Lista de exercícios 1

MAC0122 - Princípios de
Desenvolvimento de Algoritmos

Setembro 2017

1 Classes e objetos

1. Crie uma classe *Ponto* com atributos x e y. Sua classe deve suportar soma, subtração, comparação, representação em *string* e os seguintes métodos:

   (a) *distancia_para_ponto* que computa a distância euclidiana para outro ponto, dado por um argumento da função;

   (b) *distancia_para_origem* que computa a distância para a origem, isto é, o ponto $(0, 0)$;

   (c) *inclinacao* que computa a inclinação da reta que passa pela origem. Quais casos esse método pode falhar? Como você pode evitar isso?;

   (d) *reta_para* que dado um ponto como argumento, retorna uma dupla (a, b) tal que a reta $y = ax + b$ passe pelos dois pontos (o próprio objeto e o ponto dado pelo argumento da função);

2. Qual a diferença de uma cópia rasa e uma cópia profunda? E qual a diferença de uma igualdade rasa e uma igualdade profunda?

3. Vimos em aula como calcular a soma harmônica

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + ... + \frac{1}{n}$$

   Podemos calculá-la de duas formas: somando da esquerda para direita ou somando da direita para a esquerda. Qual dessas formas é estável e por quê?

2 Listas

1. Desenhe um diagrama de referências para a e b antes e depois de ser executada a terceira linha do seguinte trecho de código:

```
a = [1, 2, 3]
b = a[:]
b[0] = 5
```

2. Apesar de Python nos fornecer uma grande lista de métodos nativos, é uma boa prática e instrutivo pensar sobre como elas podem ser implementadas. Implemente funções que se comportem como:

   (a) count;

   (b) in;

   (c) reverse;

   (d) index;

   (e) insert;

3. Considere três variações de um mesmo problema.

   (a) Escreva uma função que recebe uma lista de números e retorna a soma dos números pares na lista. Qual a complexidade do seu algoritmo?

   (b) Escreva uma função que recebe uma lista de números e retorna a soma dos números pares **distintos** na lista. Qual a complexidade do seu algoritmo?

   (c) Escreva uma função que recebe uma lista de números e retorna a soma dos números pares **distintos** na lista **supondo que a lista esteja ordenada**. Qual a complexidade do seu algoritmo?

4. Dado uma lista de inteiros, retorne True se a lista é monótona[1]. Retorne False, caso contrário.

[1] https://pt.wikipedia.org/wiki/Função_monótona

3 Dicionários

1. Escreva uma função que conta o número de ocorrência de cada palavra em uma *string* dada. Crie uma versão alternativa que é passado o nome de um arquivo como parâmetro, ao invés de uma *string*.

2. Encontre o primeiro número não-repetido em um longo arquivo de números. Qual a complexidade do seu algoritmo?

3. O que é impresso pelos seguintes comandos?

```
mydict = {"cat":12, "dog":6, "elephant":23,
   "bear":20}
yourdict = mydict
yourdict["elephant"] = 999
print(mydict["elephant"])
```

4 Filas e pilhas

1. Imagine um tabuleiro quadriculado com $m \times n$ casas dispostas em m linhas e n colunas. Algumas casas estão livres e outras estão bloqueadas. As casas livres são marcadas com - e as bloqueadas com #. Há um robô na casa $(1, 1)$, que é livre. O robô só pode andar de uma casa livre para outra. Em cada passo, só pode andar para a casa que está ao norte, a leste, ao sul ou a oeste. Ajude o robô a encontrar a saída, que está na posição (m, n).

2. Uma pilha pode ser utilizada para verificar se uma expressão aritmética contendo uma sequência de parênteses e colchetes está bem formada, ou seja, se todo parênteses (e colchetes) que abre é fechado. Utilizando os métodos *append*() e *pop*() de listas, escreva uma função $bem\_formada(s)$ que recebe um string s contendo apenas os caracteres de (,), [e] e devolve True caso a expressão estiver bem formada e False caso contrário.

3. Na notação usual de expressões aritméticas, como em $(2+3)*4$, os operadores são escritos entre os operandos; por isso, a notação é chamada de infixa. Na notação polonesa inversa (ou posfixa) os operadores são escritos depois dos operandos. O mesmo exemplo em notação posfixa

é escrito como 2 3 + 4 *. Escreva uma função que reduz uma expressão posfixa em um valor. *Uma dica é usar a classe Token vista no EP4.*

5 Busca binária

1. Escreva uma função que dado uma lista de inteiros ordenado (crescente) e um inteiro x, retorne a posição de x se ele estiver na lista ou -1, caso contrário. Qual a complexidade do seu algoritmo?

2. Modifique sua função do item anterior para retornar a primeira ocorrência de x. A complexidade do seu algoritmo se manteve igual a anterior?

3. Suponha que uma lista é rotacionada em um pivô desconhecido. Por exemplo, a lista dada por 0 1 2 4 5 6 7 pode se tornar 4 5 6 7 0 1 2. Dado essa lista e um inteiro x, retorne a posição de x se ele estiver na lista ou -1, caso contrário. Você pode assumir que não há números repetidos na lista. Qual a complexidade do seu algoritmo?

6 Algoritmos elementares de ordenação

1. Escreva uma função crescente que recebe uma lista e retorna True caso a lista esteja em ordem crescente, e False caso contrário. *Dica: você pode usar essa função futuramente para testar os seus próprios algoritmos de ordenação.*

2. Implemente os algoritmos de ordenação por inserção, seleção e o algoritmo da bolha. Caso a lista já esteja ordenada, qual algoritmo é mais rápido e qual é mais lento? E no caso da lista estar parcialmente ordenada?

3. Um algoritmo de ordenação é dito estável se a posição relativa dos elementos que têm o mesmo valor se mantem após a ordenação. Diga se os algoritmos de seleção, inserção e da bolha são ou não estáveis.

7 Questões relacionadas aos EPs

1. Qual a função da classe Token na análise léxica?

2. Agora que você sabe como ordenar uma lista, como você poderia usar isso para resolver o problema de múltiplas entradas (linhas) para a mesma personagem no EP3?

3. Como você pode implementar uma ordenação de Horarios? Seria possível manter uma função de ordenação que funcione tanto para inteiros quanto para Horarios? *Dica: pense na sobrecarga de operadores e métodos especiais.*

Referências

[1] Paulo Feofiloff. *Projeto de Algoritmos (em C).* https://www.ime.usp.br/ pf/algoritmos/, 2017.

[2] Brad Miller e David Ranum. *How to Think Like a Computer Scientist: Interactive Version.* (Inglês) [*Traduzido como "Aprendendo com Python: Edição interativa (usando Python 3.x)"*]. https://python.ime.usp.br/pensepy/static/pensepy/index.html, 2012.

[3] Departamento de Ciência da Computação, IME-USP. *Princípios de Desenvolvimento de Algoritmos em Python.* https://panda.ime.usp.br/algoritmos/static/algoritmos/index.html, 2015.

[4] Departamento de Ciência da Computação, IME-USP. *Aulas de Introdução à Computação em Python.* https://panda.ime.usp.br/aulasPython/static/aulasPython/index.html, 2015.