MAC0110 – Introdução à Computação
Prova 2 – 24/05/2018 – BCC

NOME (EM LETRA DE FORMA LEGÍVEL):

ASSINATURA:

No. USP:

Instruções

1. Não destaque as folhas deste caderno.
2. A prova pode ser feita a lápis.
3. A legibilidade também faz parte da nota!
4. A prova consta de 3 questões. Verifique antes de começar a prova se o seu caderno de questões está completo.
5. Não é permitido o uso de folhas avulsas para rascunho.
6. Não é necessário apagar rascunhos no caderno de questão mas especifique qual é a resposta e qual é o rascunho.
7. Só é permitido usar os recursos dados nas aulas até o dia desta prova e deve-se seguir todas as restrições dadas também.
8. A prova é sem consulta.

Não escrever nesta parte da folha

Questão	Nota	Observação
1		
2		
3		
Total		

Boa Prova!

Questão 1 (valor=2.0+1.0)

Atenção: Nessa questão não está autorizado o uso de nenhum método da classe `str`, bem como comparações (A<B, A≤B, A==B, etc) entre strings com mais de um caractere.

a) Escreva uma função `é_lexicograficamente_menor` que recebe duas strings A e B e devolve `True` se A<B (A apareceria antes de B no dicionário) e `False` caso contrário. A condição formal a ser verificada é

$$\begin{array}{c} \text{A<B} \\ \Updownarrow \\ \exists n \geq 0 \text{ tal que } \left[\text{A[i]=B[i] } \forall i<n \text{ e } \left(\text{A[n]<B[n] ou } \begin{pmatrix} \text{A[n] não existe na string A} \\ \text{e B[n] existe na string B} \end{pmatrix} \right) \right]. \end{array}$$

Observação: você pode usar a comparação entre caracteres A[i]<B[i] em Python, e não precisa se preocupar com maíusculas/minúsculas, pontuações, etc.

```
def é_lexicograficamente_menor(A,B):
    # M = máximo índice nas duas strings
    M = min(len(A),len(B))
    # computa o n do enunciado
    n = 0
    while n<M and A[n]==B[n]:
        n += 1
    # testa os casos em que A<B
    if (n<M and A[n]<B[n]) or (n==len(A) and n<len(B)):
        return True
    else:
        return False
```

b) Escreva uma função que recebe uma lista L de palavras e devolve `True` se a lista estiver em ordem lexicográfica não-decrescente (L[i]≤L[i+1]∀i) e `False` caso contrário.

```
def está_em_ordem_lexicografica(L):
    N = len(L)
    # calcula AND de L[i]<=L[i+1] para todo i
    ordenado = True # elemento neutro do AND
    for i in range(N-1): # para acessar L[i+1]
        if é_lexicograficamente_menor(L[i+1],L[i]):
            ordenado = False
    return ordenado
```

Questão 2 (valor=3.0)

Escreva uma função que recebe uma matriz A de números reais e substitui cada elemento A[i][j] da matriz pela média dos valores das casas vizinhas, incluindo o próprio A[i][j] (considere vizinhas as casas adjacentes na horizontal/vertical/diagonal).

Dicas: (1) o número de vizinhos de A[i][j] depende de i e j; (2) os valores novos da matriz devem refletir a média dos valores *velhos* das casas vizinhas, ou seja, os valores que elas tinham ao entrar na função.

```
def produz_médias(A):
    # dimensões da matriz
    M,N = len(A),len(A[0])
    # cria uma cópia de A
    B = []
    for i in range(M):
        B.append(A[i].copy())
    # percorre a matriz, calculando as médias
    for i in range(M):
        for j in range(N):
            # percorre os vizinhos
            nvizinhos = 0
            soma = 0
            for k in range(-1,2):
                for l in range(-1,2):
                    if i+k in range(M) and j+l in range(N):
                        nvizinhos += 1
                        soma += A[i+k][l+j]
            B[i][j] = soma/nvizinhos
    # transfere valores de volta para A
    for i in range(M):
        for j in range(N):
            A[i][j] = B[i][j]
```

Questão 3 (valor=2.0+2.0)

Atenção: Nessa questão não está autorizado o uso de nenhum método da classe `list`.

Podemos representar um conjunto C=$\{c_1, \ldots, c_k\}$ de inteiros não-negativos usando uma lista L **estática** (ou seja, uma lista que não expande nem encolhe durante seu uso, e ocupa sempre a mesma posição de memória), armazenando os valores do conjunto *em ordem não-decrescente* (L[i]≤L[i+1] $\forall i$) nas primeiras k posições de L, e marcando o final do conjunto pelo valor especial -1 (terminador), que ocupa a posição de índice k. Note que uma lista estática de tamanho N permite a representação de conjuntos com até N-1 elementos.

a) Considere a função `insereElemento` abaixo, que recebe um conjunto representado por uma lista L e um tamanho N, e um valor positivo x, e tenta inserir o valor x no conjunto (caso x já não seja um elemento do conjunto e ainda haja espaço na lista).

```
def insereElemento(L,N,x):
    i = 0
    while L[i]!=-1 and L[i]<x:
        i += 1
    if L[i]==x or i>=N-1:
        return
    j = i
    while L[j]!=-1:
        L[j+1] = L[j]
        j += 1
    L[i] = x
```

Esse código possui erro(s). Mostre uma simulação passo-a-passo com uma entrada simples que produza uma saída errada, e re-escreva o código de forma a corrigi-lo. **Dica:** Lembre-se que a lista é estática (nem pense em usar `insert`) e que as operações precisam manter a ordem não-decrescente dos elementos armazenados, e o uso correto do terminador -1.

```
Considere a entrada L = [ 1, 2, -1, ? ] com N=4,
onde tentaremos inserir x = 0. A simulação fica

 i | j | L                | obs.
---+---+------------------+---------------------
 0 |   | [ 1, 2, -1, ? ]  | sai do while: L[0]!=-1 mas L[0]>=0
   |   |                  | sai do if: L[0]!=0 e i<3
   | 0 |                  | while: L[0]!=-1 (ok)
   |   | [ 1, 1, -1, ? ]  | L[1] = L[0]
nesse momento perdemos o 2 da lista L!!!
   | 1 |                  | while: L[1]!=-1 (ok)
   |   | [ 1, 1, 1, ? ]   | L[2] = L[1]
nesse momento perdemos o -1 da lista L!!!
   | 2 |                  | while: L[2]!=-1 (ok)
   |   | [ 1, 1, 1, 1 ]   | L[3] = L[2]
observe que sobrescrevemos toda lista com 1
   | 3 |                  | while: L[3]!=-1 (ok)
   |   | ERRO DE ACESSO!! | L[4] = L[3]
```

```
Vemos então 2 erros: a transferência dos elementos da lista para a
direita, que deveria abrir espaço para o x, está jogando todo o
conteúdo da lista fora, inclusive o -1, que impede o laço de
terminar corretamente; e a condição de conjunto cheio, que depende
de saber onde está o terminador.

Uma solução seria então localizar o terminador e fazer a
transferência dos elementos DA DIREITA PARA A ESQUERDA:
```

```
def insereElemento(L,N,x):
    # obs: N==len(L), nem precisava passar...
    # procura o lugar certo de x na lista
    i = 0
    while L[i]!=-1 and L[i]<x:
        i += 1
    # encontra terminador
    j = i
    while L[j]!=-1:
        j += 1
    # se x já existe ou não cabe + ninguém, sai
    if L[i]==x or j>=N-1:
        return
    # abre espaço para inserir x na posição i
    while j>=i:
        L[j+1] = L[j]
        j -= 1
    L[i] = x
```

b) Escreva uma função `ordenaLista` que recebe uma lista L de valores positivos *desordenados* e terminados por -1, e ordena a lista *no lugar* (ou seja, sem o uso de listas auxiliares) convertendo-a em um conjunto do tipo descrito acima. **Dica:** use a lista para representar um conjunto vazio e insira os elementos um a um usando a função do item (a), mesmo que não a tenha feito.

```
def ordenaLista(L):
    # lista vazia não precisa fazer nada...
    if L[0]==-1: return
    # guarda o primeiro elemento da lista
    guarda = L[0]
    # inicializa conjunto vazio
    L[0] = -1
    # os elementos a serem inseridos são
    # L[1], L[2], ... enquanto L[i]!=-1
    i = 1
    while L[i]!=-1:
        insereElemento(L,len(L),L[i])
        i += 1
    # no final, insere o primeiro elemento
    insereElemento(L,len(L),guarda)
```