decisões que lideraram o desenho da solução e sua implementação

Para todas as alíneas utilizamos o mesmo método para ler o ficheiro, em que percorremos o ficheiro todo e guardamos a informação capturada por uma expressão regular *contentRE* numa lista.

```
with open("processos.xml") as f:
    file = f.read()
    content = re.findall(contentRE, file)
```

Além disso, em todas as alíneas decidimos guardar os ids de processos lidos num *Set*. Posteriormente, ao ler a informação, é verificado se o processo já está no *Set*, de modo a não haver redundância ao ler processos repetidos.

```
for line in content:
    if line[0] not in processes:
        processes.add(line[0])
```

3.1.1 Alínea a)

Nesta alínea, decidimos usar um dicionário para armazenar o número de processo por ano. Temos, assim, em cada entrada deste dicionário uma key que corresponde a um ano, à qual está associada o número de processos resgistados nesse ano.
Desenvolvemos a seguinte ER que nos permite obter as informações relativas à data e id de um processo:

```
r'<processo id="(\d+)">(.|\n)*?<data>((\d{4})-\d{2}-\d{2})</data>'
```

Para facilitar a identificação do intervalo de anos registados criamos um minDate e um maxDate que guarda sempre a maior e menor data lida até ao momento.

```
        if date < minDate:
            minDate = date
        elif date > maxDate:
            maxDate = date
```

Por fim, ordenamos o dicionário (numProcessesYear) para que este possa ser imprimido por ordem cronológica. Para calcular o número de séculos analizados definimos a função yearToCent que recebendo a lista de chaves do dicionário calcula o número de séculos.

```
numProcessesYear = dict(sorted(numProcessesYear.items(), key=lambda p: p[0]))

numCent = len(yearToCent(numProcessesYear.keys()))
```

3.1.2 Alínea b)

Nesta alínea, necessitamos de guardar, além do id do processo, o primeiro e ultimo nome de cada candidato, bem como o ano em que essa candidatura se sucedeu.
Assim, desenvolvemos a seguinte expressão regular:

```
r'<processo id="(\d+)">(.|\n)*?<data>((\d{4})-\d{2}-\d{2})</data>(.|\n)*?
<nome>([a-zA-Z]+ )([a-zA-Z]+ )*([a-zA-Z]+)</nome>'
```

Para armazenar a informação, utilizamos dois dicionários (um para guardar a frequencia do primeiro nome em cada século e outro para guardar a frequencia do utimo nome). Em cada um desses dicionários, a chave é o século em que o processo foi realizado e o valor é outro dicionário. Este segundo dicionário tem como chave o nome (primeiro ou ultimo) e como valor a sua frequencia.

```
seculoPrimeiro = dict()
seculoUltimo = dict()

seculoPrimeiro[seculo] = dict()
seculoUltimo[seculo] = dict()
```

Por último, para apresentar os resultados mais frequentes, decidimos ordenar os dicionários por ordem decrescente de frequência e imprimir os 5 primeiros valores no ecra.

```
seculoPrimeiro = dict(sorted(seculoPrimeiro.items(),
                             key=lambda p: p[0], reverse=True))
seculoUltimo = dict(sorted(seculoUltimo.items(),
                           key=lambda p: p[0], reverse=True))
```

3.1.3 Alínea c)

Para esta alínea, criamos as seguintes ER:

```
r'<processo id="(\d+)">(.|\n)*?(<obs>((.|\n)*?)</obs>|<obs/>)'

r'(([a-zA-Z, ]+),([A-Z][a-zA-Z ]+))\.[ ]?Proc\.\d+'
```

A primeira, para identificar o id e as observações de cada processo. A segunda, para identificar os parentes presente nas observações.
A informação é armazenada num dicionário que permite guardar a frequência de cada grau de parentesco e criamos um int que representa o número de candidatos que têm parentes. Neste dicionário a chave é o grau de parentesco e o valor é a frequência.

```
withFamily = 0
frequencyFamily = dict()
```

Para cada processo que contem informação sobre os seus parentes incrementamos o withFamily e usamos um ciclo for para indentificar o número de parentes associados a cada grau de parentesco e atualizar o dicionário.

```
        if family:
            withFamily += 1
            for relativeGroup in family:
                grau = relativeGroup[2]
                numFamiliares = 1

                if isPlural(grau):
                    relatives = re.split(' e| e |, |,', relativeGroup[1])
                    numFamiliares = len(relatives)
                    grau = toSingular(grau)

                if grau in frequencyFamily:
                    frequencyFamily[grau] += numFamiliares
                else:
                    frequencyFamily[grau] = numFamiliares
```

Por fim, ordenamos o dicionário pela frequência de cada grau e imprimimos para o stdOut, juntamente com o número de candidatos com parentes e o grau mais frequente.

```
ff = dict(sorted(frequencyFamily.items(), key=lambda p: p[1], reverse=True))

for item in ff.items():
    print(str(item[0]) + ": " + str(item[1]))
```

3.1.4 Alínea d)

Para verificar se existe um pai ou uma mãe com mais de 1 filho candidato começamos para ler o nome do pai ou da mão do stdin.

```
name = input("Introduzir nome do pai ou da mãe >> ")
```

Seguidamente, escrevemos a seguinte ER que permite identificar o pai e a mãe associados a um id de processo.

```
r'<processo id="(\d+)">(.|\n)*?<pai>(.+)</pai>(.|\n)*?<mae>(.+)</mae>'
```

Por fim, verificamos se existe mais do que 1 filho candidato associado ao nome lido.

```
if (line[2] == name) or (line[4] == name):
    candidates += 1
```

3.1.5 Alínea e)

Para esta alínea, começamos por ler um ano do stdin.

```
while True:
    data = input("Introduzir ano >> ")
    try:
        data = int(data)
        break
    except ValueError:
        print("Valor Inválido")
```

Posteriormente, escrevemos 2 ER a primeira que permite identificar, caso existam, os irmãos de um candidato. A segunda, para identificar todas a informações relativas a um processo, exceto a pasta.

```
r'([a-zA-Z, ]+),(Irmao|Irmaos)\.[ ]?Proc\.(\d+)'
```

```
r'<processo id="(\d+)">(.|\n)*?<data>((\d{4})-\d{2}-\d{2})</data>(.|\n)*?<nome>(.+)</nome>
(.|\n)*?<pai>(.+)</pai>(.|\n)*?<mae>(.+)</mae>(.|\n)*?(<obs>((.|\n)*?)</obs>|<obs/>)'
```

Seguidamente, comparamos para cada processo o ano em que foi registado com o ano inicialmente lido. Caso a igualdade se verifique criamos um nodo para o filho, pai e mãe aos quais está associado um id (fornecido pela função incrementa()) que permite distingir nodos com o mesmo nome.

```
idfilho = incrementa()
idmae = incrementa()
idpai = incrementa()
dot.node(idfilho,line[5])
dot.node(idpai,line[7])
dot.node(idmae,line[9])
dot.edge(idpai, idfilho)
dot.edge(idmae, idfilho)
```

Por fim, verifamos se existem irmãos associados a este candidato, criamos um nodo para cada irmão e ligamos aos pais.

```
for relative in relatives:
    idirmao = incrementa()
    dot.node(idirmao,relative)
    dot.edge(idpai,idirmao)
    dot.edge(idmae,idirmao)
```

Capítulo 4

Testes

4.1 Testes realizados e Resultados

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

4.1.1 Alínea a

Figura 4.1: Output alínea a

4.1.2 Alínea b

```
PL/PL_Project/Projeto1  main ✗
▶ python3 b.py

--------------------------------
5 Primeiros Nomes mais frequentes
--------------------------------

SÉCULO 20
Nome Antonio    Valor 97
Nome Jose    Valor 81
Nome Manuel    Valor 80
Nome Joao    Valor 33
Nome Joaquim    Valor 16

SÉCULO 19
Nome Jose    Valor 1930
Nome Antonio    Valor 1722
Nome Manuel    Valor 1441
Nome Joao    Valor 1166
Nome Francisco    Valor 775

SÉCULO 18
Nome Manuel    Valor 3593
Nome Joao    Valor 2749
Nome Antonio    Valor 2712
Nome Jose    Valor 2250
Nome Francisco    Valor 1897

SÉCULO 17
Nome Joao    Valor 569
Nome Manuel    Valor 546
Nome Antonio    Valor 490
Nome Francisco    Valor 397
Nome Domingos    Valor 273
```

Figura 4.2: Output alínea b

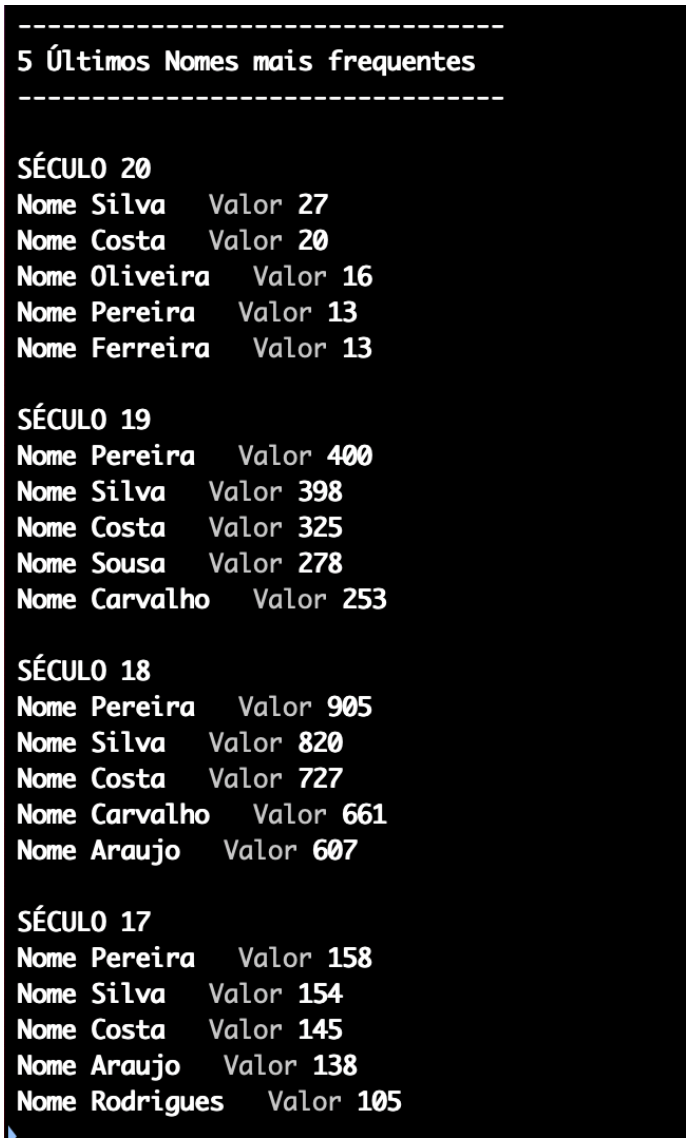

Figura 4.3: Output alínea b

4.1.3 Alínea c

```
▶ python3 c.py
Numero de candidatos com parentes eclesiásticos: 13271

Irmao: 11393
Sobrinho Materno: 1747
Tio Materno: 1684
Sobrinho Paterno: 1618
Tio Paterno: 1605
Primo: 618
Irmao Paterno: 401
Pai: 342
Filho: 342
Primo Materno: 189
Primo Paterno: 147
Tio Avo Materno: 145
Sobrinho Neto Materno: 144
Sobrinho Neto Paterno: 90
Tio Avo Paterno: 90
Irmao Materno: 46
Avo Materno: 35
Neto Materno: 32
Avo Paterno: 10
Neto Paterno: 8
Tio Bisavo Materno: 3
Parente: 3
Sobrinho Bisneto Materno: 3
Sobrinho Bisneto Paterno: 2
Sobrinho Neto: 2
Tio Avo: 2
Tio Bisavo Paterno: 1

Tipo de parentesco mais frequente: Irmao
```

Figura 4.4: Output alínea c

4.1.4 Alínea d

```
PL/PL_Project/Projeto1  main ✗
▶ python3 d.py
Introduzir nome do pai ou da mãe >> Antonio Manuel Almeida

O pai ou mãe tem 1 filhos candidatos
PL/PL_Project/Projeto1  main ✗
▶ python3 d.py
Introduzir nome do pai ou da mãe >> Teresa Maria Sousa

O pai ou mãe tem 7 filhos candidatos
PL/PL_Project/Projeto1  main ✗
▶ python3 d.py
Introduzir nome do pai ou da mãe >> Antonio Alves Barroso

O pai ou mãe tem 1 filhos candidatos
```

Figura 4.5: Output alínea d

4.1.5 Alínea e

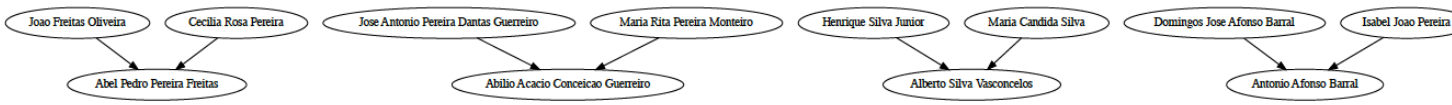

Figura 4.6: Output alínea e, Ano: 1883

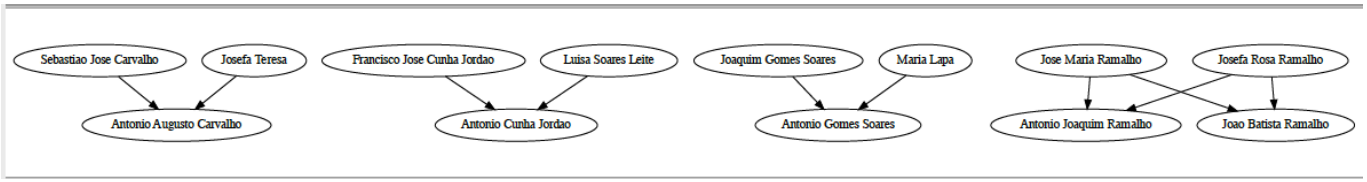

Figura 4.7: Output alínea e, Ano: 1883

Figura 4.8: Output alínea e, Ano: 1885

Figura 4.9: Output alínea e, Ano: 1885

Capítulo 5

Conclusão

O presente trabalho prático permitiu a consolidação da matéria dada nas aulas teóricas e práticas, nomeadamente sobre as expressões regulares. Estudamos diferentes formas de aplicar as expressões regulares. Com este projeto conseguimos superar alguns entraves que se verificaram ao longo da realização deste, o que levou a uma maior consolidação e compreensão de conhecimentos e a um trabalho de equipa maior. Ainda, através da realização deste trabalho, conseguimos utilizar ferramentas com as quais ainda não tínhamos feito contacto(graphviz), o que se revelou um incentivo à nossa expansão de conhecimento e autodidatismo. Em suma, o grupo considera que este trabalho prático foi um bom meio de estudo e uma experiência positiva para a avaliação e consolidação de conhecimentos relativamente à UC de Processamento dde Linguagens.

Apêndice A

Código do Programa

A.1 Alínea a)

Lista-se a seguir o código do problema desenvolvido para a primeira alínea do problema.

```
import re

def yearToCent(yearList):
    centList = [(int(year) // 100) + 1 for year in yearList]
    return list(dict.fromkeys(centList))

def printNumProcessYear(item):
    print(f'No ano ' + '\033[1m' + item[0] + '\033[0m' + ' foram registados ' +
          '\033[1m' + str(item[1]) + '\033[0m' + ' processos.')

def printNumCent(numCent):
    print(f'\nExistem registos de ' +
          '\033[1m' + str(numCent) + ' séculos' + '\033[0m' + ' diferentes.')

def printDateRange(firstDate, lastDate):
    print(f'\nHá registos entre ' + '\033[1m' + firstDate +
          '\033[0m' + ' e ' '\033[1m' + lastDate + '\033[0m' + '.')

minDate = '9999-99-99'
maxDate = '0000-00-00'
numProcessesYear = dict()
processes = set()

contentRE = re.compile(
    r'<processo id="(\d+)">(.|\n)*?<data>((\d{4})-\d{2}-\d{2})</data>')

with open("processos.xml") as f:
    file = f.read()
    content = re.findall(contentRE, file)
```

```
for line in content:
    if line[0] not in processes:
        processes.add(line[0])

        date = line[2]
        year = line[3]

        if date < minDate:
            minDate = date
        elif date > maxDate:
            maxDate = date

        if year in numProcessesYear:
            numProcessesYear[year] += 1
        else:
            numProcessesYear[year] = 1

numProcessesYear = dict(sorted(numProcessesYear.items(), key=lambda p: p[0]))

numCent = len(yearToCent(numProcessesYear.keys()))

for item in numProcessesYear.items():
    printNumProcessYear(item)

printNumCent(numCent)
printDateRange(minDate, maxDate)
```

A.2 Alínea b)

Lista-se a seguir o código do problema desenvolvido para a segunda alínea do problema.

```
import re

def printPrimeiro():
    print('\n---------------------------------')
    print('5 Primeiros Nomes mais frequentes')
    print('---------------------------------')

def printApelidos():
    print('\n---------------------------------')
    print('5 Últimos Nomes mais frequentes')
    print('---------------------------------')

def printSeculo(sec):
```

```
    print('\nSÉCULO ' + '\033[1m' + str(sec))

def printNome(nome, valor):
    print('Nome ' + '\033[1m' + nome +
          '  \033[0m' + ' Valor ' + '\033[1m' + str(valor))

seculo = 0
seculoPrimeiro = dict()
seculoUltimo = dict()
processes = set()

contentRE = re.compile(
    r'<processo id="(\d+)">(.|\n)*?<data>((\d{4})-\d{2}-\d{2})</data>
    (.|\n)*?<nome>([a-zA-Z]+ )([a-zA-Z]+ )*([a-zA-Z]+)</nome>')

with open("processos.xml") as f:
    file = f.read()
    content = re.findall(contentRE, file)

for line in content:
    if line[0] not in processes:
        processes.add(line[0])

        seculo = int(line[3]) // 100 + 1
        primeiro = line[5]
        ultimo = line[7]

        if seculo not in seculoPrimeiro:
            seculoPrimeiro[seculo] = dict()
            seculoUltimo[seculo] = dict()

        if primeiro in seculoPrimeiro[seculo]:
            seculoPrimeiro[seculo][primeiro] += 1
        else:
            seculoPrimeiro[seculo][primeiro] = 1

        if ultimo in seculoUltimo[seculo]:
            seculoUltimo[seculo][ultimo] += 1
        else:
            seculoUltimo[seculo][ultimo] = 1

seculoPrimeiro = dict(sorted(seculoPrimeiro.items(),
                             key=lambda p: p[0], reverse=True))
seculoUltimo = dict(sorted(seculoUltimo.items(),
                           key=lambda p: p[0], reverse=True))

for sec in seculoPrimeiro.keys():
    seculoPrimeiro[sec] = dict(
```

```
        sorted(seculoPrimeiro[sec].items(), key=lambda p: p[1], reverse=True)[:5])

for sec in seculoUltimo.keys():
    seculoUltimo[sec] = dict(
        sorted(seculoUltimo[sec].items(), key=lambda p: p[1], reverse=True)[:5])

printPrimeiro()
for sec in seculoPrimeiro.keys():
    printSeculo(sec)
    for item in seculoPrimeiro[sec].items():
        printNome(item[0], item[1])

printApelidos()
for sec in seculoUltimo.keys():
    printSeculo(sec)
    for item in seculoUltimo[sec].items():
        printNome(item[0], item[1])
```

A.3 Alínea c)

Lista-se a seguir o código do problema desenvolvido para a terceira alínea do problema.

```
import re

def isPlural(member):
    return (member[-1] == 's')

def toSingular(plural):
    singular = re.sub(r'(s )', r' ', plural)
    return re.sub(r's$', r'', singular)

contentRE = re.compile(
    r'<processo id="(\d+)">(.|\n)*?(<obs>((.|\n)*?)</obs>|<obs/>)')
familyRE = re.compile(r'(([a-zA-Z, ]+),([A-Z][a-zA-Z ]+))\.[ ]?Proc\.\d+')

withFamily = 0
frequencyFamily = dict()
processes = set()

with open("processos.xml") as f:
    file = f.read()
    content = re.findall(contentRE, file)

for line in content:
```

```
    if line[0] not in processes:
        processes.add(line[0])

        linha = re.sub(r'( |\n)+', r' ', line[2].strip())
        family = re.findall(familyRE, linha)

        if family:
            withFamily += 1

            for relativeGroup in family:
                grau = relativeGroup[2]
                numFamiliares = 1

                if isPlural(grau):
                    relatives = re.split(' e| e |, |,', relativeGroup[1])
                    numFamiliares = len(relatives)
                    grau = toSingular(grau)

                if grau in frequencyFamily:
                    frequencyFamily[grau] += numFamiliares
                else:
                    frequencyFamily[grau] = numFamiliares

print("Numero de candidatos com parentes eclesiásticos: " + str(withFamily) + "\n")

ff = dict(sorted(frequencyFamily.items(), key=lambda p: p[1], reverse=True))

for item in ff.items():
    print(str(item[0]) + ": " + str(item[1]))

print("\nTipo de parentesco mais frequente: " +
      max(frequencyFamily.items(), key=lambda p: p[1])[0])
```

A.4 Alínea d)

Lista-se a seguir o código do problema desenvolvido para a quarta alínea do problema.

```
import re

name = input("Introduzir nome do pai ou da mãe >> ")

candidates = 0
processes = set()
```

```
contentRE = re.compile(
    r'<processo id="(\d+)">(.|\n)*?<pai>(.+)</pai>(.|\n)*?<mae>(.+)</mae>')

with open("processos.xml") as f:
    file = f.read()
    content = re.findall(contentRE, file)

for line in content:
    if line[0] not in processes:
        processes.add(line[0])

        if (line[2] == name) or (line[4] == name):
            candidates += 1

print(f"\nO pai ou mãe tem {candidates} filhos candidatos")
```

A.5 Alínea e)

Lista-se a seguir o código do problema desenvolvido para a quinta alínea do problema.

```
import re
from graphviz import Digraph

dot = Digraph()
processes = set()
idP = 0

def incrementa():
    global idP
    idP += 1
    return str(idP)

while True:
    data = input("Introduzir ano >> ")
    try:
        data = int(data)
        break
    except ValueError:
        print("Valor Inválido")

familyRE = re.compile(r'([a-zA-Z, ]+),(Irmao|Irmaos)\.[ ]?Proc\.(\d+)')
contentRE = re.compile(
    r'<processo id="(\d+)">(.|\n)*?<data>((\d{4})-\d{2}-\d{2})</data>(.|\n)*?<nome>(.+)</nome>
    (.|\n)*?<pai>(.+)</pai>(.|\n)*?<mae>(.+)</mae>(.|\n)*?(<obs>((.|\n)*?)</obs>|<obs/>)')
```

```
with open("processos.xml") as f:
    file = f.read()
    content = re.findall(contentRE, file)

for line in content:
    if line[0] not in processes:
        processes.add(line[0])
        year = int(line[3])

        if year == data:
            linha = re.sub(r'( |\n)+', r' ', line[12].strip())
            family = re.findall(familyRE, linha)
            idfilho = incrementa()
            idmae = incrementa()
            idpai = incrementa()
            dot.node(idfilho,line[5])
            dot.node(idpai,line[7])
            dot.node(idmae,line[9])
            dot.edge(idpai, idfilho)
            dot.edge(idmae, idfilho)

            for relativeGroup in family:
                grau = relativeGroup[1]
                processes.add(relativeGroup[2])

                relatives = re.split(' e| e |, |,', relativeGroup[0])

                for relative in relatives:
                    idirmao = incrementa()
                    dot.node(idirmao,relative)
                    dot.edge(idpai,idirmao)
                    dot.edge(idmae,idirmao)

dot.render('output/graph.gv', view=True)
```