Mestrado em Matemática e Aplicações

Especialização em Matemática Financeira

2019/2020, 1st semester

Métodos Computaxcionais

Teste #1 – 20 Dezembro 2019

Duração: 2 horas

Sem Consulta)

Aluno nº _____ Nome: _____

1. (1 pt) Qual o valor da variável **x** no final do seguinte programa.

```
i = 1;
x = 0;
while x < 25:
   x = x + i;
   i = 2 * i;
```

Resposta: x =

2. (1 pt) Que valor inteiro deve tomar a variável **k** para que, no final do programa, a variável **s** tenha o valor **13**.

```
M = [[2, -4, k], [-2, k, 6], [3, -1, k]]
s = 0;
for i in range(len(M)):
   for j in range(len(M[0])):
        s = s + M[i][j]
```

Resposta: k =

3. (1 pt) Depois de correr a sequência de instruções abaixo, qual o valor da variável **s**?

```
M = [[2,3,7,9,6,0], [1,5,4,5,1,3]]
s = 0
for i = range(len(M[0])):
   s = s + (M[0][i]- M[1][i])**2
s = math.sqrt(s)
```

Resposta: x =

4. **(1.5 pt)** Dado o ficheiro de texto com nome **`xpto.txt`** contendo o texto abaixo

```
This is a file with 3 lines
This line is the second of these lines.
And this is the last line.
```

Qual o valor retornado pela chamada da função **`check_chars ("xpto.txt","a", 3)`**?

```
def check_chars(fname, ch, k):
   "documentation omitted"
   c = 0
   fid = open(fname, "r")
   lines = fid.readlines()
   fid.close()
   for line in lines:
      for x in line:
         if x == ch:
            c = c + 1;
   return c < k
```

Resposta:

5. **(1.5 pt)** Qual o valor aproximado que deve ser esperado que seja retornado pela execução da função abaixo quando é feita a chamada **`dice_extremes(900)`**.

```
def dice_extremes(n):
   c = 0;
   for k in range(n):
      d = 1 + math.floor(6*random.random())
      if d == 1 or d == 6:
           c = c + 1;
   return c
```

Resposta:

6. **(1.5 pt)** Qual o valor final de **`Q`** calculado pelo programa abaixo?

```
Q = []
for i in range(1,4):
     L = []
     for j in range(1,6):
          L.append(i*j)
     Q.append(L)
```

Resposta: `Q =`

7. **(2 pt)** Complete a especificação da função abaixo (incluindo a sua documentação) para que, dada uma lista de listas LL, a função retorne um par (L, P), em que L é o índice da lista que contém o menor elemento de LL e P é o seu índice nessa lista. Por exemplo, para

```
LL = [[4,9,7,2], [3,5,0], [1,6], [8]]
```

a função deve retornar o par `(1,2)` já que o menor elemento ocorre na posição `2` da lista `1`.

Nota 1: A posição assume que em Python o primeiro elemento de uma lista tem índice 0.

Nota 2: No caso em que haja mais de um menor elemento, a função deve retornar a posição de um qualquer desses elementos.

```
def position_smallest(LL):
```

8. **(2 pt)** Complete a especificação da função abaixo (incluindo a sua documentação) para que retorne o número de vezes que um element da lista L é maior que o element anterior. Por exemplo, para `L = [1,3,2,6,4,9,5,8]`, deve retornar `4` (pois `3>1, 6>2 , 9>4, 8>5`).

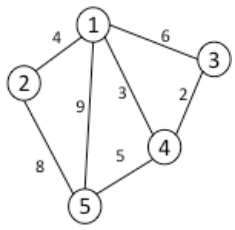
```
def increasing(L):
```

9. **(2.0 pt)** Como deve reconhecer, para uma lista V de números dada como entrada, a função abaixo retorna essa lista ordenada por ordem crescente (usando o algoritmos "bubble sort").

```
def bubble_sort(V):
    """ returns the (bubble) sorted version of vector V"""
    n = len(V)
    for k in range(n-1,0,-1):
        for i in range(0,k):
            if V[i] > V[i+1]:
                x = V[i]
                V[i] = V[i+1]
                V[i+1] = x
    return V
```

graph_x.txt

```
5; 7
1; 2; 4
1; 3; 6
1; 4; 3
1; 5; 9
2; 5; 8
3; 4; 2
4; 5; 5
```



Adapte o algoritmo acima para ordenar uma lista de arcos **A** de um grafo, por ordem decrescente dos seus pesos. Por exemplo, para o grafo indicado, a função pode receber a lista

```
A = [(1,2,4), (1,3,6), (1,4,3), (1,5,9), (2,5,8), (3,4,2), (4,5,5)]
```

(os arcos podem ser dados por uma ordem diferente) e deve retornar a lista

```
[(1,5,9), (2,5,8), (1,3,6), (4,5,5), (1,2,4), (1,4,3), (3,4,2)]
```

```
def sorted_ arcs(A):
   """Returns a graph's arcs A in decreasing order of their weights"""
```

10. (3.5 pt) Em muitos casos, é feita uma escolha aleatória, tendo em conta o seu peso, para atribuir uma tarefa a um membro de um grupo. Assumindo uma lista de triplos (idade, nome, peso), especifique uma função que retorne os primeiros e últimos nomes de um elemento escolhido aleatoriamente e proporcionalmente ao seu peso. Por exemplo, se a lista é composta pelos triplos

```
[ (45, "Ana Marta da Silva Lopes", 8),
  (23, "Pedro Miguel Costa Matos", 4),
  (39, "Clara Isabel Campos e Cunha Teles", 1),
  (56, "Carlos Jorge Santos Abreu Pinto",2)
]
```

Os possíveis nomes retornados são

"Ana Lopes", **"Pedro Matos"**, **"Clara Teles"** ou **"Carlos Pinto"**,

em que o primeiro tem o dobro da probabilidade do segundo, que tem o dobro da probabilidade do quarto, que tem o dobro da probabilidade do terceiro.

Nota: Em alternativa, retorne qualquer deles com igual probabilidade (penalização: 1.5 valores).

```
def random_choice(L):
    """Returns the first and last name of a randomly chosen triple of T"""
```

11. (3.5 pt) Assuma que é dada um conjunto de localizações numa aldeia que devem ser ligados à rede de água. A canalização pode ser colocada em várias ruas, com custos representados por um grafo pesado não dirigido, especificado no formato DIMACS num ficheiro com nome **`fname`**. Infelizmente, existe uma localização extra não considerada no ficheiro, mas em que os custos a cada uma das outras localizações são dados numa lista D, e que deverá ser considerada na infraestrutura de abastecimento de água.

Nestas condições, especifique uma função que retorne um triplo **`(T, p, d)`** em que **`T`** é a matriz de adjacência do grafo (estendido) que contem apenas os arcos que devem ser considerados para a infraestrutura, **`d`** representa a distância entre o nó extra e os outros nós (segundo a infraestrutura **`T`** adotada), e **`p`** é o nó mais afastado do nó extra.

- **Sugestão 1**: Pode usar funções auxiliares (e.g. as do anexo).
- **Sugestão 2**: Especifique uma função auxiliar, **`add_node(G,D)`**, que retorna a matriz **`G`** de adjacências original, aumentada com o nó cujas distâncias diretas aos outros nós é dada na lista **`D.`** (Note que pode usar esta função, mesmo que não a implemente; penalização de 1 valor)

```
def cheapest_water_distribution (G):
    """ returns the cheapest water distribution facility, when a node
    is added with distances D to all nodes of the original graph G as
    well as the distance to the farthest node """
```

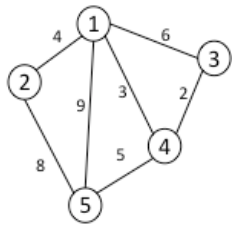
Annex

In question 11, you should consider the functions that were studied in the classes regarding weighted undirected graphs where the weights may be interpreted as distances between vertices of the graph. If no edge exists between two vertices of a graph G = <V, E>, a virtual edge with value -1 is assumed in the adjacency matrix G of the graph.

- **`def dimacs_read(filename)`**
  - returns the adjacency matrix, **`G`**, of a weighted undirected graph specified in file with **`filename`**. The first line of the graph contains the number of vertices and arcs, and the subsequent lines the triple **`<n1, n2, w>`**, where **`w`** is the weight of the edge connecting the vertices **`n1`** and **`n2`** (**`n1 < n2`**). The integers **`n1`**, **`n1`** and **`w`** are separated by semicolons ("**`;`**").

graph_x.txt

```
5; 7
1; 2; 4
1; 3; 6
1; 4; 3
1; 5; 9
2; 5; 8
3; 4; 2
4; 5; 5
```




```
G =
   0    4    6    3    9
   4    0   -1   -1    8
   6   -1    0    2   -1
   3   -1    2    0    5
   9    8   -1    5    0
```

- **`def write_graph(G, filename)`**
  - writes a graph, given by its adjacency matrix **`G`**, in file with name **`filename`**, with the format specified above.

- **`def prim(G)`**
  - Given an undirected weighted graph **`G`**, it returns a minimum spanning tree, **`T`**. Both **`T`** and **`G`** are represented by the corresponding adjacency matrices. For the graph above, **`prim(G)`** returns

```
T =
   0    4   -1    3   -1
   4    0   -1   -1   -1
  -1   -1    0    2   -1
   3   -1    2    0    5
  -1   -1   -1    5    0
```

- **`def floyd(G)`**
  - returns a matrix **`S`** with the shortest distances between any two vertices of the graph specified by its adjacency matrix, **`G`**. For the graph above **`floyd(G)`** returns

```
S =
   0    4    5    3    8
   4    0    9    7    8
   5    9    0    2    7
   3    7    2    0    5
   8    8    7    5    0
```