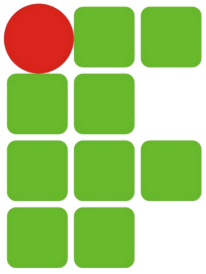


Estrutura de Dados

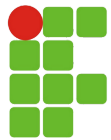
Diego Silveira Costa Nascimento

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

22 de agosto de 2018

Ementa do Curso

1. Ordenação
2. Lista
3. Pilha



Ementa do Curso

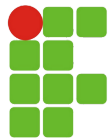
1. Ordenação
2. Lista
3. Pilha

Ordenação

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

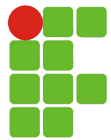
- A ideia é percorrer o vector diversas vezes;
- A cada passagem fazendo flutuar para o topo o maior elemento da sequência; e
- Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível.

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.



Ordenação por Mistura

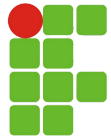
Exemplo

```
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
```

Ordenação por Mistura

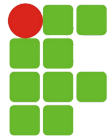
Exemplo

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



Ordenação Rápida

- 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.



Ordenação Rápida

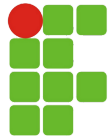
Exemplo

```
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)
```

Ordenação Rápida

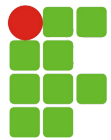
Exemplo

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



Ementa do Curso

1. Ordenação
2. Lista
3. Pilha

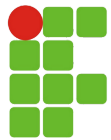


Lista Ligada

Definição

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

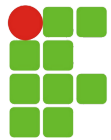
- Lista ligada;
- Lista duplamente ligada; e
- Lista circular.



Classe Elemento

Código

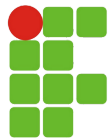
```
class Elemento:
    def __init__(self,nome):
        self.nome = nome
        self.proximo = None
```



Classe Lista Ligada

Código

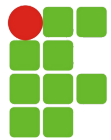
```
class ListaLigada:
    def __init__(self):
        self.inicio = None
```



Método Adicionar

Código

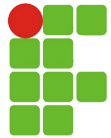
```
def adicionar(self, nome):
    novo = Elemento(nome)
    if self.inicio == None:
        self.inicio = novo
    else:
        elemento = self.inicio
        while elemento.proximo != None:
            elemento = elemento.proximo
        elemento.proximo = novo
```



Método Exibir

Código

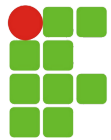
```
def exibir(self):
    elemento = self.inicio
    print(elemento.nome)
    while elemento.proximo != None:
        elemento = elemento.proximo
        print(elemento.nome)
```



Método Remover

Código

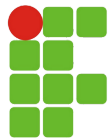
```
def remover(self, nome):
    elemento = self.inicio
    if elemento.nome == nome:
        self.inicio = elemento.proximo
    else:
        while elemento.proximo != None:
            if elemento.proximo.nome == nome:
                elemento.proximo = elemento.proximo.proximo
            else:
 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



Usando Lista Ligada

Código

```
lista = ListaLigada()

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

lista.remover('Pedro')

lista.exibir()
```



Lista Duplamente Ligada

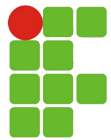
Definição

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

Classe Elemento

Código

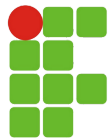
```
class Elemento:
    def __init__(self,nome):
        self.anterior = None
        self.nome = nome
        self.proximo = None
```



Classe Lista Duplamente Ligada

Código

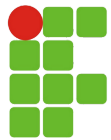
```
class ListaDuplamenteLigada:
    def __init__(self):
        self.inicio = None
```



Método Adicionar

Código

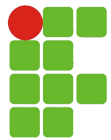
```
def adicionar(self, nome):
    novo = Elemento(nome)
    if self.inicio == None:
        self.inicio = novo
    else:
        elemento = self.inicio
        while elemento.proximo != None:
            elemento = elemento.proximo
        elemento.proximo = novo
        novo.anterior = elemento
```



Método Exibir

Código

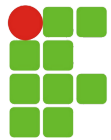
```
def exibir(self):
    elemento = self.inicio
    print(elemento.nome)
    while elemento.proximo != None:
        elemento = elemento.proximo
        print(elemento.nome)
    while elemento.anterior != None:
        elemento = elemento.anterior
        print(elemento.nome)
```



Método Remover

Código

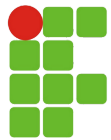
```
def remover(self, nome):
    elemento = self.inicio
    if elemento.nome == nome:
        self.inicio = elemento.proximo
        self.inicio.anterior = None
    else:
        elemento = elemento.proximo
        while elemento != None:
            if elemento.nome == nome:
                if elemento.proximo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



Usando Lista Duplamente Ligada

Código

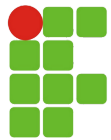
```
lista = ListaDuplamenteLigada()

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

lista.remover('Pedro')

lista.exibir()
```



Lista Circular

Definição

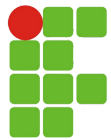
É uma estrutura de dados que implementa uma coleção de dados ligados de forma dinâmica em um único sendito, no qual o final da lista corresponde o início da própria lista.



Classe Lista Circular

Código

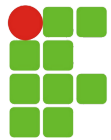
```
class ListaCircular:
    def __init__(self):
        self.inicio = None
```



Método Adicionar

Código

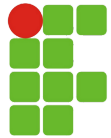
```
def adicionar(self, nome):
    novo = Elemento(nome)
    if self.inicio == None:
        self.inicio = novo
        novo.proximo = self.inicio
    else:
        elemento = self.inicio
        while elemento.proximo.nome != self.inicio.nome:
            elemento = elemento.proximo
        elemento.proximo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



Método Exibir

Código

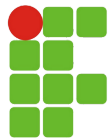
```
def exibir(self):
    elemento = self.inicio
    print(elemento.nome)
    while elemento.proximo.nome != self.inicio.nome:
        elemento = elemento.proximo
        print(elemento.nome)
    print(elemento.proximo.nome)
```



Método Remover

Código

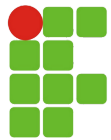
```
def remover(self, nome):
    elemento = self.inicio
    if elemento.nome == nome:
        self.inicio = elemento.proximo
        while elemento.proximo.nome != nome:
            elemento = elemento.proximo
        elemento.proximo = self.inicio
    else:
        while elemento.proximo.nome != nome:
            elemento = elemento.proximo
        elemento.proximo = elemento.proximo.proximo
```



Usando Lista Circular

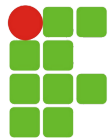
Código

```
lista = ListaCircular()

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

lista.remover('João')

lista.exibir()
```



Ementa do Curso

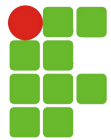
1. Ordenação
2. Lista
3. Pilha



Pilha

Definição

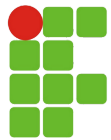
É uma estrutura de dados baseada no princípio LIFO (Last In, First Out), na qual os dados que foram inseridos primeiro na pilha serão os últimos a serem removidos.



Classe Pilha

Código

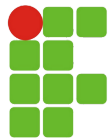
```
class Pilha:
    def __init__(self):
        self.topo = None
```



Método Empilhar

Código

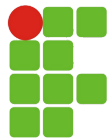
```
def empilhar(self,nome):
    novo = Elemento(nome)
    if self.topo == None:
        self.topo = novo
    else:
        novo.proximo = self.topo
        self.topo = novo
```



Método Desempilhar

Código

```
def desempilhar(self):
    if self.topo == None:
        return None
    else:
        elemento = self.topo
        self.topo = elemento.proximo
        return elemento.nome
```



Usando Pilha

Código

```
pilha = Pilha()

pilha.empilhar('João')
pilha.empilhar('Lucas')
pilha.empilhar('Pedro')
pilha.empilhar('Maria')

nome = pilha.desempilhar()

print(nome)
```