Processamento de Linguaguens e Compiladores (3º ano LCC)

Trabalho Prático 1

Relatório de Desenvolvimento

Grupo 17

Problema 3

José Pedro Gomes Ferreira
A91636

Pedro Paulo Costa Pereira
A88062

Tiago André Oliveira Leite
A91693

November 14, 2021

Contents

Chapter 1

Introdução

Neste documento vamos explicar a solução que implementamos para a resolução do probelma proposto no ambito da unidade curricular de Processsamento de Linguagens e Compiladores.
O problema proposto consiste em analisar o ficheiro de texto *calv-users.txt* e atraves da utilização da linguagem Python e da biblioteca de expressões regulares 're' extrair informação de forma a produzir alguns resultados.
No desenvolvimento do programa procuramos utilizar o conhecimento adquirido nas aulas, esperando por isso que o resultado final cumpra todos os requisitos.

Chapter 2

Problema Proposto

2.1 Descrição

Construa um ou vários programas para processar o texto ' clav-users.txt ' em que campos de informação têm a seguinte ordem: nome, email, entidade, nível, número de chamadas ao backend, com o intuito de calcular alguns resultados conforme solicitado a seguir:

- Produz uma listagem apenas com o nome e a entidade do utilizador, ordenada alfabeticamente por nome;
- Produz uma lista ordenada alfabeticamente das entidades referenciadas, indicando, para cada uma, quantos utilizadores estão registados;
- Qual a distribuição de utilizadores por níveis de acesso?
- Produz uma listagem dos utilizadores, agrupados por entidade, ordenada primeiro pela entidade e dentro desta pelo nome;
- Por fim, produz os seguintes indicadores:
    1. Quantos utilizadores?
    2. Quantas entidades?
    3. Qual a distribuição em número por entidade?
    4. Qual a distribuição em número por nível?

Para terminar, deve imprimir os 20 primeiros registos num novo ficheiro de output mas em formato Json .

2.2 Requisitos

- Utilização da linguagem Python.
- Resolver o problema com uso de expressoes regulares.
- Utilizar o modulo 're'.

Chapter 3

Concepção/desenho da Resolução

3.1 Organização

Por forma a tornar a resolução do trabalho mais simples, decidimos criar uma função especifica para resolver cada uma das alineas do problema. Assim sendo, a solução do problema vai ser composta por 6 funções mais uma que vai servir de menu para o utilizador poder escolher qual das funcionalidades do prograna quer utilizar.
Na execução do programa todas as linhas do ficheiro 'clav-users.txt' são lidas para uma variavel que depois será utlizada por cada uma da funções.
Por uma questão de simplicidade, sempre que for necessário ordenar, a ordem utilizada é a a ordem alfabetica.

3.2 Funcionalidades

3.2.1 Executar o programa.

Para executar o programa o utilizador terá que colocar na mesma diretoria os ficheiro 'main.py' e 'clav-users.txt'. De seguida basta abrir o terminal nessa diretoria e exceutar o comando:

```
>> python3 main.py
```

3.2.2 O menu.

O menu do programa é implementado pela função `menu()`.
Esta função cria um ciclo `while` que mostra ao utilizador todas as opções de funcionalidades disponiveis. De seguida lê a opção escolhida pelo utilizador e caso a opção esteja entre 1 e 6 vai invocar a função que implementa a funcionalidade selecionada. Se a opção escolhida pelo utilizador for menor que 0 ou maior que 6 vai ser mostrada uma mensagem de erro sendo solicitado ao utilizador que escolha novamente. Caso a opção seja 0, o programa termina a sua execução.
Para tornar a experiencia mais agradavel aos olhos, entre a execução de cada funcionalidade, é feita uma limpeza do ecrã atravês da impressão de 50 paragrafos.

3.2.3 Nome dos utilizadores ordenados e respetiva entidade.

Esta funcionalidade é implementada pela função `name_entity_list()`.
Na sua implementação vamos iterar por cada linha do texto e em cada uma vamos procurar o nome

de utilizador e a entidade correspondente. Para capturar o nome do utlizador vamos utilizar a função `re.match` com a expressão regular `'(\w+\s*(-?\w+\s*)*\b)'` e para capturar a entidade utilizamos a função `re.search` com a expressão regular `'ent_\w*'`. De seguida armazenamos o resultado num dicionario em que a chave é o nome do utilizador e o valor é uma lista com as entindades desse utilizador. Caso o utilizador ainda nao exista no dicionario é criada uma nova entrada, caso contrario, a entidade capturada é acrescentada na lista do utilizador.
Por fim, transformamos o dicionario numa lista composta pelos pares (nome de utilizador, entidade(s)), ordenada por nome de utilizador, e fazemos a impresão de cada utilizador e respetiva(s) entidade(s).

Nota: Na implementação desta função decidimos diferenciar utilizadores cujo nome apenas difere no facto de as letras estarem em maiusculas ou minusculas. Por exemplo 'xxxx', 'XXXX' e 'Xxxx' são utilizadores distintos.

3.2.4 Entidades ordenadas e número de utilizadores registados em cada uma.

Esta funcionalidade é implementada pela função `entity_num_elements_list()`.
Na sua implementação vamos iterar por cada linha do texto e através da função `re.search` com a expressão regular `'ent_\w*'` vamos capturar a entidade presente nessa linha. A entidade capturada é armazenada num dicionario cuja chave é o nome da entidade e o valor é o numero de utilizadores nessa entidade. Caso a entidade ainda não exista no dicionario é criada uma nova entrada com o valor 1, caso contrario, o valor dessa entidade no dicionario é incrementado em 1.
Por fim, transformamos o dicionario numa lista composta pelos pares (entidade, nº de utilizadores), ordenada por entidade, e fazemos a impresão de cada entidade e respetivo nº de utilizadores.

Nota: Na implementação desta função decidimos diferenciar utilizadores cujo nome apenas difere no facto de as letras estarem em maiusculas ou minusculas. Por exemplo 'xxxx', 'XXXX' e 'Xxxx' são utilizadores distintos.

3.2.5 Distribuiçao dos utilizadores por nível de acesso.

Esta funcionalidade é implementada pela função `dist_users_level()`.
Na sua implementação vamos iterar por cada linha do texto e partir cada uma destas pelo separador `'::'` com o uso da função `re.split`. De seguida vamos capturar o nome de utilizador no primeiro elemento (index 0) da linha partida com a função `re.match` e a expressão regular `'(\w+\s*(-?\w+\s*)*\b)'` e vamos capturar o nível de acesso no quarto elemento da linha partida (index 3) com a função `re.search` e a expressão regular `'\d+\.?\d*'`. O resultado de capturar o nível de acesso e nome de utilizador são armazenados num dicionário em que a chave é o nivel de acesso e o valor é um conjunto com os utilizadores com esse nível de acesso. Optamos por um conjunto para evitar repetições, uma vez que existem utilizadores com mais do que um nível de acesso. Caso o nível de acesso capturado nao exista no dicionario, é criada uma entrada cuja chave é o nível de acesso e cujo valor é um conjunto com o utilizador capturado, caso contrario, o utilizador é acrescentado ao conjunto de utilizadores ja existente para aquela chave. Para ser mais facil de contar o nº de utilizadores total utilizamos tambem um conjunto onde vamos colocando cada utilizador capturado.
Por fim, transformamos o dicionario numa lista composta pelos pares (nível de acesso, utilizadores) ordenada por nível de acesso e fazemos uma impressão dos utilizadores presentes em cada nível de acesso assim como a percentagem de utilizadores por nível.

Nota: Na implementação desta função decidimos diferenciar utilizadores cujo nome apenas difere no facto

de as letras estarem em maiusculas ou minusculas. Por exemplo 'xxxx', 'XXXX' e 'Xxxx' são utilizadores distintos.

3.2.6 Utilizadores agrupados por entidade, ordenados por nome e entidade.

Esta funcionalidade é implementada pela função `name_entity_group()`.
Na sua implementação vamos iterar por cada linha do texto e em cada uma vamos procurar o nome de utilizador e a entidade correspondente. Para capturar o nome do utlizador vamos utilizar a função `re.match` com a expressão regular `'(\w+\s*(-?\w+\s*)*\b)'` e para capturar a entidade utilizamos a função `re.search` com a expressão regular `'ent_\w*'`. De seguida armazenamos o resultado num dicionario em que a chave é a entidade e o valor é uma lista com os utilizadores dessa entidade. Caso a entidade ainda nao exista no dicionario é criada uma nova entrada, caso contrario, o utilizadore capturado é acrescentada na lista da entidade.
Por fim, transformamos o dicionario numa lista composta pelos pares (entidade, utilizador), ordenada por entidade, e fazemos a impresão de cada entidade e respetivos utilizadores ordenados por nome.

Nota: Na implementação desta função decidimos diferenciar utilizadores cujo nome apenas difere no facto de as letras estarem em maiusculas ou minusculas. Por exemplo 'xxxx', 'XXXX' e 'Xxxx' são utilizadores distintos.

3.2.7 Mostrar alguns indicadores.

Esta funcionalidade é implementada pela função `indicators()`.
Na sua implementação vamos iterar por cada linha do texto e partir cada uma destas pelo separador `'::'` com o uso da função `re.split`. De seguida vamos capturar o nome de utilizador no primeiro elemento (index 0) da linha partida com a função `re.match` e a expressão regular `'(\w+\s*(-?\w+\s*)*\b)'` vamos capturar a entidade no terceiro elemento da linha partida (index 2) com a função `re.search` e a expressão regular `'ent_\w*'`, e vamos capturar o nível de acesso no quarto elemento da linha partida (index 3) com a função `re.search` e a expressão regular `'\d+\.?\d*'`. Os nomes de utilizadores capturados anteriormente são armazenados num conjunto, para evitar repetiçoes, enquanto que as entidades e níveis de acesso capturados anteriormente são armazenados em dois dicionarios. Num dos dicionarios o par chaves e valores são a entidade e respetivo número de utilizadores enquanto que no outro o par chave e valores são o nível de acesso e respetivo número de utilizadores.
Por fim, fazemos um impressão do número total de utilizadores, o número total de entidades, o número de utilizadores por entidade e o número de utilizadores por nível de acesso.

Nota: Na implementação desta função decidimos diferenciar utilizadores cujo nome apenas difere no facto de as letras estarem em maiusculas ou minusculas. Por exemplo 'xxxx', 'XXXX' e 'Xxxx' são utilizadores distintos.

3.2.8 Imprimir num ficheiro Json os 20 primeiros registos.

Esta funcionalidade é implementada pela função `json_20()`.
Na sua implementação vamos iterar por cada umas das 20 primeiras linhas do texto e partir cada uma destas pelo separador `'::'` com o uso da função `re.split`. De seguida vamos capturar o nome de utilizador no primeiro elemento (index 0) da linha partida com a função `re.match` e a expressão regular `'(\w+\s*(-?\w+\s*)*\b)'`, vamos capturar o email de utilizador no segundo elemento (index 1) da linha partida com a função `re.search` e a expressão regular `'(\w+|\.|@|_|-)+'` , vamos capturar a entidade

no terceiro elemento da linha partida (index 2) com a função `re.search` e a expressão regular `'ent_\w*'`, vamos capturar o nível de acesso no quarto elemento da linha partida (index 3) com a função `re.search` e a expressão regular `'\d+\.?\d*'`, e vamos capturar o número de chamadas ao backend no quinto elemento da linha partida (index 4) com a função `re.search` e a expressão regular `'\d+'`. O resultado de cada linha é armazenado numa lista de tuplos (node de utilizador, email, entidade, nível de acesso, número de chamadas ao backend).
Finalizadas as capturas pedimos ao utilizador para escolher qual o nome do ficheiro para o qual quer imprimir os dados recolhidos. Depois abrimos o fichero e procedemos à impressão de cada registo (tuplo armazenado na lista) num formato json.

Chapter 4

Demonstração de Funcionamento

4.1 Nome dos utilizadores ordenados e respetiva entidade.

4.2 Entidades ordenadas e número de utilizadores registados em cada uma.

4.3 Distribuiçao dos utilizadores por nivel de acesso.

4.4 Utilizadores agrupados por entidade, ordenados por nome e entidade.

4.5 Mostrar alguns indicadores.

4.6 Imprimir num ficheiro Json os 20 primeiros registos.

Chapter 5

Conclusão

Síntese do Documento [**?**, **?**].
Estado final do projecto; Análise crítica dos resultados [**?**].
Trabalho futuro.

Appendix A

Código do Programa

```
import re
import unidecode


def name_entity_list():
    result = {}
    for line in text:
        user =  re.match(r'(\w+\s*(-?\w+\s*)*\b)',line).group()
        entity = re.search(r'ent_\w*',line).group()
        if user not in result:
            result[user] = [entity]
        else:
            result[user].append(', ')
            result[user].append(entity)
    result = list(result.items())
    result.sort(key=lambda x: unidecode.unidecode(x[0].casefold()))
    print("\n*** Utilizador : Entidade(s) ***\n")
    for r in result:
        print(r[0], ":", "".join(r[1]))


def entity_num_elements_list():
    entities = {}
    for line in text:
        entity = re.search(r'ent_\w*', line).group()
        if entity not in entities:
            entities[entity] = 1
        else:
            entities[entity] +=1

    result = list(entities.items())
    result.sort()
    print("\n*** Entidade : Nº Utilizadores ***\n")
    for r in result:
```

```
        print(r[0], ":", r[1])


def dist_users_level():
    levels = {}
    users = set()
    for line in text:
        broken_line = re.split(r'::', line)
        user = re.match(r'(\w+\s*(-?\w+\s*)*\b)', broken_line[0]).group()
        level = re.search(r'\d+\.?\d*',broken_line[3]).group()
        if level not in levels:
            levels[level] = set()
            levels[level].add(user)
        else:
            levels[level].add(user)
        users.add(user)
    result = list(levels.items())
    result.sort()
    total_users = len(users)
    print("\n*** Nivel de Acesso : Distribuição ***\n")
    for r in result:
        print(f"Nivel {r[0]} : {round((len(r[1])/total_users)*100)}%" )
        for user in sorted(r[1], key=lambda x: unidecode.unidecode(x.casefold())):
            print("*",user)
        if(result.index(r) != len(result)-1):
            print("")


def name_entity_group():
    result = {}
    for line in text:
        name = re.match(r'(\w+\s*(-?\w+\s*)*\b)', line).group()
        entity = re.search(r'ent_\w*', line).group()
        if entity not in result:
            result[entity] = [name]
        else:
            result[entity].append(name)

    entities = list(result.keys())
    entities.sort()
    print("\n*** Utilizadores agrupados por entidade ***\n")
    for entity in entities:
        print(f"{entity}:")
        result[entity].sort(key=lambda x: unidecode.unidecode(x.casefold()))
        for user in result[entity]:
            print("*",user)
        print("")
```

```
def indicators():
    users = set()
    entities = {}
    levels = {}
    for line in text:
        broken_line = re.split(r'::', line)
        users.add(re.match(r'(\w+\s*(\w+\s*)*\b)',broken_line[0]).group())
        entity = re.search(r'ent_\w*', broken_line[2]).group()
        level = re.search(r'\d+\.?\d*', broken_line[3]).group()

        if entity not in entities:
            entities[entity] = 1
        else:
            entities[entity] += 1
        if level not in levels:
            levels[level] = 1
        else:
            levels[level] += 1

    print("\n*** Indicadores ***\n")
    print("Número de Utilizadores:",len(users))
    print("")
    print("Número de Entidades:",len(entities.keys()))
    print("")
    print("Distribuição de utilizadores por entidade:")
    for entity in sorted(entities.keys()):
        print(f"* {entity} : {entities[entity]}")
    print("")
    print("Distribuição de utilizadores por nivel:")
    for level in sorted(levels.keys()):
        print(f"* Nivel {level} : {levels[level]}")


def json_20():
    list = []
    for i in range(20):
        broken_line = re.split(r'::', text[i])
        user = re.match(r'(\w+\s*(\w+\s*)*\b)', broken_line[0]).group()
        email = re.search(r'(\w+|\.|@|_|-)+', broken_line[1]).group()
        entity = re.search(r'ent_\w*', broken_line[2]).group()
        level = re.search(r'\d+\.?\d*', broken_line[3]).group()
        calls = re.search(r'\d+', broken_line[4]).group()
        list.append((user,email,entity,level,calls))

    file_name = input("\nDigite Nome Do Ficheiro de Output!\n>> ")
    fp = open(file_name, 'w')
```

```
    fp.write("{\n\t\"registos\":[\n")
    for i in range(len(list)):
        l = list[i]
        fp.write("\t\t{\n")
        fp.write(f"\t\t\t \"utilizador\":\"{l[0]}\",\n")
        fp.write(f"\t\t\t \"email\":\"{l[1]}\",\n")
        fp.write(f"\t\t\t \"entidade\":\"{l[2]}\",\n")
        fp.write(f"\t\t\t \"nivel de acesso\":\"{l[3]}\",\n")
        fp.write(f"\t\t\t \"número de chamadas ao backend\":\"{l[4]}\"\n")
        if i != 19:
            fp.write("\t\t},\n")
        else:
            fp.write("\t\t}\n")
    fp.write("\t]\n}\n")
    fp.close()
    print("\nFicheiro gerado com sucesso!")


def menu():
    cls = lambda: print('\n' * 50)
    inputfromuser = ""
    options = []
    options.append("Listagem com o nome e a entidade do utilizador, ordenada alfabeticamente por
    options.append("Lista ordenada alfabeticamente das entidades referenciadas, indicando, para
    options.append("Distribuição de utilizadores por níveis de acesso.")
    options.append("Utilizadores, agrupados por entidade, ordenada primeiro pela entidade e dent
    options.append("Mostrar alguns indicadores.")
    options.append("Imprimir os 20 primeiros registos num novo ficheiro de output mas em formato
    cls()
    while inputfromuser != '0':
        print("*** Selecione Opção ***\n")
        for i in range(len(options)):
            print(f"{i+1}. {options[i]}")
        print("0. Sair.")

        inputfromuser = input(">> ")
        while not inputfromuser.isdigit() or int(inputfromuser) > len(options) or int(inputfromus
            print("Opçao Invalida!")
            inputfromuser = input(">> ")

        if inputfromuser == '0':
            continue
        elif inputfromuser == '1':
            name_entity_list()
        elif inputfromuser == '2':
            entity_num_elements_list()
        elif inputfromuser == '3':
            dist_users_level()
```

```
        elif inputfromuser == '4':
            name_entity_group()
        elif inputfromuser == '5':
            indicators()
        else:
            json_20()

        input("\nPressione Enter!\n>> ")
        cls()


fp = open('clav-users.txt', 'r')
text = fp.readlines()
fp.close()
menu()
```