MATERIAL ORDENAÇÃO EM VETORES (LINGUAGEM C)

INTRODUÇÃO

Métodos de ordenação são algoritmos utilizados para organizar um conjunto de elementos em uma ordem específica. Em programação, isso é útil para trabalhar com dados de maneira mais eficiente. Existem vários métodos de ordenação, cada um com suas próprias vantagens e desvantagens. Nesta apostila, vamos explorar alguns dos principais métodos de ordenação em vetores, utilizando a linguagem de programação C.

Os métodos de ordenação podem ser divididos em duas categorias principais: ordenação interna e ordenação externa. A ordenação interna é usada quando todos os dados a serem ordenados podem ser armazenados na memória principal, enquanto a ordenação externa é usada quando os dados não cabem na memória principal e precisam ser armazenados em um dispositivo de armazenamento secundário, como um disco rígido. Os métodos de ordenação interna incluem Bubble Sort, Insertion Sort, Selection Sort, Shell Sort, Merge Sort e Quick Sort, enquanto os métodos de ordenação externa incluem Merge Sort Externo e Polyphase Sort.

Cada método de ordenação tem seu próprio desempenho, com base em fatores como o tamanho do vetor a ser ordenado, a distribuição dos elementos do vetor, o tempo de acesso à memória e o número de operações necessárias. O conhecimento desses métodos permite ao programador escolher o método mais adequado para cada situação, otimizando assim a performance e a eficiência do programa. É importante lembrar que a implementação de um método de ordenação pode ser um desafio em si mesmo, e que a escolha do método certo pode fazer toda a diferença no sucesso do projeto.

Método Bubble Sort

O Bubble Sort é um dos métodos de ordenação mais simples. O algoritmo percorre o vetor várias vezes, comparando pares de elementos adjacentes e trocando-os se estiverem fora de ordem. A cada passagem pelo vetor, o maior elemento é deslocado para o final. O processo é repetido até que não haja mais elementos para serem trocados.

É recomendado para pequenos conjuntos de dados e é frequentemente usado como um algoritmo de aprendizado ou para fins educacionais. É fácil de implementar, mas não é tão eficiente quanto outros métodos de ordenação, especialmente para grandes conjuntos de dados.

O pior caso para o Bubble Sort ocorre quando os elementos do vetor já estão ordenados em ordem reversa. Neste caso, o Bubble Sort tem que comparar e trocar todos os elementos do vetor, o que resulta em um desempenho de $O(n^2)$. Por outro lado, o melhor caso ocorre quando o vetor já está ordenado, o que significa que o Bubble Sort precisa apenas de uma passagem para verificar que o vetor está ordenado. Isso resulta em um desempenho de $O(n)$.

Exemplo de código:

```
void bubbleSort(int vetor[], int tamanho) {
    int i, j, aux;
    for (i = 0; i < tamanho - 1; i++) {
        for (j = 0; j < tamanho - i - 1; j++) {
            if (vetor[j] > vetor[j+1]) {
                aux = vetor[j];
                vetor[j] = vetor[j+1];
                vetor[j+1] = aux;
            }
        }
    }
}
```

Método Insertion Sort

O Insertion Sort é outro método de ordenação simples que percorre o vetor, inserindo cada elemento em sua posição correta em uma sublista ordenada. O processo começa com o segundo elemento do vetor e, em seguida, cada elemento subsequente é comparado com os elementos na sublista até que seja encontrado o local correto para inserção.

O Insertion Sort é outro método de ordenação simples, mas mais eficiente que o Bubble Sort. Ele é recomendado para conjuntos de dados pequenos ou moderadamente grandes, já que tem um desempenho de $O(n^2)$ no pior caso. O Insertion Sort é frequentemente usado em conjunto com outros métodos de ordenação, como o Quick Sort, para otimizar o desempenho em grandes conjuntos de dados.

O pior caso para o Insertion Sort ocorre quando o vetor está ordenado em ordem reversa. Neste caso, o algoritmo tem que comparar e mover cada elemento do vetor para a sua posição correta, o que resulta em um desempenho de $O(n^2)$. O melhor caso ocorre quando o vetor já está ordenado, pois neste caso, o algoritmo só precisa comparar cada elemento com seu antecessor e verificar que o vetor está ordenado. Isso resulta em um desempenho de $O(n)$.

Exemplo de código:

```
void insertionSort(int vetor[], int tamanho) {
    int i, j, valorAtual;
    for (i = 1; i < tamanho; i++) {
        valorAtual = vetor[i];
        j = i - 1;
        while (j >= 0 && vetor[j] > valorAtual) {
            vetor[j+1] = vetor[j];
            j--;
        }
        vetor[j+1] = valorAtual;
    }
}
```

Método Selection Sort

O Selection Sort é outro método simples de ordenação que funciona selecionando o menor elemento do vetor e colocando-o na primeira posição, depois selecionando o próximo menor elemento e colocando-o na segunda posição, e assim por diante. É recomendado para conjuntos de dados pequenos ou moderadamente grandes, já que tem um desempenho de $O(n^2)$ em todos os casos. O Selection Sort é útil para conjuntos de dados pequenos ou quando o espaço em memória é limitado.

O pior caso para o Selection Sort ocorre quando o vetor está ordenado em ordem reversa. Neste caso, o algoritmo tem que comparar cada elemento do vetor com todos os outros elementos restantes para encontrar o menor elemento e trocá-lo de posição, o que resulta em um desempenho de $O(n^2)$. O melhor caso ocorre quando o vetor já está ordenado, pois neste caso, o algoritmo só precisa comparar cada elemento com seu sucessor e verificar que o vetor está ordenado. Isso resulta em um desempenho de $O(n^2)$.

Exemplo de código:

```c
void selectionSort(int vetor[], int tamanho) {
    int i, j, indiceMenor, aux;
    for (i = 0; i < tamanho - 1; i++) {
        indiceMenor = i;
        for (j = i + 1; j < tamanho; j++) {
            if (vetor[j] < vetor[indiceMenor]) {
                indiceMenor = j;
            }
        }
        if (i != indiceMenor) {
            aux = vetor[i];
            vetor[i] = vetor[indiceMenor];
            vetor[indiceMenor] = aux;
        }
    }
}
```

Conclusão

Nesta apostila, apresentamos três algoritmos de ordenação muito utilizados na programação: Bubble Sort, Insertion Sort e Selection Sort. Cada um desses algoritmos tem suas particularidades em relação ao tempo de execução, complexidade e eficiência para diferentes tamanhos de vetores. É importante lembrar que existem muitos outros algoritmos de ordenação, como o Merge Sort e o Quick Sort, que são mais eficientes para vetores grandes. No entanto, é essencial entender os conceitos por trás desses algoritmos mais simples para compreender como os algoritmos mais complexos funcionam. Além disso, muitas vezes é possível adaptar esses algoritmos para necessidades específicas de cada aplicação.

REFERÊNCIAS

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Algoritmos: Teoria e prática. Editora Campus.

Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.

Wirth, N. (1986). Algorithms + data structures = programs. Prentice-Hall, Inc.