Algoritmos e Estrutura de Dados II

Aula 17

Tabela Hash ou Tabela de Dispersão

Prof. Dr. Dilermando Piva Jr

2º Semestre - CDN

Aula 17

Tabela Hash

Algoritmos e Estrutura de Dados II

2° Semestre - CDN

Prof. Dr. Dilermando Piva Jr.

Conteúdo Programático - Planejamento

Conteúdo Programático		
Semana	**Data**	**Temas/Atividades**
1	**07/08**	Acolhimento e Boas-vindas! Introdução a Disciplina. Formas de Avaliação e Percurso Pedagógico.
2	**14/08**	Tipo de dado abstrato. Introdução a Estrutura de Dados.
3	**21/08**	Complexidade de Algoritmos
4	**28/08**	Vetores não-Ordenados e busca sequencial
5	**04/09**	Vetores Ordenados e busca binária
6	**11/09**	Revisão de Programação Orientada a Objetos (POO)
7	**18/09**	Pilhas
8	**25/09**	Filas
9	**02/10**	Listas encadeadas
10	**09/10**	Recursão
11	***16/10***	***Primeira Avaliação Formal. (P1).*** *Correção da Avaliação após o intervalo.*
12	**18/10**	Algoritmos de Ordenação
13	**23/10**	Algoritmos de Ordenação
14	**30/10**	Árvores
15	**06/11**	Árvores
16	**13/11**	Grafos
17	**27/11**	***Apresentação PI do curso de CDN***
18	**04/12**	Tabela Hash (autoestudo)
19	***11/12***	***Segunda Avaliação Formal (P2).*** Correção da Avaliação após o intervalo
20	**18/12**	***Exame / Avaliação Substitutiva.*** Correção da Avaliação após o intervalo. Finalização Disciplina

Como organizar meu armário?

Tabela Hash ou Tabela de Dispersão

- As **tabelas hash** são a base do armazenamento e recuperação eficiente de dados no desenvolvimento de software. Ao fornecer acesso rápido aos dados por meio de chaves exclusivas, as tabelas hash permitem pesquisas, inserções e exclusões em alta velocidade, tornando-as indispensáveis em cenários onde o desempenho é crítico, como indexação de banco de dados e soluções de cache.
- A essência de uma tabela hash está em seu mecanismo de hash, **que converte uma chave em um índice de array usando uma função hash**. Este índice escolhido determina onde o valor correspondente é armazenado no array.
- Ao garantir que esta função distribua as chaves uniformemente pelo array e ao empregar técnicas avançadas de resolução de colisões, as tabelas hash podem minimizar colisões e otimizar os tempos de recuperação de dados.

Tabela Hash ou Tabela de Dispersão

As tabelas de dispersão ou *hashing table*, consistem no armazenamento de cada elemento em um determinado endereço calculado a partir da aplicação de uma função sobre a chave de busca. Matematicamente teríamos o seguinte:

$$e = f(c)$$

Onde, e é o endereço; c é a chave de busca e $f(c)$ é a função que tem como entrada a chave de busca.

Tabela Hash ou Tabela de Dispersão

Dessa forma, o processo de pesquisa sobre elementos organizados dessa forma é similar a um acesso direto ao elemento pesquisado.

Exemplo:

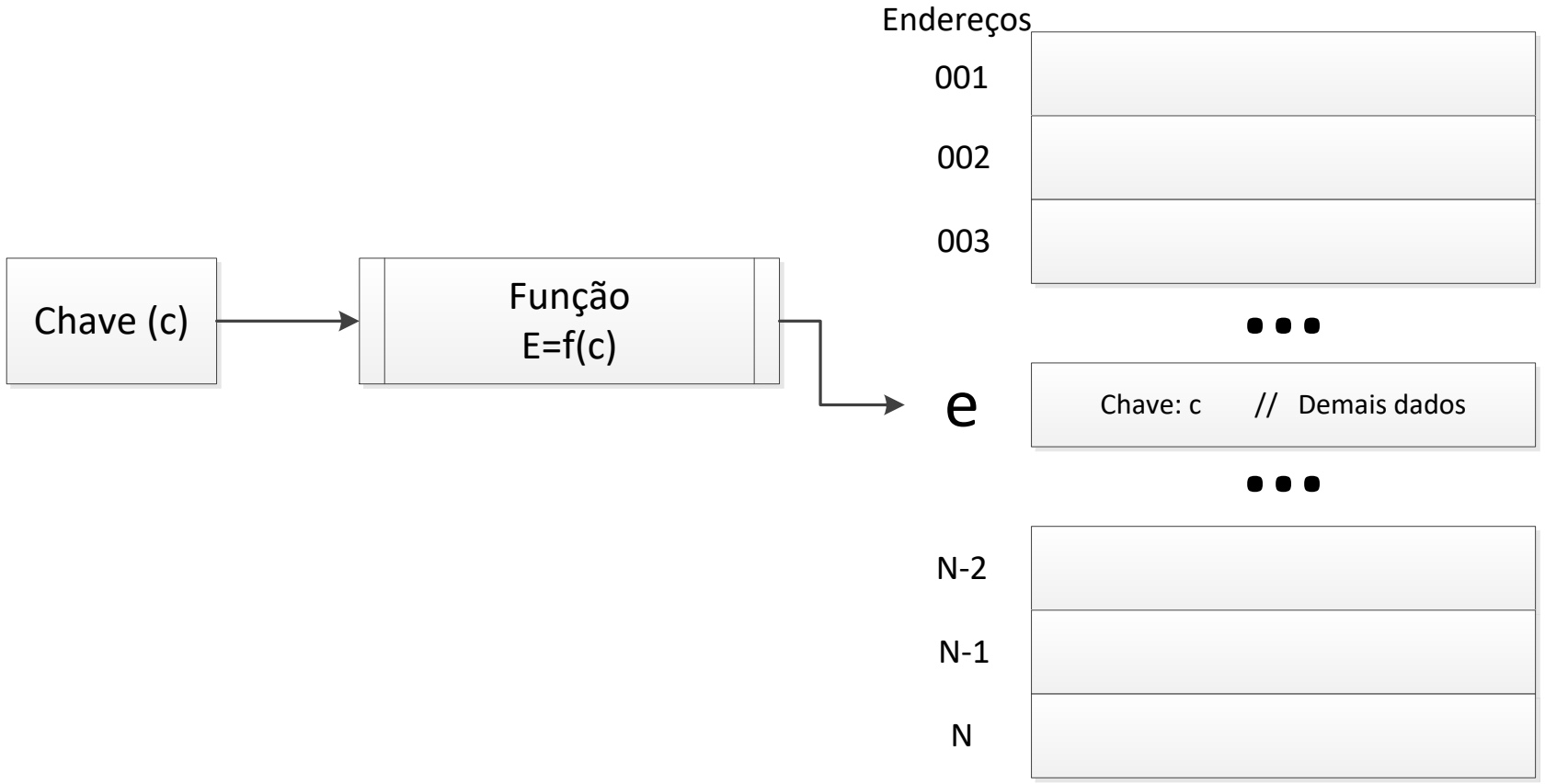


Tabela Hash ou Tabela de Dispersão

- A eficiência da pesquisa neste tipo de organização dos dados depende, indubitavelmente, da função de cálculo do endereço ($f(c)$).
- A função ideal seria aquela que pudesse gerar um endereço diferente para cada elemento da tabela (chave de busca).
- Entretanto, isso é praticamente inviável, principalmente pela dinâmica de atualização dos dados e crescimento da quantidade de elementos na tabela.

Tabela Hash ou Tabela de Dispersão

- Vamos entender esse conceito trabalhando com algo concreto.
- Supondo os dados constantes no vetor abaixo, acrescido da posição de cada elemento, teríamos:

Endereço →	0	1	2	3	4	5	6	7	8	9
Valores →	30	11	72	83	44	65	86	17	58	9

- Pensando nessa tabela e organização dos dados, poderíamos sugerir uma função de dispersão que tivesse como entrada o valor do elemento (chave de busca) e a função retornaria o endereço no vetor. Na amostra de dados, uma função possível seria:

$$f(c) = (c \bmod 10)$$

Onde **mod** corresponde ao resto da divisão inteira da chave **c** por **10** (número de elementos da tabela).

Tabela Hash ou Tabela de Dispersão

- Podemos verificar se a função de dispersão proposta é eficiente, testando alguns valores, por exemplo:

Endereço →	0	1	2	3	4	5	6	7	8	9
Valores →	30	11	72	83	44	65	86	17	58	9

Chave (c)	c mod 10	f	Acesso direto?
30	30 mod 10	0	Ok! → v[0] = 30
44	44 mod 10	4	Ok! → v[4] = 44
9	9 mod 10	9	Ok! → v[9] = 9
72	72 mod 10	2	Ok! → v[2] = 72

Tabela Hash - COLISÃO

- O problema acontece quando existe uma possibilidade de atualização de valores.

Endereço →	0	1	2	3	4	5	6	7	8	9
Valores →	30	11	72	83	44	65	86	17	58	9

- Por exemplo, vamos imaginar que queiramos mudar o valor da posição 0 (de 30 para 91). A função agora não é mais eficiente, pois se aplicarmos a função utilizando como chave o valor 91, teremos o endereço igual a 1.
- Como pode observar, o endereço 1 já está ocupado pelo valor 11.
- A esse fenômeno (dois valores distintos resultarem e um mesmo endereço quando aplicados a uma função de dispersão) chamamos de **colisão**.

Tabela Hash - COLISÃO

- A colisão é um fenômeno comum, e pode ser tratado de diversas formas.
- Apenas para você poder entender melhor o processo de colisão no mundo real, se você possui um smartphone, certamente você tem um aplicativo de gerenciamento de contatos.
- Nesse aplicativo, existe várias maneiras de acessar os contatos. Uma delas é escolhendo a letra inicial do nome.
- Quando você escolhe uma determinada letra, todos os nomes que começam com aquela letra em específico são exibidos.
- Essa é uma maneira de entendermos uma tabela de dispersão.
- A letra inicial do nome é a entrada de uma função, que separa todos os nomes que iniciam com aquela letra em específico.
- Depois disso, a busca pelo nome desejado, fica bem mais rápida. A figura ao lado ilustra essa implementação e o controle das colisões.

Tabela Hash - COLISÃO

Existem várias formas de trabalhar as colisões.

Uma das mais utilizadas é a implementação de **listas encadeadas** a partir do endereço base (Endereçamento Separado)

Exemplo:

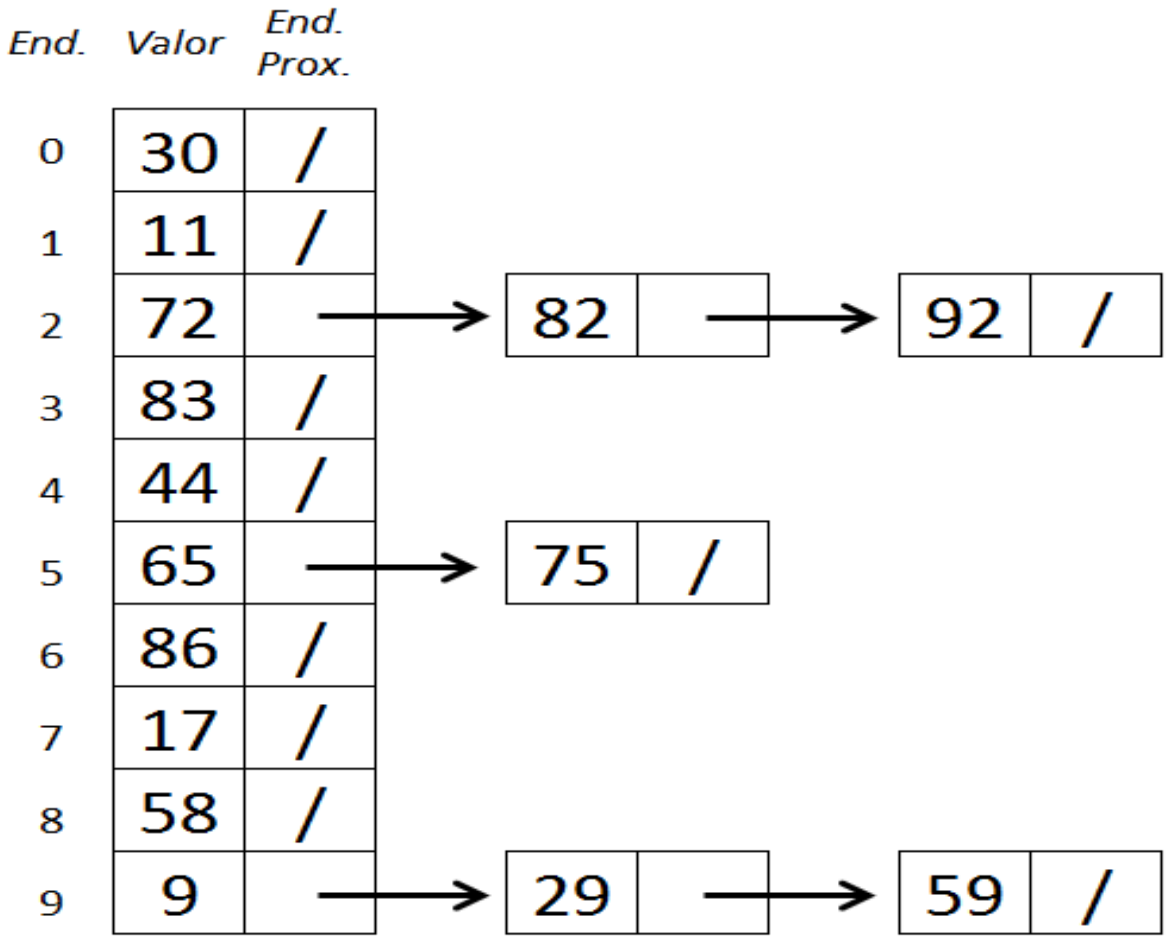


Tabela Hash - COLISÃO

Existem várias formas de trabalhar as colisões, com endereçamento aberto.

Nessa categoria, destacam-se três técnicas:

- Sondagem Linear
- Sondagem Quadrática
- Hashing Duplo

Tabela Hash - COLISÃO

Sondagem Linear

Informação a ser guardada
24

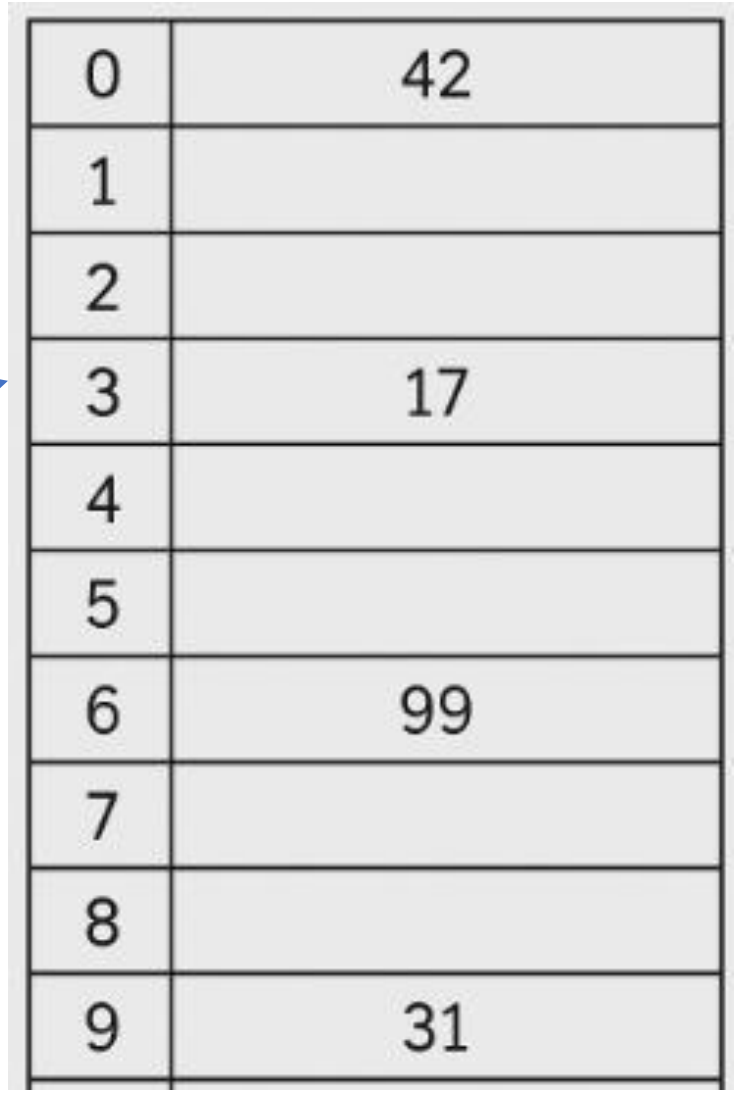
Ao gerar o endereço:

F(24) = **3**

Colisão..

Na abordagem de
Sondagem Linear

Esse valor é guardado em **4**



0	42
1	
2	
3	17
4	
5	
6	99
7	
8	
9	31

Tabela Hash - COLISÃO

Sondagem Quadrática

Informação a ser guardada
52

Ao gerar o endereço:

F(52) = **3**

Colisão..

Na abordagem de
Sondagem Quadrática **+ i ²**
Onde i = tentativa.

Esse valor é guardado em **7**

0	42
1	
2	
3	17
4	24
5	
6	99
7	
8	
9	31

Tabela Hash - COLISÃO

Hashing Duplo

Informação a ser guardada
73

Ao gerar o endereço:

F(73) = **3**

Colisão...

Na abordagem de
Hashing Duplo **f + i * f2(73)**
Onde i = tentativa.

2

Esse valor é guardado em **5**

0	42
1	
2	
3	17
4	24
5	
6	99
7	52
8	
9	31

Comparações entre Abordagens de COLISÃO

Característica	Encadeamento Separado	Sondagem Linear	Sondagem Quadrática	Hashing Duplo
Estrutura Adicional	Lista Encadeada	Nenhuma	Nenhuma	Nenhuma
Resolução de Colisão	Adicionar à Lista	Próximo Slot	Salto Quadrático	Segundo Hash
Agrupamento Primário	Não sofre	Alto	Reduzido	Mínimo
Agrupamento Secundário	Não sofre	Não sofre	Pode ocorrer	Mínimo
Uso de Memória	Mais (ponteiros)	Menos	Menos	Menos
Implementação	Simples	Simples	Moderada	Mais Complexa
Desempenho (Médio)	$O(1 + \alpha)$	$O(1/(1- \alpha))$	$O(1/(1- \alpha))$	$O(1/(1- \alpha))$

Legenda: α = *fator* de carga

Agrupamento Primário

O **agrupamento primário** ocorre quando várias chaves colidem na mesma posição inicial ou em posições próximas e, ao serem realocadas por **sondagem linear**, formam **blocos contíguos** ocupados na tabela.

Exemplo:

Suponha uma tabela de tamanho 10 e uma função hash `h(x) = x % 10`:

- Inserimos a chave 10 → `h(10) = 0`
- Depois a chave 20 → `h(20) = 0`, colisão → tenta 1
- Depois 30 → `h(30) = 0`, colisão → tenta 1 → colisão → tenta 2

Agora temos um **bloco contínuo**: posições 0, 1, 2 estão ocupadas.

➡️ Com o tempo, novas chaves que colidam com qualquer uma dessas posições terão que percorrer esse bloco, aumentando o tempo de inserção/busca.

Agrupamento Secundário

O **agrupamento secundário** ocorre em **sondagem quadrática**, quando diferentes chaves com **o mesmo índice inicial** percorrem exatamente **a mesma sequência de sondagem**.

Exemplo:

Com sondagem quadrática: `h(x) = x % 10`, `f(x, i) = (h(x) + i²) % 10`

- Chave 10 → `h(10) = 0` → tenta 0, 1, 4, 9, 6...
- Chave 20 → `h(20) = 0` também → segue **a mesma sequência**

➡️ Se a primeira chave preencher os primeiros termos da sequência, a segunda também colidirá com os mesmos, formando um **grupo "saltado"** — isso é o agrupamento secundário.

Tabela Hash - PRINCIPAIS APLICAÇÕES

- **Indexação de banco de dados**: as tabelas hash fornecem recuperação rápida de dados, o que é essencial para o desempenho dos sistemas de indexação de banco de dados.
- **Armazenamento em cache**: as tabelas hash são ideais para aplicativos de armazenamento em cache onde a pesquisa rápida de dados armazenados em cache é crucial. Eles permitem inserções, pesquisas e exclusões eficientes.
- **Desduplicação de dados**: em cenários onde a redundância de dados deve ser minimizada, as tabelas hash podem ajudar a identificar rapidamente dados duplicados.
- **Matrizes Associativas**: Muitas linguagens de programação usam tabelas hash para implementar matrizes associativas (também conhecidas como mapas ou dicionários), que podem recuperar e armazenar dados com base em chaves definidas pelo usuário.
- **Representação de dados exclusivos**: tabelas hash são úteis para manter conjuntos de itens exclusivos e são amplamente utilizadas em implementações que exigem verificações contra repetição.

Tabela Hash - IMPLEMENTAÇÃO EM PYTHON

- Em Python, a função hash pode ser chamada em qualquer objeto para retornar um valor inteiro, que chamamos de código hash ou valor hash. Vejamos alguns exemplos:

```
print(hash(“abc”))      # Exemplo de string
print(hash("123"))      # Exemplo de string numérica
print(hash(45))         # Exemplo de inteiro
print(hash(45.0))       # Exemplo de número de ponto flutuante
print(hash(45.3))       # Exemplo de outro número de ponto flutuante
print(hash(True))       # Exemplo de booleano True
print(hash(False))      # Exemplo de booleano False

try:
    print(hash([1, 2, 3]))  # Exemplo de lista (imutável)
except TypeError as e:
    print(e)  # Exibe o erro de tipo
```

VAMOS PARA A PRÁTICA ?!!!

Tabela Hash - IMPLEMENTAÇÃO EM PYTHON

Construindo a Classe HashTable

- Vamos usar uma lista de tamanho fixo para armazenar os pares (chave, valor).
- Exemplo de classe simples com sondagem linear (colisões)

Tabela Hash - IMPLEMENTAÇÃO EM PYTHON

```
class HashTable:
    __PLACEHOLDER = object()  # marcador para remoção
    def __init__(self, capacity=10):
        self.table = [None] * capacity  # lista de buckets (None, valor ou placeholder)
        self.size = 0

    def _hash(self, key):
        # Exemplo: usa o hash builtin e o módulo do tamanho
        return hash(key) % len(self.table)
```

OBSERVAÇÃO: Nesse código, __PLACEHOLDER marca posições liberadas para não interromper pesquisas em uma sequência de sondagem.

Tabela Hash - IMPLEMENTAÇÃO EM PYTHON

```
def insert(self, key, value):
        """Insere um par (chave, valor) na tabela, tratando colisões por sondagem linear."""
        idx = self._hash(key)
        start = idx
        while self.table[idx] is not None and self.table[idx] is not HashTable.__PLACEHOLDER:
            existing_key, _ = self.table[idx]
            if existing_key == key:
                # Atualiza valor para chave existente
                self.table[idx] = (key, value)
                return
            idx = (idx + 1) % len(self.table)  # avança linearmente
            if idx == start:
                raise Exception("Tabela cheia")
        # Insere na posição livre ou placeholder
        self.table[idx] = (key, value)
        self.size += 1
```

Tabela Hash - IMPLEMENTAÇÃO EM PYTHON

```
def search(self, key):
        """Retorna o valor associado à chave, ou None se não encontrado."""
        idx = self._hash(key)
        start = idx
        while self.table[idx] is not None:
            if self.table[idx] is not HashTable.__PLACEHOLDER:
                existing_key, value = self.table[idx]
                if existing_key == key:
                    return value
            idx = (idx + 1) % len(self.table)
            if idx == start:
                break
        return None
```

Tabela Hash - IMPLEMENTAÇÃO EM PYTHON

```
def remove(self, key):
    """Remove a chave da tabela, usando um placeholder para não quebrar a sondagem."""
    idx = self._hash(key)
    start = idx
    while self.table[idx] is not None:
        if self.table[idx] is not HashTable.__PLACEHOLDER:
            existing_key, _ = self.table[idx]
            if existing_key == key:
                self.table[idx] = HashTable.__PLACEHOLDER
                self.size -= 1
                return True
        idx = (idx + 1) % len(self.table)
        if idx == start:
            break
    return False
```

Tabela Hash - IMPLEMENTAÇÃO EM PYTHON

```
ht = HashTable(5)
ht.insert("aba", 123)
ht.insert("cba", 456)  # suponha que "aba" e "cba" colidam por este hash simples
print(ht.search("aba"))  # 123
ht.remove("aba")
print(ht.search("aba"))  # None (remoção feita)
```

VAMOS PARA A PRÁTICA ?!!!