Programação 2 – 2017/1
Curso: Bacharelado em Sistemas de Informação – Campus Serra
Turma: 20171.BSI.2

3ª Lista de Exercícios: Tipos Abstratos de Dados (TAD)

Leia-me Primeiro

Em todos os exercícios, construa um arquivo .py para a implementação do tipo abstrato de dados (TAD) e um arquivo de aplicação para testar as funcionalidades do TAD.

Em todas as especificações de TADs, acrescentar o método destroi(), que retorna o valor None a fim de sinalizar a conclusão do ciclo de vida (ser criado, ser utilizado, ser destruído) do TAD.

Revise as anotações sobre TAD dadas em sala de aula. Estudo o conteúdo construindo os programas pedidos abaixo:

1. Construa o TAD horário (tadhorario.py) a partir da seguinte especificação: i) uma hora é expressa por 3 números inteiros representando, respectivamente hora, minuto e segundo do ciclo de 24 horas; ii) para um dado do tipo tadhorario são permitidas as seguintes operações:

inicstr(<string>): cria, inicializa e retorna a estrutura de dados definida para ser o tadhorario com os conteúdos (hora, minutos e segundos) vindos da string parâmetro de entrada. Uma típica string de entrada possui o seguinte formato: “aahbbmccs”, ou “aa:bb:cc”. Exemplos: “14h23m7s”, “11:22:30”. Lembrando que hora (aa), minuto (bb) e segundo (ss) podem possuir apenas 1 dígito.

inicsegs(<total segundos>): cria e retorna um tadhorario que representa ser o valor do parâmetro de entrada em horas, minutos e segundos. Retorna None se a quantidade de entrada for um valor inválido, como um valor negativo, por exemplo.

cmp(<tadhorarioA>,<tadhorarioB>): retorna 0 se tadhorarioA é igual a tadhorarioB (horas iguais, minutos iguais e segundo iguais); retorna -1 se tadhorarioA representa uma data anterior à data em tadhorarioB; retorna +1 se tadhorarioA representa uma data posterior à data em tadhorarioB.

somasegs(<tadhorario>,<segundos>): retorna o tadhorario de entrada com os valores de hora, minuto e segundos alterados para refletir o novo horário que é uma quantidade de segundos à frente, ou atrás (<segundos> pode ser negativo) do horário atual.

getHora(t<tadhorario>): retorna o valor da hora do horário armazenado em tadhorario.

getMinuto(<tadhorario>): retorna o valor dos minutos do horário armazenado em tadhorario.

getSeg(<tadhorario>): retorna o valor dos segundos do horário armazenado em tadhorario.

tostr(<tadhorario>): retorna o conteúdo do tadhorario como uma string com hora, minuto e segundo separados por dois pontos, ‘:’. Exemplo: “5:35:22s”.

tostrE(<tadhorario>): retorna o conteúdo do tadhorario como uma string com hora, minuto e segundo separados pelas letras h, m e s. Exemplo: “5h35m22s”.

gerahorario(<tadhorarioA>,<tadhorarioB>): gera um tadhorario contendo um horário aleatório entre o intervalo fechado dos horários representados em tadhorarioA e tadhorarioB.

2. Construa o TAD data (taddata.py) a partir da seguinte especificação: i) uma data é expressa por 3 números inteiros representando, respectivamente, dia, mês e ano do calendário gregoriano ocidental (https://pt.wikipedia.org/wiki/Calend%C3%A1rio_gregoriano); ii) para um dado do tipo taddata são permitidas as seguintes operações:

inicstr(<string>): cria, inicializa e retorna a estrutura de dados definida para ser o taddata com os conteúdos (dia, mês e anos) vindos da string parâmetro de entrada. Uma típica string de entrada possui o seguinte formato: “dd/mm/aaaa”, ou “dd/mm/aa”. Exemplos: “10/04/2016”, “10/04/16”. Lembrando que dia e mês podem possuir apenas 1 dígito.

inicstrE(<string>): cria, inicializa e retorna a estrutura de dados definida para ser o taddata com os conteúdos (dia, mês e anos) vindos da string parâmetro de entrada. Uma típica string de entrada possui o seguinte formato: dd de <nome mês> de aaaa. Exemplos: “10 de Outubro de 2016”.

cmp(<taddataA>,<taddataB>): retorna 0 se taddataA é igual a taddataB (dias iguais, meses iguais e anos iguais); retorna -1 se taddataA representa uma data anterior à data em taddataB; retorna +1 se taddataA representa uma data posterior à data em taddataB.

somadias(<taddata>,<dias>): retorna um novo taddata com uma nova data, que é uma quantidade de dias à frente, ou atrás (<dias> pode ser negativo) da data de entrada.

getDia(t<taddata>): retorna o valor do dia da data armazenada em taddata.

getMes(<taddata>): retorna o valor do mês da data armazenada em taddata.

getAno(<taddata>): retorna o valor do ano da data armazenada em taddata.

diasemana(<taddata>): retorna o nome do dia da semana (texto com o nome do dia: 'domingo', 'segunda-feira’,'terça-feira', etc.) referente à data armazenada em taddata. Pesquise na internet o algoritmo/método que obtém o dia da semana a partir da data do calendário.

parastr(<taddata>): retorna o conteúdo do taddata como uma string com dia, mês e ano separados por barra, /. Exemplo: “10/04/2016”.

parastrE(<taddata>): retorna o conteúdo do taddata como uma string com dia, mês e ano usando o formato com nome do mês por extenso. Exemplo: “10 de Setembro de 2016”.

geradata(<taddataA>,<taddataB>): gera um taddata contendo uma data aleatória entre o intervalo fechado das datas representadas em taddataA e taddataB.

---

3. Construa o TAD equação do segundo grau (tadeq2g.py). O TAD possui as seguintes especificações: i) uma equação do segundo grau é um conjunto de 3 coeficientes reais; ii) de uma equação do segundo grau podem ser calculadas nenhuma, uma ou duas raízes reais; iii) para um dado do tipo tadeq2g são permitidas as seguintes operações:

inic(<float>, <float>,<float>): cria, inicializa e retorna a estrutura de dados definida para ser o taddata com os conteúdos dos 3 coeficientes vindos de três parâmetros numéricos de entrada.

inicstr(<string>): cria, inicializa e retorna a estrutura de dados definida para ser o taddata com os conteúdos dos 3 coeficientes vindos da string parâmetro de entrada. Uma típica string de entrada possui o seguinte formato: “ax2+bx+c”. Exemplos: “3x2-7x+1”.

fx(<tadeq2g>,<float>): retorna o valor float resultante do cálculo da equação para o valor de x igual ao do segundo parâmetro. Ou seja, calcula y = f(x).

numraizes(<tadeq2g>): retorna a quantidade de raízes que podem ser calculadas a partir da equação presente no parâmetro de entrada.

raiz1(<tadeq2g>): retorna a primeira raiz real, se existir, da equação armazenada no parâmetro de entrada. Retorna None se a raiz não existir.

raiz2(<tadeq2g>): retorna a segunda raiz real, se existir, da equação armazenada no parâmetro de entrada. Retorna None se a raiz não existir.

getA(<tadeq2g>), getB(<tadeq2g>), getC(<tadeq2g>): retornam os valores dos coeficientes a, b e c (respectivamente) do tad equação de entrada.

_delta(<tadeq2g>): retorna o valor do delta da equação de entrada. A intenção dessa função é auxiliar <u>internamente</u> o cálculo das raízes. Esta função não deve fazer parte da interface do tad que é visível para a aplicação. Python não possui nenhum mecanismo oficial de ocultamento de funções.

Por causa disso, segue-se uma convenção notacional de se prefixar com _ (caractere sublinhado/underscore) os nomes das funções que não desejamos que sejam vistas/utilizadas pelos usuários dos tads.

parastr(<tadeq2g>): retorna o conteúdo do tadeq2g como uma string com os coeficientes arranjados na forma textual da equação. Exemplos: “3x2-7x+1”, “x2-4x+3”, etc..

⏱⏱

4. Construa o TAD ponto2d (tadpto2d.py). O TAD possui as seguintes especificações: i) um ponto 2D é uma dupla, um par ordenado, representando as coordenadas x e y de um ponto no plano; ii) para um dado do tipo tadpto2d são permitidas as seguintes operações:

inic(<float>,<float>): cria, inicializa e retorna a estrutura de dados definida para ser o tadpto2d. Inicializa a estrutura de dados com os conteúdos das coordenadas x, y do plano.

igual(<tadpto2dA>,<tadpto2dB>): retorna verdadeiro se os dois pontos passados como parâmetros são iguais. Retorna falso caso contrário.

distancia(<tadpto2dA>,<tadpto2dB>): retorna a distância euclidiana entre os dois pontos passados como parâmetros..

getX(<tadpto2d>), getY(<tadpto2d>): retorna, respectivamente, as coordenadas x e y dos pontos passados como parâmetros.

noX(<tadpto2d>): retorna verdadeiro se o ponto se encontra na abcissa. Retorna falso caso contrário.

noY(<tadpto2d>): retorna verdadeiro se o ponto se encontra na ordenada. Retorna falso caso contrário.

desloca(<tadpto2d>,<float>,<float>): altera as coordenadas do ponto parâmetro de entrada para que elas reflitam um deslocamento no eixo x (primeiro parâmetro float), e um deslocamento no eixo y (segundo parâmetro float). Retorna o tadpto2d de entrada.

quadrante(<tadpto2d>): retorna o quadrante (1, 2, 3 ou 4) do sistema de coordenadas onde reside o ponto passado como parâmetro de entrada.

⏱⏱

5. Construa o TAD polinômio (tadpoli.py). O TAD possui as seguintes especificações: i) deve representar um polinômio de apenas 1 variável, contendo 1 ou vários monômios; ii) deve ser implementado utilizando-se dicionários; iii) para um dado do tipo tadpoli são permitidas as seguintes operações:

inic(<string contendo polinômio>): cria, inicializa e retorna a estrutura de dados definida para ser o tadpoli a partir do polinômio no formato string (ver exemplo logo a seguir).

```
Exemplos:
  2x4-4x3+2x2+x+5
  5x3+x
  3x4+5
  7x
  -3
```

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

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

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.

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).

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..

---

6. Construa o TAD quadrilátero (tadquadri.py). O TAD possui as seguintes especificações: i) deve representar um quadrilátero qualquer; ii) o quadrilátero sendo representado deve ser visto como uma figura que pode ser desenhada na superfície da tela do computador (canvas); iii) em um canvas, as coordenadas de desenho tem sua origem no canto esquerdo superior (ponto 0, 0). iv) para um dado do tipo tadquali são permitidas as seguintes operações:

inic(<int>,<int>,<int>,<int>): cria, inicializa e retorna a estrutura de dados definida para ser o tadquadri a partir das coordenadas x, y do canto esquerdo superior e das coordenadas x, y do canto direito inferior.

inic_la(<int>,<int>,<int largura>,<int altura>): cria, inicializa e retorna a estrutura de dados definida para ser o tadquadri a partir das coordenadas x, y do canto esquerdo superior e das dimensões largura e altura do quadrilátero.

igual(<tadquadriA>,<tadquadriB>): retorna verdadeiro se os dois quadriláteros passados como parâmetros são iguais. Retorna falso caso contrário.

area(<tadquadri>): retorna o valor da área do quadrilátero passado como parâmetro de entrada.

perimetro(<tadquadri>): retorna o valor do perímetro do quadrilátero passado como parâmetro de entrada.

getLstQuinas(<tadquadri>): retorna uma lista de tads tipo pontos2d com as quinas (cantos) do quadrilátero passado como parâmetro de entrada. A ordem segue o sentido horário a partir do canto superior esquerdo, o início do quadrilátero.

getCantoSE(<tadquadri>), getCantoSD(<tadquadri>), getCantoIE(<tadquadri>), getCantoID(<tadquadri>): retorna, respectivamente, tads tipo ponto2d representando os cantos superior esquerdo, superior direito, inferior esquerdo e inferior direito dos quadriláteros passados como parâmetros de entrada.

dentro(<tadquadri>, <tadpto2d>): retorna verdadeiro se o ponto 2D passado como segundo parâmetro está no interior ou na borda do quadrilátero passado como primeiro parâmetro. Retorna falso caso contrário.

move(<tadquadri>, <int dx>, <int dy>): retorna o quadrilátero passado como parâmetro de entrada com novos valores de cantos para refletir um deslocamento de dx posições no eixo x e dy posições no eixo y.

intersec(<tadquadriA>, <tadquadriB>): retorna um novo tadquadri representando o quadrilátero resultante da intersecção do quadrilátero A com o quadrilátero B. Se não houver essa intersecção, retornar None. Analise o problema e pesquise quais as possibilidades de intersecção entre dois quadriláteros.

Desafio: depois de resolver o problema da intersecção, tente melhorar sua solução da seguinte forma: a) usando o menor números de ifs; b) não usando ifs.

⏱⏱

7. Construa o TAD Tabela Salário (tadtabsal.py). O TAD possui as seguintes especificações: i) deve representar a tabela da figura abaixo; ii) deve possuir um conjunto de funções construtoras, analisadoras e modificadoras que você acha que seriam as apropriadas para que um programador de aplicação consiga construir programas que cadastrem e retirem dados da tabela.

POSTO / GRADUAÇÃO	JULHO/2011		
	SOLDO R$	GRAT. RISCO R$	REMUNERAÇÃO R$
CORONEL	8.725,00	3.173,00	11.898,00
TENENTE CORONEL	7.380,00	2.300,00	9.680,00
MAJOR	5.985,00	2.215,00	8.200,00
CAPITÃO	4.995,00	1.655,00	6.650,00
1º TENENTE	4.365,00	1.185,00	5.550,00
2º TENENTE	3.915,00	935,00	4.850,00
SUBTENENTE	3.420,00	930,00	4.350,00
1º SARGENTO	2.970,00	830,00	3.800,00
2º SARGENTO	2.565,00	735,00	3.300,00
3º SARGENTO	2.205,00	645,00	2.850,00
CABO	1.890,00	560,00	2.450,00
SOLDADO	1.700,00	400,00	2.100,00

Figura Tabela de Salários

Fim! (por enquanto)