João Arthur Brunet
Computação @ UFCG

Ordenação por Comparação: Insertion Sort

Direto ao ponto

O *Insertion Sort* tem como rotina base a inserção ordenada. A ideia é executar várias vezes essa rotina para ordenar um array. Para ser exato, se executarmos $N - 1$ vezes a rotina de inserção ordenada em um array o resultado é a ordenação completa do mesmo. Por isso, vamos antes entender como funciona inserção ordenada.

Inserção ordenada

Vamos analisar o caso de um array com N elementos no qual os $N - 1$ primeiros elementos estão ordenados, mas o último elemento não está no seu lugar. Isto é, precisamos encaixar o último elemento de forma que a sequência fique ordenada. No exemplo abaixo, estamos falando em inserir de forma ordenada o valor 12.

values =[9, 13, 16, 21, 32, 12]

Como a sequência está ordenada até o penúltimo índice, a ideia é comparar 12 com o valor anterior e, se 12 for menor, trocar esses valores. Essas comparações e trocas só devem parar quando 12 for maior que o elemento à esquerda ou quando 12 estiver na primeira posição do array. Para visualizar o Insertion Sort, alguns autores utilizam a metáfora de uma mão de cartas. Nesse contexto, o objetivo seria encaixar a carta 12 na mão já ordenada. Veja o passo a passo:

values = [9, 13, 16, 21, 12, 32]

values = [9, 13, 16, 12, 21, 32]

values = [9, 13, 12, 16, 21, 32]

values = [9, 12, 13, 16, 21, 32]

O código que implementa essa rotina está descrito abaixo. O índice `j` assume o valor inicial `values.length - 1` (última posição) e a condição de parada do laço é satisfeita quando esse índice alcançar 0 ou quando o valor que queremos inserir de forma ordenada já está na sua posição (`values[j] >= values[j-1]`). Se `j` alcançar 0, todo o array foi avaliado e o algoritmo deve parar. Da mesma forma, se `values[j] >= values[j-1]` o algoritmo deve parar porque o elemento que estamos querendo encaixar já está em seu lugar.

```
...
    int j = values.length - 1;

    while (j > 0 && values[j] < values[j-1]) {
            int aux = values[j];
            values[j] = values[j-1];
            values[j-1] = aux;
            j -= 1;
    }
...
```

Insertion Sort

A parte complexa desse algoritmo de ordenação nós já entendemos – a inserção ordenada.

> O Insertion Sort aplica várias vezes a inserção ordenada para ordenar uma sequência.

Vamos ver como isso é feito.

Queremos ordenar $values = [7, 1, 2, 3, 9, 5, 1]$. Se pensarmos bem, podemos ver os dois primeiros elementos desse array como sendo o cenário apresentado na seção anterior, isto é, temos que $[7, 1]$ está ordenado, exceto pela última posição. Se aplicarmos inserção ordenada em $[7, 1]$, temos como resultado $[1, 7]$.

Agora, vamos adotar a mesma postura com os três primeiros elementos: $[1, 7, 2]$. Novamente, podemos ver os 3 primeiros elementos como sendo o cenário para a inserção ordenada. Isto é $[1, 7, 2]$ está ordenado, exceto pelo último elemento. Se aplicarmos inserção ordenada em $[1, 7, 2]$, temos como resultado $[1, 2, 7]$.

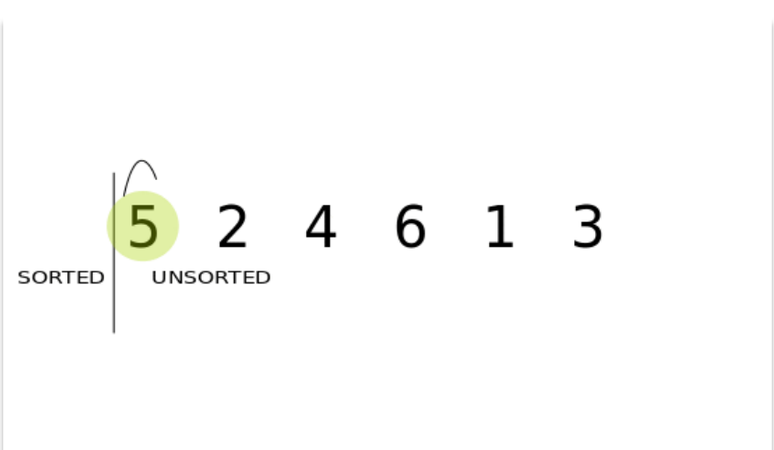
Depois, vamos adotar a mesma postura com os quatro primeiros elementos: $[1, 2, 7, 3]$. Isto é, está ordenado, exceto pelo último elemento. Então basta aplicarmos inserção ordenada de 3. O resultado é $[1, 2, 3, 7]$.

Esse processo segue até o array ficar ordenado. Você percebeu que aplicamos inserção ordenada partindo do segundo elemento do array até o final? Isso significa que basta colocarmos um ***loop*** externo ao código de inserção ordenada, fazendo `j` variar de 1 até o último elemento. Vamos ao código:

```
...
public static void insertionSort(int[] values) {
    for (int i = 1; i < values.length; i++) {

        int j = i;

        while (j > 0 && values[j] < values[j-1]) {
            int aux = values[j];
            values[j] = values[j - 1];
            values[j - 1] = aux;
            j -= 1;
        }

    }
}
...
```

A única mudança que fizemos foi adicionar o comando ***for*** com `i` variando de 1 até o final e `j` variando de acordo com `i`.

Para fixar bem, veja a animação abaixo copiada deste material. Note que a ideia é sempre inserir um elemento em uma sequência já ordenada.



Propriedades e Análise de eficiência

> O Insertion Sort é estável, in-place e $O(n^2)$.

Estabilidade é uma propriedade relacionada à ordem relativa de valores iguais no array original. Por exemplo, se houver dois valores 97 no array antes da ordenação, após a execução do algoritmo, esses dois valores devem seguir a ordem relativa inicial. Ou seja, ao término da execução do algoritmo, a primeira ocorrência do 97 deve vir antes da segunda ocorrência do 97.

O Insertion Sort é estável porque mantém a ordem relativa dos valores iguais. Isso ocorre porque as trocas são feitas sempre com vizinhos. Os valores vão sendo "afastados" um a um, e não dando saltos. Por isso, um elemento qualquer nunca trocará de posição com elementos de mesmo valor.

O Insertion Sort é *in-place* porque a ordenação é feita rearranjando os elementos no próprio array, ao invés de usar arrays ou outras estruturas auxiliares.

O pior caso do Insertion Sort é um array ordenado em ordem reversa, pois toda tentativa de inserção ordenada deve percorrer o array todo à esquerda trocando os elementos até encaixar o atual na primeira posição. Veja o exemplo:

Inserção ordenada de 20:

values = [90, 20, 16, 5, 1]

values = [20, 90, 16, 5, 1]

Inserção ordenada de 16:

values = [20, 90, 16, 5, 1]

values = [20, 16, 90, 5, 1]

values = [16, 20, 90, 5, 1]

Inserção ordenada de 5:

values = [16, 20, 90, 5, 1]

values = [16, 20, 5, 90, 1]

values = [16, 5, 20, 90, 1]

values = [5, 15, 20, 90, 1]

Inserção ordenada de 1:

values = [5, 15, 20, 90, 1]

values = [5, 15, 20, 1, 90]

values = [5, 15, 1, 20, 90]

values = [5, 1, 15, 20, 90]

values = [1, 5, 15, 20, 90]

Feito. Array ordenado.

Note que o tempo de execução é dado pela soma dos passos de cada iteração. Essa soma pode ser representada por $1 + 2 + 3 + \ldots (n - 1)$, ou seja, uma Progressão Aritmética Finita (PA) com $a_1 = 1$ e $a_n = (n - 1)$ e razão $r = 1$. A soma dos termos de uma PA é dada por: $(a_1 + a_n) * n/2$. Então, temos que o tempo de execução do algoritmo é dado por $(1 + (n - 1)) * n/2 = (n^2)/2$. Aplicando as diretrizes de simplificação, o Insertion Sort é $\Theta(n^2)$.

No melhor caso, este algoritmo é $O(n)$. Isto ocorre quando o array já está ordenado. Deste maneira, a inserção ordenada de cada elemento tem custo $O(1)$, pois todos já estão em suas devidas posições. Como a inserção ordenada é executada n vezes, o custo total é $O(n)$

É importante destacar que o Insertion Sort não é considerado um algoritmo eficiente para grandes entradas. Há alternativas $O(n * \log n)$, como Quick Sort e Merge Sort, além de alternativas lineares como o Counting Sort.

É também importante traçar o paralelo entre o Selection Sort e o Insertion Sort. O Selection efetua menos trocas do que o Insertion, pois há uma troca apenas por iteração, ou seja, no total o Selection Sort efetua n trocas. Já o insertion sort efetua ao menos uma troca por iteração, pois deve efetuar trocas para afastar cada elemento avaliado.

Por outro lado, o Insertion Sort efetua menos comparações do que o Selection Sort, pois nem sempre o elemento a ser inserido de forma ordenada deve ir até o final. Na verdade, isso só acontece no pior dos casos, em que o array está ordenado em ordem reversa. Já o Selection Sort precisa comparar todos os elementos restante cada vez para determinar quem é o menor deles.

Na teoria, ambos estão na mesma classe de complexidade, qual seja $O(n^2)$. Na prática, o Insertion Sort apresenta melhor desempenho do que o Selection

Sort.

Resumo

- O Insertion Sort nada mais é do que a execução do algoritmo de inserção ordenada repetidas vezes.
- O Insertion Sort é in-place, estável e $O(n^2)$.
- O pior caso da execução deste algoritmo manifesta-se quando a entrada está ordenada em ordem decrescente.
    - No melhor caso o Insertion Sort é $O(n)$. Isso ocorre quando o array já está ordenado.
- Na teoria, Insertion Sort, Selection Sort e Bubble Sort estão na mesma classe de complexidade, qual seja $O(n^2)$. Na prática, o Insertion Sort apresenta o melhor desempenho entre esses 3 algoritmos.

Notas

Vale a pena utilizar o VisuAlgo para visualizar a execução do Selection Sort e de outros algoritmos de ordenação.
