## **Etapa Introdução aos estudos**

### Estruturas de dados

Estruturas de dados é fundamental para entender como os dados podem ser organizados e acessados de forma eficiente dentro de um sistema computacional. Elas são responsáveis por determinar o layout dos dados na memória, bem como as operações que podem ser realizadas sobre esses dados, como inserção, remoção, busca e ordenação.

Estruturas de dados, como arrays, listas, pilhas, filas, árvores e grafos, são usadas para resolver uma ampla gama de problemas de maneira eficiente. Essas estruturas não apenas facilitam o desenvolvimento de software, mas também desempenham um papel crucial na otimização de recursos, como tempo de execução e uso de memória. Por exemplo, uma tabela hash pode realizar buscas em tempo constante, enquanto uma lista encadeada pode ser usada para cenários onde a memória precisa ser gerida dinamicamente.

Além disso, a escolha correta da estrutura de dados tem um impacto direto no desempenho dos algoritmos, influenciando a complexidade temporal e espacial das operações realizadas. Compreender o uso adequado de cada estrutura é essencial para o desenvolvimento de software eficiente e escalável, especialmente em sistemas que manipulam grandes volumes de dados ou que possuem requisitos de alto desempenho.



## 

## 

## 

## 

## 

## 

## **Etapa Aprofundamento de estudos: Estruturas de Dados e Notação Assintótica**

### Introdução ao Tema

As estruturas de dados são fundamentais no campo da ciência da computação. Elas determinam como os dados são armazenados e organizados em um programa, e essa organização é o que permite que os algoritmos operem de maneira eficiente. Uma estrutura de dados é, em essência, uma maneira de guardar e manipular informação.

Para explicar isso de forma simples, imagine que você tem vários brinquedos. Você pode organizá-los em uma fila, onde o primeiro brinquedo que entra é o primeiro a sair, ou pode empilhá-los, onde o último brinquedo colocado será o primeiro a ser retirado. Esses são exemplos de estruturas de dados no mundo real que têm correspondência direta na computação.

Agora, vamos explorar os principais tipos de estruturas de dados e entender como elas funcionam na prática, com exemplos em Python para ilustrar o uso.

### Estruturas de Dados: Conceito Teórico e Aplicação Prática

### 1. Array (Matriz)

Um array é uma estrutura de dados simples, onde os dados são organizados de forma contínua e indexada. Isso significa que podemos acessar qualquer elemento do array diretamente, usando um número chamado índice.

Aplicação prática: Imagine um armário onde cada prateleira tem um número. Se você quer pegar algo na prateleira número 3, basta ir até lá diretamente.

# Exemplo simples de Array

array = [10, 20, 30, 40, 50]

print("Acessando o terceiro elemento:", array[2]) # Output: 30

### 2. Lista Encadeada

Uma lista encadeada é como uma cadeia de nós, onde cada nó contém um dado e um ponteiro para o próximo nó. Essa estrutura é mais flexível que os arrays porque os dados não precisam estar armazenados de forma contínua.

Aplicação prática: Imagine que você está em uma fila e cada pessoa tem uma cordinha que a conecta à próxima pessoa na fila.

class No:

def \_\_init\_\_(self, dado):

self.dado = dado

self.proximo = None

class ListaEncadeada:

def \_\_init\_\_(self):

self.cabeca = None

def adicionar(self, dado):

novo\_no = No(dado)

novo\_no.proximo = self.cabeca

self.cabeca = novo\_no

# Criando uma lista encadeada

lista = ListaEncadeada()

lista.adicionar(10)

lista.adicionar(20)

lista.adicionar(30)

### 3. Pilha (LIFO)

Uma pilha é uma estrutura do tipo Last In, First Out (último a entrar, primeiro a sair). Isso significa que o último item adicionado será o primeiro a ser retirado. É como uma pilha de livros: o último colocado em cima é o primeiro a ser removido.

Aplicação prática: Imagine uma pilha de pratos. O último prato colocado no topo é o primeiro que você vai pegar.

pilha = []

# Adicionando elementos na pilha

pilha.append('Prato1')

pilha.append('Prato2')

pilha.append('Prato3')

print("Removendo o último prato:", pilha.pop()) # Removerá 'Prato3'

### 4. Fila (FIFO)

Uma fila é uma estrutura do tipo First In, First Out (primeiro a entrar, primeiro a sair). O primeiro item adicionado à fila será o primeiro a ser retirado, como uma fila de pessoas esperando para entrar em um cinema.

Aplicação prática: Pense em uma fila de banco, onde a primeira pessoa a chegar será a primeira a ser atendida.

from collections import deque

fila = deque()

# Adicionando elementos na fila

fila.append('Pessoa1')

fila.append('Pessoa2')

fila.append('Pessoa3')

print("Atendendo a primeira pessoa:", fila.popleft()) # Removerá 'Pessoa1'

### 5. Tabelas Hash

As tabelas hash são estruturas de dados que armazenam pares chave-valor. Elas permitem acesso rápido a dados, desde que conheçamos a chave. Essa estrutura é muito eficiente para busca e inserção.

Aplicação prática: Pense em um índice de um livro, onde você procura o tópico desejado (chave) e rapidamente encontra a página (valor).

tabela\_hash = {}

# Adicionando pares chave-valor

tabela\_hash['nome'] = 'Carlos'

tabela\_hash['idade'] = 25

print("Nome armazenado:", tabela\_hash['nome']) # Output: Carlos

## Aprofundamento: Notação Assintótica (Big O)

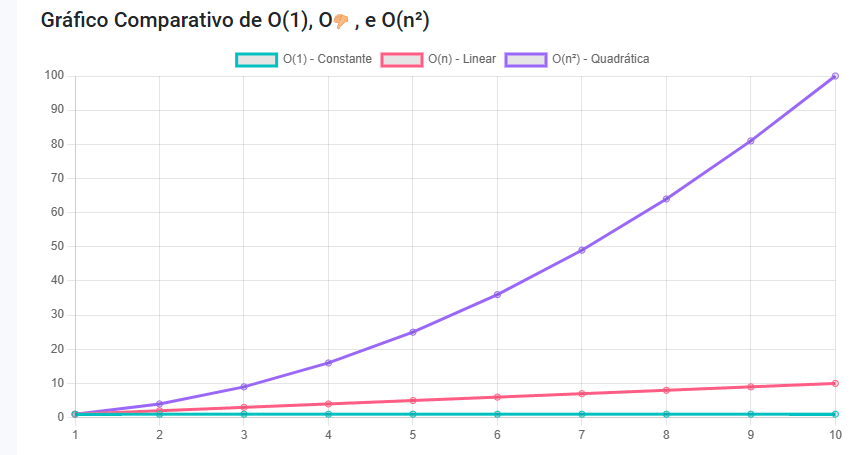
A notação assintótica, ou Big O, é uma maneira de medir o desempenho de algoritmos. Ela nos ajuda a entender como o tempo de execução de um algoritmo cresce à medida que o tamanho dos dados aumenta.

Aqui estão as principais notações usadas:

* O(1): Constante. O tempo de execução é o mesmo, independentemente do tamanho dos dados.
* O(n): Linear. O tempo de execução aumenta proporcionalmente ao tamanho dos dados
* O(n²): Quadrático. O tempo de execução aumenta drasticamente à medida que os dados crescem.

### Exemplo visual

Imagine que você precisa encontrar um brinquedo específico em uma caixa. Se a caixa estiver bem organizada (como um array), você poderá encontrar o brinquedo rapidamente (O(1)). Mas se a caixa estiver desorganizada, você terá que olhar cada brinquedo, um por um O(n)).



## Teoria e Prática com Notação Assintótica

Na prática, escolher a estrutura de dados certa é essencial para garantir a eficiência de um programa. A notação Big O ajuda os desenvolvedores a prever quanto tempo um algoritmo vai levar à medida que o número de dados cresce.

* O(1): Ideal quando você pode acessar os dados diretamente, como em uma tabela hash.
* O(n): É aceitável para algoritmos de busca lineares em uma lista ou array.
* O(n²): Deve ser evitado quando possível, pois se torna muito lento para grandes quantidades de dados.

## Conclusão

Estruturas de dados são essenciais para qualquer programador que deseje construir sistemas eficientes e bem organizados. Desde arrays simples até grafos complexos, cada estrutura tem suas vantagens e desvantagens, e saber qual usar em diferentes cenários é o que distingue um bom desenvolvedor.

Agora que exploramos a parte teórica e prática de estruturas de dados e notação assintótica, você está pronto para aplicar esses conceitos nos seus próprios projetos, utilizando Python como ferramenta para implementar essas soluções.

## **Saiba Mais:**

**Estruturas de dados:**

são formas sistemáticas de organizar e acessar dados. Elas permitem que os desenvolvedores armazenem, recuperem e manipulem dados de maneira eficiente. Em essência, são os alicerces sobre os quais os algoritmos operam.

### Para que servem?

Imagine que você está construindo uma biblioteca. Sem prateleiras, encontrar um livro específico seria uma tarefa hercúlea. Estruturas de dados são como essas prateleiras, organizando e tornando a informação acessível.

Já parou para pensar como suas playlists no Spotify são estruturadas? Ou como o botão “voltar” do seu navegador funciona para te levar à página anterior? E aqui vai outra pergunta: você sabe como funcionam os índices dos bancos de dados que tornam possíveis consultas rápidas em meio a milhões de registros? Os índices são um exemplo cristalino de estruturas de dados em ação, otimizando a busca e acesso a informações em grandes volumes de dados.



### Big techs

Já se perguntou por que empresas como Google, Facebook e Amazon dão tanta ênfase às estruturas de dados em suas entrevistas técnicas e nem tanto à qual linguagem de programação você vai usar? É porque elas reconhecem a importância fundamental dessas estruturas na criação de soluções eficientes e escaláveis. Um profissional que domina estruturas de dados é alguém que pode resolver problemas complexos de maneira otimizada.

### Linguagens de Programação vs. Estruturas de Dados

No mundo da programação, frequentemente somos bombardeados com a mais recente e mais popular linguagem ou *framework*. É fácil ser atraído por essa nova onda e esquecer dos fundamentos. No entanto, enquanto linguagens de programação e *frameworks* evoluem e mudam com o tempo, os fundamentos da Ciência da Computação, como estruturas de dados, permanecem consistentes e atemporais.

Linguagens de programação são ferramentas — elas são meios para um fim. Elas são a interface entre o desenvolvedor e a máquina, permitindo-nos traduzir nossa lógica e algoritmos em instruções que o computador pode entender e executar. A popularidade e relevância de uma linguagem podem variar com o tempo. O que é predominante hoje pode não ser amanhã.

Por outro lado, estruturas de dados são os tijolos fundamentais da Ciência da Computação. Elas são o coração da lógica e do raciocínio que impulsionam os algoritmos. Dominar estruturas de dados significa ter uma compreensão profunda de como armazenar, acessar e manipular dados de maneira eficiente. Esse conhecimento é transferível e relevante, independentemente da linguagem ou do *framework* que você está usando.

Retomando o ponto sobre as grandes empresas de tecnologia: elas valorizam profissionais que entendem os fundamentos porque sabem que esses indivíduos podem se adaptar e aprender qualquer linguagem ou ferramenta necessária. Afinal, quando você entende os princípios subjacentes, pode aplicá-los em qualquer contexto.

Um conselho: " Evite se envolver em discussões como “a linguagem A é melhor que a linguagem B” sem um contexto específico. Cada linguagem tem suas forças e fraquezas, e o que é ideal para um projeto ou situação pode não ser para outro. Em vez de se fixar em debates superficiais, concentre-se em compreender os fundamentos e em como aplicar a melhor ferramenta para o trabalho em questão." Ramon Sorage

[Vídeo | Estrutura de dados com Roberta Arcoverde](https://youtu.be/57VqgNjbzrg)