Mestrado em Matemática e Aplicações

Especialização em Matemática Financeira

2021/2022, 1º semestre

Métodos Computacionais

Teste – 3 dezembro 2021

Duração: 2 hours

(Sem consulta)

Student nº _____ Name: _____

1. (1 pt) Qual o valor retornado pela chamada **`q1(5)`**?

```
def q1(n):
   """documentation omitted."""
   i = 1
   x = n
   s = 1
   while abs(x) < 10:
      x = x + s * i
      s = -s
      i = 3 * i
```

Resposta: `x = 12`

2. (1 pt) Que valor inteiro deve ter **`k`** de forma que a chamada **`q2(k)`** retorne **`[1, 3, 9]`**?

```
def q2(k):
   """documentation omitted."""
   M = [[2,-4,k], [-2,k,6], [3,-1,k]]
   S = [];
   for i in range(len(M)):
      S.append(M[i][i] ** i)
   return S
```

Resposta: `k = -3`

3. (1 pt) Qual o valor retornado pela chamada **`q3(3)`**?

```
def q3(k):
   """documentation omitted"""
   M = [[2,3,7],[1,6,0],[3,5,4]]
   s = k
   for i in range(len(M)):
      for j in range(M[i][0]):
         s = s + j**2
   return s
```

Resposta: `x = 8`

4. **(1.5 pt)** Assuma que o ficheiro texto com nome **`xpto.txt`** contem 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 **`count_x("xpto.txt")`**?

```
def count_x(fname):
    """documentation omitted"""
    fid = open(fname, "r")
    lines = fid.readlines()
    fid.close()
    np = []
    for line in lines:
        ll = line.strip().split(" ")
        np.append(len(ll))
    return np
```

Resposta: `[7, 8, 6]`

5. **(1.5 pt)** Qual o valor aproximado que deverá ser esperado como retorno da chamada **`dice_extremes(3600)`** ?

```
def two_dices(n):
    """documentation omitted"""
    c = 0;
    for k in range(n):
        d1 = 1 + math.floor(6*random.random())
        d2 = 1 + math.floor(6*random.random())
        if d1 + d2 == 12:
            c = c + 1;
    return c
```

Resposta: `100`

6. **(1.5 pt)** Qual o valor retornado pela chamada **`mat_sq(4)`**?

```
def mat_sq(k):
    """documentation omitted"""
    Q = []
    for i in range(1,k):
        L = [(i-j)**2 for j in range(1,k)]
        Q.append(L)
    return Q
```

Resposta: `Q =`

```
[
 [0, 1, 4],
 [1, 0, 1],
 [4, 1, 0]
]
```

7. **(2.5 pt)** Complete a especificação da função **`highest_BMI`** (incluindo a sua documentação) de forma que, dada uma lista de dicionários, **`People`**, com chaves "name", "height" e "weight", retorna um tuplo com o nome e BMI da pessoa com o maior BMI (Body Mass Index). **Nota**: o BMI de uma pessoa é definido como **weight/height**2** dessa pessoa. Por exemplo, para

```
People = [ {"name":"joe", "height":1.80, "weight": 71},
           {"name":"ann", "height":1.70, "weight": 57},
           {"name":"pete","height":1.75, "weight": 85}]
```

a função deve retornar o tuplo (**`"pete"`**, 2.76), que é a pessoa com maior BMI (o seu BMI tem o valor **27.76** = 85/(1.75*1.75)), maior que o da **`ann`** (19.72) e o do **`joe`** (21.91).

```
def highest_BMI(People):
    """returns the name and bmi of the person, in the
     list of dicionnaries Ps, with highest BMI."""
     hi = 0
     for P in Ps:
         bmi = P['weight']/P['height']**2
         if bmi > hi:
            name = P["name"]
            hi = bmi
     return name, bmi
```

8. **(2.5 pt)** Complete a especificação da função abaixo (incluindo a sua documentação) de forma queretorne a media móvels das temperaturas na lista **`T`**. A função retorna uma lista em que cada valor de **`T`**, corresponde a media das últmas **`k`** temperaturas (ou menos no início da lista). Por exemplo, para a lista **`T = [12, 15, 18, 15, 12, 9]`** a função deverá retornar a lista **`[12, 13.5, 15, 16, 15, 12]`**, i.e

`[12, (12+15)/2, (12+15+18)/3, (15+18+15)/3, (18+15+12)/3, (15+12+9)/3]`

```
def mov_average(L, k):
    """returns the moving average of the last k elements
    of list T (less in the bbeguning)."""
    L = []
    for i in range(len(T)):
        j = max(0, i+1-k)
        print (j,i)
        n = i+1-j
        s = sum(T[j:i+1])
        L. append(s/n)
    return L
```

9. (2.5 pt) Dada uma lista **L** com os valores de **n** itens (em que **n** pode ser **ENORME**), o seu objetivo é descobrir um conjunto de 3 itens cujo valor somado seja tão próximo quanto possível de um valor **v**. Como **n** é mito grande, deverá usar uma abordagem estocástica que consiste em selecionar aleatoriamente 3 itens e comparar o seu valor com **v**. Esse procedimento deve ser repetido **n_it** vezes e o triplo retornado será o que mais se aproximar de **v**. **Nota:** Deverá retornar o índice que os itens ocupam na lista e não o seu valor.

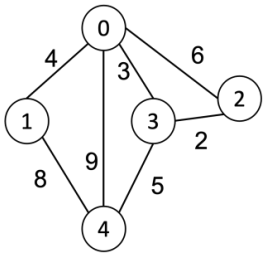
```
def select_triple(L, v, n_it):
    """Returns a triple of values from list L, whose
    sum is closest to v, byselecting randomly 3 items
    and repeating the procedure n_it times."""
    dif = math.inf
    n = len(L)
    for i in range(n_it):
        i1 = math.floor(n*random.random());
        i2 = math.floor(n*random.random());
        i3 = math.floor(n*random.random());
        if abs (v - L[i1]+ L[i2] + L[i3]) < dif:
            dif = abs (v - L[i1]+ L[i2] + L[i3])
            b1 = i1
            b2 = i2
            b3 = i3
    return b1, b2, b3
```

10. (2.5 pt) Dado um grafo pesado não direcionado **G**, o algoritmo **prim** retorna a árvore de cobertura mínima (MS), se existir. Assumindo uma implementação da função **prim** em que quer o grafo **G** quer a sua **MST**, **T**, são representadas pela sua matriz de adjacências (ver anexo), o problema a resolver é o seguinte.

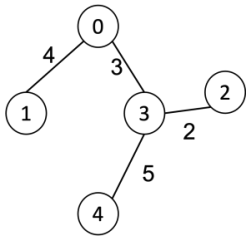
Especifique a função **MST_removed**, que tem como argumentos um grafo **G** e uma lista **L** com alguns dos seus arcos **(i,j)**, e que retorna a nova MST para o grafo (depois de remover os arcos da lista **L**) e o seu custo (i.e. a soma dos pesos dos seus arcos.

Por exemplo, para o grafo **G** abaixo, a sua **MST** é **T**, com um custo de **14** (=4+3+2+5), mas removendo os arcos **[(0,3),(1,4)]** a sua **MST** passa a ser **U**, com custo **17 (=4+6+2+5)**.

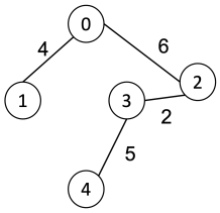
G =



T =



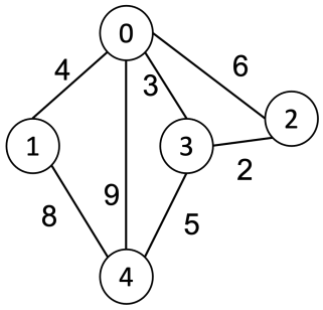
U =



def MST_removed(G,As):

```
    """Returns the Minimum Spanning Tree, and its cost,
    of graph G, after the arcs in As being removed."""
    H = G.copy()
    for a in As:
        H[a[0]][a[1]] = -1
        H[a[1]][a[0]] = -1
    U = prim(H)
    c = 0
    for i in range(len(G)):
        for j in range(len(G)):
            c = c +  U[i][j]
    return U, c
```

11. (2.5 pt) Um amigo apresenta-lhe um caminho **P** num grafo **G**, entre dois nós **p** e **q** (o grafo pode representar as estradas do mapa de uma região que ligam povoações nessa região).



O seu objetivo é descobrir o caminho de menor custo entre os nós **p** e **q**, e indicar qual a poupança obtida por usar este caminho ótimo, por comparação com o caminho que lhe foi apresentado pelo seu amigo.

Por exemplo, no grafo da direita (já usado no problema anterior), se o seu amigo lhe apresentar o caminho **[1,0,2]** deverá dizer-lhe que o caminho ótimo entre os nós **1** e **2** é, de facto, **[1,0,3,2]** que permite a poupança de 1 unidade de distância. (4+3+2= **9** vs. 4+6 = **10**).

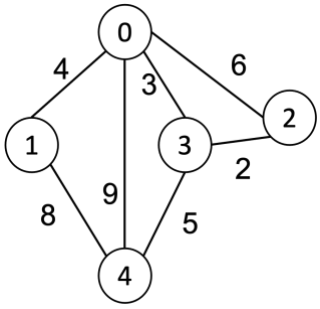
Especifique a função **better_path** que toma como argumentos um grafo **G** e um caminho **P** e retorna o melhor caminho entre os nós inicial e final de P, bem como a distância poupada através do caminho ótimo. No exemplo, deverá retornar o tuplo **([1,0,3,2], 1)**.

Suggestion: Defina a função **path_length(P,G)** para obter o custo de um caminho **P** no grafo **G**.

```
def better_path(G,P):
    """Returns the best path that between the initial and
    Final nodes of P, together with the amount saved by using
    this shortest path."""
    c1 = path_length(P, G)
    S, N = floyd(G)
    Q = path(P[0], P[-1], N)
    c2 = path_length(Q, G)
    return Q, c1-c2









def path_length(P, G):
    """Returns the length of a path P in graph G."""
    c = 0
    i = 0
    while i+1 < len(P):
        c = c + G[P[i]][P[i+1]]
        i = i+1
    return c
```

Annex

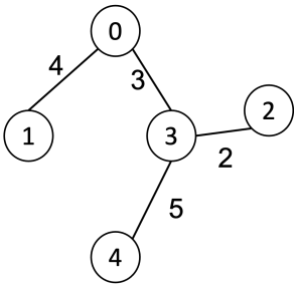
In your answers, you may consider functions that were studied in the classes regarding weighted undirected graphs, where the weights may be interpreted as distances between vertices of the graph. If no edge exists between two vertices of a graph G = <V, E>, a virtual edge with value -1 is assumed in the adjacency matrix G of the graph. Take the adjacency matrix of graph G, shown below (together with its dimacs format).




```
G = [ [-1, 4, 6, 3, 9],
      [ 4,-1,-1,-1, 8],
      [ 6,-1,-1, 2,-1],
      [ 3,-1, 2,-1, 5],
      [ 9, 8,-1, 5,-1]]
```

- **`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 Tree T (shown in the right).
  -

```
T =[  [-1, 4,-1, 3,-1],
      [ 4,-1,-1,-1,-1],
      [-1,-1, 0, 2,-1],
      [ 3,-1, 2,-1, 5],
      [-1,-1,-1, 5,-1]]
```



- **`def floyd(G)`**
  - returns a matrix **`S`** with the shortest distances between any two vertices of the graph specified by its adjacency matrix, **`G`**, together with a matrix **`N`**, where **`N[i][j]`** denotes the next node, from node **`i`**, in the shortest path from **`i`** to **`j`**.
    For the graph above **`floyd(G)`** returns matrices

```
S = [ [6, 4, 5, 3, 8],
      [4, 8, 9, 7, 8],
      [5, 9, 4, 2, 7],
      [3, 7, 2, 4, 5],
      [8, 8, 7, 5,10]]
```

```
N = [ [0, 1, 3, 3, 3],
      [0, 1, 0, 0, 4],
      [3, 3, 2, 3, 3],
      [0, 0, 2, 3, 4],
      [3, 1, 3, 3, 4]]
```

- **`def path(Next,p,q)`**
  - returns the shortest path **`P`** between nodes **`p`** and **`q`**, in a graph **`G`**, where **`Next`** is the matrix **`N`** obtained applying the **`Floyd`** function (above) to graph **`G`**. In the example above

`path(Next,1,2) = [1,0,3,2]`