Notas de Aula - AED1 – Algoritmos de Ordenação (parte 1)
Prof. Jefferson T. Oliva

A ordenação é uma das operações mais conhecidas dos sistemas de programação e tem como objetivo tornar simples e rápido o acesso e a recuperação de uma determinada informação a partir de um grande conjunto de informações. Na ordenação, cada elemento de uma sequência é posicionado de forma que Xi seja menor que o seu sucessor e assim por diante.

Um exemplo de como a ordenação é importante é a lista telefônica (provavelmente, vcs nunca usaram isso): onde é mais fácil encontrar um contato quando os nomes são mantidos ordenados. Imaginem o quão seria difícil encontrar um número de telefone caso a lista não fossem ordenado.

Terminologia básica:
- Arquivo de tamanho n é uma sequência de n itens (X1, X2, …, Xn)
- O i-ésimo componente do arquivo é chamado de item. Se voltarmos para o exemplo da lista telefônica, poderíamos observar que para cada item, há um sobrenome, nome, número de telefone e endereço
- Uma chave é associada a cada registro. Qual seria a chave para cada item em um registro telefone?
- Ordenação pela chave

Mais sobre terminologia básica:
- Ordenação interna: os dados estão na memória principal
- Ordenação externa: os dados estão em um meio auxiliar
- Ordenação estável: um algoritmo de ordenação é dito estável, se ele preserva a ordem relativa original dos registros com mesmo valor de chave. Lembram do conceito de chave primária e secundária vistas nas aulas anteriores?
- Ordenação pode ocorrer sobre os próprios registros ou sobre uma tabela auxiliar de ponteiros.

Ver figura do slide 4.

Devido à relação entre a ordenação e a busca, surge uma pergunta: o aquivo deve ser classificado ou não? Caso a operação de busca (recuperação de dados) seja realizada com frequência, é recomendada ordenação de dados.

Para medir a eficiência dos algoritmos de ordenação, consideramos o tempo para a execução do método e espaço de memória.

O tempo gasto pelo algoritmo de ordenação é comumente medido pelo número de operações críticas, como a comparação de chaves, em que é verificado se um registro é maior ou menor que o outro.

Outra operação crítica considerada na avaliação de algoritmos de ordenação é o movimento de registros, em que, caso a comparação de chaves seja positiva, é realizada a troca de posição entre os elementos comparados.

Em outras palavras, o desempenho de um algoritmo de ordenação é medido geralmente em função do tamanho do conjunto no pior caso, ou seja é considerada a notação O.

Por mais que nessa e nas próximas aulas serão apresentados diferentes tipos de algoritmos de ordenação, deve ser enfatizado que não existe um algoritmo que seja melhor que todos os outros, pois isso varia de acordo com diversos fatores, como o tamanho do conjunto de dados, pré-ordenação, repetição de chaves, entre outros.

Existem vários tipos de métodos de ordenação (exemplo: troca, seleção, inserção), mas não existe um método de ordenação considerado superior aos outros.

Nessa aula serão abordados apenas algoritmos de ordenação interna por troca e por seleção.

Método da bolha (*bubble sort*)

Ordenação por troca: em cada comparação pode haver troca de posições entre itens comparados. Caso os itens comparados não estejam posicionados na ordem correta, ambos são trocados de posição.

O algoritmo percorre o arquivo várias vezes, onde cada elemento é comparado com o seu sucessor. Caso as posições dos elementos estejam no lugar errado, são trocados de posição. Assim, a movimentação dos itens lembra bolhas.

o Bubble sort é um dos algoritmos de ordenação mais simples, pois é de fácil compreensão e implementação, mas é um dos métodos menos ineficientes, pois pode fazer até n^2 trocas. Se para 10 registros, podem ocorrer na ordem de 100 trocas, imagine fazer ordenação em de um arquivo grande, com aproximadamente 1 milhão de registros.

Implementação

```
void bubblesort(int v[], int n){
        int i, j, x;

        for (i = 0; i < n - 1; i++)
                for (j = 0; j < n - i - 1; j++)
                        if (v[j] > v[j + 1]){
                                x = v[j];
                                v[j] = v[j + 1];
                                v[j + 1] = x;
                        }
}
```

Implementação com melhoria:

```
void bubblesort(int v[], int n){
        int i, j, x, troca = 1;

        for (i = 0; i < n - 1; i++){
                troca = 0;

                for (j = 0; j < n - i - 1; j++)
                        if (v[j] > v[j + 1]){
                                x = v[j];
                                v[j] = v[j + 1];
                                v[j + 1] = x;
                                troca = 1;
                        }
        }
}
```

Exemplo de aplicação encontra-se nos slides 11-13.

Ordenação por seleção (*selection sort*)

Ideia básica
1 - Selecionar o maior elemento do conjunto
2 - Trocá-lo com o último elemento
3 - Repetir os dois passos anteriores com os n − 1 elementos restantes, após com os n − 2 e assim por diante até sobrar o primeiro elemento que será o menor do conjunto

Implementação

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

Implementação 2:

```
void selectsort(int v[], int n){
        int i, j, p, aux;
        for (i = 0; i  < n - 1;  i--){
                p = i;
                for (j = 0; j < i; j++)
                        p = j;
                if (p != 0){
                        aux = v[i];
                        v[i] = v[p];
                        v[p] = aux;
                }
        }
}
```

Ver exemplo no slide 19.

Complexidade do método: O(n^2).

Esse método de ordenação é estável.

Simples implementação.

Um dos algoritmos mais rápidos para a ordenação de vetores pequenos.

Em vetores grandes, esse algoritmo é um dos mais lentos.

Ordenação por inserção (*insertion sort*)

É o método que consiste em inserir informações num conjunto já ordenado.

Algoritmo utilizado pelo jogador de cartas
- As cartas são mantidas ordenadas nas mãos dos jogadores
- Quando o jogador compra ou recebe uma nova carta, ele procura a posição que a nova carta deverá ocupar

O algoritmo é um dos mais eficientes para pequenas entradas.

Implementação

```
void insertsort(int v[], int n){
        int i , x;

        for (i = 1; i < n; i++){
                x = v[i];

                for (j = i - 1; (j >= 0) && (x  < v[j]); j--)
                        v[j + 1] = v[j];

                v[j + 1] = x;
        }
}
```

Ver slides 25 e 26

Desempenho do método
- Melhor caso: O(n)
- Médio caso: O(n^2)
- Pior caso: O(n^2)

Deve ser utilizado quando o arquivo está "quase" ordenado.
Desvantagem: alto custo de movimentação de elementos.

Variações do Método:
=> inserção com pesquisa binária: consiste em utilizar o método da busca binária para localizar a posição a ser inserido o elemento:
→ Diminui o número de comparações mas ainda é necessário efetuar o deslocamento dos elementos para a inserção.
→ Isso sendo executado n vezes resulta em O(n^2) substituições.
→ De modo geral não ajuda!
=> Inserção em lista ligada: consiste em não mover as informações e sim efetuar as inserções nas ligações. Portanto, o tempo gasto com comparações continua sendo O(n^2),
=> A melhor variação é a inserção com incrementos decrescentes, também chamado de ordenação de Shell (shell sort)

Considerações Finais

Algoritmo	Melhor caso		
	Comparações	Trocas	Complexidade
Bubble sort	n^2	0	O(n^2)
Bubble sort com melhoria	n	0	O(n)
Selection sort	n^2	0	O(n^2)
Insertion sort	n	0	O(n)

Algoritmo	Caso médio e pior caso		
	Comparações	Trocas	Complexidade
Bubble sort	n^2	n^2	O(n^2)
Bubble sort com melhoria	n^2	n^2	O(n^2)
Selection sort	n^2	n	O(n^2)
Insertion sort	n^2	n^2	O(n^2)

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. Third edition, The MIT Press, 2009.

Horowitz, E., Sahni, S. Rajasekaran, S. Computer Algorithms. Computer Science Press, 1998.

Rosa, J. L. G. Métodos de Busca. SCE-181 - Introdução à Ciência da Computação II. Slides. Ciência de Computação. ICMC/USP, 2018.

Ziviani, N. Projeto de Algoritmos - com implementações em Java e C++. Thomson, 2007.