

Processamento de Linguagens e Compiladores

Trabalho Prático 1

Relatório de Desenvolvimento

Nome-Aluno1
(numero-al1)

Nome-Aluno2
(numero-al2)

Nome-Aluno3
(numero-al3)

16 de novembro de 2021

Resumo

O seguinte relatório documenta, justifica, analisa e expõe todas as decisões tomadas ao longo do Trabalho Prático realizado no âmbito da Unidade Curricular denominada Processamento de Linguagens e Compiladores no contexto do 3º ano do curso Licenciatura em Ciências da Computação.

O seguinte trabalho era um dos 5 que nos foi apresentados na Unidade Curricular, visto que somos o grupo 8, usando a formula $(NGr \% 7) + 1$ verificamos que nos calhou o trabalho nº2. Este consistia em criar 4 programas Python para processar o ficheiro de texto "processos.txt" com o intuito de calcular frequências de alguns elementos.

Conteúdo

1	Introdução	2
1.1	Processador de Pessoas listadas nos Róis de Confessados	2
2	Análise do problema	3
2.1	Descrição informal do problema	3
2.2	Estratégia inicial adoptada	4
3	Soluções criadas	5
3.1	Problema a	5
3.1.1	ER	5
3.1.2	Algoritmo	5
3.1.3	Exemplo de funcionamento	6
3.2	Problema b	7
3.2.1	ER	7
3.2.2	Algoritmo	7
3.2.3	Exemplo de funcionamento	8
3.3	Problema c	8
3.3.1	ER	8
3.3.2	Algoritmo	8
3.3.3	Exemplo de funcionamento	9
3.4	Problema d	10
3.4.1	ER	10
3.4.2	Algoritmo	10
3.4.3	Exemplo de funcionamento	10
4	Conclusão	12
A	Código da alínea a)	13
B	Código da alínea b)	14
C	Código da alínea c)	15
D	Código da alínea d)	16

Capítulo 1

Introdução

1.1 Processador de Pessoas listadas nos Róis de Confessados

Vamos agora enquadrar e contextualizar o trabalho. É-nos dado um ficheiro onde se encontram vários dados, entre eles nomes, datas e relações familiares de várias pessoas, que estariam guardados no arquivo municipal de Braga. Temos como intuito extrair, de forma objetiva, através de expressões regulares e outras estratégias dadas nas aulas, de forma a hipoteticamente ser possível analisar os dados em questão. No capítulo 2 iremos fazer uma breve introdução ao enunciado, descrevendo assim os problemas que nos são propostos, além dos pensamentos iniciados antes de resolver os problemas. No capítulo 3 explicaremos detalhadamente a solução para cada um dos problemas propostos, incluindo não só as expressões regulares, mas também os métodos com que estas são utilizadas de forma a extrair a informação necessária do ficheiro, decisões tomadas e ainda alguns testes realizados e resultados dos mesmos, de forma a comprovar a eficácia do nosso trabalho.

Capítulo 2

Análise do problema

2.1 Descrição informal do problema

No enunciado é nos dado um ficheiro, neste caso o “processos.txt”, sobre o qual devemos desenvolver programas em Python de forma a resolver as seguintes questões :

- a) Calcular a frequência de processos por ano
- b) Calcular a frequência de nomes
- c) Calcular a frequência dos vários tipos de relação
- d) Imprimir os 20 primeiros registos num novo ficheiro de output, em formato Json

Vejamos agora um excerto do texto dado :

```
\569::1867-05-23::Abel Alves Barroso::Antonio Alves Barroso::Maria Jose Alvares Barroso::  
Bento Alvares Barroso,Tio Paterno. Proc.32057."
```

É possível observar que toda a informação está condensada e misturada, havendo a necessidade de descobrir maneiras de identificar os vários tipos de informação pedida, sendo possível disponibilizá-la de acordo com cada uma das alíneas a cima referidas.

Desta forma, existe a necessidade de desenvolver estratégias e expressões regulares que se adequem a cada um dos problemas, desprezando a informação indesejada e mantendo apenas a que foi pedida. Ao longo da resolução dos problemas, decidimos separar a informação em grupos convenientemente decididos, tendo em conta qual parte da informação é pedida por alínea, através das estratégias que serão mais à frente explicadas.

2.2 Estratégia inicial adoptada

Quando fomos confrontados com o enunciado, após a leitura do mesmo e do ficheiro dado, discutimos várias ideias sobre qual seria a melhor forma de abordar o problema em questão. Visto que queríamos calcular a frequência de processos por ano, deliberámos a forma correta de os identificar.

Concluimos então que existia um padrão, sendo este “número do processo :: data”. Através desta ideia, pretendemos adaptar a expressão regular e o código do programa de forma a identificar este padrão e guardar a sua frequência.

Após termos chegado à conclusão da existência de padrões, efetuámos o mesmo com o problema seguinte. Deparámos-nos com o padrão em que os nomes estariam sempre antecidos por dois pontos, sendo assim o padrão predominante “::Nome Completo”. Em certas exceções é possível observar que existem nomes que se encontram da forma “.Nome Completo,”. Tal como anteriormente, após esta descoberta procedemos à tentativa de adaptação tanto do programa como da expressão regular de forma a que identificassem este padrão.

Para o problema dos vários tipos de relação, houve maior dificuldade em padronizar os dados existentes acerca dessas mesmas relações. As relações poderiam estar tanto no singular como no plural, algo importante a ter em consideração. Eventualmente concluimos que todas as relações, quer estejam no singular ou no plural, estariam compreendidas entre um ponto e uma vírgula, da forma “ , Relação.”. Após esta conclusão, procedemos à escrita do código e respetiva expressão regular.

Capítulo 3

Soluções criadas

Como referido, para melhor compreensão por parte do leitor, vamos falar das soluções criadas para cada alínea separadamente.

3.1 Problema a

3.1.1 ER

Começamos então por definir a nossa expressão regular que vai retirar todos os processos e as suas respetivas datas.

Como sabemos que os processos e o ano são da forma, por exemplo, "575::1894" criamos a seguinte ER:

```
ER = ([0-9]{3})::([0-9]{4})
```

Ou seja, vamos procurar 3 numeros de 0 a 9, que será o processo, seguido de :: e depois 4 numeros de 0 a 9, que será a data.

3.1.2 Algoritmo

Para começar a resolver este problema, primeiramente damos open ao ficheiro de texto "processos.txt":

```
f = open("processos.txt")
linhas = f.readlines()
```

Após isso, vamos iterar por cada linha do texto e usando a nossa ER, retirar os processos e os anos:

```
for i in linhas:
    new_text = re.search(r'([0-9]{3})::([0-9]{4})', i)
```

Aqui usamos o search e não o findall, porque como estamos a procurar em cada linha, só vamos encontrar uma ocorrência do processo e do ano

Depois guardamos o ano num dicionário como key e o processo como valor e criamos um contador para ser possível calcular a frequência:

```
datas = {}

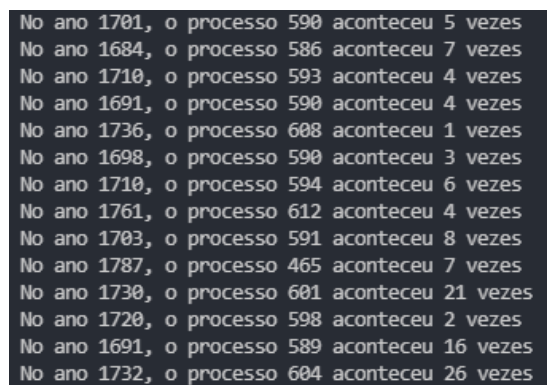
if new_text:
    data = new_text.group(2)
    processo = new_text.group(1)
    if (data,processo) not in datas:
        datas[(data,processo)] = 1
    else:
        datas[((data,processo))] += 1
```

No fim damos então print ao nosso dicionário com a seguinte estrutura:

```
for (a,b) in datas:
    print("No ano " + str(a) + ", o processo " + str(b) + " aconteceu " + str(datas[a,b]) + " ve
```

3.1.3 Exemplo de funcionamento

Executando a nossa solução para este problema vamos receber o seguinte output:



```
No ano 1701, o processo 590 aconteceu 5 vezes
No ano 1684, o processo 586 aconteceu 7 vezes
No ano 1710, o processo 593 aconteceu 4 vezes
No ano 1691, o processo 590 aconteceu 4 vezes
No ano 1736, o processo 608 aconteceu 1 vezes
No ano 1698, o processo 590 aconteceu 3 vezes
No ano 1710, o processo 594 aconteceu 6 vezes
No ano 1761, o processo 612 aconteceu 4 vezes
No ano 1703, o processo 591 aconteceu 8 vezes
No ano 1787, o processo 465 aconteceu 7 vezes
No ano 1730, o processo 601 aconteceu 21 vezes
No ano 1720, o processo 598 aconteceu 2 vezes
No ano 1691, o processo 589 aconteceu 16 vezes
No ano 1732, o processo 604 aconteceu 26 vezes
```

Figura 1. Excerto do output do programa a)

Verificamos então que a solução está correta e faz o que é pedido no enunciado do problema.

3.2 Problema b

3.2.1 ER

Como pretendemos reconhecer todos os nomes, criamos a seguinte expressão regular:

```
ER = ([a-zA-Z]+[ ]?)+
```

3.2.2 Algoritmo

Observámos que todos os nomes são antecidos por “::”, como por exemplo “::Adriano Duarte”. Sendo assim, a primeira coisa que fizemos foi um `re.split` onde o separador é “::”.

```
for linha in linhas:
    lista = re.split(r'::',linha)
```

Desta forma ao receber uma linha de texto, por exemplo, “575::1894-11-08::Aarao Pereira Silva” vamos transformar essa linha na lista “575”, “894-11-08”, “Aarao Pereira Silva”, desta forma a nossa ER consegue apanhar todos os nomes.

```
for linha in linhas:
    lista = re.split(r'::',linha)
    for elemento in lista:
        new_text = re.search(r'([a-zA-Z]+[ ]?)+',elemento)
```

Seguidamente guardamos então os nomes num dicionário e criamos um contador para calcular a frequência de cada nome.

```
relacao = {}

    if new_text:
        texto = new_text.group()
        if texto not in relacao:
            relacao[texto] = 1
        else:
            relacao[texto] += 1
```

No fim damos print aos nomes guardados no dicionário e o respetivo contador

```
for rela in relacao:
    print ("Nome: " + rela + ": " + str(relacao[rela]))
```

3.2.3 Exemplo de funcionamento

Executando a nossa solução para este problema vamos receber o seguinte output:

```
Nome: Venceslau Dias Leite: 2
Nome: Joao Dias Coelho: 2
Nome: Joaquina Soares Leite: 2
Nome: Venceslau Gabriel Dias Galas Costa: 1
Nome: Jose Antonio Galas: 1
Nome: Camila Domingues Espirito Santo: 1
Nome: Joao Marcelino Dias Galas Costa: 1
Nome: Ventura Alvares: 1
Nome: Ventura Antunes: 1
```

Figura 2. Excerto do output do programa b)

3.3 Problema c

3.3.1 ER

Pretendemos identificar e calcular a frequência dos vários tipos de relações familiares.

Observámos que todas as relações seriam escritas após nomes, sendo antecedidas por uma vírgula e após as mesmas estaria um ponto. Utilizámos o findall para encontrar todas as ocorrências dos vários tipos de relações, em conjunto com a seguinte expressão regular:

```
ER = (?i)(\birma[o]?|\bsobrinh[oa]|\bavo|\bti[oa]|\bprim[oa]|\bpai|\bfilh[oa]|\bmae)
([\ ]+?[a-zA-Z]+[.,])?
```

Incluímos todos os graus de parentesco identificados na ER, tendo em conta que estes poderiam estar no masculino ou feminino, e que poderiam ser seguidores de um semi grau de parentesco, por exemplo Sobrinho Neto.

3.3.2 Algoritmo

Começamos então por iterar por cada linha do ficheiro de texto e fazer um re.split. Depois para cada elemento nessa nova lista que o split devolve aplicamos a nossa ER.

```
for linha in linhas:
    lista = re.split(r':', linha)
    for elemento in lista:
        new_text = re.findall(r'(?i)(\birma[o]?|\bsobrinh[oa]|\bavo|\bti[oa]|\bprim[oa]|\bpai|\bfilh[oa]|\bmae)([\ ]+?[a-zA-Z]+[.,])?', elemento)
```

Tivemos ainda em conta que algumas destas relações poderiam ser apresentadas no plural, adequando assim o nosso código de forma a que fossem também incluídas na contagem dessa mesma relação, mas no singular.

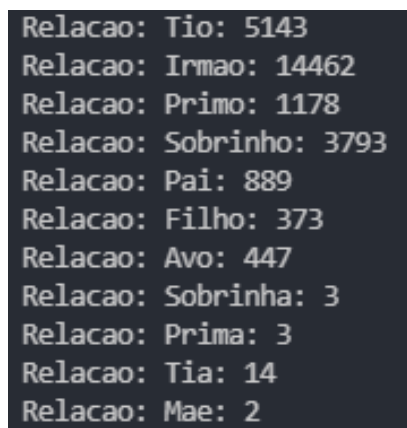
```
for palavra in a:
    por = palavra
    aux = list(por)
    if aux[len(aux)-1] == 's':
        del(aux[len(aux)-1])
    aux[0] = aux[0].upper()
    por = ''.join(aux[i] for i in range(len(aux)))
```

Por fim, de forma a ser possível calcular a frequência, criamos um contador.

```
if por not in relacao:
    relacao[por] = 1
else:
    relacao[por] += 1
```

3.3.3 Exemplo de funcionamento

Executando a nossa solução para este problema vamos receber o seguinte output:



```
Relacao: Tio: 5143
Relacao: Irmão: 14462
Relacao: Primo: 1178
Relacao: Sobrinho: 3793
Relacao: Pai: 889
Relacao: Filho: 373
Relacao: Avo: 447
Relacao: Sobrinha: 3
Relacao: Prima: 3
Relacao: Tia: 14
Relacao: Mãe: 2
```

Figura 2. Output do programa c)

Para verificar a veracidade do nosso output, vamos confirmar no ficheiro de texto se os valores estão de facto corretos.



The image shows two search bars from a text editor. The first search bar contains the word 'Tio' and shows a count of 5129 of 5143. The second search bar contains the word 'Irmão' and shows a count of 14345 of 14462.

Figura 3. Confirmação dos valores do programa c)

Como podemos verificar, o nosso programa está correto.

3.4 Problema d

3.4.1 ER

Como esta alínea pedia apenas para imprimir os primeiros 20 registos em formato Json, não foi preciso definir uma ER.

3.4.2 Algoritmo

Primeiramente como o problema pede para imprimir os primeiros 20 registos, vamos abrir o nosso ficheiro de texto, mas ler só as primeiras 20 linhas.

```
N=20
with open("processos.txt", "r") as fileoriginal:
    fileN = [next(fileoriginal) for x in range(N)]
```

De seguida, voltamos a usar a tática de fazer re.split a cada linha do nosso texto:

```
for v in fileN:
    lista = re.split(r'::|[\ ]+[\ ]+',v)
```

Depois vamos adicionar cada linha a um dicionário e no fim dessa linha passamos esse dicionário para uma lista

```
for elemento in lista:
    if elemento != '\n' and elemento != '':
        if contador == 0:
            dic["numero processo"] = elemento
        elif contador == 1:
            dic["data"] = elemento
        elif contador >= 2:
            dic["nome(s)" + str(nome)] = elemento
            nome += 1
        contador += 1
    listadics.append(dic)
```

Por fim, damos print ao nosso dicionário em formato JSON usando o seguinte comando:

```
with open("json.json", 'w') as file:
    file.write((json.dumps(listadics, indent=4, sort_keys= False)))
```

3.4.3 Exemplo de funcionamento

Ao receber, por exemplo, as seguintes linhas de texto:

```
575::1894-11-08::Aarao Pereira Silva::Antonio Pereira Silva::Francisca Campos Silva:::
582::1909-05-12::Abel Almeida::Antonio Manuel Almeida::Teresa Maria Sousa:::
```

O nosso programa tem o seguinte output:

```

1  [
2    {
3      "numero processo": "575",
4      "data": "1894-11-08",
5      "nome(s)1": "Aarao Pereira Silva",
6      "nome(s)2": "Antonio Pereira Silva",
7      "nome(s)3": "Francisca Campos Silva"
8    },
9    {
10     "numero processo": "582",
11     "data": "1909-05-12",
12     "nome(s)1": "Abel Almeida",
13     "nome(s)2": "Antonio Manuel Almeida",
14     "nome(s)3": "Teresa Maria Sousa"
15   },

```

Figura 4. Excerto do output do programa d)

Como podemos verificar, imprime corretamente as duas linhas em formato Json.

Para verificar que o output para as 20 linhas que é pedido estão de facto correto, usamos o site **json-formatter.curiousconcept.com**, onde passamos o ficheiro Json que o nosso programa gera e verificamos se tá correto ou não:

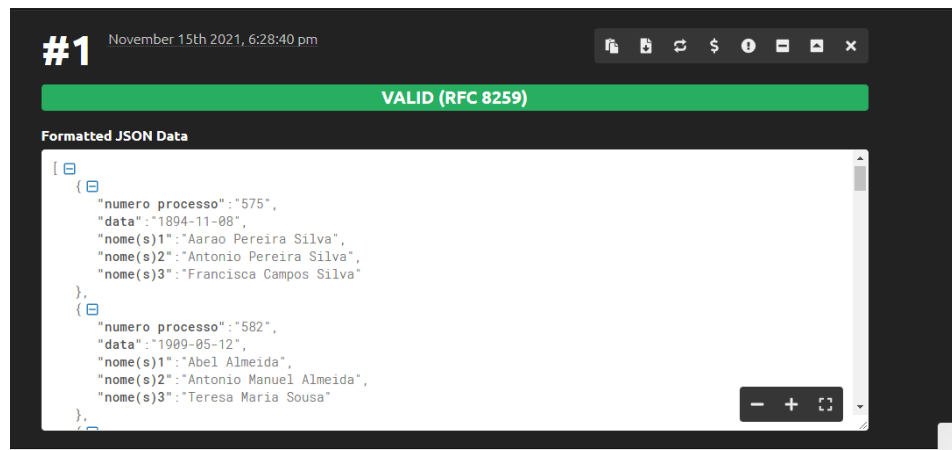


Figura 5. Confirmação do ficheiro Json

Como podemos verificar, o site validou o nosso ficheiro Json.

Capítulo 4

Conclusão

Síntese do Documento [?, ?].

Estado final do projecto; Análise crítica dos resultados [?].

Trabalho futuro.

Apêndice A

Código da alínea a)

```
1 import re
2
3 f = open("processos.txt")
4 linhas = f.readlines()
5 datas = {}
6
7 for i in linhas:
8     new_text = re.search(r'([0-9]{3})::([0-9]{4}) ', i)
9     if new_text:
10         data = new_text.group(2)
11         processo = new_text.group(1)
12         if (data, processo) not in datas:
13             datas[(data, processo)] = 1
14         else:
15             datas[((data, processo))] += 1
16 f.close()
17 for (a,b) in datas:
18     print("No ano " + str(a) + ", o processo " + str(b) + " aconteceu " + str(datas[a
    ,b]) + " vezes")
```

Apêndice B

Código da alínea b)

```
1 import re
2
3 f = open("processos.txt")
4 linhas = f.readlines()
5 relacao = {}
6
7 for linha in linhas:
8     lista = re.split(r'::', linha)
9     for elemento in lista:
10         new_text = re.search(r'([a-zA-Z]+[ ]?)', elemento)
11         if new_text:
12             texto = new_text.group()
13             if texto not in relacao:
14                 relacao[texto] = 1
15             else:
16                 relacao[texto] += 1
17
18 for rela in relacao:
19     print ("Nome: " + rela + ": " + str(relacao[rela]))
```

Apêndice C

Código da alínea c)

```
1 import re
2
3 f = open("processos.txt")
4 linhas = f.readlines()
5 relacao = {}
6
7 for linha in linhas:
8     lista = re.split(r'::', linha)
9     for elemento in lista:
10         new_text = re.findall(r'(?i)(\birma[o]?|\bsobrinh[oa]|\bavo|\bti[oa]|\bprim[
11             oa]|\bpai|\bfilh[oa]|\bmae)([ ]+?[a-zA-Z]+[. ,])?', elemento)
12         if new_text:
13             a = []
14             for k in range(len(new_text)):
15                 a.append(new_text[k][0])
16
17         for palavra in a:
18             por = palavra
19             aux = list(por)
20             if aux[len(aux)-1] == 's':
21                 del(aux[len(aux)-1])
22             aux[0] = aux[0].upper()
23             por = ''.join(aux[i] for i in range(len(aux)))
24
25             if por not in relacao:
26                 relacao[por] = 1
27             else:
28                 relacao[por] += 1
29 for rela in relacao:
30     print ("Relacao: " + rela + ": " + str(relacao[rela]))
```

Apêndice D

Código da alínea d)

```
1 import re
2 import json
3
4 listadics = []
5 N=20
6 with open("processos.txt", "r") as fileoriginal:
7     fileN = [next(fileoriginal) for x in range(N)]
8
9 for v in fileN:
10     lista = re.split(r'::| |'+',v)
11     dic = {}
12     contador = 0
13     nome = 1
14     linha = 0
15     for elemento in lista:
16         if elemento != '\n' and elemento != '':
17             if contador == 0:
18                 dic["numero processo"] = elemento
19             elif contador == 1:
20                 dic["data"] = elemento
21             elif contador >= 2:
22                 dic["nome(s)" + str(nome)] = elemento
23                 nome += 1
24                 contador += 1
25     listadics.append(dic)
26
27 with open("json.json", 'w') as file:
28     file.write((json.dumps(listadics, indent=4, sort_keys= False)))
```
