Estruturas de Dados Básicas - Listas

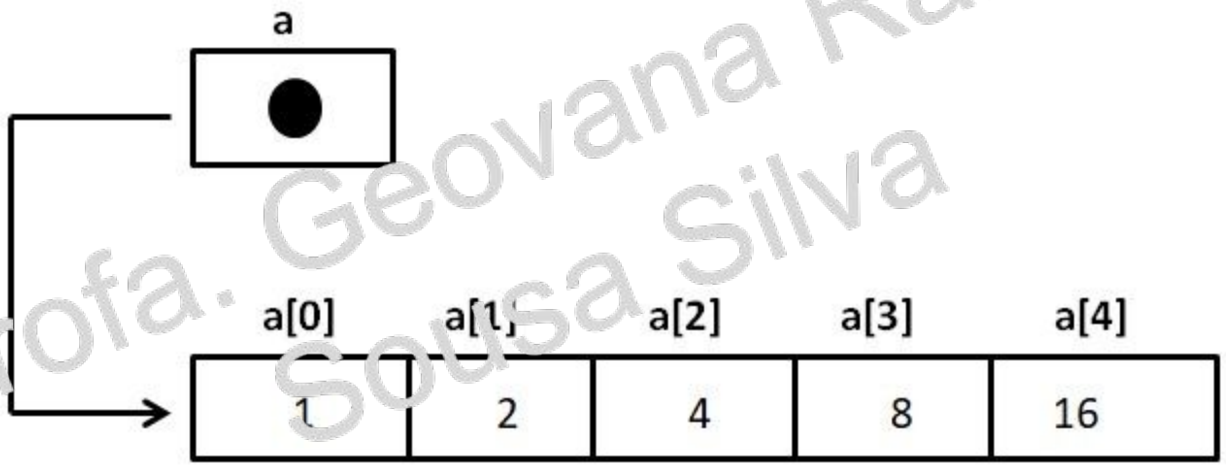
Estruturas de Dados

Universidade de Brasília
Departamento de Ciência da Computação

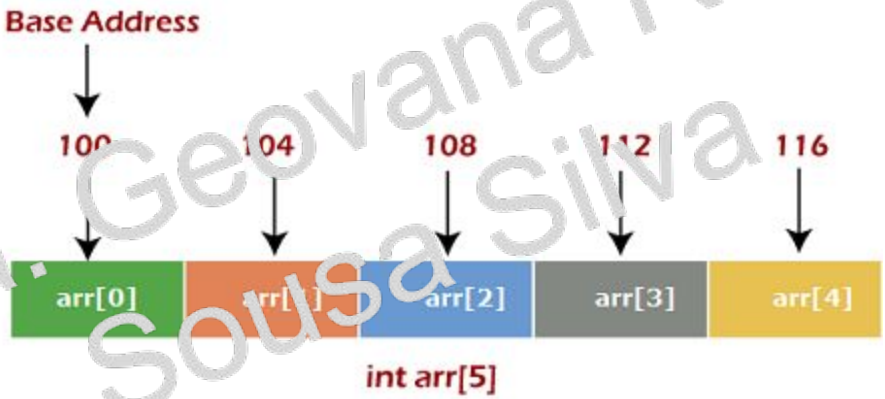
Estruturas de “Lista” - Conceitos

- Array
- Dynamic Array (Array Dinâmica)
- Linked List (Lista Encadeada)
  - Singly
  - Doubly (Duplamente)
  - Circular

Array



Array



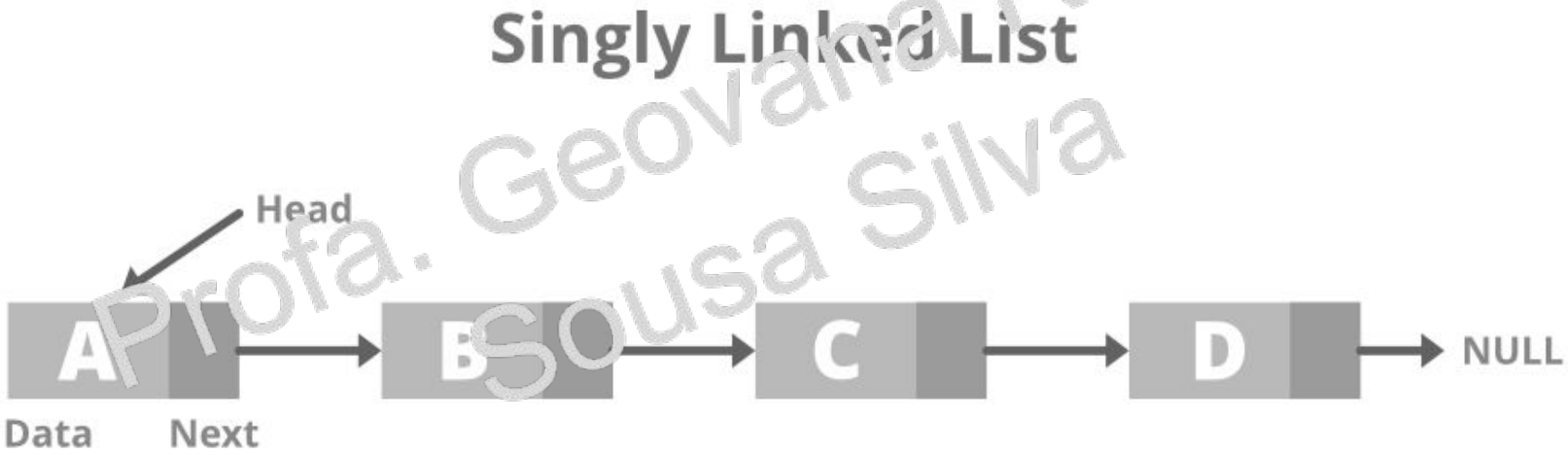
Array

- Vantagens
  - Acesso de elementos em tempo constante O(1).
  - São armazenados em locais contíguos de memória.
- Desvantagens
  - Uma vez criado, o tamanho não pode ser alterado.
  - Inserções e deleções são custosas (O(n)) já que requerem a movimentação de elementos.

Array Dinâmica

- Vantagens
  - Acesso de elementos em tempo constante O(1).
  - São armazenados em locais contíguos de memória.
  - Seu tamanho pode ser alterado durante a execução do programa.
- Desvantagens
  - Aumentar o tamanho pode requerer a realocação da estrutura inteira.
  - Inserções e deleções são custosas (O(n)) já que requerem a movimentação de elementos.

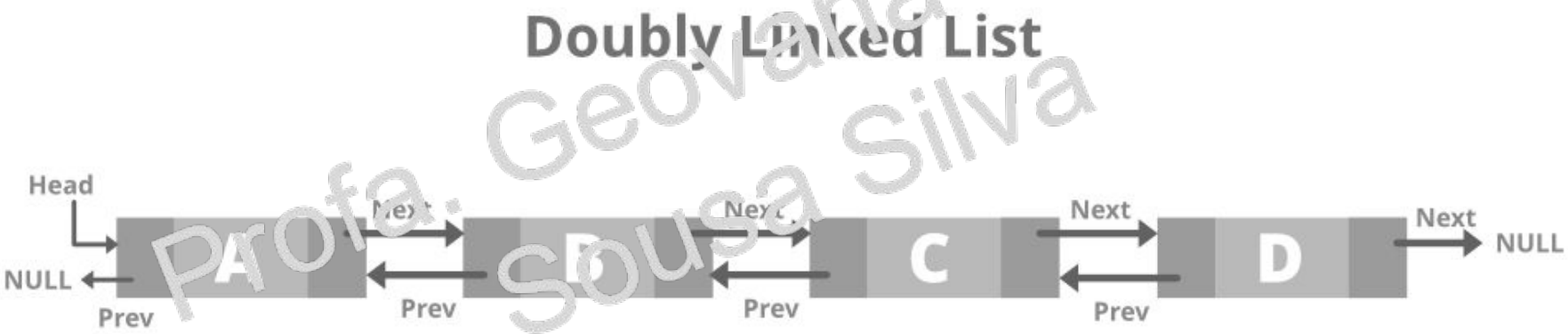
Lista Encadeada



Lista Encadeada

- Vantagens
  - Não há limite de tamanho pré-definido.
  - Inserções e deleções no meio não requerem movimentação dos demais elementos.
- Desvantagens
  - Sem acesso direto, a lista deve ser percorrida até o elemento.
  - Elementos exigem mais memória por armazenarem valor + ponteiro.

Lista Duplamente Encadeada



Lista Encadeada Circular

Comparativo

- **Array**: A inserção e deleção são O(1) apenas se forem no final do array. Caso contrário, é necessário mover os outros elementos, tornando a complexidade O(n).
- **Array Dinâmico**: Além da complexidade do array estático, possui a complexidade O(n) quando precisa realocar espaço para aumentar de tamanho.
- **Listas Encadeadas**: Para inserção e deleção serem O(1), é necessário já ter um ponteiro para o local onde a operação ocorrerá. Se você precisa primeiro localizar esse local, a operação pode degradar para O(n).

Estruturas de “Lista” - Python

- Array → array = (1, 2, 3, 4)
- Dynamic Array (Array Dinâmica) → d_array = [1, 2, 3, 4]
- Linked List (Lista Encadeada) → Implementar com classes
  - Singly
  - Doubly (Duplamente)
  - Circular

Estruturas de “Lista” - Python

Algumas estruturas satisfazem o conceito de “array dinâmica”:

- Listas, conforme o slide anterior
  - Dados de diferentes tipos (heterogênea)
- Biblioteca array → *import array as arr*
  - Dados do mesmo tipo (homogênea)

Listas Encadeadas

Não possuem implementação padrão pelo Python.

Veremos como implementar com classes.

Lista Encadeada

Os “nós” podem ser definidos como uma classe que possui os atributos data e next.

A lista pode ser uma classe que aponta para o primeiro elemento da lista.

Lista Encadeada - Classes - Python

```
class Noh:
    def __init__(self, valor):
        self.valor = valor
        self.proximo = None

class ListaEncadeada:
    def __init__(self):
        self.cabeca = None

    # Métodos com as operações
    ...
```

Operações de Listas Encadeadas

- Travessia
- Busca
- Inserção
  - Início
  - Meio
  - Fim
- Deleção

Operações de Listas Encadeadas

- **Travessia**
  a. Comece pelo nó de cabeça (ou "head") da lista.
  b. Acesse o valor armazenado no nó atual e execute a impressão do valor.
  c. Avance para o próximo nó, seguindo o ponteiro "next".
  d. Repita os passos 2 e 3 até chegar a um nó cujo ponteiro "next" é null, indicando o final da lista.

Operações de Listas Encadeadas

- **Busca**
  a. Comece pelo nó de cabeça da lista.
  b. Compare o valor no nó atual com o valor desejado.
  c. Se eles forem iguais, a busca foi bem sucedida e o nó foi encontrado.
  d. Caso contrário, avance para o próximo nó, seguindo o ponteiro "next".
  e. Repita os passos 2 a 4 até encontrar o valor ou chegar ao final da lista.

Operações de Listas Encadeadas

- **Inserção (início|meio|fim)**
  a. Um novo nó é inicialmente criado com o dado desejado. O próximo deste nó (geralmente chamado de "próximo" ou “next”) é inicialmente definido como null.

Operações de Listas Encadeadas

- **Inserção no início**
  a. O "proximo" do novo nó é ajustado para apontar para o atual nó de cabeça (ou "head") da lista.
  b. O nó de cabeça da lista é então atualizado para ser o novo nó.

Operações de Listas Encadeadas

- **Inserção no meio**
  a. Para inserir um novo nó após um nó específico (por exemplo, o i-ésimo nó), é necessário iterar até esse nó.
  b. Uma vez que o nó é encontrado, o "proximo" do novo nó é ajustado para apontar para o nó após o nó i-ésimo (ou seja, o "proximo" do i-ésimo nó).
  c. O "proximo" do nó i-ésimo é então ajustado para apontar para o novo nó.

Operações de Listas Encadeadas

- **Inserção no fim**
  a. É necessário iterar até o final da lista. Isso é feito seguindo os ponteiros "proximo" de cada nó até encontrar um nó cujo "proximo" é null, indicando que é o último nó da lista.
  b. Uma vez encontrado o último nó, o "next" desse nó é ajustado para apontar para o novo nó.

Operações de Listas Encadeadas

- **Deleção (início|meio|fim)**

Em Python, a gestão de memória é feita automaticamente por meio de um recurso chamado "coletor de lixo" (garbage collector). Isso significa que quando um objeto não tem mais referências apontando para ele, ele se torna elegível para coleta de lixo, e a memória que ele ocupa será liberada automaticamente em algum momento. Então, ao remover um nó de uma lista encadeada em Python, basta ajustar os ponteiros de forma que nenhuma parte do código tenha mais uma referência ao nó removido.

Operações de Listas Encadeadas

- **Deleção no início**
  a. Ajuste o nó de cabeça para ser o segundo nó na lista (ou seja, o "proximo" do nó de cabeça atual).

Operações de Listas Encadeadas

- **Deleção no meio**
  a. Localize o nó antecessor do nó que deseja deletar (ou seja, o nó cujo "proximo" aponta para o nó a ser deletado).
  b. Ajuste o ponteiro "proximo" do nó antecessor para apontar para o nó após o nó a ser deletado.

Operações de Listas Encadeadas

- **Deleção no fim**
  a. Localize o penúltimo nó da lista (ou seja, o nó cujo "proximo" aponta para o último nó).
  b. Ajuste o ponteiro "proximo" do penúltimo nó para null, tornando-o o novo último nó da lista.

Operações de Listas Encadeadas

- **Observações**
  a. É preciso sempre lidar com a “cabeça” de forma diferenciada, pois alterações nela devem ser refletidas no atributo da lista.
  b. É preciso fazer verificações de posições inexistentes na inserção/deleção do meio ou valores inexistentes na busca.
  c. As operações nas listas duplamente encadeadas e circulares seguem os mesmos princípios, porém há operações adicionais de ponteiros por conta dos atributos extras dos nós.

Duplamente

```
class Noh:
    def __init__(self, valor):
        self.valor = valor
        self.proximo = None
        self.anterior = None

class ListaDuplamenteEncadeada:
    def __init__(self):
        self.cabeca = None
        self.cauda = None

    # Métodos com as operações
    ...
```

Circular

```
class Noh:
    def __init__(self, valor):
        self.valor = valor
        self.proximo = None

class ListaCircular:
    def __init__(self):
        self.cabeca = None

    # Métodos com as operações
    ...
```

Para casa

- Utilizando o código de ListaEncadeada que está no GitHub:
  - faça um método que remova nós que são menores que a média da lista original (estado da lista no início da execução)
  - faça um método que insira um nó sempre na primeira posição encontrada onde o antecessor e predecessor são números maiores que o valor do nó a ser inserido. Se não houver essa posição, adicione no final.
  - faça um método que encontre o maior nó e o envie para o final da lista.

Resumo

As listas encadeadas são estruturas compostas por elementos (nós) armazenados em espaços não contíguos de memória, vinculados por ponteiros.

O Python não possui listas encadeadas por padrão, mas elas podem ser implementadas com classes.

As principais operações são a travessia, busca, inserção, e deleção.

Bibliografia