



# Estrutura de Dados

Diego Silveira Costa Nascimento

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

1 de novembro de 2018

# Ementa do Curso

- 1 Ordenação
- 2 Lista
- 3 Pilha
- 4 Fila
- 5 Espalhamento
- 6 Árvore
- 7 Complexidade



# Ementa do Curso

- 1 Ordenação
- 2 Lista
- 3 Pilha
- 4 Fila
- 5 Espalhamento
- 6 Árvore
- 7 Complexidade



## 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.



# 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.



# 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.



# 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.





- 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.



# Ementa do Curso

- 1 Ordenação
- 2 Lista**
- 3 Pilha
- 4 Fila
- 5 Espalhamento
- 6 Árvore
- 7 Complexidade



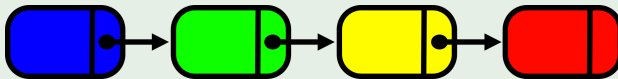
- Lista ligada;
- Lista duplamente ligada; e
- Lista circular.



## Definição

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

## Ilustração

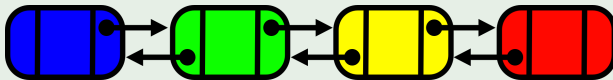


# Lista Duplamente Ligada

## Definição

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

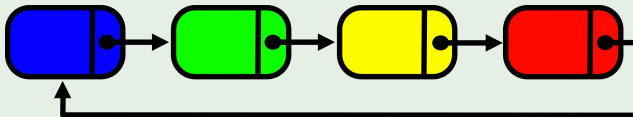
## Ilustração



## Definição

É uma estrutura de dados que implementa uma coleção de dados ligados de forma dinâmica em um único sentido, no qual o final da lista corresponde o início da própria lista.

## Ilustração



# Ementa do Curso

- 1 Ordenação
- 2 Lista
- 3 Pilha**
- 4 Fila
- 5 Espalhamento
- 6 Árvore
- 7 Complexidade

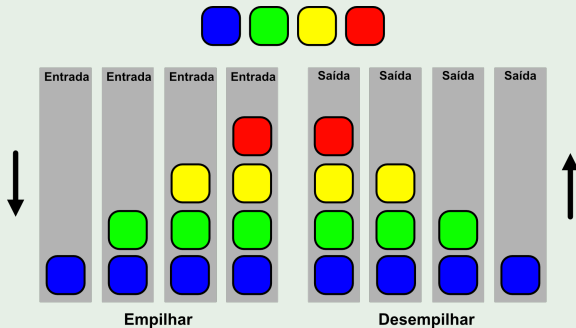


# Pilha

## Definição

É uma estrutura de dados baseada no princípio LIFO (Last In, First Out), na qual os dados que foram inseridos primeiro na pilha serão os últimos a serem removidos.

## Ilustração





# Ementa do Curso

- 1 Ordenação
- 2 Lista
- 3 Pilha
- 4 Fila**
- 5 Espalhamento
- 6 Árvore
- 7 Complexidade



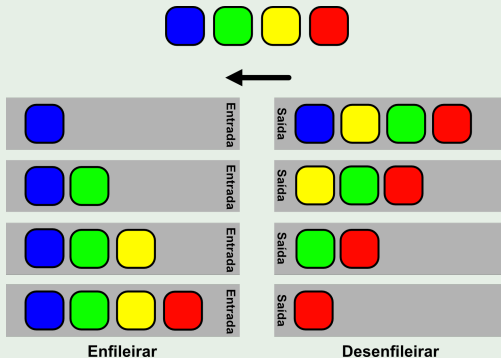
- Fila simples; e
- Fila com prioridade.



## Definição

É uma estrutura de dados baseada no princípio FIFO (First In, First Out), em que os elementos inseridos no início são os primeiros a serem removidos.

## Ilustração

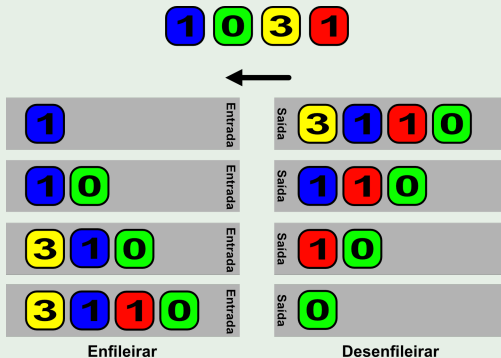


# Fila com Prioridade

## Definição

É uma estrutura de dados em que os elementos são inseridos em ordem de prioridade.

## Ilustração



# Ementa do Curso

- 1 Ordenação
- 2 Lista
- 3 Pilha
- 4 Fila
- 5 Espalhamento**
- 6 Árvore
- 7 Complexidade

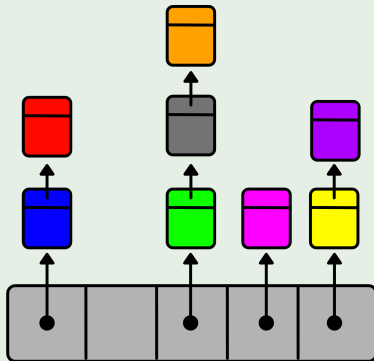


# Tabela de Espalhamento

## Definição

É uma estrutura de dados especial, que associa chaves de pesquisa a valores.

## Ilustração



# Ementa do Curso

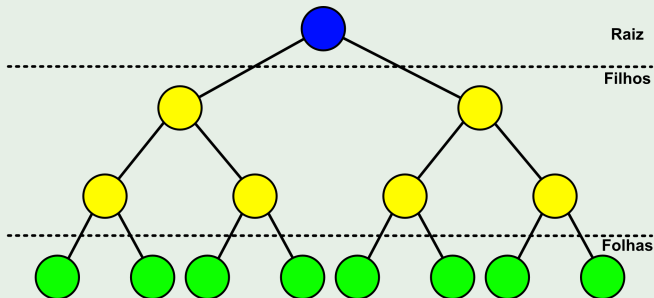
- 1 Ordenação
- 2 Lista
- 3 Pilha
- 4 Fila
- 5 Espalhamento
- 6 Árvore**
- 7 Complexidade



## Definição

É uma estrutura de dados em que cada elemento tem um ou mais elementos associados.

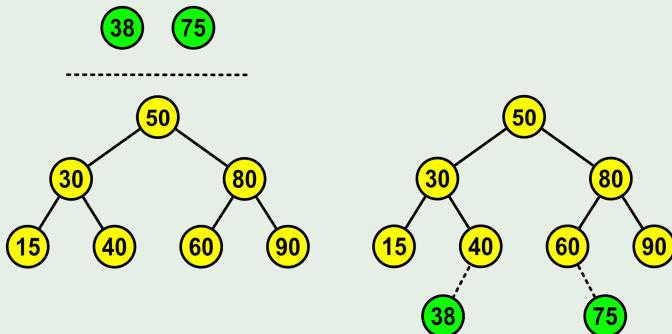
## Ilustração





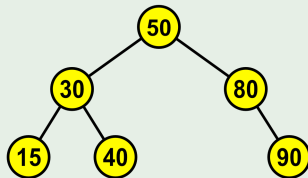
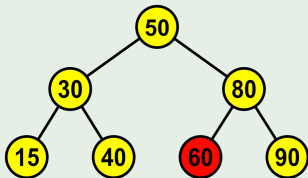
# Inserir um Nó

## Ilustração



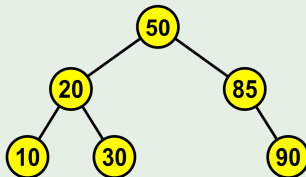
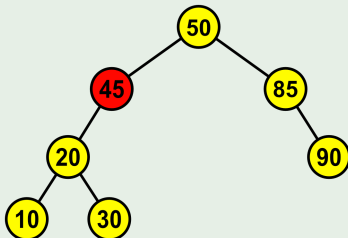
# Excluir um Nó Folha

## Ilustração



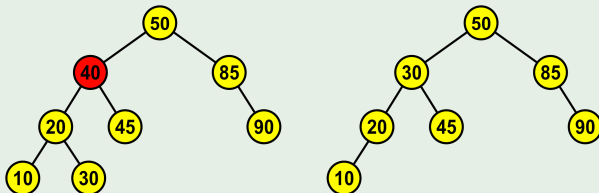
# Excluir um Nó com uma Subárvore

## Ilustração

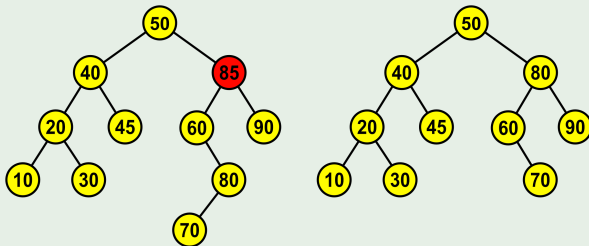


# Excluir um Nó com duas Subárvores

## Ilustração: Caso 1



## Ilustração: Caso 2



- Árvore AVL; e
- Árvore Rubro Negra.



# Ementa do Curso

- 1 Ordenação
- 2 Lista
- 3 Pilha
- 4 Fila
- 5 Espalhamento
- 6 Árvore
- 7 Complexidade**



## Definição

É um mecanismo para entender e avaliar um algoritmo em relação aos critérios desempenho, bem como saber aplica-los à problemas práticos.

- Empírica; ou
- Matemática.



## Definição

É um método de descrever o comportamento de limites de algoritmos quando aplicados a um volume muito grande de dados de entrada.

- Big  $O$  (Pior dos casos);
- Big  $\Omega$  (Melhor dos casos); e
- Big  $\Theta$  (Caso médio).





# Crescimento de Função no Big O

- $O(1)$
- $O(\log N)$
- $O(N)$
- $O(N \log N)$
- $O(N^2)$
- $O(2^N)$
- $O(N!)$



# Função $O(1)$

## Descrição

Maior parte das instruções são executadas apenas uma ou algumas vezes, independente de  $N$  – tempo de execução constante.

## Exemplo

```
def constante(n):  
    print(n)
```



# Função $O(\log N)$

## Descrição

Ocorre em programas que resolve um problema maior transformando-o em uma série de subproblemas menores, assim reduzindo o tamanho do problema por uma certa constante fracionária a cada passo – tempo de execução logarítmico. Sempre que  $N$  dobra,  $\log N$  aumenta de uma certa constante, mas  $\log N$  não dobra até que  $N$  tenha sido aumentado para  $N^2$ .

## Exemplo

```
def busca_binaria(n, array):  
    metade = len(array)//2  
    if(n == array[metade]):  
        return metade  
    elif(n > array[metade]):  
        return busca_binaria(n, array[metade:])  
    elif(n < array[metade]):  
        return busca_binaria(n, array[:metade-1])
```



# Função $O(N)$

## Descrição

Acontece quando uma pequena quantidade de processamento deve ser feito para cada elemento da entrada – tempo de execução linear. Esta é a situação ótima para um algoritmo que deve processar  $N$  entradas (ou gerar  $N$  saídas).

## Exemplo

```
def linear(n):  
    for i in range(n):  
        print(i)
```



# Função $O(N \log N)$

## Definição

Ocorre em algoritmos que quebra o problema principal em subproblemas menores, resolvendo-os e combinando as soluções – tempo de execução “linearítmico” ou  $N \log N$ . Quando  $N$  dobra o tempo de execução torna-se um pouco mais do que o dobro.

## Exemplo

Algoritmo de ordenação por mistura (Merge Sort)



# Função $O(N^2)$

## Definição

Tipicamente representa algoritmos que processa todos os pares de itens de dados – tempo de execução quadrático. Este tipo de complexidade é aceitável apenas para problemas relativamente pequenos. Quando  $N$  dobra o tempo de execução aumenta 4 vezes.

## Exemplo

```
def quadratico(n);  
    for i range(n):  
        for j range(n):  
            print(i, j)
```



## Definição

Corresponde a algoritmos que utilizam força-bruta na solução de problemas – tempo de execução exponencial. Algoritmos com esta performance são impraticáveis. Sempre que  $N$  dobra o tempo de execução é elevado ao quadrado.

## Exemplo

Encontrar a solução exata para o Problema do cacheiro viajante usando programação dinâmica.



## Definição

## Exemplo

Encontrar a solução exata para o Problema do caixeiro viajante usando busca por força bruta.

