**Estruturas de dados em Python**

“Quando falamos de uma estrutura de dados, estamos falando nas operações que essa estrutura nos permite realizar, bem como da complexidade assintótica dessas operações.” (Algoritmosempython, 2017)

A seleção da estrutura apropriada depende das ações que seu programa realizará com maior frequência. De maneira geral, o objetivo é optar por uma estrutura de dados que reduza ao máximo o custo das operações mais comuns.

Estruturas de dados podem ser classificadas em dois grupos:

**Estruturas Contíguas:**

-Os elementos são armazenados em posições de memória adjacentes, ou seja, um após o outro.

-Acesso direto aos elementos por meio de índices.

-Tamanho fixo, uma vez alocado, não pode ser facilmente modificado.

-Eficiente para acessar elementos por índice, mas ineficiente para inserções/remoções no meio da estrutura, pois pode exigir deslocamento de elementos.

**Estruturas Ligadas:**

-Os elementos são armazenados em blocos de memória chamados nós, cada nó contém dados e uma referência ao próximo nó.

-Não requer espaço contíguo na memória.

-Tamanho dinâmico, fácil de adicionar ou remover elementos.

-Acesso aos elementos requer percorrimento da estrutura, a menos que seja uma lista duplamente ligada com referências para trás.

-Eficiente para inserções/remoções, mas menos eficiente para acesso aleatório.

Em resumo, estruturas contíguas são adequadas quando o tamanho é fixo e o acesso aleatório é importante, enquanto estruturas ligadas são mais flexíveis e adequadas para inserções/remoções dinâmicas, mas têm um custo maior para o acesso aleatório de elementos. Cada uma tem suas vantagens e desvantagens, e a escolha depende dos requisitos específicos de um problema.

**Algoritmos de Busca**

“Um algoritmos de busca em vetor é um algoritmo para procurar a presença ou não de

determinado valor em uma sequência de dados em memória (em um vetor). Deve-se destacar que a realização de busca é provavelmente o algoritmo mais empregado na prática, por exemplo, sempre que acessa um sistema com usuário e senha será necessário buscar pelo seu usuário e depois verificar se a senha está correta.” (Edisciplinas, 2023)

Algoritmos de busca em Python são procedimentos ou métodos que permitem encontrar a localização ou verificar a presença de um valor específico em uma coleção de dados, como uma lista, tupla ou dicionário, usando a linguagem de programação Python. Esses algoritmos são essenciais para a manipulação eficiente de dados e desempenham um papel fundamental em diversas aplicações, desde a pesquisa de elementos em listas até a consulta de informações em estruturas de dados mais complexas. Python oferece várias formas de implementar algoritmos de busca, incluindo a busca linear, a busca binária e a busca em dicionários, cada uma adequada para diferentes tipos de coleções e requisitos de desempenho. Esses algoritmos são amplamente utilizados em programação para resolver uma variedade de problemas, desde ordenação e filtragem de dados até a pesquisa em bancos de dados e sistemas de busca na web.

**Busca Sequencial**

“Quando itens de dados são armazenados numa coleção tal qual uma lista, nós dizemos que eles têm uma relação linear ou sequencial. Cada item de dado é armazenado numa posição relativa aos demais. Em listas do Python, essas posições relativas são os índices dos itens individuais.” (Pandaime, 2019)

A busca sequencial, também conhecida como busca linear, é um método simples de busca em Python (ou em qualquer linguagem de programação) que envolve percorrer uma sequência de elementos, um por um, até encontrar o valor desejado ou determinar que o valor não está presente.

Exemplo de como realizar uma busca sequencial em Python:

Texto

Descrição gerada automaticamente

Fonte: Grupo

Neste exemplo, a função busca\_sequencial recebe uma lista e um valor de destino. Ela itera pela lista, comparando cada elemento com o valor de destino. Se o valor de destino for encontrado, a função retorna o índice desse elemento na lista. Se o valor não for encontrado após percorrer toda a lista, a função retorna -1 para indicar que o elemento não está presente.

A busca sequencial é direta e fácil de entender, mas pode ser ineficiente em listas muito grandes, pois é necessário percorrer todos os elementos, por caso o (n), onde "n" é o tamanho da lista.

**Busca Binária**

“Os algoritmos de pesquisa (ou de busca) binária também são conhecidos como pesquisa de meio intervalo. Eles retornam a posição do valor pesquisado em uma lista classificada.

Esses algoritmos usam a técnica de "dividir e conquistar" para encontrar a posição do valor.” (Freecodecamp, 2022)

A busca binária em Python é um algoritmo de busca eficiente usado para encontrar um elemento em uma lista ordenada. A principal característica da busca binária é que ela divide repetidamente a lista pela metade, eliminando metade dos elementos a cada iteração, até encontrar o elemento desejado ou determinar que o elemento não está presente. É uma técnica muito mais rápida do que a busca sequencial, especialmente em listas grandes.

Exemplo de como realizar uma busca binária em Python:

**Texto

Descrição gerada automaticamente**

Fonte: Grupo

Neste exemplo, a função busca\_binaria recebe uma lista ordenada e um valor de destino. Ela mantém duas variáveis, inicio e fim, que representam a faixa de elementos a serem pesquisados. A cada iteração do loop while, a função calcula o índice do meio da faixa e compara o elemento no meio com o valor de destino. Com base nessa comparação, a função decide se deve procurar na metade esquerda ou direita da lista e atualiza as variáveis inicio e fim de acordo.

A busca binária é muito eficiente, com uma complexidade de tempo de O(log n), onde "n" é o tamanho da lista. Isso a torna ideal para listas grandes, onde a busca sequencial seria muito mais lenta.

**Algoritmos de Ordenação**

“Ordenar conjuntos de coisas é uma tarefa costumeira em nossa vida. Em se tratando de computadores, como geralmente lidamos com grandes volumes de dados, é desejável ter alguma ordem nesses dados para facilitar sua manipulação.” (Algoritmosempython, 2017)

Algoritmos de ordenação em Python são métodos ou procedimentos usados para organizar os elementos de uma lista (ou array) em uma ordem específica, como crescente ou decrescente. A ordenação é uma tarefa fundamental em ciência da computação e é usada em uma ampla variedade de aplicativos. Existem vários algoritmos de ordenação disponíveis em Python, e cada um deles possui características e desempenhos diferentes.

Algoritmos de ordenação mais comuns em Python:

**Bubble Sort (Ordenação por Bolha):**

-Este é um algoritmo simples que compara pares de elementos adjacentes e os troca se estiverem fora de ordem.

-É ineficiente para listas grandes e não é recomendado para uso em casos reais.

**Insertion Sort (Ordenação por Inserção):**

-Este algoritmo percorre a lista e insere cada elemento na posição correta, deslocando os elementos maiores para abrir espaço.

-É eficiente para pequenas listas e quase ordenadas.

**Selection Sort (Ordenação por Seleção):**

-Este algoritmo encontra o menor elemento da lista e o coloca na posição correta.

-Não é eficiente em listas grandes, pois sempre faz o mesmo número de comparações, independentemente da entrada.

**Merge Sort (Ordenação por Mesclagem):**

-Este é um algoritmo de divisão e conquista que divide a lista em partes menores, ordena essas partes e depois mescla-as em ordem.

-É eficiente para listas grandes e tem um desempenho consistente.

**Quick Sort (Ordenação Rápida):**

-Outro algoritmo de divisão e conquista que escolhe um elemento "pivô", divide a lista em elementos menores e maiores que o pivô, e depois os ordena recursivamente.

-Geralmente é rápido e eficiente na prática.

**Heap Sort (Ordenação por Heap):**

-Este algoritmo utiliza uma estrutura de dados chamada heap para organizar os elementos em ordem crescente.

-Tem um desempenho eficiente e é adequado para listas grandes.

**Timsort:**

-O Timsort é um algoritmo híbrido baseado no Merge Sort e no Insertion Sort. Ele é usado como o algoritmo de ordenação padrão no Python e é otimizado para diferentes tipos de entradas.

Para usar esses algoritmos em Python, você pode implementá-los manualmente ou usar as funções de ordenação incorporadas, como sorted() e list.sort(). A escolha do algoritmo de ordenação depende das características da sua lista (tamanho, estado inicial, etc.) e dos requisitos de desempenho do seu aplicativo. Em muitos casos, as funções de ordenação incorporadas do Python serão suficientes e eficientes para suas necessidades.

**Listas**

“Ao longo da vida de uma pessoa desenvolvedora, independentemente da plataforma e linguagem, é comum a necessidade de lidar com listas. Um dos exemplos mais comuns de utilização de lista é o armazenamento de dados dos usuários de uma aplicação, como os nomes e endereços. A lista é uma estrutura de dados composta por elementos organizados de forma linear, na qual cada um pode ser acessado a partir de um índice, que representa sua posição na coleção (iniciando em zero).” (Kenzie, 2022)

Em Python, uma lista é uma estrutura de dados que permite armazenar uma coleção ordenada de elementos. Esses elementos podem ser de diferentes tipos, como números, strings, objetos, outras listas e assim por diante. As listas são mutáveis, o que significa que você pode adicionar, remover ou modificar elementos após a criação da lista.

Exemplo simples de uma lista em Python:



Fonte: Grupo

Neste exemplo, ‘minha\_lista’ é uma lista que contém cinco números inteiros. Você pode acessar os elementos da lista usando índices. Por exemplo:

Tela preta com letras brancas

Descrição gerada automaticamente

Fonte: Grupo

Você também pode realizar várias operações em listas, como adicionar elementos, remover elementos, verificar o tamanho da lista e muito mais. Abaixo algumas operações comuns em listas:

Texto

Descrição gerada automaticamente

Fonte: Grupo

As listas são uma das estruturas de dados mais versáteis em Python e são amplamente usadas para armazenar coleções de dados de diferentes tipos. Elas desempenham um papel fundamental na programação Python e são uma escolha comum para muitas tarefas de manipulação de dados e algoritmos.

**Tuplas**

“O Python nos disponibiliza um tipo que preenche muito melhor as características que procurávamos — as tuplas. Tuplas podem ser consideradas similares às listas, mas suas diferenças são cruciais.” (Alura, 2023)

Em Python, uma tupla é uma estrutura de dados semelhante a uma lista, mas com uma diferença fundamental: as tuplas são imutáveis, enquanto as listas são mutáveis. Isso significa que, uma vez que uma tupla é criada, seus elementos não podem ser alterados, adicionados ou removidos. As tuplas são frequentemente usadas para armazenar coleções ordenadas de elementos que não devem ser modificados.

Exemplo de uma tupla em Python:



Fonte: Grupo

Assim como nas listas, você pode acessar os elementos de uma tupla usando índices:

Texto

Descrição gerada automaticamente

Fonte: Grupo

No entanto, você não pode modificar os elementos de uma tupla depois de criá-la. Por exemplo, a seguinte operação geraria um erro:

Texto

Descrição gerada automaticamente com confiança média

Fonte: Grupo

As tuplas são frequentemente usadas em situações em que você deseja garantir que os dados não sejam acidentalmente modificados. Além disso, elas podem ser usadas como chaves em dicionários (enquanto listas não podem, porque são mutáveis), o que as torna úteis em estruturas de dados mais complexas.

Você pode criar uma tupla com ou sem parênteses. Por exemplo, as duas formas abaixo são equivalentes:

Tela preta com letras brancas

Descrição gerada automaticamente

Fonte: Grupo

Em muitos casos, as tuplas são usadas para representar coleções de dados relacionados, como coordenadas (latitude e longitude), data e hora (ano, mês, dia, hora, minuto, segundo), e assim por diante.

**Dicionário**

“Um dicionário Python é uma forma de coleção de dados em que se guarda uma chave e um valor correspondente. É similar a um dicionário mesmo, em que há sempre um termo e uma tradução.” (Hashtagtreinamentos, 2023)

Em Python, um dicionário é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor. Cada elemento em um dicionário consiste em uma chave única associada a um valor correspondente. Os dicionários são uma das estruturas de dados mais versáteis e amplamente utilizadas na linguagem.

Exemplo simples de um dicionário em Python:



Fonte: Grupo

Neste exemplo, meu\_dicionário é um dicionário que contém três pares chave-valor. As chaves são "nome", "idade" e "cidade", e os valores associados a essas chaves são "João", 30 e "São Paulo", respectivamente.

Você pode acessar os valores em um dicionário usando as chaves correspondentes:

Tela de celular com aplicativo aberto

Descrição gerada automaticamente com confiança média

Fonte: Grupo

Os valores em um dicionário podem ser de qualquer tipo de dado, incluindo números, strings, listas, tuplas e até mesmo outros dicionários. Além disso, as chaves de um dicionário devem ser únicas, mas os valores podem se repetir.

Você pode realizar várias operações em dicionários, como adicionar novos pares chave-valor, modificar valores existentes, remover pares chave-valor, verificar se uma chave está presente no dicionário e muito mais. Aqui estão algumas operações comuns em dicionários:

Texto

Descrição gerada automaticamente

Fonte: Grupo

Os dicionários são especialmente úteis quando você precisa associar informações relacionadas entre si. Eles são amplamente usados em Python para representar configurações, armazenar dados estruturados e organizar informações de várias maneiras eficientes.

**Sets**

“Os sets são uma coleção de itens desordenada, parcialmente imutável e que não podem conter elementos duplicados. Por ser parcialmente imutável, os sets possuem permissão de adição e remoção de elementos.” (FagnerPinheiro, 2020)

Em Python, um conjunto (set) é uma estrutura de dados que representa uma coleção não ordenada e sem elementos duplicados. Os conjuntos são usados quando você deseja armazenar um grupo de valores únicos, e eles são bastante eficientes para verificar a existência de um elemento em uma coleção sem a necessidade de percorrer toda a coleção. Os elementos de um conjunto podem ser de vários tipos de dados, como números, strings e outros objetos.

Exemplo de como criar e usar um conjunto em Python:

Texto

Descrição gerada automaticamente

Fonte: Grupo

Os conjuntos são úteis quando você deseja eliminar duplicatas de uma lista de elementos, verificar a associação de elementos a um conjunto ou realizar operações de conjunto, como união, interseção e diferença.

Exemplo de operações de conjunto em Python:

Texto

Descrição gerada automaticamente

Fonte: Grupo

Os conjuntos também podem ser criados usando a função set() e podem ser convertidos de e para listas e outros iteráveis usando a função set() ou list(), respectivamente.

**Referências**

TEIXEIRA, Douglas. “Estrutura de dados”, 2017.

EDISCIPLINAS. “Introdução aos algoritmos de busca em Python”, 2023.

PANDAIME. “Busca sequencial”, 2019.

FREECODECAMP. “Algoritmo de pesquisa binária”, 2022.

KRIGER, Daniel. “Lista em Python”, 2022.

ALURA. “Tupla no Python”, 2023.

HASHTAGTREINAMENTOS, “Dicionário Python”, 2023.

PINHEIRO, Fagner. “Manipulando sets no Python”, 2020.