MAC0122 Princípios de Desenvolvimento de Algoritmos

Bacharelado em Estatística, Matemática e Matemática Aplicada

Segunda Prova – 20 de outubro de 2016

Nome: _____

Assinatura: _____

Nº USP: _____ Prof: _____

Instruções:

1. Não destaque as folhas deste caderno. A prova pode ser feita a lápis.
2. A prova consta de 3 questões. A questão 2 tem 3 itens e a questão 3 tem 2 itens. Verifique antes de começar a prova se o seu caderno está completo.
3. As questões podem ser resolvidas em qualquer página. Ao escrever uma solução (ou parte dela) em página diferente do enunciado, escreva QUESTÃO X em letras ENORMES junto da solução.
4. As soluções devem ser em Python. Você pode usar apenas recursos de Python vistos em aula. Você pode definir funções auxiliares e usá-las à vontade. Cuidado com a legibilidade e, principalmente, com a TABULAÇÃO.
5. As soluções não precisam verificar consistência de dados.
6. Não é permitido o uso de folhas avulsas para rascunho, a consulta a livros, apontamentos, colegas ou equipamentos eletrônicos. Desligue o seu celular e qualquer equipamento que possa perturbar o andamento da prova.

DURAÇÃO DA PROVA: 2 horas

Questão	Valor	Nota
1	2,5	
2	3,5	
3	4,0	
Total	10,0	

Questão 1 (vale 2,5 pontos)

Escreva uma função **recursiva** `verifica_min_heap()` que respeite a especificação a seguir. Qual o **consumo de tempo** de sua função em relação ao número `n` de elementos na lista `v`?

```
def verifica_min_heap(v):
    ''' (list) -> bool

    Recebe uma lista v de números inteiros e retorna True
    se a lista for um min-heap e False caso contrário.

    A lista começa do indíce 1 (mas lembre-se que o indice 0
     existe em Python). Os exemplos abaixo mostram a lista
    com valor None no índice 0.

    Exemplos:
    >>> verifica_min_heap( [None, 1] )
    True
    >>> verifica_min_heap( [None, 2, 1] )
    False
    >>> verifica_min_heap( [None, 1, 2, 1] )
    True
    >>> verifica_min_heap( [None, 1, 4, 2, 5] )
    True
    >>> verifica_min_heap( [None, 1, 4, 2, 5, 1] )
    False
    >>>
    '''
```

Questão 2 (vale 3,5 pontos)

Uma **fatia** de uma lista `v` é qualquer sublista da forma `v[e:d]`.

> **Problema**: dada uma lista `v[p:r]` de números inteiros, encontrar uma fatia `v[e:d]` de soma máxima.

Esta questão é composta de três itens que resolvem o problema acima de duas maneiras diferentes: *força-bruta* e *divisão e conquista*.

Nesta questão você pode utilizar a função `sum()` que é nativa do Python. Essa função recebe uma lista e retorna a soma de seus elementos. Exemplo:

```
>>> sum([3, -2, 4])
5
```

Item (a) (vale 1 ponto)

Escreva uma função `fatia_max()` que respeite a especificação a seguir.
Qual o **consumo de tempo** de sua função em relação ao número `n` de elementos na lista `v[p:r]`?

```
def fatia_max(p, r, v):
    '''(int, int, list) -> int, int, int

    Recebe uma lista v[p:r] de números inteiros e retorna
    valores inteiros soma_max, e, d tais que

        soma_max == sum(v[e,d]) é a maior soma de uma fatia
        v[i:j] com p <= i <= j <= r.

    Exemplos:

    >>> v = [31, -41, 59, 26, -53, 58, 97, -93, -23, 84]
    >>> fatia_max(0,10,v)
    (187, 2, 7)
    >>> fatia_max(2,9,v)
    (187, 2, 7)
    >>> fatia_max(3,9,v)
    (155, 5, 7)
    >>> fatia_max(7,10,v)
    (84, 9, 10)
    >>>
    '''

    # inicialmente, a fatia vazia é a de soma máxima
    soma_max, e, d = 0, p, p

    for i in range(p,r):
        for j in range(i+1,r+1):
            soma = sum(v[i:j])
            if soma > soma_max:
                soma_max, e, d = soma, i, j
    return soma_max, e, d
```

Item (b) (vale 1 ponto)

Escreva uma função `fatia_max_meio()` que respeite a especificação a seguir.
Qual o **consumo de tempo** de sua função em relação ao número `n` de elementos na lista `v[p:r]`?

```
def fatia_max_meio(p, q, r, v):
    '''(int, int, int, list) -> int, int, int

    Recebe números inteiros p < q < r e uma lista v[p:r] de
    números inteiros e retorna valores inteiros
    soma_max, e, d tais que

        soma_max==sum(v[e:d]) é a maior soma de uma fatia
        v[i:j] com p <= i < q < j <= r.

    Exemplos:

    >>> v = [31, -41, 59, 26, -53, 58, 97, -93, -23, 84]
    >>> fatia_max_meio(0,1,2,v)
    (-10, 0, 2)
    >>> fatia_max_meio(0,2,3,v)
    (49, 0, 3)
    >>> fatia_max_meio(0,2,4,v)
    (75, 0, 4)
    >>> fatia_max_meio(0,2,5,v)
    (75, 0, 4)
    >>> fatia_max_meio(0,2,6,v)
    (80, 0, 6)
    >>> fatia_max_meio(0,3,6,v)
    (90, 2, 6)
    >>> fatia_max_meio(0,3,5,v)
    (85, 2, 4)
    >>> fatia_max_meio(3,5,6,v)
    (31, 3, 6)
    >>> fatia_max_meio(3,5,7,v)
    (128, 3, 7)
    >>>
    '''

    # inicialmente a fatia de soma máxima é v[q-1:q+1]
    soma_max, e, d = v[q-1]+v[q], q-1, q+1

    # teste todas as possíveis fatias
    # podíamos fazer uma solução linear aqui.
    for i in range(p,q):
        for j in range(q+1,r+1):
            soma = sum(v[i:j])
            if soma > soma_max:
                soma_max, e, d = soma, i, j

    return soma_max, e, d
```

Item (c) (vale 1,5 ponto)

Escreva uma função `fatia_max_div_conq()` que respeite a especificação a seguir.
Esta função deve utilizar a estratégia de **divisão e conquista**.

```
def fatia_max_div_conq(p, r, v):
    '''(int, int, list) -> int, int, int

    Recebe uma lista lista v[p:r] de números inteiros e retorna
    valores inteiros soma_max, e, d tais que

        soma_max == sum(v[e,d]) é a maior soma de uma fatia
        v[i:j] com p <= i <= j <= r.

    Exemplos:
    >>> v = [31, -41, 59, 26, -53, 58, 97, -93, -23, 84]
    >>> fatia_max_div_conq(0,10,v)
    (187, 2, 7)
    >>> fatia_max_div_conq(2,9,v)
    (187, 2, 7)
    >>> fatia_max_div_conq(3,9,v)
    (155, 5, 7)
    >>> fatia_max_div_conq(7,10,v)
    (84, 9, 10)
    >>>
    '''

    if r-p < 3: # n = r-p < 3
        return fatia_max(p, r, v)

    # dividir
    q = (p + r)//2

    # conquistar
    soma_max_e, p_e, r_e = fatia_max_div_conq(p, q, v)
    soma_max_d, p_d, r_d = fatia_max_div_conq(q, r, v)

    # combinar
    soma_max_q, p_q, r_q = fatia_max_meio(p, q, r, v)
    if soma_max_e > max(soma_max_d, soma_max_q):
        soma_max, e, d = soma_max_e, p_e, r_e
    elif soma_max_d > soma_max_q:
        soma_max, e, d = soma_max_d, p_d, r_d
    else:
        soma_max, e, d = soma_max_q, p_q, r_q

    return soma_max, e, d
```

Questão 3 (vale 4 pontos)

Está questão é composta de dois itens.
Item (a) (vale 2 ponto)

Escreva uma classe de nome `Vetor2D` que:

- a fim de criar um objeto da classe, o construtor recebe dois reais `x` e `y`. O objeto representa o vetor `(x,y)`. Se `x` e `y` não dados o objeto criado deve representar o vetor `(0,0)`. Exemplos:
```
>>> orig = Vetor2D()
>>> v = Vetor2D(4.7, 2.3)
```

- quando um objeto da classe `Vetor2D` é exibido através função `print()`, as coordenadas do vetor são apresentadas entre parenteses. Utilize o formato `"%.3g"`. Exemplo:
```
>>> print(orig)
(0,0)
>>> print(v)
(4.7, 2.3)
```

- possua um método `__sub__()` para subtrair um vetor de um outro vetor. Exemplo:
```
>>> u = Vetor2D(2,1)
>>> v = Vetor2D(1.9,1.1)
>>> w = u - v
>>> print(w)
(0.1,-0.1)
```

- possua um método `__mul__()` que multiplica um vetor por um valor `int` ou `float`. Exemplo:
```
>>> u = Vetor2D(3.1,-2)
>>> w = u * 0.5
>>> print(w)
(1.55,-1)
```

- possua um método `distancia()` que calcula a distância entre dois vetores. Exemplos:
```
>>> u = Vetor2D(1,1)
>>> v = Vetor2D(-1,1)
>>> orig = Vetor2D()
>>> orig.distancia(u)
1.4142135623730951
>>> orig.distancia(v)
1.4142135623730951
>>> u.distancia(v)
2.0
```

- possua um método `leitura`, que leia do teclado dois valores reais e os atribui às coordenadas `x` e `y` do objeto `Vetor2D`. Exemplo:
```
>>> u = Vetor2D()
>>> u.leitura()
Digite x: 3.14
Digite y: 2.71
>>> print(u)
(3.14,2.71)
```

```
class Vetor2D:
    def __init__(self, a=0, b=0):
        self.x = a
        self.y = b

    def __str__(self):
        s = '(%.3g, %.3g)'%(self.x, self.y)
        return s

    def __sub__(self, other):
        x = self.x - other.x
        y = self.y - other.y
        return Vetor2D(x, y)

    def __mul__(self, alfa):
        x = alfa * self.x
        y = alfa * self.y
        return Vetor2D(x, y)

    def leitura(self):
        self.x = float(input("Digite x: "))
        self.y = float(input("Ditite y: "))

    def distancia(self, other):
        d = self - other
        d2 = d.x*d.x + d.y*d.y
        return d2 ** 0.5
```

Item (b) (vale 2 ponto)

Escreva uma classe de nome `Nave` que:

- a fim de criar um objeto da classe, o construtor recebe um string `nome`, um objeto `pos` da classe `Vetor2D` e um `float vel`. Esse objeto representa uma nave espacial `nome`, que está na posição `pos` e tem **velocidade de cruzeiro** `vel`. Exemplo:
  ```
  >>> nave_x = Nave("Enterprise", Vetor2D(2.3, 5.7), 795.0)
  ```

- quando um objeto `Nave` é exibido pela função `print()`, as informações são apresentadas no seguinte formato:
  ```
  >>> print(nave_x)
  Enterprise [v = 795.0] : (2.3, 5.7)
  ```

- possua um método `viagem()`, que recebe um objeto `nova_pos` da classe `Vetor2D` e realiza a viagem da nave até essa nova posição. O método deve atualizar a posição da nave para essa nova posição e retornar o tempo necessário para realizar a viagem na velocidade de cruzeiro. As coordenadas das posições estão em quilômetros (`km`) e a velocidade é em quilômetros por segundo (`km/s`).

```
class Nave:
    def __init__(self, no=0, po=0, ve=0):
        self.nome = no
        self.pos  = po
        self.vel  = ve

    def __str__(self):
        s = "%s [%.2f]: %s"%(self.nome, self.vel, self.pos) # funciona sem str(self.pos)
        return s

    def leitura(self):
        self.nome = input("Digite o nome da nave: ")
        self.vel  = float(input("Digite a velocidade de cruzeiro: "))
        self.pos.leitura()

    def viagem(self, nova):
        '''
        faz a nave viajar para uma nova posição.
        O método atualiza a posição da nave e
        retorna o tempo necessário para percorrer
        a distância entre a posição atual da nave
        e a nova posição.
        '''
        distancia = self.pos.distancia(nova)
        tempo = distancia / self.vel
        self.pos = nova
        return tempo
```