Heapsort: Um Algoritmo Eficiente de Ordenação com Complexidade de Tempo O(n log n)

Aluna: Luana Roza de Oliveira

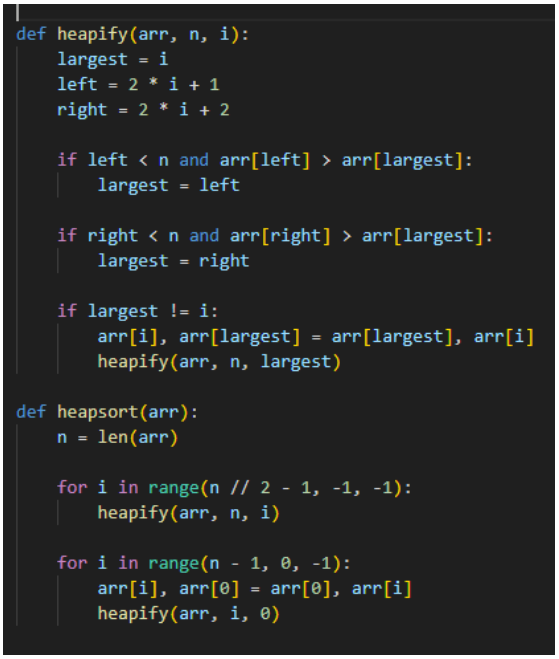
1 Introdução

O Heapsort é um dos algoritmos de ordenação mais eficientes disponíveis, com uma complexidade de tempo de O(n log n). Este artigo explora o funcionamento do Heapsort, incluindo seu código, demonstrará sua complexidade de tempo e discutirá por que ele não é um algoritmo de ordenação estável.

2 Heapsort: Uma Visão Geral

O Heapsort é baseado em uma estrutura de dados chamada Heap, mais especificamente, um Max Heap. Um Max Heap é uma árvore binária em que o valor de cada nó é maior ou igual ao valor de seus filhos. O Heapsort funciona da seguinte maneira:

- Construir um Max Heap a partir da lista de elementos a serem ordenados.
- Trocar o elemento raiz (o maior) com o último elemento da lista.
- Reduzir o tamanho do heap em 1.
- Restaurar a propriedade de Max Heap (através de uma operação chamada "heapify").
- Repetir os passos 2-4 até que o heap tenha tamanho 1.



```python
def heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and arr[left] > arr[largest]:
        largest = left

    if right < n and arr[right] > arr[largest]:
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heapsort(arr):
    n = len(arr)

    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
```

Código Heapsort

Este código implementa o algoritmo Heapsort, que é um algoritmo de classificação baseado em estruturas de dados chamadas de heaps. Aqui está uma explicação passo a passo do código:

heapify(arr, n, i) é uma função que transforma a subárvore com raiz em i em um heap máximo. Um heap máximo é uma árvore binária onde o elemento pai é sempre maior ou igual aos elementos filhos. Esta função recebe a matriz arr, seu tamanho n e o índice i do elemento raiz da subárvore.
a. Inicialmente, largest é definido como i, representando o índice do elemento atual.
b. Os índices dos elementos filhos esquerdo e direito são calculados.
c. Se o filho esquerdo existe (left < n) e é maior que o elemento atual, atualiza largest para o índice do filho esquerdo.
d. Se o filho direito existe (right < n) e é maior que o elemento armazenado em largest, atualiza largest para o índice do filho direito.

1- e. Se largest não é igual a i, troca os elementos arr[i] e arr[largest] e chama recursivamente heapify na subárvore com raiz em largest.

heapsort(arr) é a função principal que classifica a matriz arr usando o Heap Sort.
a. Calcula o tamanho da matriz n.
b. Constrói um heap máximo iterando sobre os elementos da matriz, começando do meio (índice n // 2 - 1) e indo até o início (índice 0).

2- c. Em seguida, o loop for classifica o array:

- Troca o primeiro elemento (índice 0) com o último elemento não classificado (índice i).
- Chama heapify na subárvore não classificada, agora com tamanho i.
- Repete este processo até que todos os elementos estejam classificados.

Ao final deste processo, a matriz ‘arr’ estará classificada em ordem crescente.

Este algoritmo é eficiente e tem uma complexidade de tempo de O(n log n), tornando-o uma boa escolha para classificar grandes conjuntos de dados.

3 Complexidade de Tempo

A complexidade de tempo do Heapsort é O(n log n). A construção do Max Heap tem complexidade O(n) e a etapa de ordenação subsequente envolve n-1 iterações, cada uma com uma operação de heapify de complexidade O(log n). Portanto, o tempo total é O(n + (n-1) * log n), que é assintoticamente O(n log n).

4 Por que o Heapsort não é estável

O Heapsort não é um algoritmo de ordenação estável. A razão para isso é que ele realiza trocas diretas entre elementos, o que pode alterar a ordem relativa de chaves com valores iguais. Quando dois elementos têm chaves iguais, a ordem original deles não é garantida após as trocas. Portanto, o Heapsort prioriza a ordenação em relação ao valor absoluto das chaves, mas não preserva a ordem relativa das chaves com o mesmo valor.

5 Conclusão

O Heapsort é um algoritmo eficiente de ordenação com uma complexidade de tempo de O(n log n). Ele se baseia em estruturas de dados de Heap para alcançar essa eficiência. No entanto, ele não é um algoritmo de ordenação estável, o que significa que a ordem relativa de chaves iguais pode não ser preservada. Portanto, a escolha de usá-lo ou não depende dos requisitos específicos do problema a ser resolvido.