Algoritmos e Técnicas de Programação

Engenharia Biomédica (2º ano)

Teste (aferição)

16 a 18 de Dezembro de 2025

Notas:

- Dispõe de 2:00 horas para realizar este teste;
- Comece por criar uma pasta no seu Git, junto das pastas dos TPC, de nome `AFERIÇÃO`. No fim, deverá colocar lá os resultados desta sessão de trabalho: o notebook resultante ou outro programa Python que tiver criado.

Questão 1 (4v = 1+1.5+1.5)

Especifique as seguintes **funções recursivas**. Caso não consiga, pode apresentar uma solução iterativa que será penalizada em **50%**:

a) *Contar ocorrências*: defina uma função que recebe uma lista e um elemento como argumento e devolve o número de ocorrências desse elemento na lista.

b) *Comum?*: defina uma função que recebe duas listas como argumento e devolve o valor booleano verdadeiro se as listas tiverem pelo menos um elemento comum e falso caso contrário.

c) *mergeListas*: defina uma função que recebe duas listas de números ordenadas crescentemente como argumento e devolve uma nova lista ordenada de forma crescente com os elementos das duas listas.

Questão 2 (6v = 1.5+1.5+1.5+1.5)

a) *splitLista*: defina uma função que recebe dois argumentos, uma lista de números e um número, e dá como resultado um tuplo composto por duas listas, a primeira contendo os números menores ou iguais ao número passado como argumento, e a segunda contendo os números maiores que o argumento.

```
In:  print(splitLista([1,2,3,4,5], 3))
Out: ([1,2,3], [4,5])
```

b) *multiplos23*: defina uma função que calcula quantos múltiplos de 23 existem no intervalo de 1 a `n` em que `n` é o argumento da função.

```
In:  print(multiplos23(100))
Out: 4
```

c) *max3prod*: defina uma função que recebe uma lista de números inteiros positivos e devolve o maior produto que for possível calcular multiplicando os 3 maiores inteiros da lista.

```
In:  print(max3prod([5,3,2,4,1]))
Out: 60 # Explicação: 5 * 4 * 3
```

d) *reduxInt*: defina uma função que recebe um número inteiro positivo como argumento e vai adicionar repetidamente os seu dígitos até obter apenas um dígito que é retornado como resultado.

```
In:  print(reduxInt(998)
Out: 8 # Explicação: 9 + 9 + 8 = 26; 2 + 6 = 8
```

Questão 3 (10v = 1+1+1+1+1+1+1+1+1+1)

Analise atentamente o seguinte modelo especificado em Python de uma unidade hospitalar:

```
# Hospital = [Unidade Hospitalar]
# Unidade Hospitalar = (nome_unidade, [Medico], [Camas])
# Medico = (id, nome)
# Cama = (id, ocupado)
# nome_unidade = String
# nome_medico = String
# id = Int
# ocupado = Bool
```

Como exemplo, considere a seguinte variável `hospital` como uma instância do modelo:

```
hospital = [
    (
        "Unidade de Cardiologia",
        [ (1, "Dr. João"), (2, "Dra. Maria")],
        [(101, False), (102, True), (103, False), (104, False)]
    ),
    (
        "Unidade de Pediatria",
        [ (3, "Dr. Pedro"), (4, "Dra. Clara") ],
        [ (201, True), (202, True), (203, True)]
    )
]
```

Após a sua análise, desenvolva as seguintes alíneas:

a) *listar_unidades*: Defina uma função que recebe um hospital como argumento e devolve a lista dos nomes das unidades hospitalares desse hospital.

```
In:  print(listar_unidades(hospital))
Out: ["Unidade de Cardiologia", "Unidade de Pediatria"]
```

b) *camas_disponiveis*: Defina uma função que recebe um hospital como argumento e devolve o número total de camas disponiveis nesse hospital.

```
In:  print(camas_disponiveis(hospital))
Out: 3
```

c) *listar_medicos_unidade*: Defina uma função que recebe um hospital e o nome de uma unidade hospitalar como argumento e devolve a lista dos médicos dessa unidade.

```
In:  print(listar_medicos_unidade(hospital, "Unidade de Cardiologia"))
Out: [(1, "Dr. João"), (2, "Dra. Maria")]
```

d) *existe_medico*: Defina uma função que recebe um hospital, o nome de uma unidade hospitalar e o nome de um médico, e verifica se o médico existe nessa unidade hospitalar, devolvendo `True` caso exista e `False` em caso contrário.

```
In:  print(existe_medico(hospital, "Unidade de Cardiologia", "Dra.
Maria"))
Out: True
```

e) *adicionar_medico*: Defina uma função que recebe um hospital, o nome de uma unidade hospitalar e o nome de um médico, e caso este não exista na unidade, acrescenta-o e devolve o novo hospital com o médico inserido.

```
In:  print(adicionar_medico(hospital, "Unidade de Cardiologia", "Dra.
Paula"))
Out: [("Unidade de Cardiologia",[(1, "Dr. João"), (2, "Dra. Maria"), (3,
"Dra. Paula")] ...)]
```

f) *ocupar_cama*: Defina uma função que recebe um hospital, o nome de uma unidade hospitalar e o identificador de uma cama, e caso a cama esteja livre devolve um novo hospital com a cama ocupada (não faz nada se a cama estiver ocupada).

```
In:  print(ocupar_cama(hospital, "Unidade de Cardiologia", 101))
Out: [("Unidade de Cardiologia",[...], [(101, True), (102, True)])...]
```

g) *guardar_hospital*: Defina uma função que recebe um hospital e o nome de um ficheiro e guarda a informação do hospital no ficheiro (o formato do ficheiro fica ao seu critério).

h) *carregar_hospital*: Defina uma função que recebe o nome de um ficheiro e carrega a informação do ficheiro para uma variável em memória que é devolvida como resultado (o formato do ficheiro deverá ser o mesmo que foi usado na alínea anterior).

i) *camas_ocupadas*: Defina uma função que recebe um hospital e devolve a percentagem de camas ocupadas do total de camas no hospital.

j) *unidades_ocupacao_critica*: Defina uma função que recebe um hospital e devolve uma lista de tuplos (pares), em que o primeiro elemento do tuplo é o nome de uma unidade hospitalar e o segundo é a percentagem de camas ocupadas nessa unidade, mas apenas deverão constar tuplos em que a percentagem é maior ou igual a 80%.

Bom trabalho e boa sorte

A equipa docente:

- José Carlos Ramalho
- Luís Filipe Cunha