

Python Estrutura de Dados

Módulo 07
Aula 12

Estrutura de Dados

Ordenação

A ordenação e busca são operações fundamentais em ciência da computação, com vastas aplicações no processamento e análise de dados. Esta aula abordará técnicas avançadas de ordenação e busca, explorando algoritmos eficientes e suas aplicações em problemas do mundo real.

Ordenação Avançada

Algoritmos de Ordenação

- **QuickSort:** Um algoritmo de divisão e conquista que seleciona um elemento como pivô e particiona o array em torno do pivô.
- **MergeSort:** Também baseado em divisão e conquista, divide o array em metades, ordena cada metade e depois mescla as metades ordenadas.
- **HeapSort:** Utiliza uma estrutura de dados heap para organizar os elementos do array e extrair o maior ou menor elemento repetidamente.

Comparação de Algoritmos

- Análise de complexidade de tempo e espaço.
- Casos de uso ideal para cada algoritmo.
- Vantagens e desvantagens.

QuickSort

QuickSort é um algoritmo de ordenação eficiente que segue o princípio de divisão e conquista. Ele escolhe um elemento do array para ser o pivô e particiona o array em torno desse pivô, de modo que os elementos menores que o pivô são movidos para antes dele e os elementos maiores são movidos para depois. Esse processo é repetido recursivamente para as sub-partes do array até que o array esteja completamente ordenado.

Etapas do QuickSort

- **Escolha do Pivô:** Selecionar um elemento do array como pivô. A escolha do pivô pode afetar o desempenho do algoritmo, e diferentes estratégias podem ser usadas, como escolher o primeiro elemento, o último, o mediano ou um elemento aleatório.
- **Particionamento:** Reorganizar o array de forma que todos os elementos menores que o pivô venham antes dele, e todos os elementos maiores venham depois. Ao final do particionamento, o pivô estará em sua posição final ordenada no array.
- **Recursão:** Aplicar recursivamente o mesmo processo aos sub-arrays formados pelos elementos à esquerda e à direita do pivô.

Complexidade

- **Melhor caso:** $O(n \log n)$, quando o particionamento divide o array de maneira equilibrada.
- **Pior caso:** $O(n^2)$, quando o particionamento resulta em um lado muito maior que o outro a cada passo, como quando o array já está ordenado ou o pivô é sempre o menor ou maior elemento.
- **Caso médio:** $O(n \log n)$, o que faz do QuickSort uma escolha eficiente para muitas aplicações.



O código utiliza compreensões de lista para criar os sub-arrays menores e maiores que são, então, ordenados recursivamente. O elemento na posição zero do array é escolhido como pivô para simplicidade, mas em implementações mais complexas, outras estratégias de escolha do pivô podem ser adotadas (neste exemplo utilizamos o último elemento do array) para otimizar o desempenho em diferentes conjuntos de dados.

MergeSort

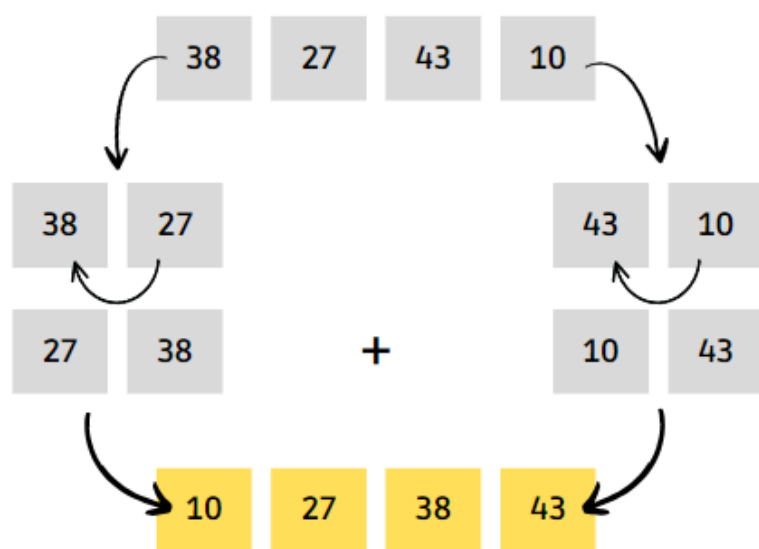
MergeSort é um algoritmo de ordenação que utiliza o princípio de divisão e conquista para ordenar elementos de um array ou lista. O processo envolve dividir a estrutura de dados original em partes menores, ordenar essas partes (se necessário) e, por fim, mesclar as partes ordenadas em uma única estrutura ordenada.

Etapas do HeapSort

- **Construção do Heap:** Transformar o array em um max heap, onde o maior elemento é o root do heap.
- **Ordenação:** Remover repetidamente o elemento máximo do heap (root), colocá-lo na parte ordenada do array e reajustar o heap para manter sua propriedade de max heap. Isso é feito até que todos os elementos sejam removidos do heap e inseridos no array em ordem.

Complexidade

- **Tempo:** $O(n \log n)$ tanto no melhor quanto no pior caso, tornando-o bastante eficiente para grandes conjuntos de dados.
- **Espaço:** $O(1)$ em termos de espaço extra, pois o HeapSort pode ser realizado in-place no array de entrada.



A principal vantagem do MergeSort é sua eficiência e previsibilidade em termos de desempenho, funcionando bem em uma variedade de conjuntos de dados, independentemente da ordem inicial dos elementos. Contudo, seu ponto de atenção é o uso adicional de memória, necessário para os arrays temporários durante a etapa de mesclagem.

O MergeSort é especialmente útil em situações onde a previsibilidade do tempo de execução é crítica, como em aplicações de tempo real, ou quando trabalha-se com grandes conjuntos de dados que não cabem inteiramente na memória (ordenando-se por meio de acesso a disco, por exemplo).

HeapSort

HeapSort é um algoritmo de ordenação eficiente que utiliza uma estrutura de dados chamada heap para organizar os elementos do array. Um heap é uma árvore binária completa onde cada nó pai é menor (min heap) ou maior (max heap) que seus nós filhos. O HeapSort aproveita as propriedades do max heap para ordenar elementos em ordem crescente (ou min heap para ordem decrescente).

Etapas do HeapSort

- **Construção do Heap:** Transformar o array em um max heap, onde o maior elemento é o root do heap.
- **Ordenação:** Remover repetidamente o elemento máximo do heap (root), colocá-lo na parte ordenada do array e reajustar o heap para manter sua propriedade de max heap. Isso é feito até que todos os elementos sejam removidos do heap e inseridos no array em ordem.

Complexidade

- **Tempo:** $O(n \log n)$ tanto no melhor quanto no pior caso, tornando-o bastante eficiente para grandes conjuntos de dados.
- **Espaço:** $O(1)$ em termos de espaço extra, pois o HeapSort pode ser realizado in-place no array de entrada.

