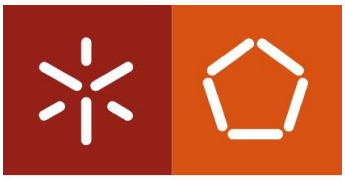

Universidade do Minho

Departamento de Informática

Processamento de Linguagens

TP1 - Exercício 2 - Grupo 71

Leonardo Marreiros (A89537)
Pedro Fernandes (A89574)

5 de abril de 2021

Resumo

O objetivo deste projeto é processar um ficheiro XML para produzir várias *queries* acerca da sua informação, utilizando expressões regulares, de modo a aumentar a capacidade de escrever as mesmas assim como aprender a filtrar informação e usar as funções do módulo 'er' do python (search(), split(), sub(), entre outras).

Conteúdo

Capítulo 1

Introdução

Os Róis de Confessados são arquivos do arcebispado contendo milhares de registos com informações como número do processo, nome, pais, datas e observações. Sendo assim, este projeto tem como objetivo o processamento de um ficheiro *XML* contendo o registo de rapazes que pretendiam seguir a vida clerical e se candidatavam aos seminários de forma a extrair todos os dados relevantes e transformá-los numa resposta simples e organizada.

Para o processamento deste ficheiro foram escritos filtros de texto utilizando expressões regulares e a linguagem de programação Python.

Em primeiro lugar iremos analisar o problema, determinar que funcionalidades implementar e identificar os desafios que serão necessários ultrapassar para implementar essas funcionalidades.

Em seguida iremos analisar a nossa solução, que expressões regulares foram escolhidas para responder aos problemas e as estruturas de dados utilizadas.

Capítulo 2

Problema

O ficheiro **Processos.xml** contém milhares de registos de rapazes que pretendiam seguir a vida clerical e se candidatavam aos seminários. Cada um desses registos contém tags incluindo nome, data, número de processo, observações, etc.

Assim, o projeto consiste em extrair de um ficheiro de registos como o fornecido toda a informação necessária e relevante dependendo do pretendido, podendo assim ser consultada e entendida de forma fácil e organizada por qualquer pessoa.

Uma vez que o ficheiro de processos é muito extenso, é impossível guardar todo o ficheiro em memória pelo que a primeira decisão tomada foi ler o ficheiro linha a linha e construir estruturas de dados como dicionários onde vai sendo adicionada informação relevante à medida que o ficheiro vai sendo lido.

O projeto tem que cumprir os seguintes requisitos:

- Calcular o número de processos por ano; apresentar a listagem por ordem cronológica e indicar o intervalo de datas em que há registos bem como o número de séculos analisados;
- Calcular a frequência de nomes próprios (primeiro nome) e apelidos (último nome) global e mostrar os 5 mais frequentes em cada século;
- Calcular o número de candidatos que têm parentes (irmão, tio, ou primo) eclesiásticos e o tipo de parentesco mais frequente;
- Verificar se o mesmo pai ou a mesma mãe têm mais do que um filho candidato;
- Utilizando a linguagem de desenho de grafos DOT desenhar todas as árvores genealógicas (com base nos triplos $<$ filho, pai, mãe $>$) dos candidatos referentes a um ano dado pelo utilizador.

Como seria de esperar, apesar de definidos, estes requisitos podem levar a diferentes interpretações que contituiu um dos desafios deste trabalho. A forma como foi abordado cada requisito assim como a solução implementada serão tópicos abordados no capítulo seguinte.

Capítulo 3

Solução

Para cada requisito/alínea foi criado um programa Python para processar o ficheiro de texto. De seguida iremos explicar as soluções encontradas para cumprir estes requisitos assim como alguns excertos de código relevantes.

3.1 Alínea A

```
for line in f:
    if res := re.search(r'<data>((.|\n)*)<\/data>',line) :
        ano = re.split(r'-', res.group(1))[0]
        data = res.group(1)

        if int(ano)% 100 != 0 :
            seculos.add(int(ano)//100 + 1)
        else:
            seculos.add(int(ano)//100)

        if (data < ano_menor):
            ano_menor = data

        elif (data > ano_maior):
            ano_maior = data

        elif ano in anos.keys():
            anos[ano] += 1
        else:
            anos[ano] = 1
anos = dict(sorted(anos.items(), key=lambda p: p[0]))
```

Para calcular o número de processos por ano, através do uso de expressões regulares capturamos a data de cada candidatura presente em cada registo que se encontra entre as tags <data> através da ER: `<data>((.|\n)*)<\/data>` e de seguida o seu ano através do resultado do primeiro elemento da função `split` quando procurados por `"-"` no grupo de captura resultante da expressão anterior. Em seguida guardamos o ano num dicionário (anos) e depois, a cada ocorrência, incrementamos o valor inteiro que dentro dele se encontra. Quanto ao intervalo de datas criamos duas variáveis: uma para armazenar o maior ano e outra para o menor. Cada vez que encontramos uma data menor ou maior estas variáveis são alteradas respetivamente. Por fim, para contabilizar os séculos

analisados foi criado um set de século que, para cada ano, adiciona o seu século correspondente. Para saber todos os séculos analisados basta ver todos os elementos existentes nesse set.

Observação: foi tomado como século 20 desde 1901 a 2000 e assim respetivamente.

3.2 Alinea B

```
if res2 := re.search(r'<data>((.|\n)*)<\/data>',line) :

    if int((re.split(r'-', res2.group(1))[0])) % 100 != 0 :
        seculo = int((re.split(r'-', res2.group(1))[0]))//100 +1
    else:
        seculo = int((re.split(r'-', res2.group(1))[0]))//100

if res := re.search(r'<nome>((.|\n)*)<\/nome>',line) :
    nome = re.search(r'^[A-Za-z]+', res.group(1)).group(0)
    apelido = re.search(r'[A-Za-z]+$', res.group(1)).group(0)
```

Para calcular a frequência dos nomes próprios e dos apelidos primeiramente foram criados dois dicionários (nomes e apelidos) que têm como chave (nome, século) e (apelido, século) respetivamente, e cujo valor de cada chave é a frequência da sua chave. Durante a leitura do ficheiro vai-se capturando através de expressões regulares a data e, de seguida, com este resultado é calculado o século de forma semelhante à última alínea. O nome completo do candidato é conseguido com a ER `<nome>((.|\n)*)<\/nome>` que irá capturar tudo o que esteja entre as tags <nome>. Através do grupo de captura resultante desta expressão, é obtido o primeiro nome com `^[A-Za-z]+`, isto é, obtendo a primeira palavra do resultado anterior. Seguindo o mesmo raciocínio, o último nome é conseguido com `[A-Za-z]+$`, capturando a última palavra. De seguida eram adicionados os dados aos dicionários e por fim estes eram ordenados, obtendo no fim uma ordenação por século decrescente e por frequência de nome ou apelido decrescente dependendo do dicionário.

```
nomes = dict(sorted(nomes.items(),
        key=lambda p: (p[0][1], p[1]),reverse=True))
apelidos = dict(sorted(apelidos.items(),
        key=lambda p: (p[0][1], p[1]),reverse=True))
```

Finalmente, estes dicionários são filtrados de forma a apenas conter informação dos cincos nomes e apelidos mais frequêntes em cada século. **Observação:** foi tomado como século 20 desde 1901 a 2000 e assim respetivamente.

3.3 Alinea C

```
if res := re.search(r'<obs>([^<]*)(<\/obs>)?', line):

    if re.search(r'<\/obs>$', res.group(0)):
        contaParentes(res.group(1))

    else:
        obsline = obsline + res.group(1).strip('\n')
elif res := re.search(r'(.*)</obs>$', line):
    obsline = obsline + ' ' + res.group(1).strip()
    contaParentes(obsline)
    obsline = ''
elif res:= re.search(r'^[^<](.*)[^>]$',line.strip()):
    obsline = obsline + ' ' + res.group(0).strip()
```

Primeiramente foi necessário normalizar todos os campos <obs>, isto porque haviam processos em que este campo possuía mais do que uma linha. Para isso quando se lia este campo verificava-se se possuía mais do que uma linha e, caso isso acontecesse, era usada uma string auxiliar onde se iam colocando sucessivamente o conteúdo das linhas, substituindo os `"\n"` por nada com a ajuda da função `strip` e reconstruido assim a string completa referente ao campo das observações. Quanto ao cálculo do número de candidatos que têm parentes eclesiásticos consideramos todos aqueles que apresentam algum irmão, tio ou primo no campo <obs> com um processo associado através da ER `([a-zA-Z ,]+)\,([a-zA-Z ]+)\.(\ ?)([Pp][Rr][Oo][Cc])` que inclui casos em que há espaços opcionais entre palavras.

```
def contaParentes(string):
    if res := re.findall(r'([a-zA-Z ,]+)\,([a-zA-Z ]+)\.(\ ?)([Pp][Rr
        ][Oo][Cc])', string):
        global irmao
        global tio
        global primo
        global candidatos
        candidatos +=1
        for p in res:

            if re.search(r'Irmaos', p[1]):
                irmao += len(re.findall(r'( e |,)', p[0])) + 1
            elif re.search(r'Irmao', p[1]):
                irmao +=1
            elif re.search(r'Tios', p[1]):
                tio += len(re.findall(r'( e |,)', p[0])) + 1
            elif re.search(r'Tio', p[1]):
                tio +=1
            elif re.search(r'Primos', p[1]):
                primo += len(re.findall(r'( e |,)', p[0])) + 1
            elif re.search(r'Primo', p[1]):
                primo +=1
```

Quanto ao parentesco mais frequente, foram corridas todas as candidaturas e observado o seu campo <obs>, por cada ocorrência de "tio", "primo" ou "irmão" era incrementada a variável que armazenava a sua quantidade. Nos casos em que havia "tios", "primos" ou "irmãos" foram contados quantos nomes se encontravam atrás deste tipo de parentesco através da soma de `","` e `" e "`. Para tal foram criadas 4 variáveis para armazenar a informação pretendida (irmão, tio, primo, candidatos). Conforme se ia lendo o ficheiro, através do método anteriormente explicado, eram incrementadas estas variáveis. Por fim era imprimido cada número pretendido assim como o tipo de parentesco mais frequente.

3.4 Alinea D

Nesta alínea assumimos que o mesmo pai e mesma mãe seriam todos os casos em que uma candidatura possuía o mesmo nome de pai e de mãe. Pensamos ainda, através do campo <obs>, ver o número de irmãos com "Proc." e adiciona-los assim como o candidato do respetivo registo, no entanto nada nos garantia que estes não poderiam apenas ser meios irmãos, ou seja, apenas serem filhos da mesma mãe ou do mesmo pai. Assim um casal acabou por ser considerado todas as ocorrência em que o pai e a mãe eram exatamente iguais, o que ainda assim não está completamente correto pois é provável haver mais do que um casal com exatamente os mesmos nomes.

```
for line in f:
```

```
    if resPai := re.search(r'<pai>((.|\n)*)<\/pai>',line) :
        pai = resPai.group(1)
    if resMae := re.search(r'<mae>((.|\n)*)<\/mae>',line) :
        mae = resMae.group(1)

        if (pai,mae) in casal.keys():
            casal[pai,mae] += 1
        else:
            casal[pai,mae] = 1

casal = {key:val for key, val in casal.items() if int(val) != 1}
casal = dict(sorted(casal.items(), key=lambda p: p[1]))
```

Assim, para armazenar a informação pretendida foi criado um dicionário de chave (pai, mãe) para o número de filhos. No fim foram removidas todas as ocorrências de apenas um filho e organizado o dicionário de forma crescente.

3.5 Alinea E

Tal como indicado no enunciado, nesta alínea, criamos todas as árvores genealógicas de pai e mãe para filho num ano indicado pelo utilizador. Começamos por pedir um ano dado pelo utilizador e filtrar o ficheiro baseado nesse ano com auxilio da expressão regular `<data>('+ano+').*<\/data>`, onde `ano` corresponde ao ano dado pelo utilizador. De seguida, capturamos o nome do candidato e o nome dos seus pais para os registos que constavam no ano pretendido e, para cada um destes casos, foi adicionado o triplo (filho,pai,mãe) a uma lista.

```
for line in f:
    if res := re.search(r'<data>('+ano+').*<\/data>', line) :
        year = re.split(r'-', res.group(1))[0]
        cont1 += 1

    if res2 := re.search(r'<nome>(.*)<\/nome>', line) :
        if (cont2 < cont1) :
            filho = res2.group(1)

    if res3 := re.search(r'<pai>(.*)<\/pai>', line) :
        if (cont2 < cont1) :
            pai = res3.group(1)

    if res4 := re.search(r'<mae>(.*)<\/mae>', line) :
        if (cont2 < cont1) :
            mae = res4.group(1)
            familia.append((filho,pai,mae))
            cont2 +=1
```

Com esta lista, criamos uma árvore de pais para filho com o auxilio da ferramenta de desenho de grafos DOT disponível através do graphviz. Nesta alínea apresentamos duas opções, numa é gerado um ficheiro com todas as árvores correspondentes aos anos fornecidos e uma outra que gera todas as árvores em ficheiros diferentes. Estes ficheiros são armazenados numa pasta (árvores) que é gerada caso exista o ano que o utilizador pretende aceder. A cada vez que o ficheiro .py é executado esta pasta é eliminada de modo a evitar problemas quanto à sobreposição de ficheiros.

```
def opcao_sep(familia):
    i = 0
```

```
    for f in familia:
        aux = str(i)
        aux = Digraph(comment='Árvore genealógica')

        aux.node('1', f[0])
        aux.node('2', f[1])
        aux.node('3', f[2])

        aux.edges(['21','31'])


        aux.render('árvores/'+str(i)+'.gv')
        i+=1

def opcao_junta(familia):
    i=0
    dot = Digraph(comment='Árvore genealógica')
    for f in familia:
        aux = str(i)
        str0 = "a"+aux
        str1 = "b"+aux
        str2 = "c"+aux
        dot.node(str0, f[0])
        dot.node(str1, f[1])
        dot.node(str2, f[2])

        dot.edge(str1,str0, constraint='true')
        dot.edge(str2,str0, constraint='true')
        i+=1

    dot.render('árvores/árvores.gv')
```

Na opção separada, cada *Digraph* tem de ter nomes de grafo e ficheiro diferentes, daí o uso da variável `i`, assim, cada ficheiro .gv irá ter um nome diferente começando em 0 e incrementando em uma unidade por cada árvore. Nesta opção utilizamos a função `edges` do *graphviz* para definir as ligações. Neste caso, é ligado o nodo pai ao nodo filho e o nodo mãe ao nodo filho também.

Na opção junta, apenas é criado um ficheiro pelo que apenas é feito um render e não é preciso ter a preocupação dos nomes dos ficheiros criados. No entanto, desta vez, uma vez que irão ficar no mesmo ficheiro, cada nodo terá de ter uma key diferente, pelo que cada nodo tem um nome diferente com o auxilio da variável `i` que é incrementada a cada iteração. Além disto, nesta versão decidimos utilizar a função `edge` ao invés da `edges` utilizada anteriormente. A diferença entre estas é que a segunda cria conexões entre nodos a partir de keys que existam, à partida que o `edge` lida com apenas uma ligação mas cria nodos (e respetivas ligações) no caso de não existir um node que tenha key com o nome indicado. No caso de existirem as keys é apenas criada a ligação entre os dois nodos indicados.

Capítulo 4

Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Durante o trabalho foram surgindo diversos problemas, principalmente devido a diversos erros e inconsistências do ficheiro fornecido, tais como:

- Existiam processos totalmente idênticos, processos com mesmo ID mas que pertenciam a pessoas diferentes;
- Problemas gramaticais, pontuação ou falta de espaços (Exemplo: "eAntonio" na descrição de dois filhos no "id = 6778");
- Algumas pessoas têm informação adicional no seu nome (Exemplo: "<mae>Ana Lopes Coelho (ou Ana Fernandes Lopes)</mae>)" no "id = 32170");
- Nem todos os parentes possuem processos;
- Algumas pessoas podem não ter pai ou mãe (Exemplo : id"30559");

Desta forma consideramos que cada processo é independente, não considera-mos relevante a informação adicional dentro do nome e nos casos de parentes sem processos estes não contam para o total de parentes eclesiástico. No caso da alínea D considera-mos que cada dois pais com o mesmo nome seriam o mesmo casal, em alternativa poderíamos ter em cada processo visto o número de irmãos que este continha, no entanto nada nos garantia que estes irmãos seriam filhos dos dois pais. Para a alínea E tivemos a mesma consideração e assim todas as árvores genealógicas são de pai, mãe para filho.

4.2 Testes realizados e Resultados

De seguida são apresentado os resultados obtidos para cada uma das alíneas do trabalho, note-se que alguns dos outputs eram muitos grandes e acabaram por ter de ser cortados de modo a fornecer uma melhor visualização.

4.2.1 Alínea A[1]

Ano: 1900	Processos: 50
Ano: 1901	Processos: 59
Ano: 1902	Processos: 83
Ano: 1903	Processos: 20
Ano: 1904	Processos: 52
Ano: 1905	Processos: 42
Ano: 1906	Processos: 63
Ano: 1907	Processos: 47
Ano: 1908	Processos: 51
Ano: 1909	Processos: 37
Ano: 1910	Processos: 26
Ano: 1911	Processos: 12

Intervalo de datas:

1616−10−29
até
1911−03−06

Séculos analisados:

XVII ; XVIII ; XIX ; XX ;

4.2.2 Alínea B

Nomes

Século XX:		
Nome: Antonio	Frequência: 91	
Nome: Jose	Frequência: 76	
Nome: Manuel	Frequência: 76	
Nome: Joao	Frequência: 34	
Nome: Joaquim	Frequência: 17	
Século XIX:		
Nome: Jose	Frequência: 2088	
Nome: Antonio	Frequência: 1828	
Nome: Manuel	Frequência: 1498	
Nome: Joao	Frequência: 1252	
Nome: Francisco	Frequência: 847	
Século XVIII:		
Nome: Manuel	Frequência: 3904	
Nome: Joao	Frequência: 3018	
Nome: Antonio	Frequência: 2875	
Nome: Jose	Frequência: 2551	
Nome: Francisco	Frequência: 2098	
Século XVII:		
Nome: Manuel	Frequência: 620	
Nome: Joao	Frequência: 614	
Nome: Antonio	Frequência: 508	
Nome: Francisco	Frequência: 435	
Nome: Domingos	Frequência: 289	

Apelidos

Século XX:		
Nome: Silva	Frequência: 25	
Nome: Costa	Frequência: 19	
Nome: Oliveira	Frequência: 16	
Nome: Pereira	Frequência: 14	
Nome: Ferreira	Frequência: 11	
Século XIX:		
Nome: Pereira	Frequência: 431	
Nome: Silva	Frequência: 426	
Nome: Costa	Frequência: 348	
Nome: Sousa	Frequência: 303	
Nome: Araujo	Frequência: 264	
Século XVIII:		
Nome: Pereira	Frequência: 982	
Nome: Silva	Frequência: 910	
Nome: Costa	Frequência: 792	
Nome: Carvalho	Frequência: 731	
Nome: Araujo	Frequência: 674	
Século XVII:		
Nome: Silva	Frequência: 180	
Nome: Pereira	Frequência: 170	
Nome: Costa	Frequência: 150	
Nome: Araujo	Frequência: 146	
Nome: Carvalho	Frequência: 110	

4.2.3 Alínea C

```
N° de candidatos que têm parentes eclesiásticos: 15229

+---------+---------+
|Irmãos   |   14448 |
+---------+---------+
|Tios     |    4116 |
+---------+---------+
|Primos   |    1060 |
+---------+---------+

Tipo de parentesco mais frequente: Irmãos
```

4.2.4 Alínea D[1]

```
+-----------------------------------------------------------------------
| Pai: Antonio Fernandes
| Mãe: Maria Fernandes
| N° de filhos: 10
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------
| Pai: Joao Machado Fagundes
| Mãe: Mariana Josefa Castro
| N° de filhos: 10
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------
| Pai: Antonio Goncalves
| Mãe: Maria Goncalves
| N° de filhos: 11
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------
| Pai: Joao Goncalves
| Mãe: Maria Goncalves
| N° de filhos: 14
+-----------------------------------------------------------------------
```

4.2.5 Alínea E[1]

Para o ano 1911, algumas árvores genealógicas são:

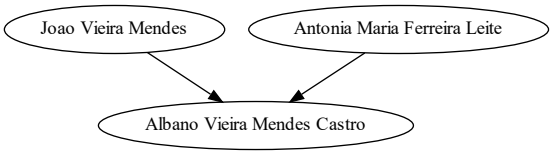

Figura 4.1: Árvore 1

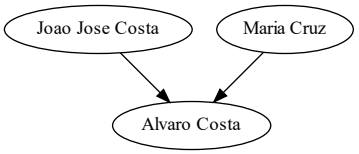

Figura 4.2: Árvore 2

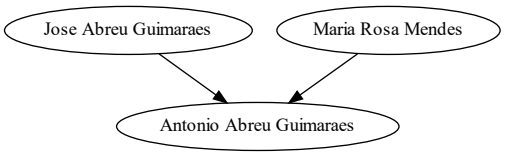

Figura 4.3: Árvore 3

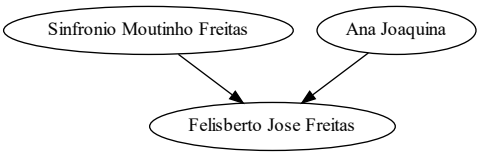

Figura 4.4: Árvore 4

[1]Resultados parciais

Capítulo 5

Conclusão

Dada por concluída o trabalho, consideramos relevante efetuar uma análise crítica do trabalho realizado.

Através do desenvolvimento deste projeto conseguimos melhorar os nossos conhecimentos relativamente à construção de filtros a partir de expressões regulares e de regras Condição-Ação. A utilidade destres filtros revelou-se realmente útil na medida em que nos permitiu, a partir de um ficheiro extenso e confuso, gerar dados relevantes e organizados de forma apelativa. A importância destes filtros na análise e recolha de dados realmente fez a diferença na apresentação de informação.

Concluído o desenvolvimento do trabalho, notamos que existiram aspetos positivos a realçar, entre eles a preocupação em analisar todos os detalhes e gralhas do ficheiro assim como a implementação de um código simples e eficaz.

Por outro lado, também existiram algumas dificuldades, tais como tentar analisar processos quase um a um de modo a tentar encontrar todas as incongruências possíveis. Apesar de ter sido necessário um cuidado e atenção redobrada pensamos ter conseguido superar esta dificuldade.

Para concluir, consideramos que houve um balanço positivo do trabalho realizado dado que as dificuldades sentidas foram superadas e foram cumpridos todos os requisitos.

Capítulo 6

Anexos

6.1 Alínea A

```
import re

def int_to_Roman(num):
        val = [
            1000, 900, 500, 400,
            100, 90, 50, 40,
            10, 9, 5, 4,
            1
            ]
        syb = [
            "M", "CM", "D", "CD",
            "C", "XC", "L", "XL",
            "X", "IX", "V", "IV",
            "I"
            ]
        roman_num = ''
        i = 0
        while  num > 0:
            for _ in range(num // val[i]):
                roman_num += syb[i]
                num -= val[i]
            i += 1
        return roman_num

f = open('processos.xml')

anos = {}
ano_menor = '~'
ano_maior = ''
seculos = set()
for line in f:
    if res := re.search(r'<data>((.|\n)*)<\/data>',line) :
        ano = re.split(r'-', res.group(1))[0]
        data = res.group(1)
```

```
            if int(ano)% 100 != 0 :
                seculos.add(int(ano)//100 + 1)
            else:
                seculos.add(int(ano)//100)

            if (data < ano_menor):
                ano_menor = data

            elif (data > ano_maior):
                ano_maior = data

            elif ano in anos.keys():
                anos[ano] += 1
            else:
                anos[ano] = 1
anos = dict(sorted(anos.items(), key=lambda p: p[0]))
s = '---------------------------------------------------\n{:9} Número de
    processos por ano:\n
    ---------------------------------------------------'.format(' ')
print(s)

for i in anos:
    print(f'\t  Ano: {i}        Processos: {anos[i]}' )


s = '---------------------------------------------------\n{:13} Intervalo
    de datas:\n---------------------------------------------------'.
    format(' ')
print(f'{s}\n \t\t   {ano_menor}\n \t\t      até\n \t\t   {ano_maior}'
    )

seculos = sorted(seculos)
s = '---------------------------------------------------\n{:13} Séculos
    analisados:\n---------------------------------------------------'.
    format(' ')
print(f'{s}\n\t   ', end='')
for s in seculos:
    print(int_to_Roman(s), end=' ; ')
print('\n')
```

6.2 Alínea B

```
import re

def int_to_Roman(num):
        val = [
            1000, 900, 500, 400,
            100, 90, 50, 40,
            10, 9, 5, 4,
            1
            ]
        syb = [
            "M", "CM", "D", "CD",
            "C", "XC", "L", "XL",
            "X", "IX", "V", "IV",
            "I"
            ]
        roman_num = ''
        i = 0
        while  num > 0:
            for _ in range(num // val[i]):
                roman_num += syb[i]
                num -= val[i]
            i += 1
        return roman_num

f = open('processos.xml')

nomes = {}
apelidos = {}
seculo = 0
for line in f:

    if res2 := re.search(r'<data>((.|\n)*)<\/data>',line) :

        if int((re.split(r'-', res2.group(1))[0])) % 100 != 0 :
            seculo = int((re.split(r'-', res2.group(1))[0]))//100 +1
        else:
            seculo = int((re.split(r'-', res2.group(1))[0]))//100

    if res := re.search(r'<nome>((.|\n)*)<\/nome>',line) :
        nome = re.search(r'^[A-Za-z]+', res.group(1)).group(0)
        apelido = re.search(r'[A-Za-z]+$', res.group(1)).group(0)



        if (nome,seculo) in nomes.keys():
            nomes[nome,seculo] += 1
        else:
            nomes[nome,seculo] = 1

        if (apelido,seculo) in apelidos.keys():
            apelidos[apelido,seculo] += 1
        else:
```

```
            apelidos[apelido,seculo] = 1

nomes = dict(sorted(nomes.items(),
        key=lambda p: (p[0][1], p[1]),reverse=True))
apelidos = dict(sorted(apelidos.items(),
        key=lambda p: (p[0][1], p[1]),reverse=True))
freqnomes = []
freqapelidos = []


def freqs(lista,listafreq):
    sec = list(lista.keys())[0][1]
    i = 0
    for n in lista:

        if((sec == n[1])):
            if(i < 5):
                listafreq.append((n[0],lista[n]))
                i+=1
            else:
                sec -=1
                i=0

freqs(nomes,freqnomes)
freqs(apelidos,freqapelidos)

def topfreq(string, listafreq):
    sec = list(lista.keys())[0][1]
    i=0
    print(f'{string}')
    print(f'Século {int_to_Roman(sec)}:')
    for f in listafreq:


        if(i < 5):
            s = '\tNome: {:11} Frequência: {:}'.format(f[0],f[1])
            print(s)
            i+=1
        else:
            sec-=1
            print(f'\nSéculo {int_to_Roman(sec)}:')
            s = '\tNome: {:11} Frequência: {:}'.format(f[0],f[1])
            print(s)
            i=1

topfreq('-------------------------------------------------\n{:20}
    Nomes\n-------------------------------------------------'.format('
    '), freqnomes)
topfreq('-------------------------------------------------\n{:18}
    Apelidos\n-------------------------------------------------'.format
    (' '), freqapelidos)
```

6.3 Alínea C

```
import re

f = open('processos.xml')

irmao = 0
tio = 0
primo = 0
obsline = ''
parentes = ''
candidatos = 0


def contaParentes(string):
    if res := re.findall(r'([a-zA-Z ,]+)\,([a-zA-Z ]+)\.(\ ?)([Pp][Rr
        ][Oo][Cc])', string):
        global irmao
        global tio
        global primo
        global candidatos
        candidatos +=1
        for p in res:

            if re.search(r'Irmaos', p[1]):
                irmao += len(re.findall(r'( e |,)', p[0])) + 1
            elif re.search(r'Irmao', p[1]):
                irmao +=1
            elif re.search(r'Tios', p[1]):
                tio += len(re.findall(r'( e |,)', p[0])) + 1
            elif re.search(r'Tio', p[1]):
                tio +=1
            elif re.search(r'Primos', p[1]):
                primo += len(re.findall(r'( e |,)', p[0])) + 1
            elif re.search(r'Primo', p[1]):
                primo +=1



for line in f:

    if res := re.search(r'<obs>([^<]*)(<\/obs>)?', line):

        if re.search(r'<\/obs>$', res.group(0)):
            contaParentes(res.group(1))

        else:
            obsline = obsline + res.group(1).strip('\n')
    elif res := re.search(r'(.*)</obs>$', line):
        obsline = obsline + ' ' + res.group(1).strip()
        contaParentes(obsline)
        obsline = ''
    elif res:= re.search(r'^[^<](.*)[^>]$',line.strip()):
```

```
        obsline = obsline + ' ' + res.group(0).strip()


s = '-------------------------------------------------------------\nN° de
    candidatos que têm parentes eclesiásticos: {:}\n
    -------------------------------------------------------------\n'.format(
    candidatos)
print(s)

pdict = {'Irmãos':irmao, 'Tios': tio, 'Primos': primo}
pdict = dict(sorted(pdict.items(), key=lambda p: p[1], reverse=True))
print('\t       +----------+----------+')
for d in pdict:
    print(f'\t       |{d:10}| {pdict[d]:8}  |')
    print('\t       +----------+----------+')


s = '\n-------------------------------------------------------------\n
    Tipo de parentesco mais frequente: {:6}\n
    -------------------------------------------------------------\n'.format(
    list(pdict.keys())[0])
print(s)
```

6.4 Alínea D

```
import re

f = open('processos.xml')

casal = {}
pai = ''
mae = ''
for line in f:

    if resPai := re.search(r'<pai>((.|\n)*)<\/pai>',line) :
        pai = resPai.group(1)
    if resMae := re.search(r'<mae>((.|\n)*)<\/mae>',line) :
        mae = resMae.group(1)

        if (pai,mae) in casal.keys():
            casal[pai,mae] += 1
        else:
            casal[pai,mae] = 1

casal = {key:val for key, val in casal.items() if int(val) != 1}
casal = dict(sorted(casal.items(), key=lambda p: p[1]))

for c in casal:
    print(f'+-----------------------------------------------------\
      n| Pai: {c[0]}\n| Mãe: {c[1]}\n| N° de filhos: {casal[c]}\n
      +-----------------------------------------------------')
```

6.5 Alínea E

```
from graphviz import Digraph
import re
import os
import shutil
try:
    shutil.rmtree("árvores")
except OSError as e:
    pass

ano = input ("Insira um ano:\n> ")
opcao = input("\nGerar um único ficheiro     (1)\nGerar múltiplos
   ficheiros  (2)\n> ")
print('A gerar ficheiros....')

cont1 = 0
cont2 = 0
f = open('processos.xml')
familia = []
mae = ''
pai = ''
filho = ''
for line in f:
    if res := re.search(r'<data>('+ano+').*<\/data>', line) :
        year = re.split(r'-', res.group(1))[0]
        cont1 += 1

    if res2 := re.search(r'<nome>(.*)<\/nome>', line) :
        if (cont2 < cont1) :
            filho = res2.group(1)

    if res3 := re.search(r'<pai>(.*)<\/pai>', line) :
        if (cont2 < cont1) :
            pai = res3.group(1)

    if res4 := re.search(r'<mae>(.*)<\/mae>', line) :
        if (cont2 < cont1) :
            mae = res4.group(1)
            familia.append((filho,pai,mae))
            cont2 +=1


def opcao_sep(familia):
    i = 0
    for f in familia:
        aux = str(i)
        aux = Digraph(comment='Árvore genealógica')

        aux.node('1', f[0])
        aux.node('2', f[1])
        aux.node('3', f[2])

        aux.edges(['21','31'])
```

```
        aux.render('árvores/'+str(i)+'.gv')
        i+=1

def opcao_junta(familia):
    i=0
    dot = Digraph(comment='Árvore genealógica')
    for f in familia:
        aux = str(i)
        str0 = "a"+aux
        str1 = "b"+aux
        str2 = "c"+aux
        dot.node(str0, f[0])
        dot.node(str1, f[1])
        dot.node(str2, f[2])

        dot.edge(str1,str0, constraint='true')
        dot.edge(str2,str0, constraint='true')
        i+=1

    dot.render('árvores/árvores.gv')

if(opcao == '1'):
    opcao_junta(familia)
else:
    opcao_sep(familia)

if len(familia)==0:
    print('\nNão há nenhuma ocorrência do ano inserido!')
else:
    print('\nFicheiros Gerados!')
```