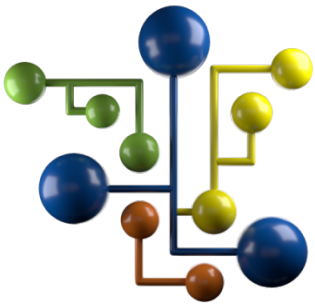

Data Science Academy

www.datascienceacademy.com.br

Introdução à Lógica de Programação

Antes de iniciar a Aula 7 do curso de Introdução à Lógica de programação, vamos à correção do exercício que deixei para você ao final da Aula 6.

Você fez o exercício? Você pode assistir aulas em vídeos, ler livros e artigos, ouvir Podcasts e vai aprender. Mas o conhecimento só é consolidado quando você experimenta por si mesmo! Seu cérebro é forçado a lembrar o que aprendeu e então o conhecimento fica registrado na sua memória. E isso ninguém pode fazer por você. Portanto, pratique até ter certeza de que compreendeu!

Aqui está a solução, comentada linha a linha. O único detalhe é a ordenação da lista (tema da Aula 7) e premissa para a Pesquisa Binária. Execute o programa no Jupyter Notebook para melhor visualização (cuidado com a identação):

```
# Função principal do programa
def main():

--# Lista fora de ordem
--autores = ['Monteiro Lobato', 'José de Alencar', 'Cecília Meireles', 'Carlos Drummond de Andrade', 'Machado de Assis', 'Clarice Lispector', 'Graciliano Ramos', 'Guimarães Rosa', 'Ruth Rocha', 'Luis Fernando Veríssimo']

--# Lista ordenada
--autores_ordenados = sorted(autores)

--# Solicita que o usuário digite o nome do autor
--autor = input('Digite o nome do autor para pesquisar na lista: ')

--# Grava a posição retornada pela pesquisa binária
--position = binary_search(autores_ordenados, autor)

--# Imprime mensagem de acordo com o resultado da pesquisa binária
--print("\nLista Ordenada de Autores: \n")
--print(autores_ordenados)
--if position == -1:
----print("\n")
----print("Desculpe, mas esse autor não faz parte da lista.")
--else:
----print("\n")
----print(autor, "é parte da lista e está na posição", position + 1, "( equivalente ao índice", position, ") na lista ordenada de autores.")

# Função para a pesquisa binária
def binary_search(autores_ordenados, autor):
```

```
--# Variáveis de controle
--primeiro_elemento = 0
--ultimo_elemento = len(autores_ordenados) - 1
--position = -1
--achei = False

--# Loop
--while not achei and primeiro_elemento <= ultimo_elemento:

----# Calcula o meio
----meio = (primeiro_elemento + ultimo_elemento) // 2

----# Verifica o meio e compara os elementos
----if autores_ordenados[meio] == autor:
------achei = True
------position = meio
----elif autores_ordenados[meio] > autor:
------ultimo_elemento = meio - 1
----else:
------primeiro_elemento = meio + 1

--return position

# Executa o programa
main()
```

Se tiver dúvidas sobre a solução acima, fique à vontade para postar nos comentários abaixo!

Agora: Aula 7 - Ordenação

Suponha que você tenha um monte de músicas no seu computador. Para cada artista, você tem um contador de plays. Você quer ordenar uma lista de artistas, do artista mais tocado para o menos tocado, para que possa categorizar os seus artistas favoritos. Como você pode fazer isso?

Uma maneira seria pegar o artista mais tocado da lista de músicas e adicioná-lo a uma nova lista. Faça isso de novo para encontrar o próximo artista mais tocado. Continue fazendo isso e então você terminará com uma lista ordenada.

Vamos pensar como engenheiros da computação e avaliar quanto tempo isso demoraria a ser executado. Lembre-se de que o tempo de execução O(n) significa que você precisa passar por todos os elementos da lista uma vez. Por exemplo, executar uma pesquisa simples na lista de artistas significa olhar para cada artista uma vez. Para encontrar o artista com o maior número

de plays você precisa verificar cada item da lista. Isso tem tempo de execução O(n). Então você tem uma operação com tempo de execução O(n) e precisa repetir essa operação n vezes.

Isso tem tempo de execução O(n × n) ou O(n^2). Algoritmos de ordenação são muito úteis. Agora você pode ordenar:

• Nomes em uma agenda telefônica
• Datas de viagem
• Emails (do mais novo ao mais antigo).

A Ordenação é uma habilidade que todo desenvolvedor de software e todo Cientista de Dados precisa ter algum conhecimento. Não apenas para passar nas entrevistas, mas como um entendimento geral da própria programação.

Os diferentes algoritmos de Ordenação são uma demonstração perfeita de como o design do algoritmo pode ter um efeito tão forte na complexidade, velocidade e eficiência do programa.

Vamos fazer um tour por alguns dos principais algoritmos de Ordenação e ver como podemos implementá-los em Python!

• Ordenação Por Bolha (Bubble Sort)

A Ordenação por "bolha" é normalmente ensinada nas aulas introdutórias de cursos de graduação em Ciência da Computação (parte fundamental em Data Science), uma vez que demonstra claramente como a Ordenação funciona, sendo simples e fácil de entender. A Ordenação por bolha percorre a lista e compara pares de elementos adjacentes. Os elementos são trocados se estiverem na ordem errada. A passagem pela parte não ordenada da lista é repetida até que a lista seja ordenada. Como a Ordenação por bolhas passa repetidamente pela parte não ordenada da lista, ela tem uma complexidade de pior caso de O (n^2).

Aqui você encontra o pseudocódigo do Buble Sort: https://pt.wikipedia.org/wiki/Bubble_sort

Aqui o código em Python (atenção com a identação):

```
# Ordenação por Bolha
def bubble_sort(lista):

--# Função que realiza a troca dos elementos
--def troca(i, j):
----lista[i], lista[j] = lista[j], lista[i]

--n = len(lista)
--trocado = True
```

```
--x = -1
--while trocado:
----trocado = False
----x = x + 1
----for i in range(1, n - x):
------if lista[i - 1] > lista[i]:
--------troca(i - 1, i)
--------trocado = True

--return lista

# Lista não ordenada
listaNum = [9, 3, 5, 4, 6, 2, 7, 1, 8]

# Ordena a lista
bubble_sort(listaNum)
```

• Ordenação Por Seleção (Selection Sort)

A Ordenação Por Seleção também é bastante simples, mas frequentemente supera a ordenação por bolhas. Com a Ordenação Por Seleção, dividimos nossa lista / array de entrada em duas partes: a sub-lista de itens já ordenados e a sub-lista de itens restantes a serem ordenados que compõem o restante da lista. Primeiro, encontramos o menor elemento na sub-lista não ordenada e o colocamos no final da sub-lista ordenada. Assim, estamos continuamente pegando o menor elemento não ordenado e colocando-o em ordem na sub-lista ordenada. Esse processo continua iterativamente até que a lista seja totalmente ordenada.

Aqui o pseudocódigo e um exemplo em Python:
https://pt.wikipedia.org/wiki/Selection_sort

• Ordenação Por Mistura (Merge Sort)

A Ordenação Merge Sort é um exemplo perfeitamente elegante de um algoritmo de divisão e conquista. É simples e usa as duas etapas principais de um algoritmo:

(1) Divide continuamente a lista não ordenada até que você tenha N sub-listas, onde cada sub-lista possui 1 elemento "não ordenado" e N é o número de elementos na lista original.

(2) Mescla repetidamente as sub-listas juntando 2 de cada vez para produzir novas sub-listas ordenadas até que todos os elementos tenham sido totalmente mesclados em uma única lista ordenada.

Aqui o pseudocódigo e um exemplo em Python:
https://pt.wikipedia.org/wiki/Merge_sort

• Ordenação Rápida (Quick Sort)

A ordenação rápida também é um algoritmo de divisão e conquista, como o Merge Sort. Embora seja um pouco mais complicado, na maioria das implementações ele executa significativamente mais rápido que a ordenação Merge Sort e raramente atinge a pior complexidade de O (n^2). Possui 3 etapas principais:

(1) Primeiro selecionamos um elemento que chamaremos de pivô da matriz.

(2) Movemos todos os elementos menores que o pivô, para a esquerda do pivô; movemos todos os elementos maiores que o pivô, para a direita do pivô. Isso é chamado de operação de partição.

(3) Aplicamos recursivamente as duas etapas acima separadamente em cada uma das sub-matrizes de elementos com valores menores e maiores que o último pivô.

Aqui o pseudocódigo e um exemplo em Python:
https://pt.wikipedia.org/wiki/Quicksort

Esses são os principais algoritmos de ordenação e podem ser úteis quando precisamos ordenar um conjunto de dados durante o pré-processamento. Muitos algoritmos de Machine Learning usam esses tipos de ordenação internamente para operação do algoritmo.

Um dos maiores benefícios de estudar lógica de programação é o desenvolvimento do raciocínio. Por isso programação é muito recomendado para crianças, mas todos nós obtemos benefícios ao desenvolver nossas habilidades cognitivas, ao mesmo tempo que aprendemos um importante skill em Data Science.

Agora é com você! No link abaixo você encontra mais de 20 exercícios de pesquisa binária e ordenação. Os exercícios possuem enunciado e depois você pode clicar no link abaixo do enunciado e conferir a resposta. Tente resolver alguns exercícios e compreender bem os algoritmos que estudamos esta semana até aqui.

https://www.w3resource.com/python-exercises/data-structures-and-algorithms/

A tabela abaixo resume o tempo de execução e complexidade dos principais algoritmos de pesquisa e ordenação. Fonte:

https://www.hackerearth.com/practice/notes/sorting-and-searching-algorithms-time-complexities-cheat-sheet/

#aula07