INF 112 – Programação 2

Aula: Ordenação

Ordenação

Ordenação: dado um conjunto de elementos, ordenar tais elementos significa reorganizá-los de forma ascendente ou descendente de acordo com algum critério.

Ordenação

Em geral, deseja-se ordenar números, caracteres, strings e, mais importante, registros contendo vários campos. No caso dos registros, um campo deve ser escolhido como “chave” na ordenação.

Ordenação

Formalmente, dada uma sequência s_1, s_2, .., s_k de chaves, o objetivo é encontrar uma permutação i_1, i_2, … , i_k dos índices 1, 2 … k, de modo que $s_{i1} \leq s_{i2} \leq \ldots \leq s_{ik}$

Ordenação

Em geral, o objetivo da ordenação é facilitar a recuperação destes elementos. Além disso, a ordenação é muitas vezes utilizadas como método auxiliar em vários algoritmos importantes.

Ordenação

Existem vários algoritmos de ordenação.

Um algoritmo pode ser melhor do que outro mas não existe um algoritmo de ordenação que seja “o melhor de todos” em todas as situações: alguns são simples de implementar, mas lentos; outros são rápidos, mas complexos; outros são rápidos, mas usam muita memória, etc...

Assim, o conhecimento sobre os vários algoritmos existentes é importante para um profissional da área de computação.

Principais características dos métodos

Eficiência de CPU: normalmente se mede a eficiência do método com base no número de operações de comparação realizadas

Outra medida muito utilizada é o número de movimentos realizados. Em geral, avalia-se o número de operações de leitura e escrita dos dados

Uso de memória: algoritmos que fazem ordenação "*in-place*" são os que não precisam de espaço extra na memória para serem executados.

Principais características dos métodos

Ordenação estável: alguns métodos de ordenação são estáveis, ou seja, se dois elementos possuem chaves "iguais", a ordem relativa entre eles após a ordenação será mantida.

Isso pode ser útil, por exemplo, ao ordenar "estudantes": podemos ordená-los por ordem alfabética e, então, ordená-los por CRA. Após as duas ordenações, os estudantes estarão ordenados por CRA mas os que tiverem CRAs iguais serão apresentados em ordem alfabética.

Principais características dos métodos

Ordenação interna: quando os elementos estão armazenados na memória principal do computador. A maior parte dos algoritmos é para memória interna.

Ordenação externa: quando os elementos estão na memória secundária (disco, fita , etc.)

Ordenação *direta/indireta*

Algumas vezes, a ordenação é feita de forma “indireta”. Isso é útil, principalmente, quando os dados a serem ordenados são registros muito grandes que possuem chaves de comparação “rápida”.

A ordenação indireta é feita ordenando-se um conjunto de índices/ponteiros para os elementos.

1	2	3	4	5	6	7
G	**P**	**D**	**A**	**V**	**R**	**F**

Vetor original

Ordenação *direta/indireta*

Algumas vezes, a ordenação é feita de forma “indireta”. Isso é útil, principalmente, quando os dados a serem ordenados são registros muito grandes que possuem chaves de comparação “rápida”.

A ordenação indireta é feita ordenando-se um conjunto de índices/ponteiros para os elementos.

1	2	3	4	5	6	7
G	**P**	**D**	**A**	**V**	**R**	**F**

Vetor original

1	2	3	4	5	6	7
A	**D**	**F**	**G**	**P**	**R**	**V**

Vetor ordenado (de forma direta)

Ordenação *direta/indireta*

Algumas vezes, a ordenação é feita de forma “indireta”. Isso é útil, principalmente, quando os dados a serem ordenados são registros muito grandes que possuem chaves de comparação “rápida”.

A ordenação indireta é feita ordenando-se um conjunto de índices/ponteiros para os elementos.

1	2	3	4	5	6	7
G	**P**	**D**	**A**	**V**	**R**	**F**

Vetor original

1	2	3	4	5	6	7
A	**D**	**F**	**G**	**P**	**R**	**V**

Vetor ordenado (de forma direta)

1	2	3	4	5	6	7
4	**3**	**7**	**1**	**2**	**6**	**5**

Vetor ordenado (de forma indireta)

Ordenação direta/indireta

Por simplicidade, vamos supor que temos um arranjo de inteiros e desejamos ordenar tal arranjo de forma direta.

1	2	3	4	5	6	7
4	**5**	**2**	**1**	**7**	**5**	**3**

Vetor original

1	2	3	4	5	6	7
1	**2**	**3**	**4**	**5**	**6**	**7**

Vetor ordenado (de forma direta)

Método da Bolha

Algoritmo da “bolha”: algoritmo de implementação bastante simples.

Ideia: o algoritmo varre o vetor repetidas vezes verificando os pares de elementos adjacentes. Se um par estiver fora de ordem, ele é trocado.

Na primeira varredura, o maior elemento do array “sobe até a última posição”. Na segunda varredura, o segundo maior elemento sobe até a penúltima posição....

Esse processo lembra “bolhas” subindo em um líquido até a superfície.

Método da Bolha

```
void bubbleSort(int *v, int n) {
    for(int i = 0; i < n-1; i++)
        for(int j = 0; j < n-1-i; j++)
            if (v[j] > v[j+1]) {
                swap(v[j],v[j+1]);
                /*int aux = v[j];
                v[j] = v[j+1];
                v[j+1] = aux;*/
        }
}
```

Método da Bolha

Rastreie a ordenação do arranjo: [5,7,4,2,1]

Algumas questões:

- Se o arranjo tiver tamanho *n*, quantas comparações o algoritmo fará?
- Em qual situação o algoritmo se comporta (em termos de tempo) de forma pior?
- O algoritmo *Bubble Sort* funciona? É estável?
- O algoritmo da bolha realiza ordenação "*in-place*"?

Ordenação por Seleção

Princípio de ordenação muito simples!

Encontre o menor elemento e o coloque no início do vetor;

Encontre o segundo menor elemento e o coloque na segunda posição do vetor;

...

Ordenação por Seleção

```
void SelectionSort(int *v, int n) {
    for (int i = 0; i < n-1; i++) {
        // acha a posicao do menor elemento
        // entre as posições (i) e (n-1)
        int posMenor = i;
        for (int j = i+1; j < n; j++)
            if (v[j] < v[posMenor])
                posMenor = j;

        // troca o menor elemento (que está na
        // posicao posMenor) com o elemento (i)
        int aux = v[i];
        v[i] = v[posMenor];
        v[posMenor] = aux;
    }
}
```

Ordenação por Seleção

Rastreie a ordenação do arranjo: [5,7,4,2,1]

Ordenação por Inserção

Imagine que temos um arranjo de tamanho *n*, onde os *k* primeiros elementos já estão ordenados.

A ideia do algoritmo de inserção é aumentar o número de elementos já ordenados inserindo (de 1 em 1) elementos na parte já ordenada do arranjo.

Inicialmente, o conjunto de elementos ordenados é vazio.

Ordenação por Inserção

```
void insertionSort(int *v, int n) {
    for (int i = 1; i < n; i++) {
        // o arranjo entre as posicoes [0,i) já está ordenado
        int elemInserir = v[i];
        int j = i-1;
        while(j >= 0 && v[j] > elemInserir) {
            v[j+1] = v[j];
            j--;
        }
        v[j+1] = elemInserir;
    }
}
```

Ordenação por Inserção

Rastreie a ordenação do arranjo: [5,7,4,2,1]

Ordenação por Inserção

O algoritmo é bastante eficiente para pequenas quantidades de dados.

Dos métodos mais simples, em geral, é o mais eficiente na prática.

É eficiente caso o arranjo já esteja ordenado (ou parcialmente ordenado).

Pode realizar bem mais movimentos de dados do que o método da seleção.

Métodos elementares

Foram vistos vários métodos elementares.

A maioria deles não é eficiente não prática (principalmente para entradas grandes).

Cenas dos próximos capítulos...