Fatec - Registro

E. Ricardo, P. Lucas, D. Bruno, E. Bruno, M. Leonardo, S. Renan, O. Filipe, R
. Mauricio, A. Marcelo, M. Matheus, K. Sthevens.

Relatório Insertion Sort e Merge Sort

Implementação dos Algoritmos Insertion Sort e Merge Sort

Registro - SP

2025

Introdução

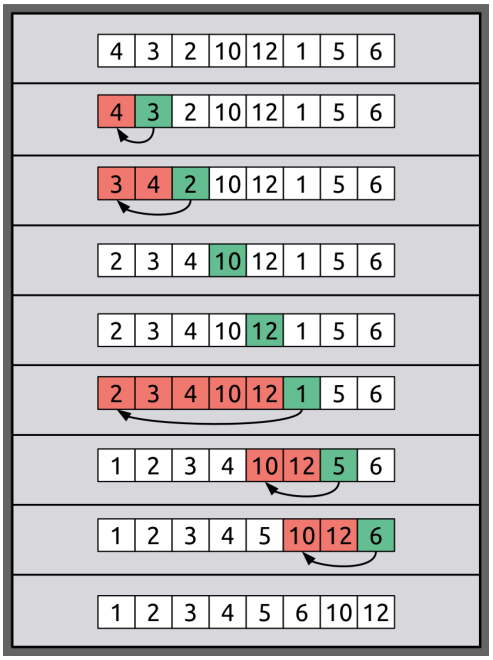
Os algoritmos de ordenação são fundamentais na computação, pois organizam os elementos de uma coleção (como vetores ou listas) em uma ordem específica, seja ela crescente ou decrescente, tal organização é essencial para otimizar a busca e a manipulação de dados em diversas aplicações. Existem várias técnicas de ordenação, cada uma com características próprias em termos de complexidade de tempo, uso de memória e estabilidade. Entre os algoritmos mais conhecidos estão o Bubble Sort, Insertion Sort, Merge Sort e Quick Sort. Nesse relatório será explicado e exemplificado os algoritmos de Insertion Sort e Merge Sort.

Resumo

É notório que ambos os métodos de ordenação, Insertion Sort e Merge Sort, têm seu espaço no mercado de programação. O Insertion Sort, por exemplo, apresenta grande facilidade de implementação e compreensão, sendo, portanto, amplamente utilizado como método de ensino para os demais algoritmos de ordenação. Por outro lado, o Merge Sort, por ser um algoritmo mais enxuto, é capaz de operar com eficiência em sistemas com restrições de tempo, devido à sua execução que sempre ocorre em um tempo determinado para “n” elementos.

Insertion Sort

O algoritmo de ordenação por inserção (Insertion Sort) funciona percorrendo uma sequência passo a passo, assim selecionando um elemento e o comparando com os anteriores, após isso o inserindo na posição correta em relação aos que já foram organizados. Esse processo se repete até que todos os elementos estejam ordenados. Veja um exemplo a seguir:



Artigo: akira - ciência da computação

Nesta imagem, observa-se o funcionamento do algoritmo de ordenação por inserção (Insertion Sort) passo a passo:

1. Na primeira divisão desta imagem observa-se uma sequência completa, porém fora de ordem, no qual o algoritmo irá trabalhar.

2. Na segunda divisão, o elemento “3” é selecionado. Ele é comparado com o valor à sua esquerda, o número “4”. Como “3” é menor, ele é deslocado para a posição anterior, ocupando o lugar de “4” na sequência.

3. Em seguida, o algoritmo seleciona o número “2”, que é comparado com os elementos já ordenados à sua esquerda. Como “2” é menor que todos eles, ele é movido para o início da sequência, assumindo a primeira posição.

4. Nas próximas etapas, os elementos “10” e “12” são avaliados, mas como já estão em posição correta (maiores que os anteriores), não necessitam de movimentação.

5. A seguir, o número “1” é selecionado. Ele é comparado com todos os elementos à sua esquerda, por ser o menor valor da sequência, é deslocado até a primeira posição, reorganizando todos os outros elementos que são maiores que ele.

6. Por fim, os elementos “5” e “6” são processados. Como estão fora de ordem em relação aos anteriores, o algoritmo realiza as comparações necessárias e os posiciona corretamente, completando a ordenação em ordem crescente.

Com base em seu funcionamento, o algoritmo de ordenação Insertion Sort é amplamente utilizado em ambientes educacionais e de aprendizagem, principalmente devido à sua simplicidade de implementação e fácil compreensão. Essas características tornam este algoritmo uma ferramenta essencial para o ensino introdutório de estruturas de ordenação na programação. Apesar de não ser o mais eficiente para grandes volumes de dados, seu valor pedagógico é significativo.

Logo em seguida observa-se um exemplo de sua implementação prática na linguagem C, com todos seus conceitos de ordenação em funcionamento:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int v[5] = {5, 9, 2, 6, 1}, n = 5;
    int i = 0;
    int j = 1;
    int aux = 0;

    while(j < n){
        aux = v[j];
        i = j - 1;

        while((i >= 0) && (v[i] > aux)){
            v[i + 1] = v[i];
            i = i - 1;
        }

        v[i + 1] = aux;
        j = j + 1;

    }

    for(i = 0; i < n; i++){
        printf("%d", v[i]);
    }

    return 0;
}
```

Exemplo ordenação - Code Blocks.

No início do código, é realizada a declaração de um vetor denominado “v”, contendo cinco elementos não ordenados. Também é definida a variável “n” igual a 5, representando o tamanho do vetor. Em seguida, são declaradas as variáveis auxiliares “i” e “j”, utilizadas para percorrer os elementos do vetor, além da variável “aux”, que tem a finalidade de armazenar temporariamente os valores durante o processo de ordenação.

Na sequência, observa-se a utilização do primeiro laço de repetição do tipo “while”, cuja função é selecionar o segundo elemento do vetor e posicioná-lo adequadamente em relação ao primeiro elemento. O valor é atribuído à variável “aux” e a variável “i” recebe o valor de “j” menos um, iniciando o processo de comparação com os elementos anteriores.

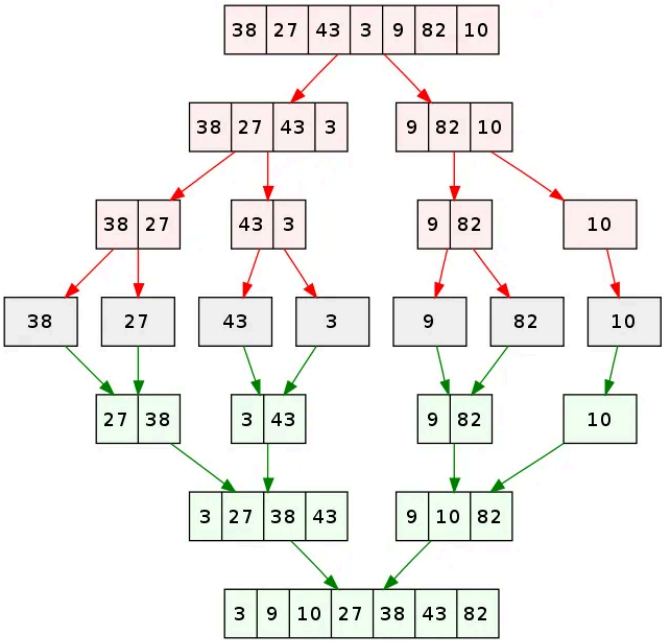
Em seguida inicia-se o segundo laço de repetição “while”, responsável pela execução do processo de ordenação. Esse laço garante que não sejam acessadas posições inválidas no vetor, por meio da verificação de “i” maior ou igual a zero, e compara os elementos “v” na posição “i” com o valor armazenado em “aux”. Caso o elemento seja maior, ocorre seu deslocamento para a posição seguinte, enquanto “i” é movimentado para que novas comparações possam ser realizadas com os elementos à esquerda.

Após o término desse processo, o valor armazenado em “aux” é inserido na posição correta dentro da sequência, garantindo que a parte inicial do vetor esteja ordenada até o índice atual. Com isso, é implementado um laço de repetição do tipo “for”, responsável por exibir os elementos do vetor já ordenados. O programa é encerrado com a instrução “return zero”, indicando sua execução bem-sucedida, sem falhas.

Merge Sort

O método do Merge Sort segue o conceito de divisão e conquista, sua ordenação ocorre através da divisão do vetor em vetores menores, sendo assim, com a ajuda da recursividade o algoritmo inicia um processo de combinação entre os vetores menores, no qual compara e ordena os seus elementos, como mostrado na imagem a seguir:

Imagem 2:



Fonte: https://guilherme-rmendes95.medium.com/algoritmos-merge-sort-ef12dadeba2a

Na primeira parte da imagem, como se trata de um vetor de tamanho ímpar, é decidido se a maior parte será separada para a esquerda ou para a direita; no exemplo da imagem, a maior parte foi separada para a esquerda. Em seguida, observa-se a divisão dos dois vetores nomeados por LEFT e RIGHT: na divisão à esquerda, o vetor foi dividido em dois pares; e na divisão à direita, como o número de elementos é ímpar, formou-se um par e um elemento ficou isolado (no caso, o [10]).

Depois, todos os vetores se dividem novamente até que se chegue à unidade. Em seguida, inicia-se a etapa de conquista, onde os números se comparam entre si e formam pares ordenados, como mostrado na imagem 2; assim, temos três pares de números já ordenados do menor para o maior. Após isso, os pares se comparam e começam a se ordenar, tanto os da direita quanto os da esquerda, até que restam apenas dois vetores: o da direita, contendo quatro índices já ordenados, e o da esquerda, contendo três índices já ordenados, chegando à última etapa, que é a união dos dois vetores e sua respectiva ordenação.

O Merge Sort apresenta um desempenho notável em aplicações com restrição de tempo, pois sua execução ocorre sempre em um tempo determinado para n elementos (Reinaldo, Universidade Federal de Ouro Preto, s.d.).

A implementação do código do Merge Sort na linguagem C exemplificado nas imagens a seguir:

```c
#include <stdio.h>
#include <stdlib.h>

void merge(int * v, int left, int middle, int right){

    int helper[5];
    for (int i = left; i <= right; i++){
        helper[i] = v[i];
    }

    int i = left;
    int j = middle + 1;
    int k = left;

    while(i <= middle && j <= right){
        if (helper[i] <= helper[j]){
            v[k] = helper[i];
            i++;
        }else{
            v[k] = helper[j];
            j++;
        }
        k++;

    }

    while(i <= middle){
        v[k] = helper[i];
        i++;
        k++;
    }

     while (j <= right) {
        v[k] = helper[j];
        j++;
        k++;
    }

}
```

Exemplo ordenação - Code Blocks.

```c
void mergeSort(int * v, int left, int right) {

  if(left >= right)
      return;

      else {

      int middle = (left + right) / 2;
      mergeSort(v, left, middle);

      mergeSort(v, middle + 1, right);

      merge(v, left, middle, right);


  }
}

  int main(){
      int v[5] = {2,6,8,4,1};
      int n = 5;

      mergeSort(v, 0, n - 1);


  for(int i = 0; i < n; i++){
          printf("%d", v[i]);

      }


   return 0;

}
```

Exemplo ordenação - Code Blocks.

No início do código temos a declaração de uma função contendo, o vetor (v) e os índices finais, iniciais e ponto de divisão do vetor (left, right, middle). Após isso temos declarado a variável “helper” do tipo inteiro possuindo 5 índices; em seguida é criado uma estrutura de repetição no qual é definido a variável “i = left” e que se “i” for menor ou igual a “right” o “i” vai receber um incremento, então temos que sob essas condições que o “helper[i]” recebe o “v” na posição “i”; em sequência é definida três variáveis inteiras, sendo: i = left; j = middle + 1; k = left;

Em seguida temos uma estrutura de repetição do laço “while” que diz: enquanto o “i” for menor ou igual a “middle” && “j” menor ou igual a “right”, então… temos que se “helper” no índice “i” for menor ou igual a “helper” no índice “j”, então… temos que o “i” recebe um incremento, porém se essa condição não for verdadeira ocorre que o “v” no índice “k” recebe a variável helper no índice “j”, logo “j” recebe um incremento. E temos que “k” sempre que rodar dentro do laço “for” receberá o incremento inevitavelmente.

Após esse laço é criado um novo laço while que diz: enquanto o “i” for menor ou igual a o middle obtemos que v no índice k recebe o helper no índice “i”, e tanto o “i” quanto o “k” recebem um incremento descrito como “i++” e “k++”.

Em sequência temos outro laço que diz: enquanto “j” for menor ou igual a right tem se que “v” no índice k receber helper no índice “j” e tanto “j” como “k” recebem um incremento.

Logo após, é declarada a função denominada “mergeSort”, cuja responsabilidade é realizar a divisão recursiva do vetor a ser ordenado. Inicialmente, observa-se a presença de uma estrutura condicional do tipo “if”, que verifica se o vetor está em sua condição base, ou seja, se possui apenas um elemento ou nenhum. Caso essa condição seja verdadeira, entende-se que o vetor já está ordenado, encerrando-se, assim, a chamada recursiva.

Na estrutura condicional “else”, presente logo abaixo, o vetor é dividido em duas partes a partir do cálculo do ponto médio. Cada uma dessas partes é então submetida, recursivamente, à mesma função de ordenação, permitindo que os subvetores sejam organizados individualmente antes da junção final.

Após a implementação da função de ordenação, é definida a função principal “main”, responsável por declarar o vetor com os elementos a serem ordenados, bem como a variável que representa seu tamanho. Em seguida, utiliza-se um laço de repetição do tipo “for” para percorrer o vetor e exibir os elementos já ordenados na tela. O código é finalizado com a instrução return zero, indicando o encerramento bem-sucedido do programa.

Conclusões

Com base no que foi abordado nos tópicos anteriores, os algoritmos de ordenação são ferramentas essenciais para a manipulação e otimização de dados em sistemas computacionais. O relatório demonstrou, por meio da análise do Insertion Sort e do Merge Sort, como diferentes abordagens podem ser aplicadas para ordenar coleções de elementos, tanto em ordem decrescente quanto em ordem crescente, conforme a necessidade de cada caso.

Por um lado, o Insertion Sort se destaca por sua simplicidade e facilidade de implementação, sendo um excelente recurso didático, por outro, o Merge Sort evidencia sua eficiência em ambientes com restrição de tempo, devido à estratégia de divisão e conquista, que garante uma execução em tempo determinado para “n” elementos.

Referências

Merge Sort: Algoritmo, Exemplo, Complexidade, Código:

https://www-wscubetech-com.translate.goog/resources/dsa/merge-sort?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt&_x_tr_pto=tc&_x_tr_hist=true

Lista Encadeada:
https://www.wscubetech.com/resources/dsa/linked-list-data-structure

Ordenação: MergeSort :

http://www.decom.ufop.br/reinaldo/site_media/uploads/2013-02-bcc202/aula_13_-_mergesort_(v2).pdf

Reinaldo, F. (s.d.). Desempenho do Merge Sort em aplicações com restrição de tempo. Universidade Federal de Ouro Preto.

Algoritmo Insertion Sort:

https://pt.khanacademy.org/computing/computer-science/algorithms/insertion-sort/a/insertion-sort

Algoritmos de Ordenação: Insertion Sort:

https://henriquebraga92.medium.com/algoritmos-de-ordenação-iii-insertion-sort-bfade66c6bf1