Universidade do Minho
Escola de Engenharia

Processamento de Linguagens

MIEI - 3ºAno

Trabalho Prático 1

Francisco Correia Franco A89458
Carlos Miguel Luzia de Carvalho A89605
José Pedro Carvalho Costa A89519

A89458

A89605

A89519

5 de abril de 2021

Conteúdo

1 Resumo **1**

2 Introdução **2**

3 Análise e Especificação **3**
3.1 Descrição informal do problema 3
3.2 Especificação dos requisitos 3
3.2.1 Dados 3
3.2.2 Pedidos 4

4 Concepção/Desenho da Resolução **5**
4.1 Estruturas de Dados 5
4.2 Algoritmos 5
4.2.1 Problema a) 5
4.2.2 Problema b) 7
4.2.3 Problema c) 8
4.2.4 Problema d) 8

5 Codificação e Testes **10**
5.1 Alternativas, Decisões e Problemas de Implementação 10
5.2 Testes realizados e resultados 10
5.2.1 Problema a 11
5.2.2 Problema b 12
5.2.3 Problema c 12
5.2.4 Problema d 13

6 Conclusão **14**

7 Anexo de Código **15**
7.1 Problema A 15
7.2 Problema B 16
7.3 Problema C 18
7.4 Problema D 20

1 Resumo

Este relatório tem como objetivo documentar o primeiro trabalho prático da disciplina de Processamento de Linguagens sobre espressões regulares em python.

Primeiro será feita uma introdução ao projeto com uma descrição do enunciado. Depois será apresentada a implementação da solução com as estruturas de dados e os filtros de texto utilizados. Serão também apresentadas ilustrações de exemplos e serão discutidos alguns problemas na implementação. Por fim uma avaliação crítica do trabalho desenvolvido.

2 Introdução

No âmbito de UC de Processamento de Linguagens foi-nos pedido desenvolver o BibTexPro, um processador de BibTex. Para tal foram desenvolvidos vários filtros de texto para poder realizar todas as tarefas propostas. Para a realização deste trabalho foi necessário aplicar os conhecimentos de expressões regulares, bem como familiarizarmo-nos com o formato bibtex. Iremos apresentar toda a linha de raciocino para a concretização das várias tarefas pedidas no enunciado.

3 Análise e Especificação

3.1 Descrição informal do problema

Este projeto foca-se no exercício 4 do enunciado, BibTexPro - Um processador de Bibtex. No enunciado é pedido para desenvolver um ou mais programas em python para processar qualquer base de dados BibTex a partir de um ficheiro exemplo, de modo a realizar várias tarefas.

3.2 Especificação dos requisitos

3.2.1 Dados

O BibTex é simultaneamente um formato para criar uma base de dados textual de registos bibliográficos e uma ferramenta de formatação compatível com o processador de documentos LaTeX. BibTex permite que se façam, no meio de um documento LaTeX, citações bibliográficas a outros documentos públicos que estejam carateriados e classificados numa determinada base de dados escrita nesse formato, com objetivo de facilitar a produção da referência e o respetivo bloco de texto que identifica o documento citado e que vai aparecer na secção' final do documento LaTeXem edição.

```
@InProceedings{CPBFH07e,
  author =    {Daniela da Cruz and Maria João Varanda Pereira
               and Mário Béron and Rúben Fonseca and Pedro Rangel Henriques},
  title =     {{Comparing Generators for Language-based Tools}},
  booktitle = {Proceedings of the 1.st Conference on Compiler
               Related Technologies and Applications, CoRTA'07
               --- Universidade da Beira Interior, Portugal},
  year =      {2007},
  editor =    {},
  month =     {Jul},
  note  =     {}
}
```

1. Exemplo de uma entrada em BibTex

3.2.2 Pedidos

Foi-nos pedido para implementar um ou vários programas em python para filtrar o documento BibTex.

a) Calcular o número de entradas por *categoria*; apresente a istagem em formato HTML por ordem alfabética.

b) Criar um índice de autores, que mapeie cada autor nos respetivos registos identificados pela respetiva chave de citação (a 1ª palavra a seguir à chaveta, que é única em toda a BD).

c) Transforme cada registo num documento JSON válido; a BD original deve ser então um documento JSON válido formado por uma lista de registos.

d) Construa um Grafo que mostre, para um dado autor (definido na altura pelo utilizador) todos os autores que publicam normalmente com o autor em causa. Recorrendo à linguagem **DOT** do **GraphViz**, gere um ficheiro com esse grafo de modo a que possa, posteriormente, usar uma das ferramentas que processam **DOT** para desenhar o dito grado de associação de autores

4 Concepção/Desenho da Resolução

4.1 Estruturas de Dados

Para a realização deste trabalho utilizamos 2 estruturas já implementadas na versão 3.9 do python, nomeadamente arrays e dicionários.

Para a contrução do HTML, apresentação do índice de autores e construção do grafo (alínes a, b e d respetivamente), utilizamos o dicionário pois este permite-nos armazenar para uma chave única, informações relativas a ela.

Para a alínea c) utilizamos um array pois precisavamos de guardar a informação antes de a escrever para o ficheiro e esta é uma estrutura simples e suficiente para a realização desta tarefa.

4.2 Algoritmos

´

4.2.1 Problema a)

Nesta alinea começámos em primeiro lugar por analisar a expressão e identificar o objetivo, analisando o exemplo de bibtex fornecido podemos concluir que quereriamos retirar unicamente o conjunto de letras imediatamente a seguir ao @. Para isso utilizamos uma expressão regular *(+)*. Uma vez tendo a categoria separada para cada bloco de texto *BibTex*, normalizamos todas estas string de forma a todas poderem ser comparadas como iguais ou não, não interferindo assim letras maíusculas ou espaços.De seguida tendo já todas as strings normalizadas utilizámos o dicionário para contar quantas vezes aparece no texto cada uma das categorias, sendo a chave a categoria em causa e o valor a contagem do aparecimento desta.

```
categoria={}

for linha in f:
    #talvez nao seja preciso ser um findall
  campos = re.findall(r'\@(\w+)',linha)
  for c in campos:
        caux=c.lower()
        if caux in categoria:
            categoria[caux]+=1
        else:
            categoria[caux]=1
```

Código referente à separação da categoria e sucessiva normalização e inserção no dicionário

Tendo já o dicionário criado com a informação necessária partimos então para a realização do resto da alinea, passando este dicionário para um documento html.

```
    html = open("a.html","w")
    html.write('''<!DOCTYPE html>
<html>
    <head>
        <title>Relatório</title>
        <meta charset="UTF-8"/>
    </head>
    <body>
        <h1>Categorias</h1>
        <ol>
''')

    for i in sorted(categoria):
        html.write("            <li>" +i + ": "+ str(categoria[i]) + "</li>\n")

    html.write('''      </ol>
    </body>
</html>\n''')
```

Código referente à transposição do dicionário criado para HTML

4.2.2 Problema b)

Para resolver este problema começamos por usar a função *split* com a expressão regular '@' para separar os blocos de entradas do *BibTex*. De seguida em cada bloco vamos procurar e guardar a chave correspondente e utilizando um match, de seguida fazemos search dos autores desse bloco. Se forem encontrados autores, usamos a função *sub*, para retirar *enters* ou espaços a mais, de modo a não criarem interferência, e depois usamos a função *split* para encontrar os vários autores na entrada. Feita esta normalização, percorremos o vetor dos autores do bloco atual e fazemos uma última normalização, para nomes que comecem pelo apelido e depois tenham uma virgula e o resto do nome. Por fim começamos a construir o índice de autores, onde vamos então verificar se já existe uma entrada para cada autor no dicionário. Caso não exista adicionamos o autor e a categoria na entrada do autor correspondente e caso já exista apenas adicionamos a chave à lista de chaves que cada autor possui.

```
categoria=[]
autoresOcorr={}

#HOW TO SPLI
l = f.read()
categoria = re.split('@',l)

#separa dentro de cada categoria a chave única e coloca em campo e os autores separa tmb
for i in categoria:
    campo = re.match(r'(\w+\{([^,]+))', i)

    if campo:
        if autor := re.search(r'\b(?i:author) *= *\"([^"]*)\",?|\b(?i:author) *= *\{(.*)\},?',i):
            #print(autor)

            lsa=autor.group(1)
            if not(lsa):
                lsa=autor.group(2)

            lsa=re.sub(r'( |\n)+', r' ', lsa)
            autores = re.split(r' +and +',lsa)


            for aut in autores:
                if aux:=re.search(r'([\w\. ]+), ([\w\. ]+)',aut):
                    aut = aux.group(2) +' ' +aux.group(1)


                if aut in autoresOcorr:
                    autoresOcorr[aut.strip()].append(campo[2])
                else:
                    autoresOcorr[aut.strip()]=[campo[2]]
```

Código referente à sequência de etapas referidas a cima

4.2.3 Problema c)

Na resolução desta alínea utilizamos a mesma ideia que na b, mas de uma forma mais geral, ou seja, demos *split* pela expressão regular '@', obtendo os blocos do *BibTex*. Depois, fizemos um *match* e um *split* para obter o tipo e a referência, como na b, e adicionamos ao ficheiro *json* . De seguida, ao contrario da b, fizemos um *findall*, pois vamos precisar de todos os campos do bloco, substituindo (?i:author) por (+) para ser mais geral, e adicionamos o caso do valor do campo ser totalmente interiro e, por isso, não precisar de aspas ou chavetas. Por fim, é necessário percorrer o resultado do *findall* e adiciona-lo ao ficheiro *json*.

```
for i in categoria:

    if autor := re.search(r'\b(?i:author) *= *\"([^"]*)\",?|\b(?i:author) *= *\{(.*)\},?',i):
        #print(autor)

        lsa=autor.group(1)
        if not(lsa):
            lsa=autor.group(2)


        lsa=re.sub(r'( |\n)+', r' ', lsa)
        autores = re.split(r' +and +',lsa)


        if autores.__contains__(ex_aut):

            for aut in autores:
                if aux:=re.search(r'([\w\. ]+), ([\w\. ]+)',aut):
                    aut = aux.group(2) +' ' +aux.group(1)

                if not aut.__eq__(ex_aut) and aut not in autoresRelat:
                    autoresRelat.append(aut)
```

Código referente à separação de autores associados

4.2.4 Problema d)

Para este problema também nos seguimos muito pelo raciocínio tomado em b,sendo que dividimos o conteúdo do ficheiro BibTex em blocos e para cada um desses blocos retirávamos e normalizávamos os autores. Posto isto, para cada um desses blocos procuramos se o nome inserido pelo utilizador consta na lista de autores, caso conste vamos adicionar todos os nomes de autores que ainda não tenham sido acrescentados, a uma lista com todas as associações do autor a procurar.

```
for i in categoria:

    if autor := re.search(r'\b(?i:author) *= *\"([^"]*)\",?|\b(?i:author) *= *\{(.*)\},?',i):
        #print(autor)

        lsa=autor.group(1)
        if not(lsa):
            lsa=autor.group(2)


        lsa=re.sub(r'( |\n)+', r' ', lsa)
        autores = re.split(r' +and +',lsa)


        if autores.__contains__(ex_aut):

            for aut in autores:
                if aux:=re.search(r'([\w\. ]+), ([\w\. ]+)',aut):
                    aut = aux.group(2) +' ' +aux.group(1)

                if not aut.__eq__(ex_aut) and aut not in autoresRelat:
                    autoresRelat.append(aut)
```

Código referente à separação de autores associados

Tendo já a lista com os autores relacionados com o autor em causa vamos então construir o gráfico, recorrendo à linguagem DOT do GraphViz desta associação de autores. Sendo que todos os autores na lista vão estar ligados ao autor fornecido pelo utilizador.

```
g = Graph(format='png')
g.node(str(0), ex_aut)
k=1

for i in autoresRelat:
    g.node(str(k), i)
    g.edge(str(0), str(k), constraint='true')
    k+=1

g.render('graphviz/doutput', view=True)

print(autoresRelat)
```

Código referente à separação de autores associados

5 Codificação e Testes

5.1 Alternativas, Decisões e Problemas de Implementação

Nas alínes b), c) e d), decidimos ler o ficheiro todo e fazer *split* pela expressão regular '@', guardando os blocos de entradas num array. Em ficheiros muito grandes esta forma acaba por não ser muito eficaz pois estamos a utilizar demasiada memoria para guardar todos os blocos de entradas. Uma alternativa a isto seria ler o ficheiro por linhas e tratar cada uma na sua vez, assim não era necessário estar a utilizar memória disponível para armazenar informação do ficheiro.

Um dos problemas que enfrentamos foi o de poder haver grupos de chavetas aninhados. Nos casos em que não havia chavetas aninhados uma expressão que apanhasse qualquer coisa exceto '}' serviria perfeitamente, mas ao ter mais grupos de chavetas abrem-se muitas exceções aquele caso. Esta situação tornou-se um grande problema pelo facto do modulo *re* não suportar padrões recursivos. A solução que encontramos foi de que cada campo só podia ter no máximo um elemento aninhado delineado por chavetas.

5.2 Testes realizados e resultados

Relativamente aos testes que fomos realizando para a realização deste projecto, fomos maioritariamente imprimindo a informação no ecrã para confirmar a sua veracidade, fomos também analisando casos mais específicos ou situações por ventura mais alarmantes como uso duplicado de chavetas, parênteses, virgulas ou aspas. Fomos também a medida que desenvolvemos as expressões regulares utilizando o regex101, como forma mais eficaz de identificar possíveis erros ou imperfeições nas nossas **ER**

Assim sendo passando a fase de implementação e respetivos testes estes foram os resultados que obtivemos, utilizando o ***exemplo-utf8.bib*** disponibilizado pelos docentes.

5.2.1 Problema a

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Relatório</title>
        <meta charset="UTF-8"/>
    </head>
    <body>
        <h1>Categorias</h1>
        <ol>
            <li>article: 33</li>
            <li>book: 1</li>
            <li>incollection: 5</li>
            <li>inproceedings: 112</li>
            <li>mastersthesis: 1</li>
            <li>misc: 1</li>
            <li>phdthesis: 1</li>
            <li>techreport: 11</li>
        </ol>
    </body>
</html>
```

Resultados obtido alínea a

5.2.2 Problema b

```
A. M. Simões:  ['speaker:sepln2001']
Alberto M. Simões:  ['freq2002', 'jspell2002', 'dag2002', 'elpub2002', 'parguess2002', 'sepln2003']
Alberto Manuel Brandão Simões:  ['teseambs']
Alberto Manuel Simões:  ['APL2k2.Parguess', 'APL2k2.Synthesis', 'xata:xmldt', 'xata:museudapessoa', 'elpub2003']
Alberto Simões:  ['elpub06-blind', 'avalon:jspell', 'avalon:avalinha', 'xata07:xmltmx', 'MP07', 'epia-music-2007']
Alda Gancarski:  ['GDH06', 'GH07b', 'GH07a', 'GFH08']
Alda Lopes Gancarski:  ['FGH08', 'FCHGD09a', 'FCHGD09b']
Alexandre Carvalho:  ['elpub06-blind']
Ana Silva:  ['Gis99', 'RSea99']
Anne Doucet:  ['GDH06']
António R. Fernandes:  ['elpub06-blind']
Anália Lourenço:  ['epia-music-2007']
Bastian Cramer:  ['OPHCC2010']
Bruno Defude:  ['FCHGD09a', 'FCHGD09b']
Bruno Martins:  ['cp3a:terminum2003', 'cp3a:kvec2003']
```

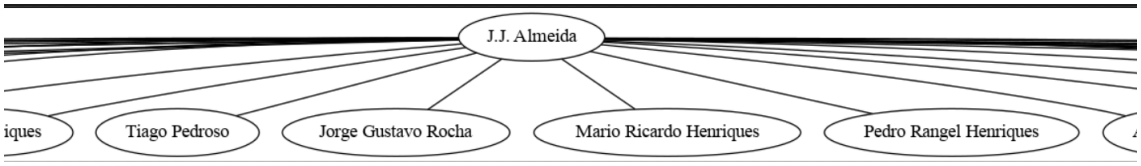
Resultados obtido alínea b

5.2.3 Problema c


```
"referencias":[
{
    "tipo":"inproceedings",
    "codigo":"graminteractivas1990",
    "author":"F. M�rio Martins and J.J. Almeida and P.R. Henriques",
    "title":"Mecanismos para Especifica��o e Prototipagem de Interfaces
Utilizador-Sistema",
    "note":"(Gram�ticas Interactivas guardadas)",
    "booktitle":"3$�$ Encontro Portugu�s de Computa��o Gr�fica",
    "address":"Coimbra",
    "year":"1990",
},
{
    "tipo":"techreport",
    "codigo":"tlc89",
    "author":"J.J. Almeida and J.B. Barros",
    "title":"Teoria das Linguagens ",
    "institution":"umdi",
    "type":"Texto did�ctico",
    "year":"1988",
    "keyword":"compiladores",
}
{
    "tipo":"techreport",
    "codigo":"estruturasdedados90",
    "author":"J.B. Barros and J.J. Almeida",
    "title":"Estruturas de Dados",
    "institution":"umdi",
    "type":"Texto did�ctico",
    "year":"1990",
    "keyword":"algoritmos",
}
```


Resultados obtido alínea c

5.2.4 Problema d



Resultados obtido alínea d

6 Conclusão

Este projeto permitiu-nos consolidar a matéria lecionada nas aulas relativamente e expressões regulares e constatamos o seu poder expressivo e versatilidade. Numa primeira fase tratou-se da construção das expressões para filtrar e armazenar a informações necessárias tendo estes sido um pouco complicados ja que nem todos os blocos de entradas possuiam a mesma estrutura. Além disso tivemos especial atenção aos casos de nome de autores repetidos mas escritos em formato diferente sendo necessário normalizar todos os nomes para evitar entradas repetidas nos dicionários. Por fim apresentamos as soluções conforme era pedido em cada alínea.

Em suma destaca-se a importância da escolha das estruturas e a forma de ler o ficheiro pois influenciam diretamente a eficácia do programa e ainda as expressões regulares pois são estas que manipulam toda a informação do ficheiro.

7 Anexo de Código

7.1 Problema A

```
import re

filepath = input("Insira o filepath\n")

try:
        f = open(filepath, "r", encoding='utf-8')

        categoria={}

        for linha in f:

          campos = re.findall(r'\@(\w+)',linha)
          for c in campos:
                        caux=c.lower()
                        if caux in categoria:
                                categoria[caux]+=1
                        else:
                                categoria[caux]=1


        html = open("a.html","w")
        html.write('''<!DOCTYPE html>
<html>
    <head>
        <title>Relatório</title>
        <meta charset="UTF-8"/>
    </head>
    <body>
            <h1>Categorias</h1>
            <ol>
''')

        for i in sorted(categoria):
                html.write("                    <li>" +i + ": "+
                str(categoria[i]) + "</li>\n")

        html.write('''          </ol>
        </body>
</html>\n''')

except Exception as e:
    print(e)
```

7.2 Problema B

```
import re

filepath = input("Insira o filepath\n")

try:
    f = open(filepath, "r", encoding='utf-8')

    categoria=[]
    autoresOcorr={}

    #HOW TO SPLIt
    l = f.read()
    categoria = re.split('@',l)

    #separa dentro de cada categoria a chave
    #única e coloca em campo e os autores separa tmb
    for i in categoria:
        campo = re.match(r'(\w+\{([^,]+))', i)

        if campo:
            if autor := re.search(r'\b(?i:author) *=

            *\"([^"]*)\",?|\b(?i:author) *= *\{(.*)\},?',i):

                lsa=autor.group(1)
                if not(lsa):
                    lsa=autor.group(2)

                lsa=re.sub(r'( |\n)+', r' ', lsa)
                autores = re.split(r' +and +',lsa)


                for aut in autores:
                    if aux:=re.search(r'([\w\. ]+), ([\w\. ]+)',aut):
                        aut = aux.group(2) +' ' +aux.group(1)


                    if aut in autoresOcorr:
                        autoresOcorr[aut.strip()].append(campo[2])
                    else:
                        autoresOcorr[aut.strip()]=[campo[2]]



    for i in sorted(autoresOcorr):
        print(i + ": ",autoresOcorr[i])
```

```
        print()

except Exception as e:
    print(e)
```

7.3 Problema C

```
import re

filepath = input("Insira o filepath\n")
#print(filepath)
try:
        f = open(filepath, "r", encoding='utf-8')

        json = open("c.json","w")

        categoria=[]
        autores = []



        #HOW TO SPLI
        l = f.read()
        categoria = re.split('@',l)

        allc=".\n"

        json.write("{\n\t\"referencias\":[")
        for i in categoria:

                tipo = re.match(r'(\w+)\{([^,]+)',i)

                if tipo:
                        g=re.split('{',tipo.group())

                        json.write("\n\t{\n\t\t\"tipo\":\""+g[0]+"\",")


                        json.write("\n\t\t\"codigo\":\""+g[1]+"\",")

                        info = re.findall(r'\b(\w+) *= *(\d*),|\b(\w+) *=
                        *\"([^"]*)\",?|\b(\w+) *=
                        *\{([^\}]*(?:\{[^\}]*\})+[^\}]*)\},?|\b(\w+) *=
                        *\{([^\}]*[^\}]*)\},?',i)

                        for (c,v,c1,v1,c2,v2,c3,v3) in info:
                                print(c,": ",v, " || ",c1,": ",v1," || ",c2,": ",v2,
                                ",v3)
                                print()
                                if c:
                                        json.write("\n\t\t\"" + c +"\":\""+v+"\",")
                                elif c1:
                                        if re.search(r'^\d*$',v1):
```

```
                                        json.write("\n\t\t\"" + c1 +"\":"+v1
                                    else:
                                        json.write("\n\t\t\"" + c1 +"\":\""+
                                elif c2:
                                    if re.search(r'^\d*$',v2):
                                        json.write("\n\t\t\"" + c2 +"\":"+v2
                                    else:
                                        json.write("\n\t\t\"" + c2 +"\":\""+
                                elif c3:
                                    json.write("\n\t\t\"" + c3 +"\":\""+v3+"\","
                                else:
                                    pass

                            json.write("\n\t}")


                json.write("\n\t]\n}\n")

        except Exception as e:
            print(e)
```

7.4 Problema D

```
import re
from graphviz import Graph


filepath = input("Insira o filepath\n")

try:
    f = open(filepath, "r", encoding='utf-8')
    ex_aut = input("Insira o nome do autor a gerar o gráfio\n")

    categoria=[]
    autoresRelat=[]

    #HOW TO SPLI
    l = f.read()
    categoria = re.split('@',l)


    #separa dentro de cada categoria a chave única e coloca em campo e os
    autores separa tmb
    for i in categoria:

        if autor := re.search(r'\b(?i:author) *= *\"([^"]*)\",?|\b(?i:author) *
        *\{(.*)\},?',i):

            lsa=autor.group(1)
            if not(lsa):
                lsa=autor.group(2)


            lsa=re.sub(r'( |\n)+', r' ', lsa)
            autores = re.split(r' +and +',lsa)


            if autores.__contains__(ex_aut):

                for aut in autores:
                    if aux:=re.search(r'([\w\. ]+), ([\w\. ]+)',aut):
                        aut = aux.group(2) +' ' +aux.group(1)

                    if not aut.__eq__(ex_aut) and aut not in autoresRelat:
                        autoresRelat.append(aut)


    g = Graph(format='png')
    g.node(str(0), ex_aut)
```

```
        k=1

        for i in autoresRelat:
            g.node(str(k), i)
            g.edge(str(0), str(k), constraint='true')
            k+=1

        g.render('graphviz/doutput', view=True)

        print(autoresRelat)

    except Exception as e:
        print(e)
```