Processamento de Linguagens (3º ano MIEI)

Trabalho Prático 1

Relatório de Desenvolvimento

Carlos Preto
(A89587)

Pedro Veloso
(A89557)

3 de abril de 2021

Resumo

O trabalho prático tem como principal objetivo a aplicação de conhecimentos lecionados nas aulas de Processamento de Linguagens, bem como aumentar a capacidade de escrita de expressões regulares. Através destas expressões é possível desenvolver Processadores de Linguagens Regulares.

O problema atribuído ao grupo foi o problema 3, tendo o grupo optado por resolver adicionalmente o problema 4. Ao longo do relatório serão discutidas as diferentes fases associadas à resolução das alíneas de cada um desses mesmos problemas.

Conteúdo

1 Introdução **2**

2 Problema 3: Análise e Especificação **3**
2.1 Descrição informal do problema . . . 3
2.2 Especificação dos Requisitos . . . 3
2.2.1 Dados . . . 3
2.2.2 Pedidos . . . 3

3 Problema 3: Concepção/desenho da Resolução **4**
3.1 Estruturas de Dados . . . 4
3.2 Algoritmos . . . 5

4 Problema 3: Codificação e Testes **13**
4.1 Alternativas, Decisões e Problemas de Implementação . . . 13
4.2 Testes realizados e Resultados . . . 14

5 Problema 4: Análise e Especificação **23**
5.1 Descrição informal do problema . . . 23
5.2 Especificação dos Requisitos . . . 23
5.2.1 Dados . . . 23
5.2.2 Pedidos . . . 23

6 Problema 4: Concepção/desenho da Resolução **24**
6.1 Estruturas de Dados . . . 24
6.2 Algoritmos . . . 24

7 Problema 4: Codificação e Testes **26**
7.1 Alternativas, Decisões e Problemas de Implementação . . . 26
7.2 Testes realizados e Resultados . . . 26

8 Conclusão **29**

A Código do Programa 3 **30**

B Código do Programa 4 **40**

Capítulo 1

Introdução

De maneira a saber qual o Problema a realizar, foi necessário aplicar a fórmula (27 % 5) + 1, sendo o resultado obtido 3, logo o grupo terá de realizar o Problema 3.
O tema deste problema será referente a um processador de BibTex, onde é fornecido um ficheiro BIB denominado 'exemplo-utf8.bib'. Com a ajuda de ficheiros fornecidos pelos docentes, foi possível aprender um pouco mais em relação a essa formatação.
O problema encontra-se divido em quatro alíneas, a primeira pede para calcular o número de entradas por categoria, apresentado os resultados num ficheiro HTML por ordem alfabética, na segunda alínea é necessário criar um índice de autores que mapeie cada autor nos respetivos registos identificados pela respetiva chave de citação. A terceira alínea pede que se transforme cada registo num documento JSON válido, e por fim, na quarta alínea, é necessário produzir um grafo para um dado autor introduzido pelo utilizador, onde indicará os autores que já publicaram com este. De maneira a produzir esse grafo, é necessário criar um ficheiro DOT com a respetiva informação.
O grupo optou também por realizar o Problema 4, por motivos de curiosidade e de consolidaçáo de conhecimentos.
Com este relatório o grupo espera conseguir explicar todo o raciocínio e trabalho por detrás de cada uma das alíneas, da forma mais correta e simples possível, de modo a facilitar a compreensão da resolução de cada uma das alíneas.

Estrutura do Relatório

O relatório estará dividido em 8 capítulos. O capítulo 1 será referente à Introdução do relatório, onde se abordarão tópicos como o enquadramento e contexto do tema proposto, bem como os diferentes objetivos e resultados obtidos. O capítulo 2 será referente à Análise e Especificação do Problema 3, abordando tópicos como Descrição informal do problema e especificação dos requisitos, isto é, dos dados e os pedidos para o problema em questão. Por sua vez, o capítulo 3 falará sobre a Concepção/desenho da Resolução, onde o grupo irá identificar as estruturas de dados usadas, bem como os algoritmos de cada alínea. No capítulo 4 aborda-se o tópico de Codificação e Testes do Problema 3, onde serão apresentadas as alternativas, decisões e problemas de implementação, e ainda exemplos de testes e respetivos resultados realizados que contribuíram para o sucesso da resolução de cada alínea.
Uma vez que o grupo optou por, adicionalmente, realizar o Problema 4, os capítulos 5, 6 e 7 serão semelhantes aos capítulos 2, 3 e 4, só que serão agora referentes ao Problema 4. Por fim é apresentada uma breve Conclusão, onde se discute os resultados obtidos e o trabalho futuro, sendo que, nos Apêndices A e B, é apresentado o código completo de cada uma das alíneas dos problemas 3 e 4, respetivamente.

Capítulo 2

Problema 3: Análise e Especificação

2.1 Descrição informal do problema

Com a resolução deste problema pretende-se, com a utilização de expressões regulares, que seja possível encontrar valores para certos campos de um ficheiro BibTeX. Um ficheiro BibTeX, permite que se façam, no meio de um documento LATEX, citações bibliográficas a outros documentos públicos que estejam caracterizados e classificados numa determinada BD escrita nesse formato. Para tal, existem cerca de 20 categorias de referências, sendo que para categoria existem campos, sendo alguns obrigatórios e outros opcionais.

2.2 Especificação dos Requisitos

2.2.1 Dados

Para a resolução deste problema é fornecido um ficheiro no formato bib, contendo uma ou mais referência. Sendo um ficheiro válido considera-se que todas as referências pertencem a uma das 20 categorias existentes, iniciadas pelo caractere '@' e que nessa mesma linha consta o nome único da referência.
Referente aos campos, para o documento ser válido, deve-se iniciar os seus valores com (") ou com {, seguido do valor, voltando a fechar com a aspa ou chaveta correspondente. Contudo, como a conversão do BibTex para Latex transforma as palavras no interior de uma frase em minúscula, caso se pretenda manter a maiúscula deve-se colocar a palavra entre chavetas. Existem também 2 campos que podem ter múltiplos valores, sendo eles o campo referente aos autores e aos editores, estando os vários nomes separados pela palavra "and". Nesses campos caso se pretenda indicar apenas um valor utiliza-se o padrão {{valor}}.

2.2.2 Pedidos

Neste problema é pedido que se contabilize o número de vezes que uma determinada categoria é utilizada num ficheiro BibTex, apresentando o resultado por ordem alfabética e num ficheiro html (alínea a). Pede-se também, que se apresente o nome de todos os autores, associado ao nome único da referência (alínea b). Na alínea c, é pedido que se converta um documento com o formato BibTex em um ficheiro com o formato json. Por fim na alínea d, é pedido que se construa um grafo onde se associe um dado autor a outros autores com quem publica normalmente.

Capítulo 3

Problema 3: Concepção/desenho da Resolução

Para a realização deste problema, construiu-se um programa em python para cada uma das alíneas. Assim será realizada uma analise, em separado, para cada programa.

3.1 Estruturas de Dados

Alínea A: Através do ficheiro 'biblayout.pdf' fornecido pelos docentes, é possível saber quais as diferentes categorias de referência presentes num ficheiro BD Bibtex. Assim, utilizou-se um dicionário como estrutura de dados, onde cada chave será o nome da categoria e o valor correspondente a essa chave será será o número de ocorrências dessa categoria no nosso ficheiro 'exemplo-utf8.bib'.
O dicionário terá portanto 14 chaves, tendo cada uma um valor inicial de 0. No final de se ler o ficheiro, é necessário criar um ficheiro HTML, onde será escrita toda a informação adquirida.

Alínea B e C: Para a realização destas alíneas utiliza-se dicionários, definidos no python, como estrutura de dados. Assim cada referência dá entrada no primeiro dicionário, onde a chave será o nome único da referência e o valor será um outro dicionário. Neste segundo dicionário a chave será o nome dos campos da referência e valor será o valor desse mesmo campo. Com esta estrutura de dados é possível agregar o valor de cada campo, que no ficheiro podem estar em diferentes, em uma única string o que facilita o seu posterior processamento.
A escolha de dicionários deve-se ao facto, de que a sua utilização facilita a associação de cada linha que se encontra sem o nome do campo, para o seu respetivo campo, pois basta ter um apontador para a posição do último campo encontrado.
Na alínea b, ao contrário, da aliena c, regista-se, no segundo, dicionário apenas os valores do campo autor. Já na alínea c, regista-se os valores de todos os campos.

Alínea D: Para a resolução da alínea D, optou-se novamente por utilizar o dicionário como estrutura de dados. No dicionário a chave corresponde a cada autor que já publicou com um autor (definido pelo utilizador), e o valor correspondente a essa chave será o número de vezes que um autor publicou com o autor definido pelo utilizador. Inicialmente o dicionário estará vazio, sendo preenchido à medida que se for lendo o ficheiro.
No final de se ler o ficheiro, são criadas duas cópias do dicionário original ordenado por ordem alfabética dos autores, sendo estes usados para se ir iterando no dicionário. Por fim, é nessário criar um ficheiro DOT,

onde será escrita toda a informação adquirida, e de maneira a que, posteriormente, se possa usar uma das ferramentas que processam DOT 8 para desenhar o dito grafo de associações de autores.

3.2 Algoritmos

Alínea A: Inicialmente, o ficheiro 'exemplo-utf8.bib' é aberto de maneira a se poder aceder à sua informação. Para isso, é realizada a leitura do ficheiro linha a linha, onde conforme o conteúdo dessa mesma linha será realizada alguma ação.
Sabe-se que a linha contém uma determinada categoria, caso se encontre o símbolo '@', seguido de mais que um letra, podendo esta ser maiúscula ou minúscula.

Listing 3.1: Algoritmo usado para verificar a possibilidade da existência de uma categoria

```
1       categoriaMatch = re.search(r'@[a-zA-Z]+', line)
```

Caso a condição se verifique, é necessário obter a secção da linha que obteve match com a nossa expressão regular, sendo essa secção a categoria encontrada. Caso contrário, ter-se-á de continuar para a próxima.

Listing 3.2: Algoritmo usado para obter a categoria encontrada

```
1       categoria = categoriaMatch.group(0).lower()
```

Após encontrada a categoria, percorre-se o nosso dicionário, e caso se encontre a chave correspondente a essa mesma categoria, é necessário verificar se a chave tem o mesmo comprimento que a categoria encontrada. Tal é necessário, porque, por exemplo, quando se encontrasse a categoria '@book', esta iria dar match quer com '@book', quer com '@booklet', o que estaria, efetivamente, incorreto.
Caso cumpra todos os requisitos, é incrementado o valor dessa chave, caso contrário, continua-se a iterar pelo dicionário.

Listing 3.3: Algoritmo usado para encontrar e incrementar categoria correspondente no dicionário

```
1       for key, value in categorias.items():
2         sameCategoria = re.search(r'(?i:' + categoria + r')', key)
3         if sameCategoria and len(key) == len(categoria):
4                 categorias[key] += 1
```

No final de se obter todas as categorias encontradas, é necessário escrever a informação no nosso dicionário para um ficheiro HTML. Para tal, é criado um ficheiro entitulado 'alineaA.html', onde posteriormente se escreve o código HTML necessário, bem como listar cada um dos items no nosso dicionário.

Listing 3.4: Algoritmo usado para gerar ficherio HTML

```
1       html = open("alineaA.html", "w")
2
3       html.write('<!DOCTYPE html>')
4       html.write('<html>')
5       html.write('<head>')
6       html.write('<meta charset=\"utf-8\"')
7       html.write('</head>')
8       html.write('<body>')
9       html.write('<h1>Entradas por Categoria</h1>')
10      html.write('</body>')
11      html.write('<ul>')
12
13      for key, value in sorted(categorias.items()):
14        html.write('<li>')
15        html.write(key + ': ' + str(value))
16        html.write('</li>')
17
18      html.write('</body>')
19      html.write('</html>')
20
21      html.close()
```

Alínea B: A resolução desta alínea é dividida em duas fases, a fase de leitura dos dados e a fase de processamento dos dados recolhidos.
Na primeira fase, é realizada a leitura do ficheiro, linha a linha, armazenado em memória os dados referentes aos autores de cada referência.
Posteriormente, na segunda fase é realizado o tratamento desses dados. Uma vez que uma referência pode ter vários autores é necessário, realizar um split do conteúdo. Contudo, como também é possível que o campo autor seja constituído apenas por um valor (que contenha a palavra "and"), antes de realizar o split é necessário testar a existência de uma lista de autores.

Listing 3.5: Algoritmo usado para verificar a possibilidade da existência de uma lista de autores

```
1       if(not re.match(r'^{{',v.strip())):
2         lista = True
```

Após esta verificação procede-se há remoção de caracteres desnecessários, como por exemplo as aspas ou chavetas.

Listing 3.6: Algoritmo de remoção de caracteres

```
1       frase = re.sub(r'{',r'',v.strip())
2       frase = re.sub(r',$',r'',frase.strip())
3       frase = re.sub(r'}',r'',frase.strip())
4       frase = re.sub(r'"',r'',frase.strip())
5       frase = re.sub(r'\\',r'',frase.strip())
```

Por fim é escrito o resultado, associando os autores ao nome da referência.

Alínea C: Similar à alínea B, para a resolução desta problema realiza-se a leitura do ficheiro, armazenando em memória o seu conteúdo.
Posteriormente, percorre-se a estrutura de dados analisando os dados e escrevendo o ficheiro json correspondente. Ao analisar os campos e os seus valores, começa-se por perceber se o campo é um autor ou um editor, uma vez que estes campos admitem listas como valor. Caso o campo analisado seja um autor ou um editor, é ainda executado um algoritmo que verifica se pode existir mais de um valor.

Listing 3.7: Análise de múltiplos valores

```
1     lista = False
2     if((s == 'author' or s == 'editor') and not re.match(r'^{{',v.strip())):
3             lista = True
```

Após a verificação é eliminado do valor do campo caracteres como chavetas, aspas, barras, virgulas e ainda comandos referentes ao latex, desnecessários em ficheiros json.

Listing 3.8: Eliminação dos caracteres

```
1     frase = re.sub(r'\\[^{]+{',r'',v.strip())
2     frase = re.sub(r'{',r'',frase.strip())
3     frase = re.sub(r',$',r'',frase.strip())
4     frase = re.sub(r'}',r'',frase.strip())
5     frase = re.sub(r'"',r'',frase.strip())
6     frase = re.sub(r'\\',r'',frase.strip())
```

Tendo o conteúdo do campo limpo, é ainda necessário verificar se é um valor numérico. Caso estejamos na presença de tais valores é necessário tratar, também o seu conteúdo, uma vez que valores do tipo 0001 ou 0000 são inválidos em json.

Listing 3.9: Tratamento de valores numéricos

```
1     number = False
2         if(re.search(r'^[0-9]+$',frase.strip())):
3             frase = re.sub(r'^0+$',r'0',frase.strip())
4             frase = re.sub(r'^0+([1-9]+)',r'\1',frase.strip())
5             number = True
```

Por fim, realiza-se a projeção do resultado, tendo em conta a diferença usada em listas e valores numéricos, da normal representação de strings no formato json.

Alínea D: Através dos vários exemplos encontrados no ficheiro 'exemplo-utf8.bib', é possível perceber que o nome de um autor pode vir em diferentes formatos. Consideremos o exemplo do autor 'Pedro Rangel Henriques'. Este autor pode aparecer no ficheiro como 'P. Rangel Henriques', 'P. R. Henriques', 'Pedro R. Henriques', 'P. Henriques', 'Pedro Henriques' e 'Rangel Henriques, Pedro'. Esta diversidade de formatos torna a resolução desta alínea bastante complexa, pelo que, para melhor se explicar a sua resolução, divide-se a explicação em diferentes fases.
A primeira fase corresponde à normalização dos autores. De maneira a normalizar os nomes foi criada um função que substitui toda a acentuação de uma palavra, bem como os todos os símbolos excepto o traço, o

ponto e a vírgula. Assim, por exemplo, o nome 'Mário Be?!rón' passa a ser 'Mario Beron', o que posteriormente beneficiará na comparação de nomes, pois já não nos teremos de preocupar com a acentuação e erros das palavras.

Listing 3.10: Normalização

```
1
2 def normal(name):
3     name = re.sub(u"[àáâãäå]", 'a', name)
4     name = re.sub(u"[èéêë]", 'e', name)
5     name = re.sub(u"[ìíîï]", 'i', name)
6     name = re.sub(u"[òóôõö]", 'o', name)
7     name = re.sub(u"[ùúûü]", 'u', name)
8     name = re.sub(u"[ýÿ]", 'y', name)
9     name = re.sub(u"[ß]", 'ss', name)
10    name = re.sub(u"[ñ]", 'n', name)
11    name = re.sub(r'[^\w' + r' \-\.\,' + r']', '', name)
12    return name
```

A segunda fase será referente ao input do utilizador, onde caso o seu input seja nulo, ou contenha apenas símbolos, será necessário que o utilizador volte a inserir o seu autor pretendido, até que este tenha um formato válido. No final, normaliza-se o nome inserido pelo utilizador.

Listing 3.11: Tratamento input utilizador

```
1 nomeInput = input('Insira o nome do Autor: ')
2
3 while not re.search(r'[a-zA-Z]+', nomeInput):
4     print('Nome Inválido!')
5     nomeInput = input('Insira o nome do Autor: ')
6
7 nome = normal(nomeInput)
```

A terceira fase corresponde à identificação dos autores no ficheiro, sendo necessário percorrer linha a linha o nosso ficheiro. Inicialmente, é necessário perceber se estamos numa linha correspondente aos autores.

Listing 3.12: Identificação linha Autores

```
1     authorLine = re.search(r'author *={*', line.strip())
```

Se não estivermos numa linha de autores avançamos para a próxima linha, porém, se estivermos, será necessário saber onde terminam os autores, pois pode haver casos onde se tem mais que uma linha de autores. Para isso, cria-se uma flag com o valor 'False', e vai-se avançando nas linhas até encontrar o fim dos autores. Quando se encontrar o final, coloca-se a flag a 'True', para assim se poder parar de iterar nas linhas. Sempre que não se encontrar o fim da linha dos autores, vai-se concatenando as linhas, para assim no final obtermos uma única linha com todos os autores presentes numa determinada categoria.

Listing 3.13: Concatenar Linhas

```
1     else:
2         nextLine = next(bib)
3         line = line + nextLine
```

Quando for encontrado o fim, a linha é tratada, de maneira a remover as mudanças de linha, os duplos espaços seguidos no meio de palavras e vários símbolos que não sejam necessários. Após tal procedimento, faz-se 'split' da nossa linha pela palavra 'and', para assim obtermos uma lista com todos os nomes.

Listing 3.14: Tratamento da linha

```
1     deleteEnter = re.sub(r'\n', ' ', line.strip())
2     deleteEndingLine = re.sub(r'\"*\{*\}*,$', '', deleteEnter.strip())
3     deleteBeginningLine = re.sub(r' *author* *= *{*\"* *', '', deleteEndingLine)
4     fixedLine = re.sub(r' +', ' ', deleteBeginningLine)
5
6     diferentAuthors = fixedLine.split(' and')
```

De seguida, cria-se uma lista nova, onde iremos colocar todos os nomes sem os espaços iniciais e finais, bem como colocar os nomes onde o primeiro e último nome vêem trocados no formato correto.

Listing 3.15: Tratamento dos nomes

```
1     authors = []
2
3     for author in diferentAuthors:
4         normalAuthor = normal(author)
5         newAuthor = normalAuthor.strip()
6
7         if ',' in newAuthor:
8
9             duplo = newAuthor.split(',')
10            firstName = duplo[1].strip()
11            lastName = duplo[0].strip()
12
13            corretAuthor = firstName + ' ' + lastName
14            authors.append(corretAuthor)
15
16        else:
17            authors.append(newAuthor)
```

Após estes passos, é altura de verificar se o autor pretendido pelo utilizador se encontra nessa lista. Para isso, divide-se o nome do autor pretendido pelo ' ', e vai-se percorrer cada autor na lista de autores. A expressão que nos diz se o autor está nessa linha é construída à custa dos diferentes nomes do autor do utilizador. Por exemplo, se o nome pretendido pelo utilizador for 'Pedro Rangel Henriques', a nossa expressão irá procurar por '(?i:P)(.?)(edro)?(?)(?i:R)?(.?)(angel)?(?)(?i:Henriques)$'. Esta expressão obriga a que a primeira letra do autor seja a mesma, e que o último nome do autor esteja sempre presente, sendo que campos como o nome do meio e resto do primeiro nome foram definidos como opcionais. Assim, a expressão dará match com todos os tipos de formatos possíveis para o nome de um autor.

Listing 3.16: Construção expressão regular

```
normalizedName = normal(name)

expression = r'^(?i:' + dividedName[0][0] + r').*(' + dividedName[0][1:] + r')?( ?)'
lastWord = len(dividedName) - 1
counter = 1

while counter <= lastWord:
    if counter<lastWord:
        expression += r'(?i:' + dividedName[counter][0] + r')?(.?)(' + dividedName[
            counter][1:] + r')?( ?)'
        counter+=1
    else:
        expression += r'(?i:' + dividedName[counter] + r')$'
        break

sameAuthor = re.search(expression, normalizedName)
```

Caso o autor esteja nessa lista, então iremos colocar os restantes autores dessa lista no nosso dicionário. Caso já esteja, o seu valor no dicionário é incrementado.

Listing 3.17: Adição de autores ao dicionário

```
if sameAuthor:
    for author in authors:
        normalizedAuthor = normal(author)

        expression = r'^(?i:' + dividedName[0][0] + r').*(' + dividedName[0][1:] + r
            ')?( ?)'
        lastWord = len(dividedName) - 1
        counter = 1

        while counter <= lastWord:
            if counter<lastWord:
                expression += r'(?i:' + dividedName[counter][0] + r')?(.?)(' +
                    dividedName[counter][1:] + r')?( ?)'
                counter+=1
            else:
                expression += r'(?i:' + dividedName[counter] + r')$'
                break
        isOurAuthor = re.search(expression, normalizedAuthor)

        if isOurAuthor:
            pass
        else:
            if normalizedAuthor in colabs:
                colabs[normalizedAuthor] += 1
            else:
                colabs[normalizedAuthor] = 1
    break
else:
    pass
```

Após ter os nomes todos no dicionário, e já tendo percorrido todas as linhas do ficheiro, trata-se de remover os valores vazios que lá se encontrem, pois estes não simbolizam nenhum autor.

Listing 3.18: Eliminação de valores nulos

```
    for key, value in list(colabs.items()):
        if key == '':
            colabs.pop(key, None)
```

Após estes procedimentos, temos no dicionário todos os autores que já publicaram com o autor introduzido pelo autor. Porém, neste dicionário, os autores 'J.C. Almeida' e 'João Carlos Almeida', são considerados autores diferentes, o que não corresponde à realidade. Para resolver esse problema para todos os autores, será necessário iterar pelo nosso dicionário, por isso, criamos duas cópias do nosso dicionário, ordenados por ordem alfabética.

Listing 3.19: Criação de duas cópias do dicionário original

```
    orderedColabs = sorted(colabs.items())
    loopedColabs = sorted(colabs.items())
```

Com estes 2 dicionários, vamos percorrer cada elemento do 'orderedColabs', constrói-se uma expressão semelhante à referida acima para o autor, sendo que depois se vai ver cada elemento do 'loopedColabs' e verificar se os dois nomes dão match. O critério definido para dar match foi ter, obrigatoriamnete, a mesma letra inicial e o nome final idêntico e completo, sendo que campos como o nome do meio e resto do primeiro nome foram definidos como opcionais. Caso deem match, definiou-se que seria a maior palavra que ficaria com os valores da mais pequena, sendo depois a mais pequena colocada com o valor -1, para posteriormente saber-se que é para eliminar.

Listing 3.20: Unir nomes idênticos

```
if match:
    div2 = key2.split(' ')

    if len(div1) > len(div2) and colabs[key2] > 0 and colabs[key1] > 0:
        colabs[key1] += colabs[key2]
        colabs[key2] -= colabs[key2] + 1
    else:
        if len(div1) == len(div2):
            if len(key1) > len(key2) and colabs[key2] > 0 and colabs[key1] > 0:
                colabs[key1] += colabs[key2]
                colabs[key2] -= colabs[key2] + 1
            else:
                if len(key1) < len(key2) and colabs[key1] > 0 and colabs[key2] > 0:
                    colabs[key2] += colabs[key1]
                    colabs[key1] -= colabs[key1] + 1
                else:
                    if (key1 != key2) and colabs[key2] > 0 and colabs[key1] > 0:
                        colabs[key1] += colabs[key2]
                        colabs[key2] -= colabs[key2] + 1
        else:
            if len(div1) < len(div2) and colabs[key1] > 0 and colabs[key2] > 0:
                colabs[key2] += colabs[key1]
                colabs[key1] -= colabs[key1] + 1
```

No fim, trata-se de eliminar os elementos do dicionário que tenham valor -1 (significa que se fundiram com outra palavra), e depois cria-se um ficheiro DOT com no nome do autor pretendido pelo utilizador, e onde se escreve toda a informação presente no dicionário principal.

Listing 3.21: Eliminar nomes que fundiram e criar ficheiro DOT

```
1 for key, value in sorted(colabs.items()):
2     if value == -1:
3         colabs.pop(key, value)
4
5 dotFile = open(nome + '.dot', "w")
6 dotFile.write("digraph G {")
7
8 for key, value in sorted(colabs.items()):
9     dotFile.write('\"' + nome + '\" -> \"' + key + '\"[label=\"' + str(value) +
          '\"];')
10
11 dotFile.write("}")
12 dotFile.close()
```

Capítulo 4

Problema 3: Codificação e Testes

4.1 Alternativas, Decisões e Problemas de Implementação

Alínea A: No geral a alínea A não implicou problemas de Implementação. Inicialmente, começou-se por criar uma expressão regular para cada uma das categorias, porém, rapidamente se verificou que essas expressões poderiam ser resumidas a uma única expressão que iria apanhar todas as categorias presentes. Após tomada essa decisão, o código ficou mais simples e eficiente.
A escrita do ficheiro HTML é feita através da criação de um ficheiro e escrita nesse mesmo, pois considera-se ser a opção mais simples e eficaz.

Alínea B e C: Para a realização destas alíneas, primeiramente tentou-se criar o resultado à medida que se avançava no ficheiro, contudo devido às diferentes possibilidades (o campo e o seu respetivo valor encontram-se na mesma linha, ou na linha encontra-se apenas o nome do campo, existindo ainda o caso onde o valor do campo encontra-se em várias linhas) tornou-se complicado desenvolver uma estratégia que permitisse tal abordagem.
Posteriormente, desenvolveu-se um algoritmo capaz de armazenar a informação recorrendo apenas a um dicionário. Contudo nessa abordagem, uma vez que se obtinha todos os campos e respetivos valores numa única string, seria necessário realizar um split, capaz de dividir os campos e seus respetivos valores. Primeiramente utilizou-se o símbolo '=' na realização do split, o que se tornou falível, uma vez que existia valores de campos que utilizavam esse símbolo.
Assim, partindo do algoritmo já criado obteve-se a solução final, que com o uso de dois dicionários, consegue realizar a divisão de uma forma natural, à medida que lê o documento.

Alínea D: A alínea D trouxe bastantes problemas de implementação e antes da sua resolução final foram testadas diferentes alternativas. O grupo tentou pensar em todos os casos possíveis e implementar mecanismos para controlar tais casos. O facto do input do utilizador ser uma incógnita, ou seja, este poder escrever o que lhe apetecer faz com que se tenham de tomar decisões de controlo e normalização.
Para além do input do utilizador, e talvez o maior desafio da alínea, é importante saber que um autor pode ser escrito de diversas formas, sendo por isso necessário criar expressões regulares que apanhem esses casos todos, sendo que essa expressão varia conforme o tamanho do nome do autor.
Posteriormente, quando se insere um autor no dicionário, há a hipótese de esse mesmo já existir no dicionário mas com um nome diferente. Nesses casos, fica-se com o nome de maior comprimento e elimina-se o de menor, passando os valores do menor para o maior.
Através destas decisões, os resultados obtidos serão os mais completos e compactos possíveis.

4.2 Testes realizados e Resultados

Mostram-se a seguir alguns testes feitos (valores introduzidos) e os respectivos resultados obtidos, para cada alínea do problema:

Alínea A A alínea A apenas necessitou de um único teste, onde se procurou no ficheiro 'exemplo-utf8.bib' todas as ocorrências das diferentes categorias.
Observou-se que estão presentes 165 entradas de categorias no ficheiro, tal como pode ser verificado pelo somatório de todas as entradas de cada categoria no ficheiro HTML produzido, sendo que o número de entradas de cada categoria também foi posteriormente confirmado.

```
OUTPUT:
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"
</head>
<body>
    <h1>Entradas por Categoria</h1>
</body>
    <ul>
    <li>@article: 33</li>
    <li>@book: 1</li>
    <li>@booklet: 0</li>
    <li>@inBook: 0</li>
    <li>@inCollection: 5</li>
    <li>@inProceedings: 112</li>
    <li>@manual: 0</li>
    <li>@mastersthesis: 1</li>
    <li>@misc: 1</li>
    <li>@online: 0</li>
    <li>@phdthesis: 1</li>
    <li>@proceedings: 0</li>
    <li>@techreport: 11</li>
    <li>@unpublished: 0</li>
    </body>
</html>
```

Alínea B

Teste 1: Com este teste pretende-se avaliar o normal funcionamento do programa desenvolvido. De notar, a existência de dois autores.

```
INPUT:
@inproceedings{AH97,
  title = "Dynamic Dictionary = cooperative information sources",
  author = "J.J. Almeida and P.R. Henriques",
```

```
  year = 1998,
  address = "Australia",
  url = "http://natura.di.uminho.pt/~jj/bib/agentes97.ps.gz",
  keyword ="dictionary, Agentes",
  booktitle = "Proc. II Conference Knowledge-based Intelligent Electronic Systems ({Kes98})",
  month = "April",
}
OUTPUT:
  AH97:["J.J. Almeida","P.R. Henriques"]
```

Teste 2: Neste teste pretende-se provar o funcionamento do programa para autores que possuem um único valor.

```
INPUT:
@techreport{Camila,
  author ={{projecto and Camila}},
  editor ={L.S. Barbosa and J.J. Almeida and J.N. Oliveira and Luís Neves},
  title = "\textsc{Camila} - A Platform for Software Mathematical Development",
  url="http://camila.di.uminho.pt",
  type="(Páginas do projecto)",
  institution = "umdi",
  year=1998,
  keyword = "FS",
}
OUTPUT:
  Camila:"projecto and Camila"
```

Teste 3: Com a realização do teste seguinte, prova-se o bom funcionamento do programa, mesmo que o nome do campo esteja escrito em maiúscula.

```
INPUT:
@INPROCEEDINGS{CH2010a,
  AUTHOR =       {Daniela da Cruz and Pedro Rangel Henriques},
  TITLE =        {Exploring, Visualizing and Slicing the Soul of XML Documents},
  BOOKTITLE =    {Proceedings of 25th Symposium On Applied Computing - Document Engineering},
  YEAR =         {2010},
  note =         {to be published}
}
OUTPUT:
  CH2010a:["Daniela da Cruz","Pedro Rangel Henriques"]
```

Teste 4: Com a realização deste teste verifica-se o funcionamento do programa, para campos onde o valor apareça em mais que uma linha.

```
INPUT:
@inproceedings{mp2001,
  author= {J.J. Almeida and  J. Gustavo Rocha and  P. Rangel Henriques and
    Sónia Moreira and Alberto Simões},
  title = {{Museu da Pessoa} -- Arquitectura} ,
  booktitle = {Congresso Nacional de Bibliotecários, Arquivistas e
```

```
    Documentalistas},
  address = {Porto},
  url = "http://natura.di.uminho.pt/~jj/bib/museuDaPessoa2001.ps.gz",
  month = "Maio",
  year= 2001,
}
OUTPUT:
  mp2001:["J.J. Almeida"," J. Gustavo Rocha"," P. Rangel Henriques","Sónia Moreira",
          "Alberto Simões"]
```

Por fim realizou-se o teste para o ficheiro "exemplo-utf8.bib", fornecido.

Alínea C

Teste 1: Com este teste pretende-se avaliar o normal funcionamento do programa desenvolvido. De notar, a existência do símbolo '=' no valor do campo "title".

```
INPUT:
@inproceedings{AH97,
  title = "Dynamic Dictionary = cooperative information sources",
  author = "J.J. Almeida and P.R. Henriques",
  year = 1998,
  address = "Australia",
  url = "http://natura.di.uminho.pt/~jj/bib/agentes97.ps.gz",
  keyword ="dictionary, Agentes",
  booktitle = "Proc. II Conference Knowledge-based Intelligent Electronic Systems ({Kes98})",
  month = "April",
}
OUTPUT:
"AH97" : {
                "title" : "Dynamic Dictionary = cooperative information sources",
                "author" : ["J.J. Almeida","P.R. Henriques"],
                "year" : 1998,
                "address" : "Australia",
                "url" : "http://natura.di.uminho.pt/~jj/bib/agentes97.ps.gz",
                "keyword" : "dictionary, Agentes",
                "booktitle" : "Proc. II Conference on Knowledge-based Intelligent
                               Electronic Systems (Kes98)",
                "month" : "April"
        }
```

Teste 2: Com a realização deste teste, analisou-se o funcionamento para um input com valores numéricos, strings com a seu valor escrito em diferentes linhas e valores que devem ser representados sobre forma de uma lista.

```
INPUT:
@InProceedings{MP07,
    author =       {Alberto Simões and Rúben Fonseca and José João Almeida},
```

```
    title =        {{Makefile::Parallel} Dependency Specification Language},
    booktitle =    {Euro-Par 2007},
    year =         {2007},
    address =      {Rennes, France},
    month =        {August},
    pages =  {33--41},
    editor =   {Anne-Marie Kermarrec and Luc Bougé and Thierry Priol},
    volume =   {4641},
    series =   {LNCS},
    publisher =  {Springer-Verlag},
    abstract =     {  Some processes are not easy to be programmed from scratch
  for
    parallel machines (clusters), but can be easily split on simple
    steps. Makefile::Parallel is a tool which lets users to specify how processes
    depend on each other.

    The language syntax resembles the well known Makefile
    makefiles format, but instead of specifying files or targets
    dependencies, Makefile::Parallel specifies processes (or jobs) dependencies.

    The scheduler submits jobs to the cluster scheduler (in our case,
    Rocks PBS) waiting them to end. When each process finishes,
    dependencies are calculated and direct dependent jobs are submitted.

    Makefile::Parallel language includes features to specify parametric rules,
  used
    to split and join processes dependencies.  Some tasks can be split
    into n smaller jobs working on different portions of files. At the
    end, another process can be used to join the results.
  }
}
OUTPUT:
"MP07" : {
                "author" : ["Alberto Simões","Rúben Fonseca","José João Almeida"],
                "title" : "Makefile::Parallel Dependency Specification Language",
                "booktitle" : "Euro-Par 2007",
                "year" : 2007,
                "address" : "Rennes, France",
                "month" : "August",
                "pages" : "33--41",
                "editor" : ["Anne-Marie Kermarrec","Luc Bougé","Thierry Priol"],
                "volume" : 4641,
                "series" : "LNCS",
                "publisher" : "Springer-Verlag",
                "abstract" : "Some processes are not easy to be programmed from scratch
                              for parallel machines (clusters), but can be easily split on
                              simple steps. Makefile::Parallel is a tool which lets users
                              to specify how processes depend on each other.
```

```
                                The language syntax resembles the well known Makefile
                                makefiles format, but instead of specifying files or targets
                                dependencies, Makefile::Parallel specifies processes (or jobs)
                                dependencies.  The scheduler submits jobs to the cluster
                                scheduler (in our case, Rocks PBS) waiting them to end.
                                When each process finishes, dependencies are calculated
                                and direct dependent jobs are submitted.
                                Makefile::Parallel language includes features to specify
                                parametric rules, used to split and join processes
                                dependencies.  Some tasks can be split into n smaller jobs
                                working on different portions of files. At the end, another
                                process can be used to join the results."
        }
```

Teste 3: Com a realização deste teste verificou-se o funcionamento do programa, para campos com valor numérico (campo: year).

```
INPUT:
@inproceedings{mp2001,
  author= {J.J. Almeida and  J. Gustavo Rocha and  P. Rangel Henriques and
    Sónia Moreira and Alberto Simões},
  title = {{Museu da Pessoa} -- Arquitectura} ,
  booktitle = {Congresso Nacional de Bibliotecários, Arquivistas e
    Documentalistas},
  address = {Porto},
  url = "http://natura.di.uminho.pt/~jj/bib/museuDaPessoa2001.ps.gz",
  month = "Maio",
  year= 2001,
}
OUTPUT:
"mp2001" : {
                "author" : ["J.J. Almeida","J. Gustavo Rocha","P. Rangel Henriques",
                            "Sónia Moreira","Alberto Simões"],
                "title" : "Museu da Pessoa -- Arquitectura",
                "booktitle" : "Congresso Nacional de Bibliotecários, Arquivistas
                               e Documentalistas",
                "address" : "Porto",
                "url" : "http://natura.di.uminho.pt/~jj/bib/museuDaPessoa2001.ps.gz",
                "month" : "Maio",
                "year" : 2001
           }
```

Por fim realizou-se a conversão do ficheiro "exemplo-utf8.bib", fornecido. Contudo, para verificar que o resultado era um ficheiro json válido, utilizou-se um website que analisa o conteúdo de um documento e procura a existência de erros.

Alínea D

Teste 1: Verificar se caso o utilizador não insira um nome válido o programa deteta que este não é válido

```
INPUT: .,.,'
OUTPUT: Nome Inválido!
```

Teste 2: Verificar se independentemente do formato do input do utilizador, os resultados para esse autor seriam os mesmos. De maneira a não ficar muito extenso, apenas vamos considerar o autor 'Alberto Manuel Simoes', onde testamos quais os resultados para 'Alberto Manuel Simoes', 'Alberto M. Simoes' e 'A. Simoes'.

```
INPUT: Alberto Manuel Simões
OUTPUT:
digraph G {
    "Alberto Manuel Simoes" -> "Alexandre Carvalho"[label="1"];
    "Alberto Manuel Simoes" -> "Ana Frankenberg-Garcia"[label="1"];
    "Alberto Manuel Simoes" -> "Ana Pinto"[label="1"];
    "Alberto Manuel Simoes" -> "Anabela Barreiro"[label="1"];
    "Alberto Manuel Simoes" -> "Analia Loureno"[label="2"];
    "Alberto Manuel Simoes" -> "Antonio R. Fernandes"[label="1"];
    "Alberto Manuel Simoes" -> "Belinda Maia"[label="1"];
    "Alberto Manuel Simoes" -> "Bruno Martins"[label="1"];
    "Alberto Manuel Simoes" -> "Cristina Mota"[label="1"];
    "Alberto Manuel Simoes" -> "Debora Oliveira"[label="1"];
    "Alberto Manuel Simoes" -> "Diana Santos"[label="1"];
    "Alberto Manuel Simoes" -> "Eckhard Bick"[label="2"];
    "Alberto Manuel Simoes" -> "Elisabete Ranchhod"[label="1"];
    "Alberto Manuel Simoes" -> "Eugenio Ferreira"[label="1"];
    "Alberto Manuel Simoes" -> "Isabel Rocha"[label="1"];
    "Alberto Manuel Simoes" -> "J. Alves de Castro"[label="3"];
    "Alberto Manuel Simoes" -> "J. Gustavo Rocha"[label="2"];
    "Alberto Manuel Simoes" -> "Jose Joao Almeida"[label="34"];
    "Alberto Manuel Simoes" -> "Luis Cabral"[label="1"];
    "Alberto Manuel Simoes" -> "Luis Costa"[label="1"];
    "Alberto Manuel Simoes" -> "Luis Sarmento"[label="1"];
    "Alberto Manuel Simoes" -> "Marcirio Chaves"[label="1"];
    "Alberto Manuel Simoes" -> "Miguel Rocha"[label="1"];
    "Alberto Manuel Simoes" -> "Nuno Cardoso"[label="1"];
    "Alberto Manuel Simoes" -> "P. Rangel Henriques"[label="3"];
    "Alberto Manuel Simoes" -> "Paulo A. Rocha"[label="2"];
    "Alberto Manuel Simoes" -> "Paulo Silva"[label="1"];
    "Alberto Manuel Simoes" -> "Rachel Aires"[label="1"];
    "Alberto Manuel Simoes" -> "Rosario Silva"[label="1"];
    "Alberto Manuel Simoes" -> "Ruben Fonseca"[label="1"];
    "Alberto Manuel Simoes" -> "Rui Vilela"[label="2"];
    "Alberto Manuel Simoes" -> "Sonia Moreira"[label="1"];
    "Alberto Manuel Simoes" -> "Susana Afonso"[label="1"];
    "Alberto Manuel Simoes" -> "Xavier Gomez Guinovart"[label="2"];
}
```

```
INPUT: Alberto M. Simoes
OUTPUT:
digraph G {
    "Alberto M. Simoes" -> "Alexandre Carvalho"[label="1"];
    "Alberto M. Simoes" -> "Ana Frankenberg-Garcia"[label="1"];
    "Alberto M. Simoes" -> "Ana Pinto"[label="1"];
    "Alberto M. Simoes" -> "Anabela Barreiro"[label="1"];
    "Alberto M. Simoes" -> "Analia Loureno"[label="2"];
    "Alberto M. Simoes" -> "Antonio R. Fernandes"[label="1"];
    "Alberto M. Simoes" -> "Belinda Maia"[label="1"];
    "Alberto M. Simoes" -> "Bruno Martins"[label="1"];
    "Alberto M. Simoes" -> "Cristina Mota"[label="1"];
    "Alberto M. Simoes" -> "Debora Oliveira"[label="1"];
    "Alberto M. Simoes" -> "Diana Santos"[label="1"];
    "Alberto M. Simoes" -> "Eckhard Bick"[label="2"];
    "Alberto M. Simoes" -> "Elisabete Ranchhod"[label="1"];
    "Alberto M. Simoes" -> "Eugenio Ferreira"[label="1"];
    "Alberto M. Simoes" -> "Isabel Rocha"[label="1"];
    "Alberto M. Simoes" -> "J. Alves de Castro"[label="3"];
    "Alberto M. Simoes" -> "J. Gustavo Rocha"[label="2"];
    "Alberto M. Simoes" -> "Jose Joao Almeida"[label="34"];
    "Alberto M. Simoes" -> "Luis Cabral"[label="1"];
    "Alberto M. Simoes" -> "Luis Costa"[label="1"];
    "Alberto M. Simoes" -> "Luis Sarmento"[label="1"];
    "Alberto M. Simoes" -> "Marcirio Chaves"[label="1"];
    "Alberto M. Simoes" -> "Miguel Rocha"[label="1"];
    "Alberto M. Simoes" -> "Nuno Cardoso"[label="1"];
    "Alberto M. Simoes" -> "P. Rangel Henriques"[label="3"];
    "Alberto M. Simoes" -> "Paulo A. Rocha"[label="2"];
    "Alberto M. Simoes" -> "Paulo Silva"[label="1"];
    "Alberto M. Simoes" -> "Rachel Aires"[label="1"];
    "Alberto M. Simoes" -> "Rosario Silva"[label="1"];
    "Alberto M. Simoes" -> "Ruben Fonseca"[label="1"];
    "Alberto M. Simoes" -> "Rui Vilela"[label="2"];
    "Alberto M. Simoes" -> "Sonia Moreira"[label="1"];
    "Alberto M. Simoes" -> "Susana Afonso"[label="1"];
    "Alberto M. Simoes" -> "Xavier Gomez Guinovart"[label="2"];
}
```

```
INPUT: A. Simões
OUTPUT:
digraph G {
    "A. Simoes" -> "Alexandre Carvalho"[label="1"];
    "A. Simoes" -> "Ana Frankenberg-Garcia"[label="1"];
    "A. Simoes" -> "Ana Pinto"[label="1"];
    "A. Simoes" -> "Anabela Barreiro"[label="1"];
    "A. Simoes" -> "Analia Loureno"[label="2"];
    "A. Simoes" -> "Antonio R. Fernandes"[label="1"];
    "A. Simoes" -> "Belinda Maia"[label="1"];
    "A. Simoes" -> "Bruno Martins"[label="1"];
    "A. Simoes" -> "Cristina Mota"[label="1"];
    "A. Simoes" -> "Debora Oliveira"[label="1"];
    "A. Simoes" -> "Diana Santos"[label="1"];
    "A. Simoes" -> "Eckhard Bick"[label="2"];
    "A. Simoes" -> "Elisabete Ranchhod"[label="1"];
    "A. Simoes" -> "Eugenio Ferreira"[label="1"];
    "A. Simoes" -> "Isabel Rocha"[label="1"];
    "A. Simoes" -> "J. Alves de Castro"[label="3"];
    "A. Simoes" -> "J. Gustavo Rocha"[label="2"];
    "A. Simoes" -> "Jose Joao Almeida"[label="34"];
    "A. Simoes" -> "Luis Cabral"[label="1"];
    "A. Simoes" -> "Luis Costa"[label="1"];
    "A. Simoes" -> "Luis Sarmento"[label="1"];
    "A. Simoes" -> "Marcirio Chaves"[label="1"];
    "A. Simoes" -> "Miguel Rocha"[label="1"];
    "A. Simoes" -> "Nuno Cardoso"[label="1"];
    "A. Simoes" -> "P. Rangel Henriques"[label="3"];
    "A. Simoes" -> "Paulo A. Rocha"[label="2"];
    "A. Simoes" -> "Paulo Silva"[label="1"];
    "A. Simoes" -> "Rachel Aires"[label="1"];
    "A. Simoes" -> "Rosario Silva"[label="1"];
    "A. Simoes" -> "Ruben Fonseca"[label="1"];
    "A. Simoes" -> "Rui Vilela"[label="2"];
    "A. Simoes" -> "Sonia Moreira"[label="1"];
    "A. Simoes" -> "Susana Afonso"[label="1"];
    "A. Simoes" -> "Xavier Gomez Guinovart"[label="2"];
}
```

Como se pode verificar, os resultados são os mesmos, pelo que se pode concluir que o programa cumpre os requisitos. Posteriormente, verificou-se que os autores presentes nos resultados são os que aparecem no ficheiro, e com os seus valores corretos.

Teste 4: Verificação de desenho do grafo de associação de autores, através de ferramentas que processam DOT.
Neste exemplo, usou-se o output obtido através do autor 'Bastian Cramer'.

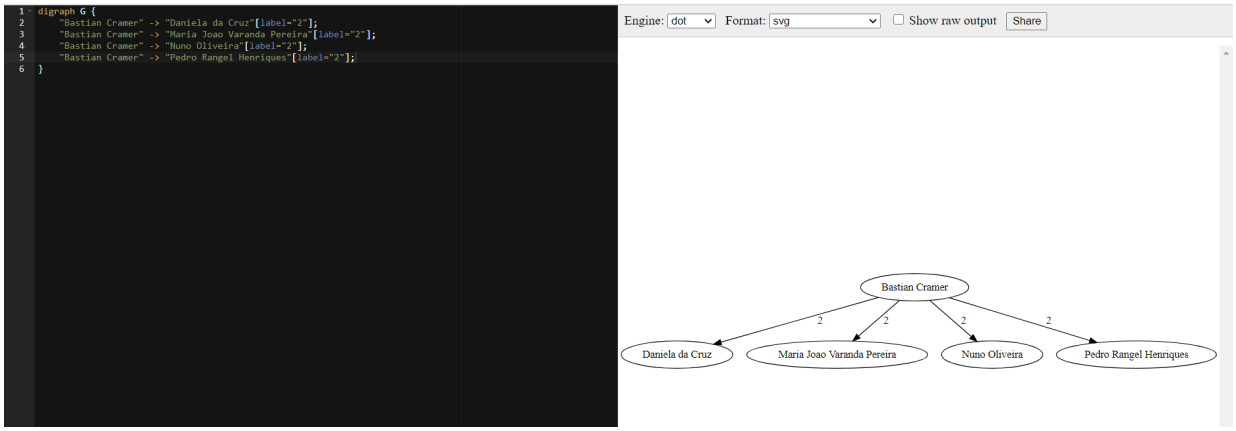

Figura 1: Grafo para 'Bastian Cramer'

Capítulo 5

Problema 4: Análise e Especificação

5.1 Descrição informal do problema

O principal objetivo deste problema consiste em converter um ficheiro com o formato csv para um ficheiro com o formato json. Um ficheiro csv (Comma Separated Values), consiste num documento onde os campos de cada valor estão separados por um ponto e virgula ';'. É possível encontrar os nomes de cada campo na primeira linha do ficheiro (também separados por pontos e virgulas).
Contudo, para este problema considera-se a possibilidade da existência de listas de valores, ou seja é possível haver campos que em vez de conter um único valor contém uma lista. Para tal acontecer é necessário que os valores registados apresentem o seguinte padrão: (valor, ... ,valor). Além de registar os valores com a forma indicada é necessário no seu nome aparecer um * indicando a existência de uma lista de valores. Considera-se também a possibilidade de realizar operações (soma, máximo, mínimo e média) sobre as listas, colocando a operação - sum, avg, max, min - a seguir ao * no nome do campo. Para a realização de operações é necessário que os valores presentes na lista suportem tais operações.

5.2 Especificação dos Requisitos

5.2.1 Dados

Para a realização do problema deve ser fornecido um documento com o formato csv, onde a primeira linha contém o nome dos campos, sendo possível especificar, usando um *, o uso de listas na representação dos valores referentes ao nome. A realização de operações sobre as listas devem ser especificadas utilizando o nome do campo, escrevendo o tipo da operação - sum, avg, max, min - a seguir ao *.
Por fim, os valores devem aparecer nas linhas seguintes, também separados por ';'. Para utilizar as listas, os valores devem aparecer entre parênteses e separados por virgulas. Também é fundamental que para cada nome, exista um valor, mesmo que seja nulo, e para cada valor deve existir um nome correspondente na primeira linha.

5.2.2 Pedidos

Recebido um csv válido, é pedido a construção do ficheiro json correspondente. Assim, é necessário criar um conjunto de pares { "campo": "valor", ... }, que representam cada linha presente no ficheiro csv. No atributo campo estará presente o nome referente ao valor e no atributo valor estará o seu valor ou uma lista de valores. Desta forma cada conjunto terá tantos pares, "campo": "valor", como nomes de valores presentes na primeira linha.

Capítulo 6

Problema 4: Concepção/desenho da Resolução

Para a realização do problema, é necessário ler o documento csv e retirar as informações necessárias. Com o objetivo de construir o ficheiro json, sem utilizar muitos recursos, optou-se por ler o ficheiro linha a linha e ir escrevendo o correspondente, armazenado em memória o mínimo possível.

6.1 Estruturas de Dados

Como é necessário saber os nomes correspondentes aos valores que são apresentados ao longo do ficheiro, armazenou-se numa lista, após ler a primeira linha do ficheiro, os nomes de cada valor. Uma vez que os nomes aparecem de forma sequencial, associou-se a cada nome um índice da lista.
Optou-se por não utilizar mais nenhuma estrutura de dados auxiliar uma vez que, com a informação já guardada, é possível ir construindo o conjunto de pares à medida que que se lê cada linha do ficheiro.

6.2 Algoritmos

A resolução deste problema, passa por duas fases, a fase de armazenamento dos dados necessários (nomes dos valores) e a fase de leitura/processamento das linhas que contém os valores.
Para a fase de armazenamento dos nomes, realiza-se a leitura da primeira linha do documento e posteriormente um split sobre a linha. Analisando a figura 6.1, observa-se a realização desta primeira fase, da qual resulta a lista "cabeçalho"com os nomes dos valores.

Listing 6.1: Algoritmo de armazenamento dos nomes

```
1       line = f.readline()
2       cabecalho = re.split(r';',line)
```

A segunda fase, sendo mais complexa, pode ser dividida em casos. O caso 1, onde o valor deve ser representado sem a utilização de listas. O caso 2, onde se utiliza listas para representar o valor, contudo não se realiza nenhuma operação e o caso 3 onde se realiza uma operação sobre a lista.
Para distinguir estes casos utiliza-se a seguinte expressão regular, capaz de diferenciar os 3 casos.

```
([^*]+) *\* *(.*)
```

Como a expressão regular vai procurar por uma palavra, seguida pela chave *, percebe-se que ao encontrar estamos no caso 2 ou 3, caso não seja encontrado o * estamos no caso 1. Considerou-se também que entre o nome do valor e a chave * podem existir 0, 1 ou mais espaços.
Para diferenciar o caso 2 do 3 basta haver um conjunto de caracteres a seguir à chave *. Uma vez mais entre a chave e a palavra podem haver ou não espaços.
Contudo, ao contrario do caso 1, onde basta escrever o par correspondente, no caso 2 e 3 ainda existe mais algum trabalho a realizar.
No caso onde os valores não sofrem nenhuma operação (caso 2), é necessário substituir os parênteses curvos pelos retos, retirar os espaços entre os elementos e caso os elementos não sejam numéricos coloca-los entre aspas.

Listing 6.2: Algoritmo de tratamento dos elementos no caso 2

```
1       val = re.sub(r'^\(', r'[', val)
2       val = re.sub(r'\)$', r']', val)
3
4       val = re.sub(r'^\[ *', r'[', val)
5       val = re.sub(r' *]$', r']', val)
6       val = re.sub(r' *, *', r',', val)
7
8       if(re.search(r'[^0-9\[\]\,]',val)):
9         val = re.sub(r'"', r'', val)
10        val = re.sub(r'^(\[)(.)', r'\1"\2', val)
11        val = re.sub(r'(.),', r'\1",', val)
12        val = re.sub(r'(.)]$', r'\1"]', val)
```

Para o caso 3, é necessário eliminar os parênteses curvos e realizar o split dos elementos. Posteriormente é necessário identificar a operação a realizar.

Listing 6.3: Algoritmo de tratamento dos elementos no caso 3

```
1       val = re.sub(r'^\( *',r'',val)
2       val = re.sub(r' *\)$',r'',val)
3       valores = re.split(r',',val)
```

Por fim, para evitar problemas de abrir e fechar aspas, eliminou-se dos valores e dos nomes todas as aspas existentes.

Capítulo 7

Problema 4: Codificação e Testes

7.1 Alternativas, Decisões e Problemas de Implementação

Ao contrario do realizado era possível ler o ficheiro todo e guarda-lo numa estrutura de dados, deixando o processamento dos dados para uma fase posterior. Contudo, considerou-se essa estratégia muito ineficiente. Uma vez que os valores encontram-se todos na mesma linha, foi possível realizar o processamento dos dados conforme se lia o ficheiro, evitando acumular em memória os dados de um ficheiro que pode ser enorme.
Optando por processar a informação à medida que se ia avançando no documento trouxe o problema da indeterminação do fim do documento. Esta informação era fundamental para saber quando colocar uma virgula entre os conjuntos. Assim para ultrapassar este problema fecha-se os conjuntos, seguidos da virgula, no inicio da leitura de um novo conjunto, e fecha-se o ultimo conjunto fora da leitura do documento.

7.2 Testes realizados e Resultados

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

Teste 1: Com a realização deste teste pretende-se avaliar se o programa é capaz de funcionar corretamente, mesmo que a chave * esteja separada do nome do valor, além do normal funcionamento do programa.

```
INPUT:
      número;nome;curso;notas  *
      A71823;Ana Maria;MIEI;(12,14,15,18)
      A89765;João Martins;LCC;(11,16,13)
      A54321;Paulo Correia;MIEFIS;(17)
OUTPUT:
       {
          {
                "número" : "A71823",
                "nome" : "Ana Maria",
                "curso" : "MIEI",
                "notas" : [12,14,15,18]
          },

          {
```

```
            "número" : "A89765",
            "nome" : "João Martins",
            "curso" : "LCC",
            "notas" : [11,16,13]
        },

        {
            "número" : "A54321",
            "nome" : "Paulo Correia",
            "curso" : "MIEFIS",
            "notas" : [17]
        }
    }
```

Teste 2: Neste teste pretende-se avaliar o funcionamento do requisito referente a operações sobre listas. Para além de avaliar este requisito, também se pretende verificar que o programa funciona para o caso em que a operação venha distante da chave * ou quando existe espaços desnecessários no valor do campo.

```
INPUT:
    número;nome;curso;notas*   min
    A71823;       Ana Maria    ;MIEI;(12,14,15,18)
    A89765;João "Martins";LCC;(     11,16              ,13)
    A54321;Paulo Correia;MIEFIS;(17)
OUTPUT:
    {
        {
            "número" : "A71823",
            "nome" : "Ana Maria",
            "curso" : "MIEI",
            "notas" : 12.0
        },

        {
            "número" : "A89765",
            "nome" : "João Martins",
            "curso" : "LCC",
            "notas" : 11.0
        },

        {
            "número" : "A54321",
            "nome" : "Paulo Correia",
            "curso" : "MIEFIS",
            "notas" : 17.0
        }
    }
```

Teste 3: Com este teste pretende-se verificar o funcionamento da listagem de valores para elementos não numéricos.

```
INPUT:
      número;nome;curso;amigos*
      A71823;       Ana Maria    ;MIEI;(Miguel, Zé)
      A89765;João Martins;LCC;(Pedro,Carlos     )
      A54321;Paulo Correia;MIEFIS;(  "Tiago")
OUTPUT:
       {
          {
                "número" : "A71823",
                "nome" : "Ana Maria",
                "curso" : "MIEI",
                "amigos" : ["Miguel","Zé"]
          },

          {
                "número" : "A89765",
                "nome" : "João Martins",
                "curso" : "LCC",
                "amigos" : ["Pedro","Carlos"]
          },

          {
                "número" : "A54321",
                "nome" : "Paulo Correia",
                "curso" : "MIEFIS",
                "amigos" : ["Tiago"]
          }
       }
```

Capítulo 8

Conclusão

Com o desenvolvimento do trabalho, o grupo consolidou o conhecimento sobre as expressões regulares, e percebeu o quão importante estas podem ser para a generalização dos casos de estudo, pois ao invés de criar expressões para cada um dos casos possíveis, consegue-se criar uma única expressão que consiga apanhar todos esses casos, não tendo depois no futuro de se preocupar com o formato de uma determinada frase.
Ao longo do relatório foram apresentados os diversos raciocínios e algoritmos por trás de cada uma das alíneas, fundamentado estes com exemplos de código das alíneas.
Numa análise geral, o grupo pensa ter atingido os objetivos propostos para cada uma das alíneas do Problema 3, tendo realizado inúmeros testes de maneira a assegurar que os resultados obtidos seriam os corretos.
Para além do problema 3, o grupo também decidiu realizar o Problema 4, de maneira a aprofundar ainda mais o conhecimento.

Apêndice A

Código do Programa 3

De seguida lista-se o código desenvolvido para a resolução do problema 3, distinguindo-se cada alínea por um programa.

O CÓDIGO desenvolvido, para a resolução da alínea A do problema 3 é o seguinte:

```
import re

bib = open('exemplo-utf8.bib', encoding='utf-8')

categorias = {
    "@article" : 0,
    "@book" : 0,
    "@inBook" : 0,
    "@misc" : 0,
    "@online" : 0,
    "@inCollection" : 0,
    "@proceedings" : 0,
    "@inProceedings" : 0,
    "@mastersthesis" : 0,
    "@phdthesis" : 0,
    "@techreport" : 0,
    "@booklet" : 0,
    "@manual" : 0,
    "@unpublished" : 0
}

for line in bib:
    categoriaMatch = re.search(r'@[a-zA-Z]+', line)
    if categoriaMatch:
        categoria = categoriaMatch.group(0).lower()

        for key, value in categorias.items():
            sameCategoria = re.search(r'(?i:' + categoria + r')', key)
            if sameCategoria and len(key) == len(categoria):
                categorias[key] += 1
```

```
    else:
        pass

bib.close()

html = open("alineaA.html", "w")

html.write('<!DOCTYPE html>')
html.write('<html>')
html.write('<head>')
html.write('<meta charset=\"utf-8\"')
html.write('</head>')
html.write('<body>')
html.write('<h1>Entradas por Categoria</h1>')
html.write('</body>')
html.write('<ul>')

for key, value in sorted(categorias.items()):
    html.write('<li>')
    html.write(key + ': ' + str(value))
    html.write('</li>')

html.write('</body>')
html.write('</html>')

html.close()
```

O CÓDIGO desenvolvido, para a resolução da alínea B do problema 3 é o seguinte:

```
import re
f = open('exemplo-utf8.bib', encoding='utf-8')

temp = {}
for line in f:
    y = re.search(r'^\@[^{]+{([^,]+),',line.strip())
    if(y):
        res = ''
        dentro = True
        chave = ''
        temp2 = {}
        while(dentro):
            subline = f.readline()
            if(re.search(r'^}$',subline.strip())):
                dentro = False
            else:
                if(pal := re.search(r'([^="]+) *= *("|{|[0-9])?(.*)',subline)):
                    chave = pal.group(1).strip().lower()

                    if(chave =='author' and chave in temp2 and pal.group(2) and pal.group(3)):
                        temp2[chave] = temp2[chave] + ' ' + pal.group(2).strip()
                                       + pal.group(3).strip()
                    else:
                        if(chave =='author' and pal.group(2) and pal.group(3)):
                            temp2[chave] = pal.group(2).strip() + pal.group(3).strip()
                else:
                    if(chave =='author' and chave in temp2):
                        temp2[chave] = temp2[chave]+ ' ' + subline.strip()
                    else:
                        if(chave =='author'):
                            temp2[chave] = subline.strip()

        temp[y.group(1)] = temp2

f.close()

b = open("alineaB.txt", "w", encoding='utf-8')

for (x,y) in temp.items():
    frase = ''
    for (s,v) in y.items():

        lista = False
        if(not re.match(r'^{{',v.strip())):
            lista = True
```

```
        frase = re.sub(r'{',r'',v.strip())
        frase = re.sub(r',$',r'',frase.strip())
        frase = re.sub(r'}',r'',frase.strip())
        frase = re.sub(r'"',r'',frase.strip())
        frase = re.sub(r'\\',r'',frase.strip())

        if(lista):
            frase = re.sub(r' and ',r'","',frase.strip())
            frase = '["'+ frase + '"]'
        else:
            frase = '"' + frase + '"'

    b.write(x + ':'+ frase + '\n')

b.close()

print('Concluido!')
```

O CÓDIGO desenvolvido, para a resolução da alínea C do problema 3 é o seguinte:

```
import re
f = open('exemplo-utf8.bib', encoding='utf-8')
temp = {}
for line in f:
    y = re.search(r'^\@[^{]+{([^,]+),',line.strip())
    if(y):
        res = ''
        dentro = True
        chave = ''
        temp2 = {}
        while(dentro):
            subline = f.readline()
            if(re.search(r'^}$',subline.strip())):
                dentro = False
            else:
                if(pal := re.search(r'([^="]+) *= *("|{|[0-9])?(.*)',subline)):
                    chave = pal.group(1).strip()

                    if(chave in temp2):
                        if(pal.group(2)):
                            temp2[chave] = temp2[chave] + ' ' + pal.group(2).strip()
                            if(pal.group(3)):
                                temp2[chave] = temp2[chave] + pal.group(3).strip()
                    else:
                        if(pal.group(2)):
                            temp2[chave] = pal.group(2).strip()
                            if(pal.group(3)):
                                temp2[chave] = temp2[chave] + pal.group(3).strip()
                else:
                    if(chave in temp2):
                        temp2[chave] = temp2[chave]+ ' ' + subline.strip()
                    else:
                        temp2[chave] = subline.strip()

        temp[y.group(1)] = temp2

f.close()

json = open("alineaC.json", "w", encoding='utf-8')

json.write('{\n')

i = 1
ultimo = len(temp)
for (x,y) in temp.items():
    json.write(f'\t"{x}" : '+'{' + '\n')
```

```
    j = 1
    ultimo2 = len(y)
    for (s,v) in y.items():

        lista = False
        if((s == 'author' or s == 'editor') and not re.match(r'^{{',v.strip())):
            lista = True

        frase = re.sub(r'\\[^{]+{',r'',v.strip())
        frase = re.sub(r'{',r'',frase.strip())
        frase = re.sub(r',$',r'',frase.strip())
        frase = re.sub(r'}',r'',frase.strip())
        frase = re.sub(r'"',r'',frase.strip())
        frase = re.sub(r'\\',r'',frase.strip())

        number = False
        if(re.search(r'^[0-9]+$',frase.strip())):
            frase = re.sub(r'^0+$',r'0',frase.strip())
            frase = re.sub(r'^0+([1-9]+)',r'\1',frase.strip())
            number = True

        if(lista):
            frase = re.sub(r' and ',r'","',frase.strip())
            frase = '["'+ frase + '"]'
        else:
            if(not number):
                frase = '"' + frase + '"'

        if(j == ultimo2):
            json.write('\t\t"' + s + '"' +' : '+ frase + '\n')
        else:
            json.write('\t\t"' + s + '"' +' : '+ frase + ',' + '\n')

        j = j +1

    if(i == ultimo):
        json.write('\t}' + '\n')
    else:
        json.write('\t},' + '\n')

    json.write('' + '\n')

    i = i + 1

json.write('}' + '\n')
json.close()
print('Conversão Concluida!')
```

O CÓDIGO desenvolvido, para a resolução da alínea D do problema 3 é o seguinte:

```
import re

def normal(name):
    name = re.sub(u"[àáâãäå]", 'a', name)
    name = re.sub(u"[èéêë]", 'e', name)
    name = re.sub(u"[ìíîï]", 'i', name)
    name = re.sub(u"[òóôõö]", 'o', name)
    name = re.sub(u"[ùúûü]", 'u', name)
    name = re.sub(u"[ýÿ]", 'y', name)
    name = re.sub(u"[ß]", 'ss', name)
    name = re.sub(u"[ñ]", 'n', name)
    name = re.sub(r'[^\w' + r' \-\.\,' + r']', '', name)
    return name

nomeInput = input('Insira o nome do Autor: ')

while not re.search(r'[a-zA-Z]+', nomeInput):
    print('Nome Inválido!')
    nomeInput = input('Insira o nome do Autor: ')

nome = normal(nomeInput)

bib = open('exemplo-utf8.bib', encoding='utf-8')
colabs = {}

for line in bib:
    authorLine = re.search(r'author *={*', line.strip())

    if authorLine:
        fimAuthors = False

        while not fimAuthors:
            endFound = re.search(r'}*,', line.strip())

            if endFound:
                fimAuthors = True

                deleteEnter = re.sub(r'\n', ' ', line.strip())
                deleteEndingLine = re.sub(r'\"*\{*\}*,$', '', deleteEnter.strip())
                deleteBeginningLine = re.sub(r' *author* *= *{*\"* *', '', deleteEndingLine)
                fixedLine = re.sub(r' +', ' ', deleteBeginningLine)

                diferentAuthors = fixedLine.split(' and')

                authors = []
```

```
for author in diferentAuthors:
    normalAuthor = normal(author)
    newAuthor = normalAuthor.strip()

    if ',' in newAuthor:
        duplo = newAuthor.split(',')
        firstName = duplo[1].strip()
        lastName = duplo[0].strip()

        corretAuthor = firstName + ' ' + lastName
        authors.append(corretAuthor)

    else:
        authors.append(newAuthor)

dividedName = nome.split(' ')

for name in authors:
    normalizedName = normal(name)

    expression = r'^(?i:' + dividedName[0][0] + r').*(' +
    dividedName[0][1:] + r')?( ?)'
    lastWord = len(dividedName) - 1
    counter = 1

    while counter <= lastWord:
        if counter<lastWord:
            expression += r'(?i:' + dividedName[counter][0] + r')?(.?)
            (' + dividedName[counter][1:] + r')?( ?)'
            counter+=1
        else:
            expression += r'(?i:' + dividedName[counter] + r')$'
            break

    sameAuthor = re.search(expression, normalizedName)

    if sameAuthor:
        for author in authors:
            normalizedAuthor = normal(author)

            expression = r'^(?i:' + dividedName[0][0] + r').*(' +
            dividedName[0][1:] + r')?( ?)'
            lastWord = len(dividedName) - 1
            counter = 1

            while counter <= lastWord:
                if counter<lastWord:
                    expression += r'(?i:' + dividedName[counter][0] + r')?
```

```
                                    (.?)(' + dividedName[counter][1:] + r')?( ?)'
                                    counter+=1
                                else:
                                    expression += r'(?i:' + dividedName[counter] + r')$'
                                    break

                            isOurAuthor = re.search(expression, normalizedAuthor)
                            if isOurAuthor:
                                pass
                            else:
                                if normalizedAuthor in colabs:
                                    colabs[normalizedAuthor] += 1
                                else:
                                    colabs[normalizedAuthor] = 1
                        break
                    else:
                        pass
            else:
                nextLine = next(bib)
                line = line + nextLine
    else:
        pass


bib.close()

for key, value in list(colabs.items()):
    if key == '':
        colabs.pop(key, None)


orderedColabs = sorted(colabs.items())
loopedColabs = sorted(colabs.items())

for key1, value1 in orderedColabs:
    div1 = key1.split(' ')

    expression = r'^(?i:' + div1[0][0] + r').*(' + div1[0][1:] + r')?( ?)'
    lastWord = len(div1) - 1
    counter = 1

    while counter <= lastWord:
        if counter<lastWord:
            expression += r'(?i:' + div1[counter][0] + r')?(.?)(' +
            div1[counter][1:] + r')?( ?)'
            counter+=1
        else:
            expression += r'(?i:' + div1[counter] + r')$'
```

```
            break

    for key2, value2 in loopedColabs:
        match = re.search(expression, key2)

        if match:
            div2 = key2.split(' ')

            if len(div1) > len(div2) and colabs[key2] > 0 and colabs[key1] > 0:
                colabs[key1] += colabs[key2]
                colabs[key2] -= colabs[key2] + 1
            else:
                if len(div1) == len(div2):
                    if len(key1) > len(key2) and colabs[key2] > 0 and colabs[key1] > 0:
                        colabs[key1] += colabs[key2]
                        colabs[key2] -= colabs[key2] + 1
                    else:
                        if len(key1) < len(key2) and colabs[key1] > 0 and colabs[key2] > 0:
                            colabs[key2] += colabs[key1]
                            colabs[key1] -= colabs[key1] + 1
                        else:
                            if (key1 != key2) and colabs[key2] > 0 and colabs[key1] > 0:
                                colabs[key1] += colabs[key2]
                                colabs[key2] -= colabs[key2] + 1
                else:
                    if len(div1) < len(div2) and colabs[key1] > 0 and colabs[key2] > 0:
                        colabs[key2] += colabs[key1]
                        colabs[key1] -= colabs[key1] + 1


for key, value in sorted(colabs.items()):
    if value == -1:
        colabs.pop(key, value)

dotFile = open(nome + '.dot', "w")
dotFile.write("digraph G {")

for key, value in sorted(colabs.items()):
        dotFile.write('\"' + nome + '\" -> \"' + key + '\"[label=\"' + str(value) + '\"];')

dotFile.write("}")
dotFile.close()
```

Apêndice B

Código do Programa 4

Lista-se a seguir o CÓDIGO que foi desenvolvido, para a resolução do problema 4.

```
import re

def sumValores(v):
    res = 0
    for val in v:
        res = res + float(val)
    return res

def maxValores(v):
    res = -9999999999999999999
    for val in v:
        val = float(val)
        if(val > res):
            res = val
    return res

def minValores(v):
    res = 9999999999999999999
    for val in v:
        val = float(val)
        if(val < res):
            res = val
    return res

f = open('exemploP4_2.csv', encoding='utf-8')

line = f.readline()
cabecalho = re.split(r';',line)

print('{')
i = 0
for line in f:
```

```
if(i != 0):
    print('\t},')
    print('')

i = i + 1
sublinha = re.split(r';',line)
print('\t{')
for fazer in cabecalho:
    ultimo = False

    val = sublinha[cabecalho.index(fazer)]

    if(re.search(r'\n',val)):
        ultimo = True

    val = val.strip()

    if(m := re.search(r'([^*]+) *\* *(.*)',fazer)):
        # Se entrou aqui tem de ser uma lista de numeros
        # mas tira as aspas caso os numeros estejam entre aspas
        val = re.sub(r'"', r'', val)
        # Retira o nome do campo
        nome = m.group(1).strip()
        if(m.group(2)):
            faz = m.group(2)

            #elimina os parenteses referentes á lista
            val = re.sub(r'^\( *',r'',val)
            val = re.sub(r' *\)$',r'',val)
            valores = re.split(r',',val)

            if(faz == 'sum'):
                # calcula a soma dos valores
                valSum = sumValores(valores)
                if(ultimo):
                    print(f'\t\t"{nome}" : {valSum}')
                else:
                    print(f'\t\t"{nome}" : {valSum},')
            elif(faz == 'avg'):
                # calcula a media dos valores
                valAvg = sumValores(valores) / len(valores)
                if(ultimo):
                    print(f'\t\t"{nome}" : {valAvg}')
                else:
                    print(f'\t\t"{nome}" : {valAvg},')
            elif(faz == 'max'):
                # calcula o maximo dos valores
                valMax = maxValores(valores)
```

```
                    if(ultimo):
                        print(f'\t\t"{nome}" : {valMax}')
                    else:
                        print(f'\t\t"{nome}" : {valMax},')
                elif(faz == 'min'):
                    # calcula o minimo dos valores
                    valMin = minValores(valores)
                    if(ultimo):
                        print(f'\t\t"{nome}" : {valMin}')
                    else:
                        print(f'\t\t"{nome}" : {valMin},')
            else:
                # lista normal
                val = re.sub(r'^\(', r'[', val)
                val = re.sub(r'\)$', r']', val)

                #Tira espaços entre elementos da lista
                val = re.sub(r'^\[ *', r'[', val)
                val = re.sub(r' *]$', r']', val)
                val = re.sub(r' *, *', r',', val)

                #Se é uma lista de strings mete as aspas
                #elimina aspas no meio para evitar erros de abrir e fechar a string
                if(re.search(r'[^0-9\[\]\,]',val)):
                    val = re.sub(r'"', r'', val)
                    val = re.sub(r'^(\[)(.)', r'\1"\2', val)
                    val = re.sub(r'(.),', r'\1","', val)
                    val = re.sub(r'(.)]$', r'\1"]', val)

                if(ultimo):
                    print(f'\t\t"{nome}" : {val}')
                else:
                    print(f'\t\t"{nome}" : {val},')
        else:
            # não é lista
            #elimina aspas no meio para evitar erros de abrir e fechar a string
            val = re.sub(r'"', r'', val)
            if(ultimo):
                print(f'\t\t"{fazer.strip()}" : "{val}"')
            else:
                print(f'\t\t"{fazer.strip()}" : "{val}",')
print('\t}')
print('')
print('}')
f.close()
```