www.estudar.com.vc

Lista de Exercícios

P3 2018

Programação em Python Poli

USP

1. Tema

Simule o código abaixo e selecione as opções correspondentes à saída impressa do programa:

```
def g(L, v, a, b, n):
    c = (a+b)//2
    print(a,c,b)
    if (v==L[c])   :   return n-c;
    elif (a==c)    :   return -1;
    elif (v>L[c])  :   return g(L,v,a,c, n);
    elif (v<L[c])  :   return g(L,v,c,b, n);
    else:
        return -;

def f(L, v, n):
    return g(L, v, 0, 4, n(;

print(f([4,3,1,-5], -5, 4));
print(g([11,5,4,1], 5, 1, 4, 4));
```

Abaixo estão listadas as impressões, em ordem (primeira linha é a primeira impressão, segunda linha é a segunda impressão, e assim por diante). Selecione a única correta por linha:

a. A primeira impressão é:

A. $0,4,2$ **B.** -1 **C.** $0,-5,4$ **D.** $0,2,5$ **E.** 3 **F.** $2,3,4$ **G.** $0,2,4$ **H.** $0,3,4$

b. A segunda impressão é:

A. $0,4,4$ **B.** $2,3,4$ **C.** $2,2,4$ **D.** $0,4,-5$ **E.** 3 **F.** -1 **G.** $0,3,4$ **H.** $1,2,4$

c. A terceira impressão é:

A. $0,2,5$ **B.** 1 **C.** $0,4,2$ **D.** -1 **E.** $1,3,4$ **F.** 3 **G.** $1,2,4$ **H.** $0,-5,4$

d. A quarta impressão é:

A. $0,4,2$ **B.** 3 **C.** $1,2,4$ **D.** $0,2,4$ **E.** $2,3,4$ **F.** $0,4,-5$ **G.** -1 **H.** $0,2,5$

e. A quinta impressão é:

A. $1,1,4$ **B.** $2,3,4$ **C.** $1,1,2$ **D.** $0,2,4$ **E.** 3 **F.** $0,-5,4$ **G.** -1 **H.** $0,5,2$

f. A sexta impressão é:

A. $0,2,4$ **B.** -1 **C.** $0,2,5$ **D.** $1,2,1$ **E.** $1,2,4$ **F.** $1,1,2$ **G.** 1 **H.** 3

2. Tema

As 5 funções abaixo foram projetadas para ordenar qualquer lista de valores inteiros de modo a resultar em lista de ordem decrescente. Entretanto, alguns falham para algumas configurações de lista:

I.

```
def ordena (lista):
    for i in range(len(lista)-1):
```

```
        for j in range(len(lista)-1):
            if (lista[j+1] >= lista[j]):
                q = lista[j];
                lista[j] = lista[j+1];
                lista[j+1] = q;
```

II.

```
def ordena (lista):
    i = 0;
    while (i<len(lista)):
        x = lista[i];
        j = i-1;
        while (j>=0 and lista[j]<x):
            lista[j+1] = lista[j];
            j = j-1;
        lista[j+1] = x;
        i += 1;
```

III.

```
def ordena (lista):
    for i in range(len(lista)-1):
        for j in range(len(lista)-2):
            if (lista[j+1] > lista[j]):
                q = lista[j];
                lista[j] = lista[j+1];
                lista[j+1] = q;
```

IV.

```
def ordena (lista):
    for i in range(len(lista)):
```

```
        maior = lista[i];
        indice = i;
        for j in range(i, len(lista)):
            if (lista[j] > maior):
                maior = lista[j];
                índice = j;
                lista[indice] = lista[i];
```

V.

```
def ordena (lista):
    for i in range(len(lista)):
        maior = lista[i];
        indice = i;
        for j in range(i, len(lista)):
            if (lista[j] > maior):
                maior = lista[j];
                índice = j;
        if (indice != i):
            x = lista[indice];
            lista[indice] = lista[i];
            lista[i] = x;
```

Quais alternativas abaixo contêm códigos que ordene (de forma decrescente) qualquer configuração inicial da lista:

A. V
B. II
C. I
D. III
E. IV

3. TEMA

Elaboração própria

Supondo que uma matriz M_{mxn} tenha, em cada linha, seus elementos ordenados de forma crescente, monte a função $acrescenta(.)$ para que ela seja invocada pela função $ordena\_matriz(.)$, de modo a gerar uma ordenação crescente completa dos elementos da matriz (ou seja, ao final da execução de $ordena\_matriz(.)$, seja gerado um vetor v com todos os dados iniciais de M $(m_{00}, m_{01}, \dots, m_{10}, m_{11}, \dots, m_{m-1,n-1})$, de modo que $v_0 \leq v_1 \leq v_2 \leq \cdots \leq v_{mxn-1})$.

Por exemplo, para a matriz $3x4$ abaixo, a saída seria o vetor $v = (0, 1, 1, 2, 3, 5, 6, 7, 9, 10, 15, 18)$:

$$M = \begin{pmatrix} 1 & 2 & 10 & 18 \\ 1 & 3 & 9 & 15 \\ 0 & 5 & 6 & 7 \end{pmatrix}$$

```
def ordena_matriz(M, nlnhs, ncols):
    v1 = [0]*(nlnhs*ncols);
    v2 = [0]*(nlnhs*ncols);
    prontos = 0;
    for linha in range(nlnhs):
        if (linha %2 == 0):
            acrescenta(M, linha, v1,v2, prontos, ncols);
        else:
            acrescenta(M, linha, v2,v1, prontos, ncols);
        prontos += ncols
    if (nlnhs %2 == 0):  print(v1);
    else:  print(v2);
```

```
def acrescenta (M, lnh, vetAnt, vetNov, prontos, ncols):
    c_ant = 0; c_nov = 0; c_mat = 0;
    while (L1):
        if (L2):
            L3
            c_mat += 1; c_nov += 1;
        elif (L4):
            L5
            c_ant += 1; c_nov += 1;
        elif (L6):
            L7
            c_mat += 1; c_nov += 1;
        else:
            L8
            c_ant += 1; c_nov += 1;
    print("parcial:", vetNov);
```

Para cada um dos 8 itens a seguir, correspondendo às lacunas no código acima, assinale a única resposta que torna o programa acima correto:

L1:

A. $vetAnt[c\_ant] < vetNov[c\_nov]$

B. $c\_mat < prontos \ or \ c\_ant < ncols$

C. $c\_mat < ncols \ or \ c\_ant < prontos$

D. $c\_mat < ncols$

E. $M[c\_ant][c\_nov]$

F. $M[c\_ant][c\_mat]$

G. $c\_ant < prontos$

L2:

A. $prontos < c\_nov$

B. $prontos < c\_ant$

C. $c\_ant < c\_nov$

D. $c\_ant < prontos$

E. $c\_ant <= prontos$

F. $c\_ant > prontos$

G. $c\_ant >= prontos$

L3:

A. $vetAnt[c\_ant] = vetAnt[c\_mat]$

B. $vetAnt[c\_nov] = M[lnh][c\_mat]$

C. $vetNov[c\_mat] = M[lnh][c\_nov]$

D. $vetAnt[c\_ant] = vetAnt[c\_nov]$

E. $vetNov[c\_nov] = M[lnh][c\_mat]$

F. $vetNov[c\_nov] = vetAnt[c\_ant]$

G. $vetNov[c\_mat] = vetAnt[c\_nov]$

L4:

A. $c\_mat >= c\_nov$

B. $c\_mat >= ncols$

C. $c\_nov >= ncols$

D. $c\_mat > ncols$

E. $c\_ant < ncols$

F. $c\_ant < prontos$

G. $c\_nov > ncols$

L5:

A. $vetAnt[c\_ant] = vetNov[c\_nov]$
B. $vetNov[c\_ant] = vetAnt[c\_nov]$
C. $vetNov[c\_nov] = M[c\_ant]\ [c\_nov]$
D. $vetNov[c\_nov] = M[c\_ant][c\_nov]$
E. $vetNov[c\_nov] = vetAnt[c\_ant]$
F. $vetAnt[c\_ant] = M[c\_ant][c\_nov]$
G. $vetAnt[c\_ant] = vetNov[c\_mat]$

L6:
A. $vetAnt[c\_ant] > M[lnh]\ [c\_mat]$
B. $vetAnt[c\_ant] < vetNov[c\_nov]$
C. $vetAnt[c\_nov] < vetNov[c\_ant]$
D. $vetAnt[c\_nov] > M[lnh][c\_mat]$
E. $vetAnt[c\_ant] < M[lnh][c\_mat]$
F. $vetAnt[c\_ant] < vetNov[c\_nov]$
G. $vetAnt[c\_nov] < M[lnh][c\_mat]$

L7:
A. $vetNov[c\_nov] = M[lnh][c\_mat]$
B. $vetAnt[c\_ant] = vetAnt[c\_mat]$
C. $vetAnt[c\_nov] = M[lnh][c\_mat]$
D. $vetAnt[c\_ant] = vetAnt[c\_nov]$
E. $vetNov[c\_mat] = M[lnh][c\_nov]$
F. $vetNov[c\_mat] = vetAnt[c\_nov]$
G. $vetNov[c\_nov] = vetAnt[c\_ant]$

L8:
A. $vetNov[c\_nov] = vetAnt[c\_ant]$

B. $vetNov[c\_nov] = M[c\_ant][c\_nov]$
C. $vetNov[c\_ant] = vetAnt[c\_nov]$
D. $vetAnt[c\_ant] = vetNov[c\_mat]$
E. $vetNov[c\_nov] = M[c\_ant][c\_nov]$
F. $vetAnt[c\_ant] = vetNov[c\_nov]$
G. $vetAnt[c\_ant] = M[c\_ant][c\_nov]$

4. TEMA

Nesta questão você deve elaborar um programa que, dada uma matriz A_{mxn}, um vetor x_{nx1} e um vetor b_{mx1}, determina se $Ax = b$; nesse caso, ele devolve (i, Ax), sendo i o primeiro índice tal que $A_i x \neq b_i$. Pode-se supor que os dados sejam inteiros.

Por exemplo, considere a matriz $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}$ e o vetor $x = [4; 3; 2; 1]$, com produto $Ax = \begin{bmatrix} 20 \\ 30 \end{bmatrix}$. Assim, usando como vetor lado direito $b = [20; 30]$ ou $b = [20; 31]$, a resposta seria, respectivamente, $(-1, None)$ ou $(1, [20; 30])$.

```
#TRECHO A - ID 0
main()

#TRECHO B - ID 0
def produto_escalar (v1, v2):
    n = len(v1[0])

#TRECHO C - ID 0
def produto_escalar (v1, v2):
```

```
    n = len(v1)

#TRECHO D - ID 0
def produto_matriz_vetor (A, x, b):
    m = len(A[])
    n = len(A[][])

#TRECHO E - ID 0
def produto_matriz_vetor (A, x, b):
    m = len(A)
    n = len(A[0])

#TRECHO F - ID 0
def produto_matriz_vetor (A, x, b):
    m = len(A[0])
    n = len(A[0][0])

#TRECHO G - ID 1
    soma = 0

#TRECHO H - ID 1
    erro = True;

#TRECHO I - ID 1
    erro = False;

#TRECHO J - ID 1
    ind = -1

#TRECHO K - ID 1
```

```
    ind = 0

#TRECHO L - ID 1
    y = [];

#TRECHO M - ID 2
        esc = produto_escalar(A[i], x);

#TRECHO N - ID 3
            erro = False
            ind = i

#TRECHO O - ID 3
            erro = True
            ind = i

#TRECHO P - ID 2
        if (soma != 0):
            soma += v1[i] * v2[i];

#TRECHO Q - ID 2
        soma += v1[i] * v2[i];

#TRECHO R - ID 2
        y = y + esc;

#TRECHO S - ID 2
        y.append(esc);

#TRECHO T - ID 2
```

```
        if (ind == -1 and esc != b[i]):

#TRECHO U - ID 2
        if (ind == 0 and esc != b[i]):

#TRECHO V - ID 1
    if (erro):

#TRECHO X - ID 1
    for i in range(1,n):

#TRECHO Y - ID 1
    for i in range(0,n+1):

#TRECHO Z - ID 1
    for i in range(0,n):

#TRECHO AA - ID 1
    for i in range(1,m):

#TRECHO BB - ID 1
    for i in range(0,m):

#TRECHO CC - ID 1
    for i in range(1,m+1):

#TRECHO DD - ID 1
    return soma;

#TRECHO EE - ID 1
```

```
    if (soma == 0):
        return True
    else:
        return False

#TRECHO FF - ID 2
        return y, ind

#TRECHO GG - ID 1
    return -1, None

#TRECHO HH - ID 1
    return None, -1

#TRECHO II - ID 2
        return False, None

#TRECHO JJ - ID 2
        return ind, y

#TRECHO KK - ID 2
        return i, y

#TRECHO LL - ID 0
def main(): # A*x = [20; 30]
    A = [ [1, 2, 3, 4], [2, 3, 4, 5]];
    x = [4, 3, 2, 1]; b = [20, 31];
    resp, y = produto_matriz_vetor(A, x, b)
    if (resp == -1):
        print("Confere, resultado =", y)
```

```
    else:
        print("Errado, linha =", resp)
```

Assinale a única alternativa que contém os blocos corretos (dentre os abaixo) na ordem correta.

A. A, G, Z, P, DD, E, L, H, K, BB, M, S, U, O, V, JJ, GG, LL, A
B. A, C, G, Z, Q, EE, E, L, I, K, BB, M, S, U, O, V, JJ, GG, LL
C. A, G, X, Q, DD, E, L, I, J, AA, M, S, T, O, V, KK, II, LL
D. C, G, X, Q, EE, E, L, I, J, AA, M, R, T, O, V, JJ, GG, LL, A
E. B, G, X, Q, DD, F, L, I, J, AA, M, S, T, O, V, JJ, GG, LL, A
F. C, G, Z, Q, DD, E, L, I, J, BB, M, S, T, O, V, JJ, GG, LL, A
G. C, G, Z, Q, EE, DD, L, I, K, BB, M, S, U, O, V, JJ, GG, LL, A
H. A, C, G, Z, Q, EE, E, L, I, J, BB, M, R, T, O, V, JJ, GG, LL
I. B, G, Z, Q, DD, F, L, I, J, BB, M, S, T, O, V, JJ, GG, LL, A
J. A, B, G, X, Q, DD, F, L, I, J, AA, M, S, T, O, V, JJ, GG, LL
K. C, G, Y, Q, DD, E, L, I, J, CC, M, S, T, O, V, JJ, GG, LL, A
L. C, G, X, Q, DD, E, L, I, J, AA, M, S, T, O, V, JJ, GG, LL, A
M. C, G, X, Q, DD, E, L, I, J, AA, M, S, T, O, V, KK, II, LL, A
N. A, G, Z, Q, DD, E, L, H, K, BB, M, S, T, N, V, KK, GG, LL, A
O. C, G, Z, Q, EE, E, L, I, J, BB, M, R, T, O, V, JJ, GG, LL, A
P. C, G, Z, Q, DD, E, L, H, J, BB, M, S, T, O, V, KK, II, LL, A
Q. B, G, Y, Q, DD, F, L, I, J, CC, M, S, T, O, V, JJ, GG, LL, A

Gabarito

1. Alternativa D.

2. Alternativa B.

3. Alternativa A.

4. Alternativa F.