Aluno: Bruno Fetzer

Introdução a ordenação

A ordenação é o ato de colocar um conjunto de dados em uma determinada ordem predefinida. A principal vantagem é que a ordenação torna o acesso e a busca por dados muito mais eficientes. Exemplo:

Fora de ordem: 5, 2, 1, 3, 4

Ordenado: 1, 2, 3, 4, 5

Conceitos Fundamentais

Chave de Ordenação

Para ordenar um conjunto de dados é preciso usar como base uma chave específica. Essa chave de ordenação é o "atributo" ou "campo" do item utilizado para a comparação. Acessando esse atributo é que sabemos se um elemento está à frente ou não de outros elementos no conjunto ordenado.

Tipos de Ordenação

É possível usar qualquer tipo de chave, desde que tenha uma regra de ordenação bem definida. Os tipos de ordenação mais comuns são:

- **Numérica:** 1, 2, 3, 4, 5
- **Lexicográfica (ordem alfabética):** Amanda, Antônio, Bruno, Rodrigo.

A ordenação também pode ser classificada pela sua direção:

- **Crescente:** 1, 2, 3, 4, 5 ou Amanda, Antônio, Bruno, Rodrigo.
- **Decrescente:** 5, 4, 3, 2, 1 ou Rodrigo, Bruno, Antônio, Amanda

Classificação dos Algoritmos de Ordenação

Um algoritmo de ordenação tem como objetivo organizar os elementos de uma sequência em uma certa ordem predefinida. Eles podem ser classificados de diversas formas:

- **Ordenação Interna (in-place):** Ocorre quando o conjunto de dados que será ordenado cabe inteiramente na memória principal. Qualquer elemento pode ser imediatamente acessado.
- **Ordenação Externa:** Ocorre quando o conjunto de dados a ser ordenado não cabe na memória principal e está armazenado em memória secundária (ex: um arquivo). Os elementos são acessados sequencialmente ou em grandes blocos.

Estabilidade do Algoritmo

Um algoritmo é considerado **estável** se a ordem relativa dos elementos com chaves iguais não muda durante a ordenação

- **Dados não ordenados com chaves iguais:** 5a, 2, 5b, 3, 4, 1
- **Ordenação Estável:** O valor 5a virá antes do valor 5b no resultado final, preservando a ordem original: 1, 2, 3, 4, 5a, 5b
- **Ordenação Não Estável:** A ordem relativa entre 5a e 5b pode ser trocada: 1, 2, 3, 4, 5b, 5a

Algoritmos de Ordenação

Ordenação por Bolha (Bubble Sort)

A origem do nome "Bubble Sort" vem da ideia de bolhas flutuando em um tanque de água em direção ao topo, até encontrarem seu próprio nível. O algoritmo movimenta uma posição por vez: compara os dois primeiros elementos e, se o primeiro for maior que o segundo, trocam de lugar. Faz a mesma coisa com o próximo par de elementos, até chegar no fim. Nesse primeiro ciclo, o maior elemento é "empurrado" para a última posição. O processo se repete, ignorando os elementos que já estão em seu lugar final, até que a lista fique em ordem crescente.

Este algoritmo não é eficiente, e seu desempenho diminui à medida que o número de elementos no array aumenta.

Ordenação por Seleção (Selection Sort)

O algoritmo Selection Sort, a cada ciclo, procura o menor valor em toda a porção não ordenada do array para ocupar a primeira posição disponível. Na próxima iteração, procura o próximo menor valor e o coloca na segunda posição, e assim sucessivamente, até que o array esteja ordenado.

Assim como o Bubble Sort, também não é um algoritmo eficiente. Ele precisa percorrer todo o array a cada iteração, fazendo com que o tempo de execução seja sempre O(N2). Porém, na maioria dos casos, ele supera o desempenho do Bubble Sort, pois realiza menos trocas.

Ordenação por Inserção (Insertion Sort)

A origem do nome se assemelha ao processo de ordenar cartas de baralho nas mãos: pega-se uma carta de cada vez e a insere em seu devido lugar na mão, mantendo as cartas da mão sempre em ordem. O algoritmo percorre o array e, a cada iteração, verifica se o elemento atual está na posição correta em relação aos seus antecessores. Se os antecessores forem maiores que ele, o elemento se

desloca para a esquerda até encontrar sua posição correta. Na prática, o Insertion Sort é mais eficiente que a maioria dos algoritmos de ordem quadrática e, para conjuntos pequenos de dados, supera até mesmo o Quick Sort.

Vantagens:

- **Estável:** A ordem dos elementos iguais não muda durante a ordenação.
- **Online:** Pode ordenar os dados na medida em que os recebe, sem precisar de todo o conjunto de uma vez.

Algoritmo Merge Sort

Conhecido como ordenação por "intercalação", é um algoritmo recursivo que usa a ideia de "dividir para conquistar". Ele divide os dados em conjuntos cada vez menores para depois ordená-los e combiná-los (intercalá-los) em um array maior e ordenado.

Em relação à eficiência, o Merge Sort possui um gasto extra de espaço de memória, pois cria uma cópia do array para cada chamada recursiva.

Algoritmo Quick Sort

Conhecido como ordenação por "partição", é um algoritmo recursivo que usa a ideia de "dividir para conquistar". Ele se baseia no problema da separação: rearranjar um array de forma que os valores menores que um certo valor, chamado "pivô", fiquem na parte esquerda, enquanto os valores maiores fiquem na parte direita. Esse processo cria duas partições, e o algoritmo é aplicado recursivamente a cada uma delas até que reste apenas um elemento por partição. O ponto principal e a parte mais delicada do Quick Sort é a escolha do pivô e a implementação da função de particionamento, que precisa ser rápida e não deve usar um array auxiliar para ser eficiente.

Algoritmo Heap Sort

É um algoritmo de ordenação que compete em desempenho com o Quick Sort. Ele transforma o array de dados em uma estrutura do tipo heap, que é uma árvore binária completa (com exceção do seu último nível). Essa estrutura permite a recuperação e remoção eficiente do elemento de maior valor do array, construindo o array ordenado de trás para a frente.

Algoritmo Counting Sort

É um algoritmo de ordenação para pequenos valores inteiros dentro de um certo intervalo. Ele consiste em contar o número de vezes que cada valor inteiro aparece no array. A partir dessa contagem, o algoritmo realoca os valores, já ordenados, de volta no array em tempo linear.

Ordenação de um array de struct

Para ordenar uma struct, é preciso escolher um dos seus campos para servir como "chave" de comparação. Se a chave for um número, a comparação dos valores é direta, porém para ordenar por um campo de texto, é preciso usar a função strcmp().

Ordenação externa

Na ordenação externa, o conjunto de dados a ser ordenado não cabe na memória principal (está armazenado em memória secundária), nesse tipo de ordenação os dados são acessados em sequência ou em grandes blocos, não podendo acessar imediatamente cada elemento.

Merge sort externo

O algoritmo parte do principio de que é mais fácil ordenar um conjunto com poucos dados do que um com muitos. O algoritmo divide os dados em conjuntos cada vez menores para depois ordená-los e combiná-los por meio de merge. A diferença está em considerar que os dados estão dentro de um arquivo e que a etapa de intercalação será realizada entre arquivos.

Busca em arrays

A busca é procurar a presença de um valor específico dentro de um conjunt ode dados, como um array. É usada a chave de busca para a comparação e a eficiência da operação depende da estrutura e da ordem dos dados.

Busca sequencial ou linear

Percorre o array do início ao fim, comparando cada elemento com a chave de busca. É aplicável a arrays não ordenados e no pior caso (quando o elemento é o último ou não existe), exige O(N) comparações.

Busca sequencial ordenada

Quando o array está ordenado, a busca pode ser otimizada, pois pode cancelar a busca antes de terminar de percorrer completamente o array, no momento que o valor do array ultrapassar a chave de busca.

Busca binária

Exige que o desenvolvedor especifique qual campo da estrutura será usado como chave de busca para a comparação, a única diferença na implementação é o acesso ao campo correto da struct e o uso de funções adequadas para comparação, como strcmp().