



Estrutura de Dados

Diego Silveira Costa Nascimento

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte
diego.nascimento@ifrn.edu.br

19 de agosto de 2018

1 Introdução

2 Ordenação

3 Lista



1 Introdução

2 Ordenação

3 Lista



- Consolidar os conhecimentos sobre programação previamente adquiridos;
- Identificar e desenvolver modelos matemáticos, determinando que classes de problemas podem ser resolvidos com o uso deles;
- Criar representações concretas dos objetos e desenvolver rotinas capazes de atuar sobre essas representações, de acordo com o modelo considerado;
- Fornecer domínio de alocação dinâmica de memória;
- Apresentar as principais estruturas de dados: lista, fila, pilha, árvores e tabelas de dispersão;
- Introduzir aspectos básicos da complexidade de algoritmos;
- Apresentar os principais processos de ordenação e pesquisa de dados.



- Proporciona reúso de código;
- Diminui custos de desenvolvimento e manutenção;
- Melhora o desempenho do sistema; e
- Organiza as informações.



Algoritmo

É uma sequência finita e lógica de instruções ou passos, especificados em uma determinada linguagem, que mostram como resolver determinado problema.

Estrutura de Dados

É um modo particular de armazenamento e organização de dados em um computador de modo que possam ser usados de modo eficiente.

Programa

É uma expressão em linguagem formal inteligível por um computador (Algoritmos + Estruturas de dados).



Ementa do Curso

1 Introdução

2 Ordenação

3 Lista



Definição

Uma ordenação consiste em colocar os elementos de um conjunto de dados de forma organizada (ascendente ou descendente) de acordo seus valores.

- Ordenação por inserção (Insert Sort);
- Ordenação por seleção (Select Sort);
- Ordenação por flutuação (Bubble Sort);
- Ordenação por mistura (Merge Sort); e
- Ordenação rápida (Quick Sort).



Ordenação por Inserção

- Eficiente quando aplicado a um pequeno número de elementos;
- Percorre um vetor de elementos da esquerda para a direita;
- À medida que avança vai deixando os elementos mais à esquerda ordenados; e
- Assemelha-se a ordenação de cartas de um jogo de baralho.

Exemplo

```
valores = [5, 8, 9, 2, 1]

for i in range(1, len(valores)):
    aux = valores[i]
    j = i
    while (j > 0) and (aux < valores[j - 1]):
        valores[j] = valores[j - 1]
        j -= 1
    valores[j] = aux

print(valores)
```

Ordenação por Seleção

- Baseado em passar sempre o menor valor do vetor para a primeira posição;
- Depois o de segundo menor valor para a segunda posição; e
- Assim é feito sucessivamente com os $(n - 1)$ elementos restantes.

Exemplo

```
valores = [5, 8, 9, 2, 1]

for i in range(0, len(valores) - 1):
    index_menor = i
    for j in range(i + 1, len(valores)):
        if valores[j] < valores[index_menor]:
            index_menor = j
    if valores[index_menor] < valores[i]:
        valores[i], valores[index_menor] = valores[index_menor], valores[i]

print(valores)
```



Ordenação por Flutuação

- A ideia é percorrer o vector diversas vezes;
- A cada passagem fazendo flutuar para o topo o maior elemento da sequência; e
- Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível.

Exemplo

```
valores = [5, 8, 9, 2, 1]

for i in range(len(valores) - 1, 0, -1):
    for j in range(0, i):
        if (valores[j] > valores[j + 1]):
            valores[j], valores[j + 1] = valores[j + 1], valores[j]

print(valores)
```



Ordenação por Mistura

- Do tipo dividir-para-conquistar;
- Dividir: Dividir os dados em subsequências pequenas; e
- Conquistar: Classificar as metades recursivamente aplicando o merge sort.



Exemplo

```
def merge_sort(lista):
    if len(lista) > 1:
        centro = len(lista) // 2
        sublista_esquerda = lista[:centro]
        sublista_direita = lista[centro:]

        merge_sort(sublista_esquerda)
        merge_sort(sublista_direita)

    i = j = k = 0
    while i < len(sublista_esquerda) and j < len(sublista_direita):
        if sublista_esquerda[i] < sublista_direita[j]:
            lista[k] = sublista_esquerda[i]
            i += 1
        else:
            lista[k] = sublista_direita[j]
            j += 1
        k += 1
    while i < len(sublista_esquerda):
        lista[k] = sublista_esquerda[i]
        i += 1
        k += 1
    while j < len(sublista_direita):
        lista[k] = sublista_direita[j]
        j += 1
        k += 1
```

Exemplo

```
valores = [5, 8, 9, 2, 1]
merge_sort(valores)
print(valores)
```



- Escolha um elemento da lista, denominado pivô;
- Rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele;
- Ao fim do processo o pivô estará em sua posição final e haverá duas sublistas não ordenadas; e
- Recursivamente ordena as sublistas de elementos menor e a maior. sort.



Exemplo

```
def quick_sort(lista, index_inicio=None, index_fim=None):
    if index_inicio == None and index_fim == None:
        index_inicio = 0
        index_fim = len(lista) - 1

    pivo = lista[(index_inicio + index_fim) // 2]
    i = index_inicio
    j = index_fim

    while i < j:
        while lista[i] < pivo:
            i += 1

        while lista[j] > pivo:
            j -= 1

        if i < j:
            lista[i], lista[j] = lista[j], lista[i]
            i += 1
            j -= 1

    if j > index_inicio:
        quick_sort(lista, index_inicio, j)
    if i < index_fim:
        quick_sort(lista, j+1, index_fim)
```


Exemplo

```
valores = [7,1,3,9,8,4,2,7,4,2,3,5]  
quick_sort(valores)  
print(valores)
```



Ementa do Curso

1 Introdução

2 Ordenação

3 Lista



Definição

É uma estrutura de dados que implementa uma coleção de dados ligados (encadeados) de forma dinâmica em um único sentido.



Código

```
class Elemento:
    def __init__(self, nome):
        self.nome = nome
        self.proximo = None
```



Classe Lista Ligada

Código

```
class ListaLigada:  
    def __init__(self):  
        self.inicio = None
```



Código

```
def adicionar(self, nome):  
    novo = Elemento(nome)  
    if self.inicio == None:  
        self.inicio = novo  
    else:  
        elemento = self.inicio  
        while elemento.proximo != None:  
            elemento = elemento.proximo  
        elemento.proximo = novo
```



Código

```
def exibir(self):  
    elemento = self.inicio  
    print(elemento.nome)  
    while elemento.proximo != None:  
        elemento = elemento.proximo  
        print(elemento.nome)
```



Código

```
def remover(self, nome):
    elemento = self.inicio
    if elemento.nome == nome:
        self.inicio = elemento.proximo
    else:
        while elemento.proximo != None:
            if elemento.proximo.nome == nome:
                elemento.proximo = elemento.proximo.proximo
            else:
                elemento = elemento.proximo
```



Código

```
lista = ListaLigada()

lista.adicionar('João')
lista.adicionar('Pedro')
lista.adicionar('Marcos')
lista.adicionar('Lucas')

lista.remover('Pedro')

lista.exibir()
```



Definição

É uma estrutura de dados que implementa uma coleção de dados ligados de forma dinâmica em sentido duplo.



Código

```
class Elemento:
    def __init__(self,nome):
        self.anterior = None
        self.nome = nome
        self.proximo = None
```



Classe Duplamente Ligada

Código

```
class ListaLigada:  
    def __init__(self):  
        self.inicio = None
```



Código

```
def adicionar(self, nome):
    novo = Elemento(nome)
    if self.inicio == None:
        self.inicio = novo
    else:
        elemento = self.inicio
        while elemento.proximo != None:
            elemento = elemento.proximo
        elemento.proximo = novo
        novo.anterior = elemento
```



Código

```
def exibir(self):  
    elemento = self.inicio  
    print(elemento.nome)  
    while elemento.proximo != None:  
        elemento = elemento.proximo  
        print(elemento.nome)  
    while elemento.anterior != None:  
        elemento = elemento.anterior  
        print(elemento.nome)
```



Código

```
def remover(self, nome):
    elemento = self.inicio
    if elemento.nome == nome:
        self.inicio = elemento.proximo
        self.inicio.anterior = None
    else:
        elemento = elemento.proximo
        while elemento != None:
            if elemento.nome == nome:
                if elemento.proximo != None:
                    elemento.anterior.proximo = elemento.proximo
                    elemento.proximo.anterior = elemento.anterior
                else:
                    elemento.anterior.proximo = None
            elemento = elemento.proximo
```



Código

```
lista = ListaDuplamenteLigada()

lista.adicionar('João')
lista.adicionar('Pedro')
lista.adicionar('Marcos')
lista.adicionar('Lucas')

lista.remover('Pedro')

lista.exibir()
```

