ALGORITMOS DE ORDENAÇÃO

ANA LUÍSA PEREIRA DOS SANTOS

Estudante de Engenharia de Software
Universidade Federal de Goiás
Trabalho referente a disciplina de Introdução a Programação

TÓPICOS

- Algoritmos de Ordenação;
- Ordenação por Inserção;
- Etapas da Ordenação por Inserção;
- Códigos com Ordenação por Inserção;
- Ordenação por Seleção;
- Etapas da Ordenação por Seleção;
- Códigos com Ordenação por Seleção.

ALGORITMOS DE ORDENAÇÃO

Algoritmo de ordenação é um método ou procedimento para reorganizar os elementos de uma lista em uma sequência específica, geralmente em ordem *crescente* ou *decrescente*. Os algoritmos de ordenação são fundamentais na computação porque a ordenação pode, muitas vezes, reduzir a complexidade de um problema.

ALGORITMOS DE ORDENAÇÃO

Imagine que você tem uma lista de notas de alunos em um exame, e você precisa classificá-las em ordem crescente para saber quais alunos tiveram as melhores e piores pontuações.

Fazer isso manualmente pode ser um processo muito trabalhoso, principalmente se a lista for grande. Se a lista tiver dezenas, centenas ou milhares de notas, o esforço necessário para comparar e ordenar cada uma das notas se torna inviável e sujeito a erros.

ORDENAÇÃO POR INSERÇÃO (*INSERTION SORT*)

O algoritmo de ordenação por inserção é um método estável que constrói aos poucos uma lista ordenada. Ele começa assumindo que o primeiro elemento da lista já está ordenado e, a partir daí, compara cada elemento subsequente com os elementos anteriores, inserindo-os na posição correta dentro da parte da lista que já está ordenada.

Ele divide a lista em duas partes, uma ordenada e a outra desordenada. Assim, ele vai pegando sempre o primeiro elemento da lista desordenada e colocando em sua posição correta na lista ordenada. Esse processo ocorre até que a lista esteja completamente ordenada.

ETAPAS DA ORDENAÇÃO POR INSERÇÃO

1- Primeiramente, considere a primeira posição da lista como ordenada.

- A primeira posição é tratada como uma lista ordenada com um único elemento.

2- Em seguida, pegue o primeiro elemento da lista não ordenada e coloque-o na posição correta na lista ordenada.

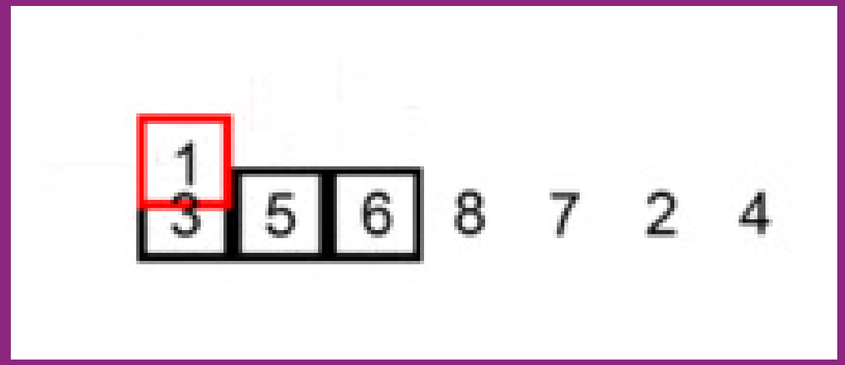
- Compare esse elemento com os elementos da lista ordenada (da direita para a esquerda) e insira-o na posição correta, deslocando os elementos maiores uma posição para a direita.

ETAPAS DA ORDENAÇÃO POR INSERÇÃO

3- Por fim, repita o passo 2 até que todos os elementos da lista não ordenada sejam inseridos na lista ordenada.

- **Continue pegando o próximo elemento da lista não ordenada e inserindo-o na posição correta na lista ordenada até que todos os elementos estejam ordenados.**

Exemplo visual:



CÓDIGOS COM ORDENAÇÃO POR INSERÇÃO

Pseudocódigo (Portugol):

```
função ordenacaoPorInsercao(lista)
    para i de 1 até comprimento(lista) - 1 faça

        chave <- lista[i]
        j <- i - 1

        enquanto j >= 0 e lista[j] > chave faça
            lista[j + 1] <- lista[j]
            j <- j - 1

        lista[j + 1] <- chave
    fim para
fim função

função principal()
    lista <- [5, 2, 9, 1, 5, 6]

    ordenacaoPorInsercao(lista)

    escreva("Lista ordenada: ", lista)
fim função
```

Golang:

```
func insertionSort(arr []int) {
    for i := 1; i < len(arr); i++ {
        key := arr[i]
        j := i - 1

        for j >= 0 && arr[j] > key {
            arr[j+1] = arr[j]
            j = j - 1
        }

        arr[j+1] = key
    }
}

func main() {
    arr := []int{5, 2, 9, 1, 5, 6}

    insertionSort(arr)

    fmt.Println("Array ordenado:", arr)
}
```

ORDENAÇÃO POR SELEÇÃO (*SELECTION SORT*)

O algoritmo de ordenação por seleção funciona inicialmente encontrando o menor valor na lista e trocando-o com o primeiro elemento. Com isso, o primeiro elemento já está em sua posição correta. Em seguida, o algoritmo busca o menor valor na porção restante da lista e o troca com o primeiro elemento dessa porção. Esse processo se repete em cada iteração, reduzindo a parte não ordenada da lista e aumentando a parte ordenada. Isso continua até que todos os elementos tenham sido comparados e a lista esteja completamente ordenada.

ETAPAS DA ORDENAÇÃO POR SELEÇÃO

1- Receber uma lista de entrada:

- O algoritmo começa com uma lista de elementos que precisam ser ordenados.

2- Encontrar o menor elemento da lista e trocá-lo com o primeiro elemento:

- O algoritmo percorre toda a lista para encontrar o menor elemento.
- Uma vez encontrado, este menor elemento é trocado com o primeiro elemento da lista.
- Após esta troca, o primeiro elemento da lista está no seu lugar correto e não precisa mais ser considerado nas próximas iterações.

ETAPAS DA ORDENAÇÃO POR SELEÇÃO

3- Considerar apenas o restante da lista que não está ordenado e ir para o passo 2

- O algoritmo então considera a sub-lista que exclui o primeiro elemento (que já está ordenado).
- Ele repete o processo de encontrar o menor elemento nesta sub-lista e trocá-lo com o primeiro elemento da sub-lista.

4- Repetir o processo para os elementos restantes:

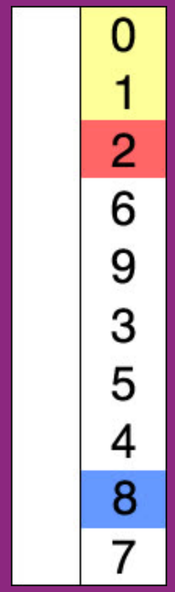
- O algoritmo continua esse processo de redução da sub-lista não ordenada, movendo o menor elemento encontrado para a posição correta no início da sub-lista.
- Em cada iteração, a parte ordenada da lista aumenta em um elemento, e a parte não ordenada diminui.

ETAPAS DA ORDENAÇÃO POR SELEÇÃO

5- Quando o restante da lista estiver vazio, então a lista estará ordenada:

- **O algoritmo termina quando a sub-lista não ordenada está vazia, ou seja, todos os elementos foram ordenados e estão em suas posições corretas.**

Exemplo visual:



CÓDIGOS COM ORDENAÇÃO POR SELEÇÃO

Pseudocódigo (Portugol):

```
funcao ordenacaoPorSelecao(lista: vetor[1..6] de inteiro)
var
   i, j, menorIndice, temp: inteiro
inicio
   para i de 1 ate comprimento(lista) - 1 faca
      menorIndice <- i
      para j de i + 1 ate comprimento(lista) faca
         se lista[j] < lista[menorIndice] entao
            menorIndice <- j
         fimse
      fimpara
      temp <- lista[menorIndice]
      lista[menorIndice] <- lista[i]
      lista[i] <- temp
   fimpara
fimfuncao

funcao principal()
inicio
   lista[1] <- 5
   lista[2] <- 2
   lista[3] <- 9
   lista[4] <- 1
   lista[5] <- 5
   lista[6] <- 6

   ordenacaoPorSelecao(lista)

   escreva("Lista ordenada: ")
   para i de 1 ate comprimento(lista) faca
      escreva(lista[i], " ")
   fimpara
fimfuncao
```

Golang:

```
package main

import "fmt"

func ordenacaoPorSelecao(lista []int) {
    tamanho := len(lista)
    for i := 0; i < tamanho-1; i++ {
        menorIndice := i
        for j := i + 1; j < tamanho; j++ {
            if lista[j] < lista[menorIndice] {
                menorIndice = j
            }
        }
        lista[i], lista[menorIndice] = lista[menorIndice], lista[i]
    }
}

func main() {
    lista := []int{5, 2, 9, 1, 5, 6}

    ordenacaoPorSelecao(lista)

    fmt.Println("Lista ordenada:", lista)
}
```

Ana Luísa Pereira dos Santos

ana_luisa23@discente.ufg.br

Engenharia de Software

Introdução a Programação

REFERÊNCIAS

https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao#google_vignette

https://www.freecodecamp.org/portuguese/news/algoritmos-de-ordenacao-explicados-com-exemplos-em-python-java-e-c/

https://www.estrategiaconcursos.com.br/blog/algoritmo-ordenacao-insercao/

https://www.estrategiaconcursos.com.br/blog/algoritmo-ordenacao-por-selecao/