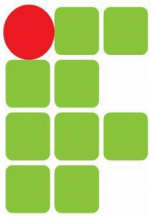

Instituto Federal do Espírito Santo
Campus Serra
Coordenadoria de Informática
Curso Superior Bacharelado em Sistemas de Informação

3ª Avaliação de Programação 2 – Semestre 2018-1

Turma Prof. Ernani Filho

Leia com atenção

- **O material para a prova estará contido na pasta *paraProva3.zip*.**
- **Para a entrega da prova, compacte os arquivos.py em um arquivo .ZIP e faça a entrega como uma atividade no moodle. Nomeie o arquivo zip com *<seu nome>.zip*.**
- **Toda questão estará correta somente se produzir os resultados solicitados e obedecer às especificações e restrições fornecidas no enunciado.**
- **As soluções das questões devem ser construídas utilizando APENAS o conteúdo de python visto em sala de aula.**
- **EVITE PERDER PONTOS POR FALTA DE INFORMAÇÃO: na dúvida sobre quais recursos/comandos da linguagem usar, solicite a orientação do professor.**

Instruções e restrições para TODAS as questões da prova

(a) A escolha dos nomes de variáveis e parâmetros de entrada das funções também FAZ parte do conteúdo avaliado.
(b) NÃO utilizar interrupções de loop do tipo *break* nem resumos do tipo *continue*.
(c) Quando for necessário algum tipo de sequência, fazer uso APENAS de listas, tuplas, strings e dicionários.
(d) Uma vez definido o tipo da variável, NÃO alterá-lo ao longo de todo o programa.
(e) Para responder às questões, utilizar EXATAMENTE os nomes das funções fornecidos nos enunciados.
(f) Nos arquivos q1.py e q2.py, escreva o SEU NOME nos cabeçalhos dos arquivos.

Observação

O único equipamento eletrônico permitido durante a confecção da prova é o computador do laboratório. A prova será anulada caso esta regra seja violada.

Questão 1 (15 pontos, tempo estimado: casa)

Palavras-chave: TAD, matriz, interface, aritmética amtricial, arquivos.

No arquivo **main3aAVALmat.py** fornecido como material da prova, construa o código que é pedido nos enunciados a seguir (1.1. TAD matriz e 1.2. Aplicação).

1.1) TAD Matriz

Construa o arquivo tadmatriz.py a ser utilizado como um módulo por outros programas. O arquivo implementa um tipo abstrato de dados matriz (TAD) que segue a especificação a seguir. O trabalho será considerado correto somente se a expecificação abaixo for seguida.

Especificação TAD Matriz (tadmatriz.py):

a) Requisito: a matriz não deve armazenar elementos de valor zero.

b) Modelo/Estrutura de dados: A implementação da estrutura de dados que representa o TAD deve, obrigatoriamente, obedecer a uma das duas opções a seguir.

[1] **Estruturas de dados para a construção do TAD matriz**:

(i) [<int lins>, <int cols>, {<chave tupla lin, col: dado elem <float>>}]

(ii) {'lins': <int>, 'cols': <int>. 'dados': {<chave tupla lin, col: dado elem <float>> }}

Onde lins e cols são, respectivamente, quantidade de linhas e quantidade de colunas.

c) Interface:

c.1) cria(quant. linhas, quant. colunas): Retorna uma estrutura de dados [1] do tipo tad matriz representando uma matriz de dimensões <linhas> por <colunas>.

c.2) criaLst(matriz lista de listas): Retorna uma estrutura de dados do tipo tad matriz a partir de uma matriz lista de listas passada como argumento de entrada.

c.3) destroi(): Retorna *None*.

c.4) getElem(tadMat, lin, col): Retorna o elemento armazenado na posição *lin*, *col* da matriz de entrada. Retorna *None* se o elemento não existir (lin e col forem valores que extrapolam as dimensões da matriz).

c.5) setElem(tadMat, lin, col, valor): Armazena o elemento na posição lin, col da matriz de entrada. Retorna *None* se a posição não existir (lin e col forem valores que extrapolam as dimensões da matriz).

c.6) soma(tadMatA, tadMatB): Soma duas matrizes e retorna uma terceira matriz (NOVA) com o resultado da soma. Soma deve usar o conceito matemático de soma de matrizes. Retorna *None* se as matrizes não puderem ser somadas.

c.7) vezesK(tadMat, k): Altera a matriz de entrada multiplicando os seus elementos por k. Retorna uma referência para tadMat.

c.8) multi(tadMatA, tadMatB): Multiplica dois tad matrizes e retorna um terceiro (NOVO) tad matriz com o produto resultante. multi deve usar o conceito matemático de multiplicação de matrizes. Retorna *None* se as matrizes não puderem ser multiplicadas.

c.9) clona(tadMat): Retorna um tad matriz cópia do tad matriz de entrada.

c.10) diagP(tadMat): Retorna uma lista com a diagonal principal do tad matriz passado como entrada. Retorna *None* se a diagonal principal não existir (matriz não quadrada).

c.11) diagS(tadMat): Retorna uma lista com a diagonal secundaria do tad matriz passado como entrada. Retorna *None* se a diagonal secundaria não existir (matriz não quadrada).

c.12) quantLinhas(tadMat): Retorna a quantidade de linhas da matriz de entrada.

c.13) quantColunas(tadMat): Retorna a quantidade de colunas da matriz de entrada.

c.14) carrega(<arquivo>): Carrega uma matriz a partir de um arquivo texto de nome <arquivo>. Retorna uma matriz do tipo tad matriz preenchida com o conteúdo arquivo. No arquivo, a matriz está representada da seguinte forma: elementos separados por espaço, cada linha de texto é uma linha da matriz.

c.15) salva(tadMat,<arquivo>): Salva uma matriz em um arquivo texto de nome <arquivo>. Retorna uma referência para o tad matriz de entrada. No arquivo, a matriz deve estar representada da seguinte forma: elementos separados por espaço, cada linha de texto é uma linha da matriz.

1.2) Aplicação

Construa a aplicação **main3aAVALmat.py** da seguinte forma. A aplicação deve, obrigatoriamente, fazer uso do conteúdo do tadmatriz.py. A aplicação deve também ler e processar LINHA A LINHA o arquivo **bdaritmat.csv**. Um exemplo do conteúdo e formato do arquivo está disponível na figura 1. No arquivo, cada letra maiúscula representa o nome de uma matriz. A letra minústuca **t** repreenta a operação transposta de uma matriz m qualquer. Por último, constantes inteiras são apenas valores multiplicadores. Usando a figura 1 como exemplo, vemos que a primeira linha contém apenas o nome de uma matriz. Isto significa que os dados dessa matriz devem ser carregados na memória e transformados em um tad matriz. A partir daí, encontramos em cada linha uma dupla separada por vírgula contendo uma operação aritmética e um nome de matriz (exceção da operação de transposição, apenas a letra **t** minúscula). A interpretação da tarefa pode ser obtida diretamente da observação da figura 1 e das explicações dada pelo professor em sala de aula.

Os dados da matriz que aparece em cada linha de **bdaritmat.csv** estão armazenados nos arquivos de nome <nome da matriz>.txt (Ex. A.txt, B.txt, etc.). Os arquivos <nome da matriz>.txt estão dentro da pasta **bdmatrizes**.

Conteúdo bdaritmat.csv

J	
+, A	J + A
−, H	(J + A) − H
*, M	((J + A) − H) * M
*,25	(((J + A) − H) * M) * 25
t	((((J + A) − H) * M) * 25)t
*, I	(((((J + A) − H) * M) * 25)t) * I
+, Q	((((((J + A) − H) * M) * 25)t) * I) + Q

Resultado:

matriz ((((((J + A) − H) * M) * 25)t) * I) + Q

Figura 1: conteúdo exemplo do arquivo bdaritmat.csv

Ao final do processamento, a função main da aplicação deverá: i) salvar a matriz resultado no arquivo Q1.txt; ii) imprimir na tela, de forma organizada (sem colchetes de listas) os elementos das diagonais principal e secundárias da matriz resultado.

Utilize o arquivo MR.txt, contendo a matriz resposta, como um gabarito para testar e calibrar a aplicação principal main3aAVALmat.py. Construa uma aplicação genérica porque durante a correção o professor poderá utilizar como entrada arquivos do tipo bdaritmat.csv com conteúdos diferentes daquele exemplo exibido no exemplo da figura 1.

Questão 2 (20 pontos, tempo estimado: casa)

Palavras-chave: TAD, matriz, interface, aritmética amtricial, arquivos.

No arquivo **main3aAVALpoli.py** fornecido como material da prova, construa o código que é pedido nos enunciados a seguir (2.1. TAD polinomio e 2.2. Aplicação).

2.1) TAD Polinomio

Construa o arquivo tadmatriz.py a ser utilizado como um módulo por outros programas. O arquivo implementa um tipo abstrato de dados matriz (TAD) que segue a especificação a seguir. O trabalho será considerado correto somente se a expecificação abaixo for seguida.

Especificação TAD Polinomio (tadpoli.py):

a) Requisito: O polinômio será sempre monovariável (x variável independente).
O polinomio terá sempre graus inteiros, maiores ou iguais a zero, e coeficientes reais (floats).

b) Modelo/Estrutura de dados: A implementação da estrutura de dados que representa o TAD deve, obrigatoriamente, obedecer a uma das duas opções a seguir.descrição abaixo:

[1] **Estruturas de dados para a construção do TAD matriz**:

{<int grau>: <float coef>, <int grau>: <float coef>, .. , <int grau>: <float coef>}

Onde coef é o coeficiente de X (xis), coeficiente da variável independente.

A figura a seguir resume a ideia da estrutura de dados do tad polinomio.

Polinômio x TAD polinômio.

P(x) = $a_n x^n + a_{n-1} x^{n-1} \ldots + a_2 x^2 + a_1 x^1 + a_0$

Dicionário: {n:a_n, n-1:a_{n-1}, … , 2:a_2, 1:a_1, 0:a_0}

Dicionário com entradas do tipo chave igual a grau e conteúdo igual a coeficiente do termo do polinômio (monômio).

Exemplo:

Polinômio (matemática): $-2x^9-x^4+5x^2+3x-9$ → **TAD:** {9:-2, 4:-1, 2:5, 1:3, 0:-9}

Figura 2

c) Interface:

c.1) inic(<string contendo um polinômio>): cria e retorna a estrutura de dados [1] definida para ser o tadpoli. O valor inicial da estrutura é definido a partir da string polinômio argumento de entrada (ver nos exemplos a seguir amostras de polinômios tipicamente encontrados em livors de Cálculo I). Observe que uma constante pode ser visto como um polinômio de grau zero.

Na string de entrada não existirão dois termos com o mesmo grau, e podem ou não existir espaços ao lado dos operadores + e - .

Exemplos:

2x4-4x3+2x2+x+5
-5x3+x
-6 ← Polinômio constante, equivalente -6x^0
7x-3

c.2) igual(<tadpoliA>, <tadpoliB>): retorna verdadeiro se os dois polinômios passados como argumentos são iguais. Retorna falso caso contrário.

c.3) somapoli(<tadpoliA>, <tadpoliB>): soma os polinômios armazenados nos parâmetros de entrada. Retorna um polinômio resultado da soma (um novo tadpoli). A soma obedece às regras da aritmética de polinômios vistas na álgebra.

c.4) multipoli(<tadpoliA>, <tadpoliB>): multiplica os polinômios armazenados nos parâmetros de entrada. Retorna o polinômio resultado da multiplicação (um novo tadpoli). A multiplicação obedece às regras da aritmética de polinômios vistas na álgebra.

c.5) fx(<tadpoli>, <float>): retorna o valor float resultante do cálculo do polinômio para o valor de x igual ao do segundo parâmetro. Ou seja, calcula y = f(x).

c.6) **dfx(<tadpoli>):** retorna um tad poli contendo uma equação polinomial correspondente à derivada do tadpoli passado como argumento. A obtenção da derivada do polinomio obedece às regras de derivadas de funções visto em Cálculo I.

c.6) parastr(<tadpoli>): retorna o conteúdo do tadpoli como uma string com os coeficientes arranjados na forma polinomial textual (note a existência de espaços entre os operadores): Exemplos: **"**2x4 – 4x3 + 2x2 + x + 5**"**, **"**5x3 + x**",** etc.

2.2) Aplicação

Construa a aplicação **main3aAVALpoli.py** da seguinte forma. A aplicação deve, obrigatoriamente, fazer uso do conteúdo do módulo tadpoli.py. A aplicação deve também ler e processar LINHA A LINHA o arquivo **bdaritpoli.csv**. Um exemplo do conteúdo e formato do arquivo está disponível na figura 2. Usando a figura 2 como referência, vemos que a primeira linha contém apenas um polinômio, sem nenhuma operação associada a ele. Este é o polinômio inicial, ponto de partida. A partir daí, encontramos em cada linha uma dupla separada por vírgula contendo uma operação aritmética e um polinômio. A interpretação da tarefa pode ser obtida diretamente da observação da figura 2 e das explicações dada pelo professor em sala de aula.

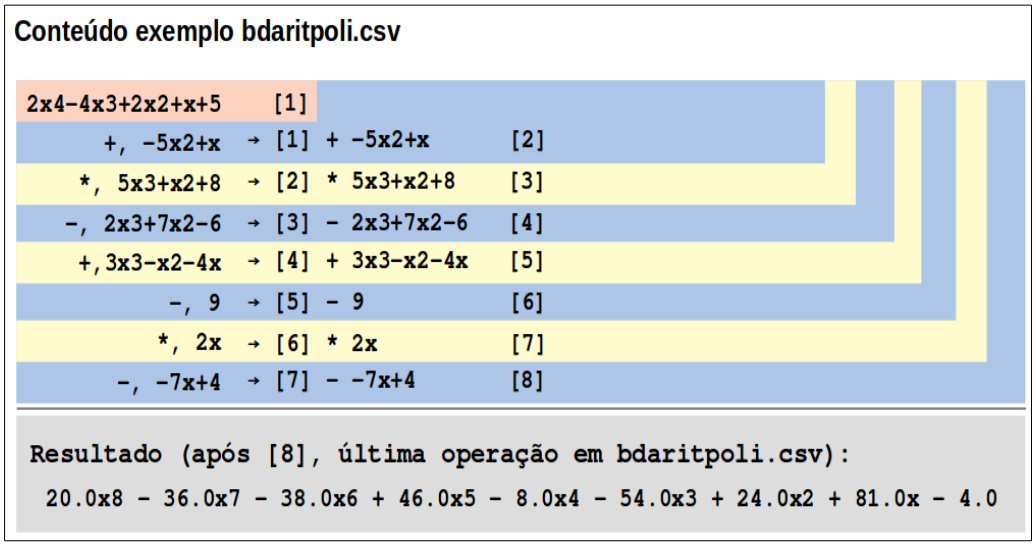

Figura 2: conteúdo exemplo do arquivo bdaritpoli.csv

Ao final do processamento, a função main da aplicação deverá exibir na tela: i) o polinômio resultante no formato string, como um dos polinômios de entrada; ii) o polinômo equivalente à derivada do polinômo resultante; iii) O valor do polnômio resultante quando X for igual a 3.75; iv) o valor da derivada do polinômio resultante quando X for igual a 20.34.

Utilize a figura 2 como um gabarito para testar e calibrar a aplicação main3aAVALpoli.py. Construa uma aplicação genérica porque durante a correção o professor poderá utilizar como entrada arquivos do tipo bdaritpoli.csv com conteúdos diferentes dos do exemplo mostrado na figura 2.

Processo de correção:

O trabalho será corrigido segundo o algoritmo a seguir:

Início

Passo 1: montar matriz de similaridade para verificação de plágio.

Passo 2:

Para cada aluno **faça**:

if NÃO houve detecção de plágio[*] **então**:

if saída da aplic. principal (main2aAVAL...py) estiverem corretos **então**:

- Corrigir as funções da interface de cada TAD; [**]

else:

- Corrigir as funções da interface de cada TAD; [**]
- Aplicar desconto de 50% na nota final;

else:

- Chamada dos alunos para esclarecimentos;
- Nota avaliação igual a zero;

fim para

Passo 3: chamada dos alunos para explicação rápida do código enviado.

Fim.

[*] Detecção de plágio

A detecção de plágio será avaliada por meio do cálculo de similaridade entre códigos. Os seguintes programas serão utilizados para avaliar a similaridade:

- MOSS (Measure of Software Similarity) system provided by Stanford: https://theory.stanford.edu/~aiken/moss/
- Jplag (detecting software plagiarism) system provided by Institute for Program Structures and Data Organization: http://jplag.ipd.kit.edu/

[**] Correção das funções de cada TAD

Cada função receberá um valor de 0 a 5: **0** (função não faz o que é pedido no enunciado); **1** (função não trata os parâmetros de entrada corretamente, nem retorna o tipo do dado que foi pedido); **2** (função trata os parâmetros e tipo de saída corretamente mas o código não obedece à especificação); **3** (função trata os parâmetros e tipo de saída corretamente, o código obedece à especificação mas não resolve o problema); **4** (função trata os parâmetros e tipo de saída corretamente , o código obedece à especificação mas não resolve totalmente o problema); **5** (função trata os parâmetros e tipo de saída corretamente , o código obedece à especificação e resolve totalmente o problema).

Conteúdo do arquivo *aval-3-2018-1.zip*

- *prova3-BS2-2018-1-para-casa.pdf*: este documento pdf que você está lendo.
- *main3aAVALmat.py*: template da aplicação principal que importará o módulo tadmat e resolverá o problema proposto pela figura 1.
- *main3aAVALpoli.py*: template da aplicação principal que importará o módulo tadpoli e resolverá o problema proposto pela figura 2.
- *bdmatrizes*: pasta contendo uma série de arquivos matrizes a serem utilizados como dados de entrada para o processamento de *main3aAVALmat.py.*
- *MR.txt*: matriz resultante do processamento mostrado na figura 1, questão 1. A ser utilizada como gabarito para testes de *main3aAVALmat.py.* O conteúdo de MR.txt é específco para as matrizes que residem em bdmatrizes.
- *bdaritmat.csv*: arquivo contendo a aritmética matricial mostrada na figura 1. A ser utilizado como dado de entrada de *main3aAVALmat.py.*
- *bdaritpoli.csv*: arquivo contendo a aritmética polinomial mostrada na figura 2. A ser utilizado como dado de entrada de *main3aAVALpoli.py.*

Bom trabalho!