Fundamentos de Algoritmos e Estrutura de Dados #3 – Listas Encadeadas e Tabela Hash

Prof. André Gustavo Hochuli

gustavo.hochuli@pucpr.br
aghochuli@ppgia.pucpr.br

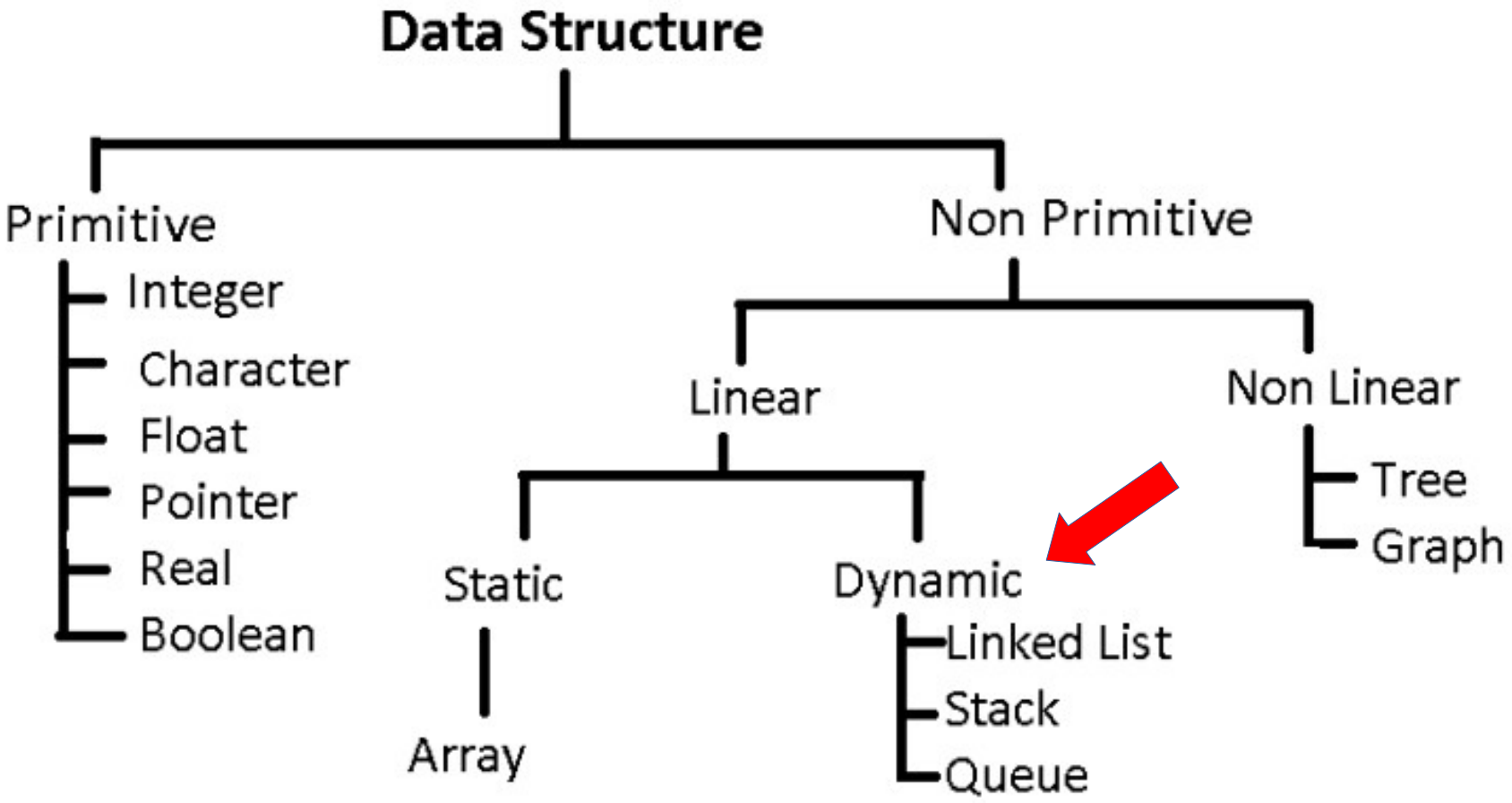
Plano de Aula

- **Listas Encadeadas**
  - **Filas e Pilhas**
- **Tabela Hash**
  - **Contextualização do Problema**
  - **Funções Hash**
  - **Colisões**
- **Exercícios e Análises**

Atividades Avaliativas

- **Trabalho #1 (30%): Estruras Lineares vs Não-Lineares**
  - **29/08 até 14/09**
  - **Códificação e Relatório em Formato de Artigo**
- **Trabalho #2 (20%): Grafos e Problemas de Busca**
  - **12/09 até 21/09**
  - **Codificação e Apresentação**
  - **Apresentação em horário de aula no formato híbrido dia 26/06**
- **Avaliação Individual Online (50%)**
  - **Formato de Prova**
  - **Dia 26/06**
- **Média**
  - **TR1*.3 + TR2*.2 + AVAL*.5 = 10**
- **Presença: +2h de aula**

Estruturas de Dados



Estruturas de Dados

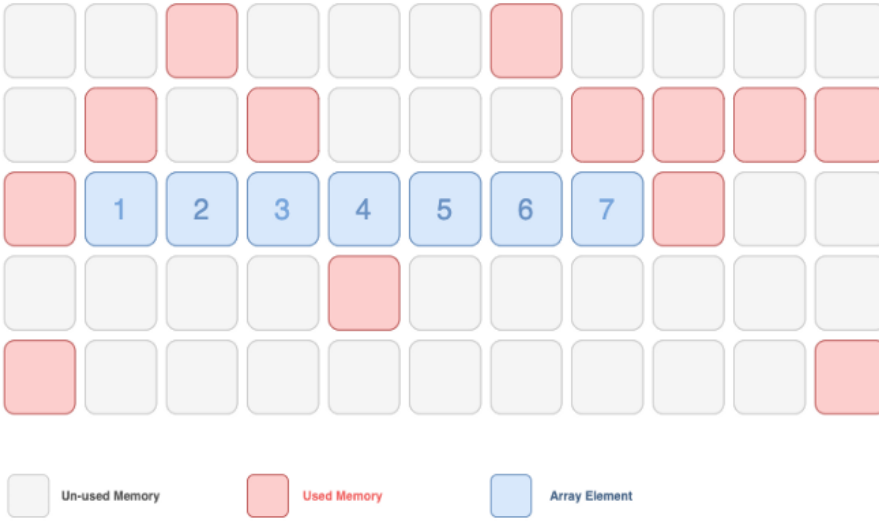
- **NOTA:**

- Embora a linguagem C seja tradicionalmente utilizada no ensino de Estruturas de Dados por permitir maior controle sobre memória e ponteiros, neste material adotamos Python e Programação Orientada a Objetos (POO) com fins didáticos.

- O objetivo é priorizar a compreensão conceitual das estruturas e de seus algoritmos, utilizando abstrações que facilitam a aprendizagem. Essa abordagem permite ao estudante focar no raciocínio algorítmico e na arquitetura das estruturas, sem se prender inicialmente a detalhes de implementação de baixo nível.

Estruturas de Dados Estáticas

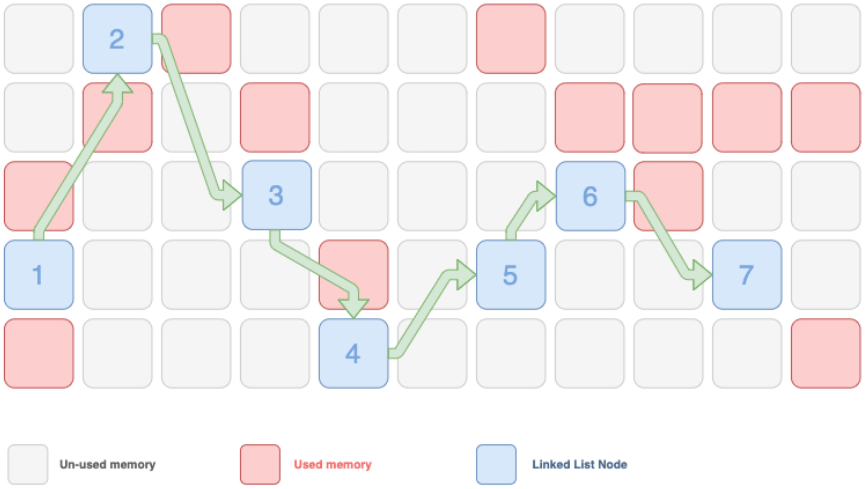
- **Alocação contígua (*Static*)**
  - **Vantagens**
    - **Acesso é rápido e sequencial**
    - **Baixo Overhead**
    - **Requer baixo nível de programação**



  - **Desvantagem**
    - **Inviável para grandes massas de dados**
    - **Limitado ao número de blocos sequenciais livres**

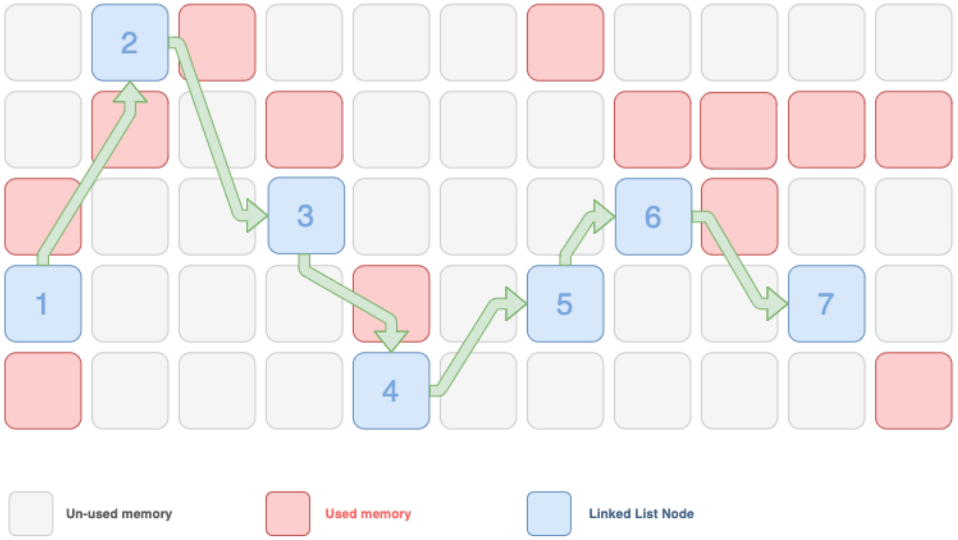
Estruturas de Dados Dinâmicas

- **Alocação não-contígua (*Dynamic*)**
  - **Vantagens**
    - **Armazenar grandes massas de dados**
    - **Memória física é o limite**
  - **Desvantagem**
    - **Overhead**
    - **Elevado nível de abstração**
  - **Arquiteturas**
    - **Listas Ligadas**
    - **Árvores**
    - **Grafos,....**

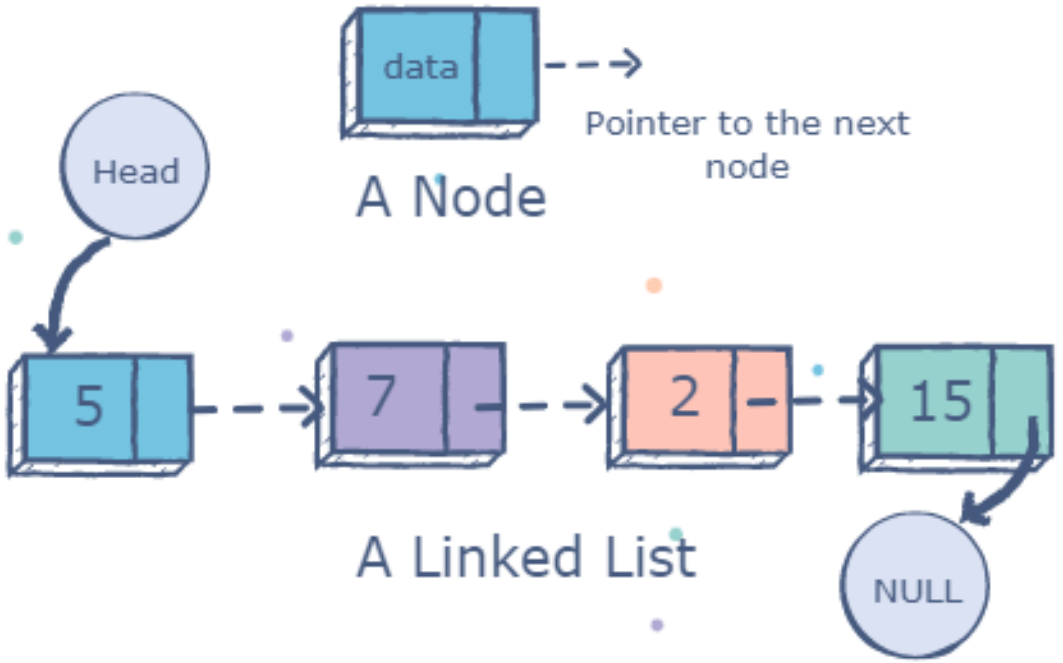


Ponteiros

- **Ponteiros**
  - **Alocam endereços de memória**
  - **Referenciam outros dados**
  - **Eficiência no acesso e modificação de dados**



Listas Ligadas



Listas Ligadas

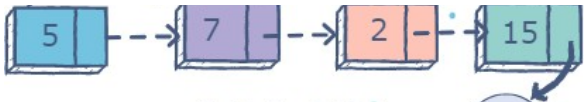
- **Implementação**


```python
class Node:
    # Function to initialize the node object
    def __init__(self, data):
        self.data = data
        self.next = None
```

Listas Ligadas

- **Implementação**


```python
class Node:
    # Function to initialize the node object
    def __init__(self, data):
        self.data = data
        self.next = None
```

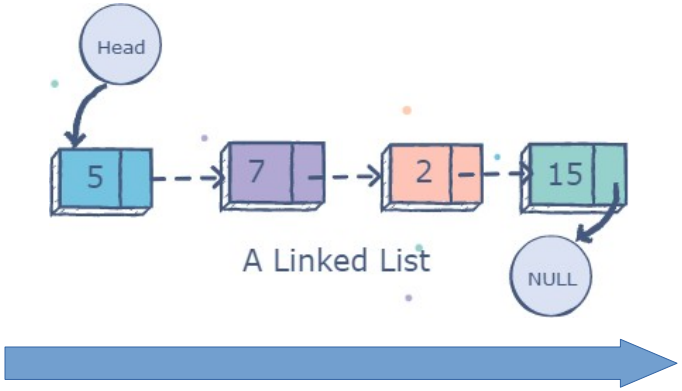



```python
#Criando a estrutura inicial de "ligação" entre os dados:
arr = Node(5)
arr.next = Node(7)
arr.next.next = Node(2)
arr.next.next.next = Node(15)


print("arr.data =",arr.data)
print("arr.next =",arr.next)
print("arr.next.data =",arr.next.data)
print("arr.next.next =",arr.next.next)
print("arr.next.next.data =",arr.next.next.data)
print("arr.next.next.next =",arr.next.next.next)
print("arr.next.next.next.data =",arr.next.next.next.data)
print("arr.next.next.next.next =",arr.next.next.next.next)
```

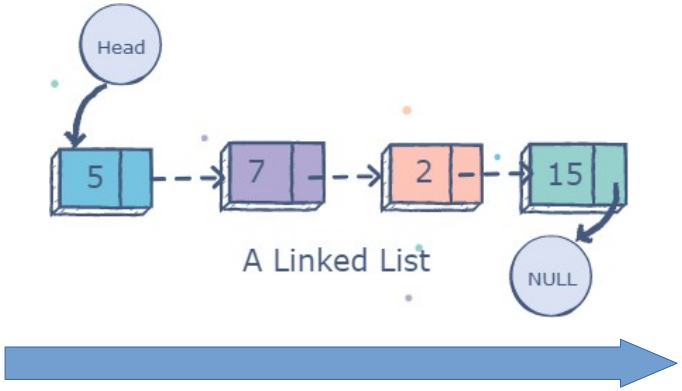
Listas Ligadas

- **E como percorrer a lista a partir do primeiro elemento?**



Listas Ligadas

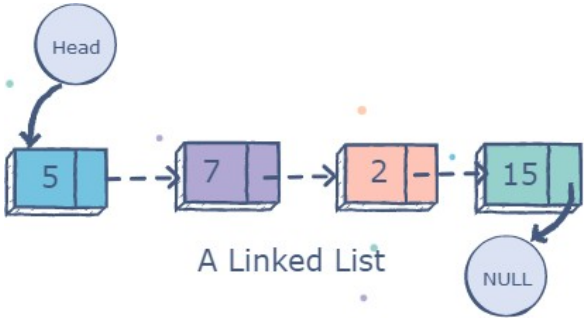
- **E como percorrer a lista a partir do primeiro elemento?**




```
1  head = a
2  while head is not None:
3      print(head.data)
4      head = head.next
```

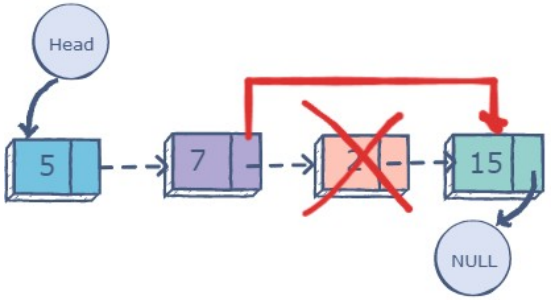
Listas Ligadas

- **Como remover um elemento ? (i.e 2)**



Listas Ligadas

- **Como remover um elemento ? (i.e 2)**
  - **Atualizando as referências**

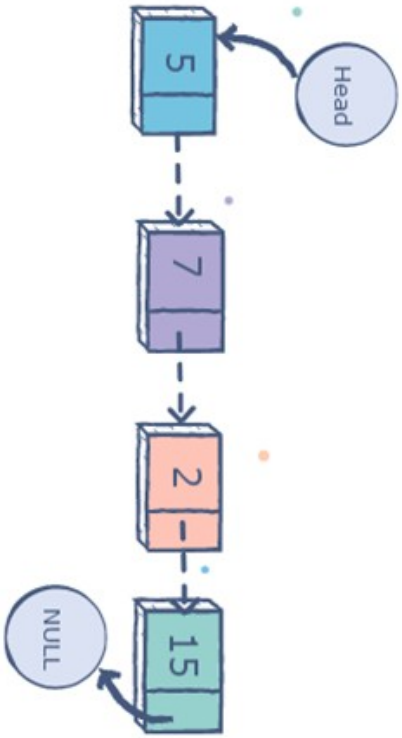



```
while head is not None and head.data == value:
    head = head.next

current = head
while current is not None and current.next is not None:
    if current.next.data == value:
        temp = current.next
        current.next = current.next.next #atualiza a referência
        #apenas ilustrativo, python tem garbage collector
        del temp
    else:
        current = current.next
```

Pilha (Stack)

- **Pilha ou (*Stack*)**
  - **Inserção e Remoção da cabeça (Last In – First Out) - LIFO**
- **Aplicações**
  - **Recursão (Programação)**
  - **Reverter Vetores**
  - **Histórico de Navegação**
  - **Etc**



Pilha (Stack)

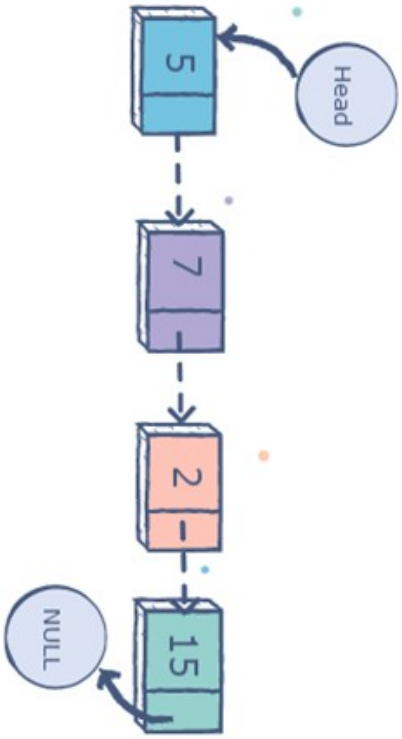
- **Pilha ou (*Stack*)**
  - **Inserção**

```
def push(head, value):
    new_node = Node(value)
    new_node.next = head
    return new_node  # novo topo
```

  - **Remoção**

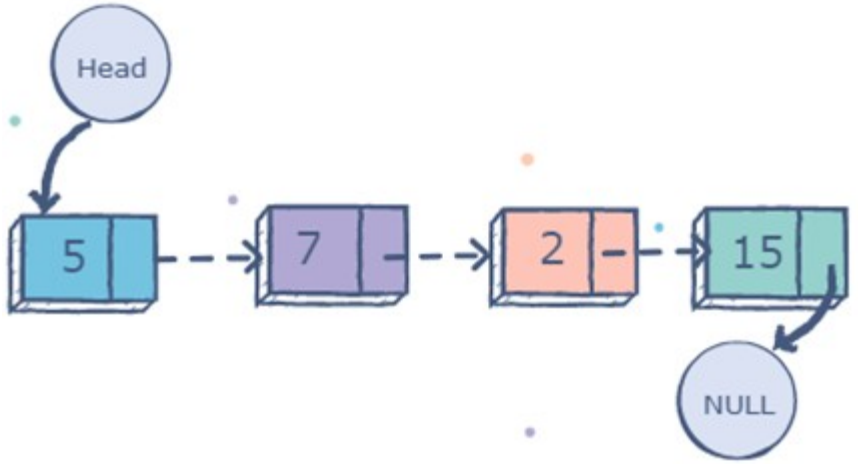
```
def pop(head):
    if head is None:
        return None, None  # pilha vazia

    value = head.data
    temp = head
    head = head.next
    del temp  #libera memória
    return top, value
```



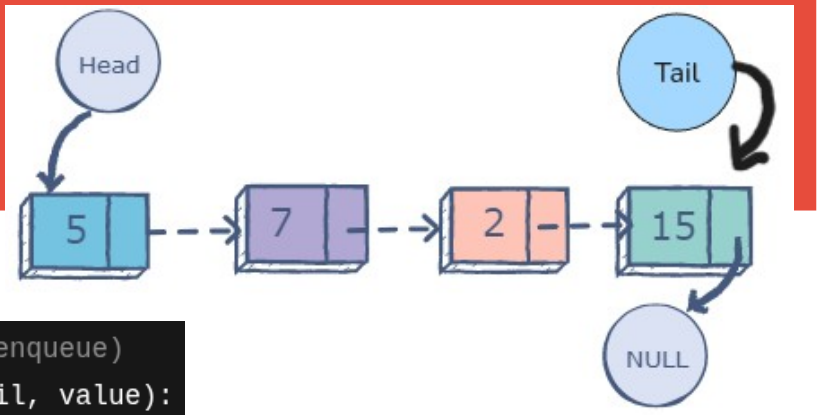
Fila (Queue)

- Fila (Queue)
  - Inserção da Cauda
  - Remoção da Cabeça
  - First In – Last Out (FIFO)



- Aplicações
  - Compartilhamento de Recursos
    - CPU, Interrupções, Harwades e Periféricos
  - Controle de Acesso
  - Transfêrência de Dados
  - Playlists

Fila (Queue)



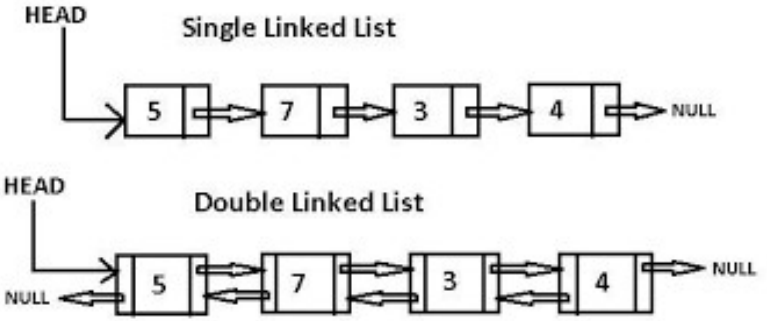
- **Utiliza dois ponteiros**
- **Inserção na Cauda**
- **Remoção da Cabeça**

```python
# Inserir no final (enqueue)
def enqueue(head, tail, value):
    new_node = Node(value)
    if head is None:
        head = tail = new_node
    else:
        tail.next = new_node
        tail = new_node
    return head, tail
```

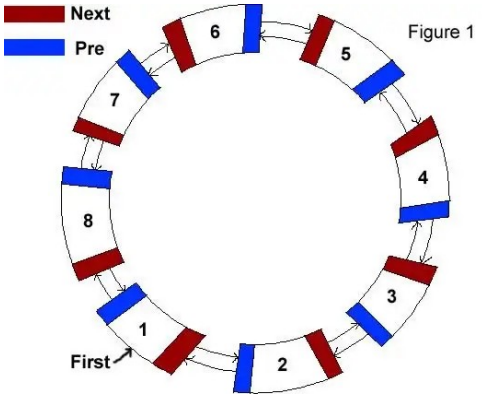
```python
# Remover do início (dequeue)
def dequeue(head, tail):
    if head is None:
        return None, None, None  # fila vazia
    value = head.data
    temp = head
    head = head.next
    if head is None:
        tail = None  # fila ficou vazia
    del temp
    return head, tail, value
```

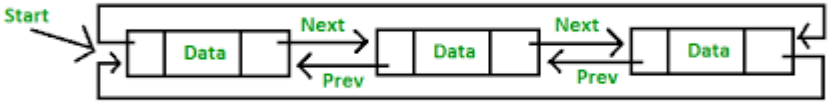
Outras Arquiteturas

- **Duplamente Encadeada**



- **Circular**





Implementação e Discussão

Let's Code!!

LINK ==> Linked Lists and Hash Table.ipynb

Exercícios

- **Implemente um lista encadeada, com inserção e remoção em qualquer posição**
- **Implemente um algoritmo de ordenação utilizando listas ligadas**
- **Implemente o jogo da torre de Hanói**
- **Implemente a busca por elemento recursiva**

Hash Table

Problema

- **Buscar um elemento em tempo constante independente da chave**

Família	1	2	3	4	5	6
	José Maria	Leila	Artur	Jolinda	Gisela	Alciene

- **Imagine um problema para armazenar identificadores de 11 dígitos (i.e CPF)**
  - **10^11 = 100.000.000.000 (100bi)**

- **A busca é custosa: O(n)**

- **Busca Binária O(log n)?**
  - **Aplicar métodos de ordenação a cada 'evento' é custoso**
  - **Recursivos (i.e QuickSort - N Log N)**
  - **Não Recursivos (i.e Insertion N^2)**

Dicionário de Dados

- **Tipo de dados abstrato que representa um objeto/entidade**
- **Implementa as funções Inserir, Buscar e Remover**
- **Utiliza chaves para indexar a informação (função hash)**

```
contatos_lista = [('Yan', '1234-5678'), ('Pedro', '9999-9999'),
                  ('Ana', '8765-4321'), ('Marina', '8877-7788')]

contatos = dict(contatos_lista)
print(contatos)
{'Yan': '1234-5678', 'Pedro': '9999-9999', 'Ana': '8765-4321',
  'Marina': '8877-7788'}
 print(contatos['Ana'])
 8765-4321
```

Key == Ana

Função Hash e Tabela Hash

- **Funções Hash ou Funções de Espalhamento é uma função de mapeamento do dados para outro domínio**
- **Não permite caminho inverso (reconstrução)**
- **A colisão é um fator importante**

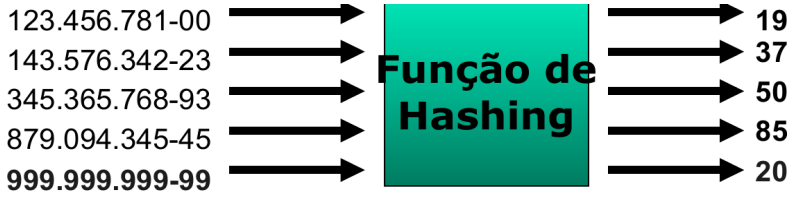
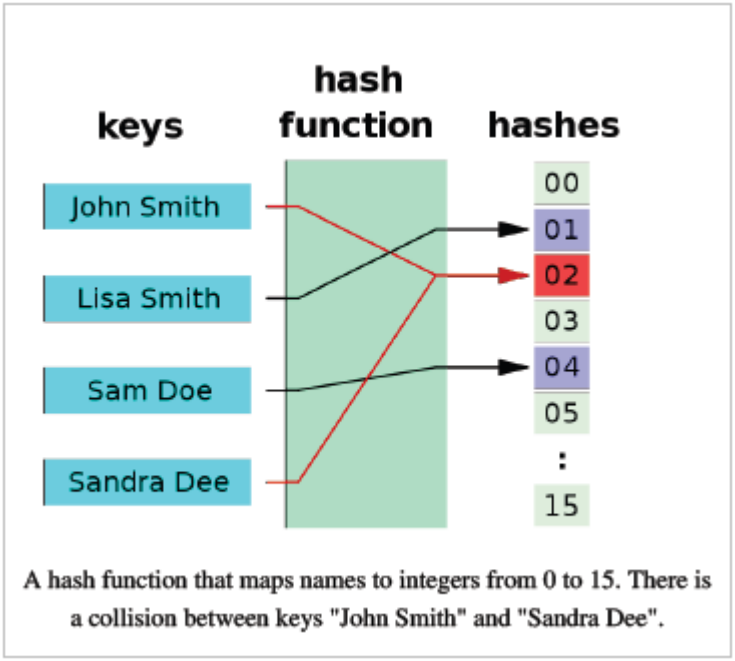


Tabela Hash

19	**123.456.781-00; João da Silva; Av. Canal. Nº 45.**
20	
...	
37	**143.576.342-23; Carla Soares; Rua Celso Oliva. Nº 27.**
...	
50	**345.365.768-93; Gustavo Liberato; Av. Atlântica. S/N.**
...	
85	**879.094.345-45 ; Helena Camargo; Rua B. Nº 100.**
...	



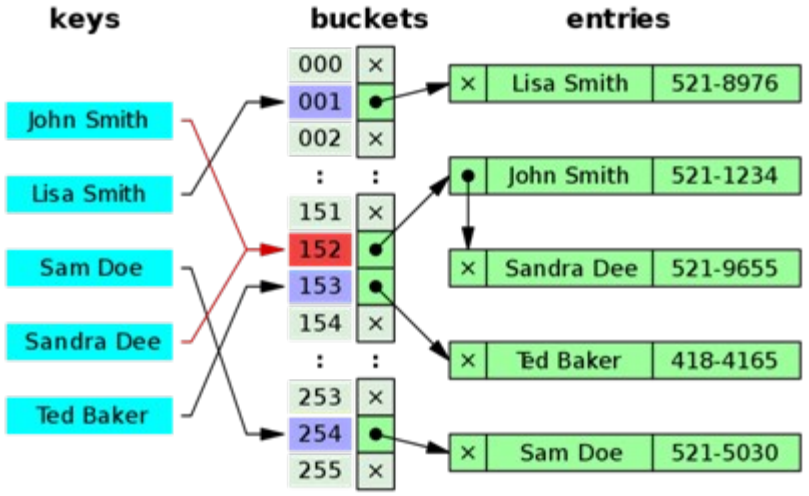
A hash function that maps names to integers from 0 to 15. There is a collision between keys "John Smith" and "Sandra Dee".

Funções Hash - h(k)

- **Módulo**
  - **Valor de M é crítico (M colisões)**

```
int hashcode(int k){
        return (k % m);
}
```

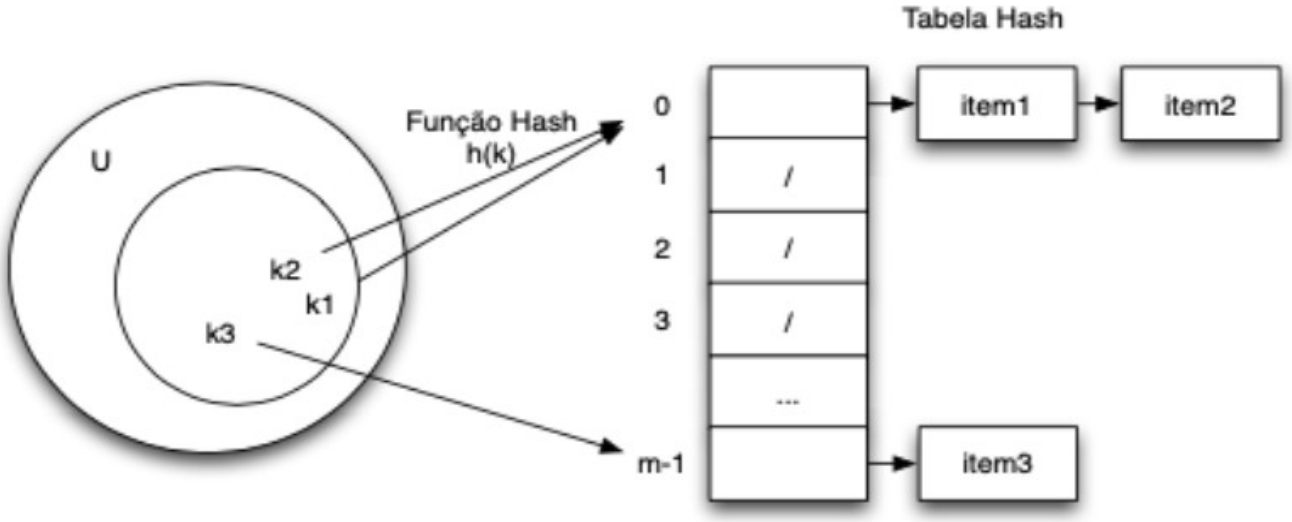
- **Outros métodos**
  - **Multiplicação**
  - **Fibonacci**
  - **Etc...**



- **Qualidade da hash determina:**
  - **Colisões vs Custo computacional**

Colisão

- **Um ou mais itens mapeados para a mesma chave**
- **Solução*: Encadeamento**
- **Tamanho da lista (#colisões) depende de h(x)**



- ***Existem outras soluções disponíveis no estado da arte**

Exercício

- **Implementar uma tabela hash**
- **Avaliar diferentes funções hash e seus parâmetros**
- **Analises Críticas e Comparações de Desempenho**
  - **i.e Número de Colisões**

Implementação e Discussão

Let's Code!!

LINK ==> Linked Lists and Hash Table.ipynb