Professor
Edson Pedro FERLIN

Estruturas de Dados

Tabela Hash

Prof. Edson Pedro Ferlin

Professor
Edson Pedro FERLIN

Estruturas de Dados

AULA 5 - TABELA HASH

Tabelas Hash

Conceitos • Função Hash • Tratamento de Colisões

Problemática

- E se quisermos uma estrutura de dados que nos possibilite encontrar valores rapidamente?
- Exemplos:
  - Buscar usuário pelo e-mail
  - Buscar produto pelo código
  - Buscar CEP
  - Buscar palavra em um dicionário

Existe uma solução?

- As estruturas de dados que estudamos até então somente levam em conta o valor relativo de um elemento em relação aos demais;
- Precisamos de uma solução que possibilite acessar o valor de uma determinada pergunta sem precisar verificar os demais valores;
- A solução é o uso de uma Tabela Hash.

Como localizar uma informação rapidamente?

Imagine encontrar o CPF de um cliente entre **10 milhões de registros.**

Busca Sequencial	`O(n)`
Busca Binária	`O(log n)`
Árvore Binária de Busca	`O(log n)`
Tabela Hash	`O(1) média`

?

Pergunta para discussão

Qual estrutura proporciona o acesso mais rápido a um registro?

Aplicação das Tabelas Hash

1. **Bancos de dados**
2. **Cache de navegadores**
3. **DNS — nomes de domínio**
4. **Compiladores (símbolos)**
5. **Dicionários (dict) do Python**
6. **Conjuntos (set)**
7. **Sistemas de autenticação**
8. **Indexação e buscas**

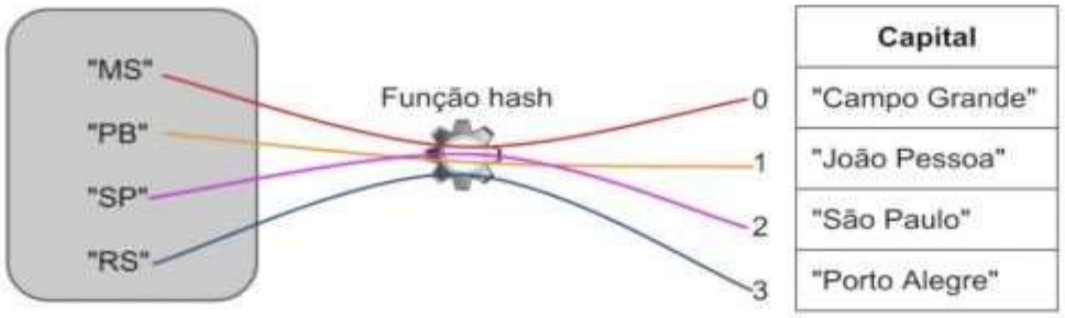
O que é uma Tabela Hash?

É uma estrutura de dados que armazena pares **chave → valor**, usando uma **função hash** para determinar a posição onde cada elemento será armazenado.

Conceito

- Uma tabela Hash ou tabela de dispersão é uma estrutura de dados que permite transformar uma chave em um endereço na memória por meio da aplicação de uma função de Hash;
- A intenção é atingir diretamente, se possível, o local onde a "resposta" para a "pergunta da chave" se encontra;
- Na realidade, o método aproveita a possibilidade de acesso randômico à memória para alcançar uma complexidade média por operação de O(1), sendo o pior caso, entretanto, O(n).

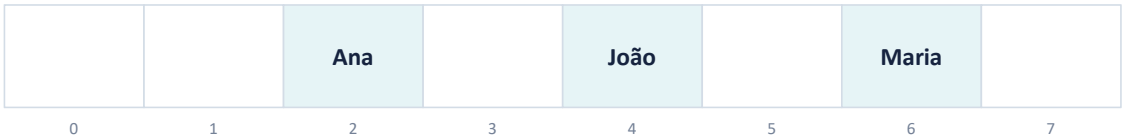
Esquema de Funcionamento



Estrutura geral

`h(chave) → posição na tabela`

Tabela



Componentes

Chave	Key — identifica o dado
Valor	Value — o dado armazenado
Função Hash	Converte a chave em índice
Índice	Posição calculada na tabela
Tabela	Vetor que guarda os elementos

Complexidade das Estruturas de Dados

Estrutura	Complexidade de busca
Lista	`O(n)`
Vetor Ordenado	`O(log n)`
BST Balanceada	`O(log n)`
Tabela Hash	`O(1) – média`

A tabela hash oferece acesso praticamente constante, independente do volume de dados.

Função Hash

Como transformar uma chave em um índice da tabela

Conceito de Função Hash

Uma função hash transforma uma chave em um índice válido da tabela.

Objetivos

1

Rapidez

cálculo imediato do índice

2

Distribuição uniforme

espalhar as chaves na tabela

3

Poucas colisões

evitar chaves no mesmo índice

Características das Funções Hash

- Produzir um número baixo de colisões
  - Idealmente h(x) != h(y) sempre que x != y
- Ser facilmente computável
  - Essa característica garante o desempenho da tabela de hash, ou seja, que o tempo de acesso a seu valor não será alto
- Ser uniforme
  - Idealmente, a função h deve ser tal que todos os compartimentos possuam a mesma probabilidade de serem escolhidos

Funções Hash

- Para atender a esses critérios, normalmente são utilizadas funções matemáticas básicas para implementar as funções de Hash
- Uma aparente limitação seria o fato de as chaves de um tabela hash não precisarem ser numéricas
  - Essa limitação é facilmente sanada, pois a todo dado não numérico corresponde uma representação numérica no computador
- Exemplos comuns de funções Hash
  - Método da Divisão
  - Método da Dobra

Método da Divisão

- Este método é particularmente fácil e eficiente, sendo por isso muito empregado. A chave x é dividida pela dimensão da tabela m, e o resto da divisão é usado como endereço-base. Isto é

  h(x) = h mod m, resultando em endereços no intervalo [0, m − 1].

Assume a table with 8 slots:

Hash key = key % table size

4 = 36 % 8

2 = 18 % 8

0 = 72 % 8

3 = 43 % 8

6 = 6 % 8

[0]	72
[1]	
[2]	18
[3]	43
[4]	36
[5]	
[6]	6
[7]	

Método da Dobra

- Suponha a chave como uma sequência de dígitos escritos num pedaço de papel
- O método em questão consiste basicamente em "dobrar" esse papel, de maneira que os dígitos se sobreponham
- Estes devem então ser somados, sem levar em consideração o "vai um"

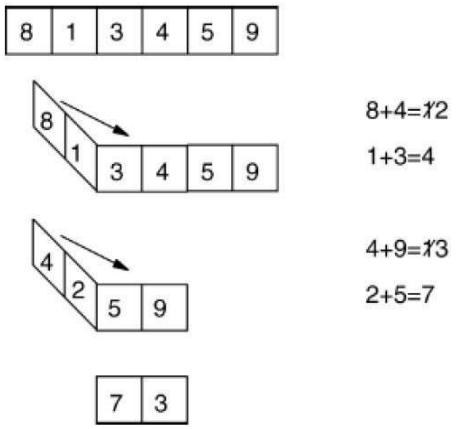


FIGURA 10.4 Método da dobra.

Exemplo matemático

Considere uma tabela de tamanho 10. Usamos o resto da divisão (módulo):

```
h(chave) = chave mod 10
```

O módulo garante que o índice esteja sempre entre 0 e 9.

Exemplo: distribuindo as chaves

Chave	h(x) = x mod 10
23	3
35	5
42	2
58	8
91	1

Tabela resultante

0	1	2	3	4	5	6	7	8	9
	91	42	23		35			58	

Cada chave ocupa a posição indicada pela função hash.

Exemplo em Python

```
def hash_func(chave, tamanho):
    return chave % tamanho

print(hash_func(23, 10))   # 3
print(hash_func(42, 10))   # 2
print(hash_func(91, 10))   # 1
```

O operador %

Em Python, `%` retorna o resto da divisão — exatamente o que a função hash precisa para gerar o índice.

Atividade - Função Hash

Utilizando **`h(x) = x mod 7`** , calcule os índices das chaves abaixo.

15	22	31	49	55
índice = ?	índice = ?	índice = ?	índice = ?	índice = ?

Características de uma boa Função Hash

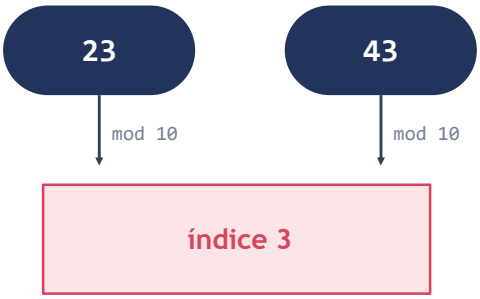
- ✓ **Simples** — fácil de calcular
- ✓ **Determinística** — mesma chave → mesmo índice
- ✓ **Uniforme** — distribui bem as chaves
- ✓ **Rápida** — sem custo computacional alto
- ✓ **Poucas colisões** — minimiza conflitos de índice

Colisões

Quando chaves diferentes disputam o mesmo índice

O que é uma colisão?

Ocorre quando duas chaves diferentes produzem o mesmo índice.

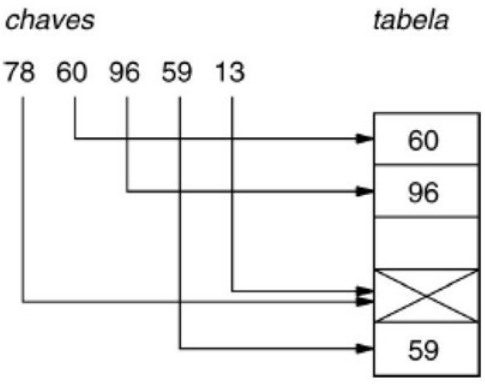



```
23 mod 10 = 3
43 mod 10 = 3
```

Mesmo índice para chaves distintas → colisão.

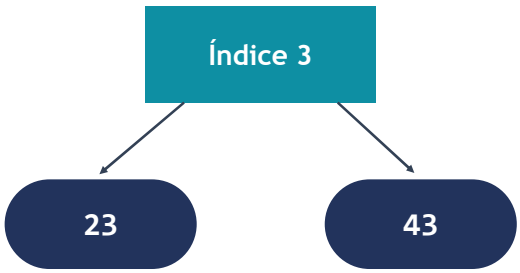
Exemplo de Colisão

- Uma colisão ocorre quando duas chaves diferentes retornam o mesmo endereço de memória



O problema das colisões

As duas chaves precisam ocupar o índice 3. Qual delas armazenar?



A solução: métodos de tratamento de colisões.

Como tratar as colisões?

- Existem várias estratégias para tratar colisões;
- Os exemplos mais famosos são:
  - Encadeamento: Cada posição da tabela armazena uma lista
  - Endereçamento aberto: Quando existe uma colisão, o elemento da colisão é posicionado em outra posição na tabela; essa posição é determinada através de outro cálculo
- Mesmo existindo formas de se tratar as colisões, a quantidade de colisões determina a eficiência do algoritmo de acesso aos dados.

Métodos de tratamento das colisões

Existem dois métodos principais para resolver colisões:

1

Encadeamento Separado

Cada posição guarda uma lista encadeada com todas as chaves daquele índice.

2

Endereçamento Aberto

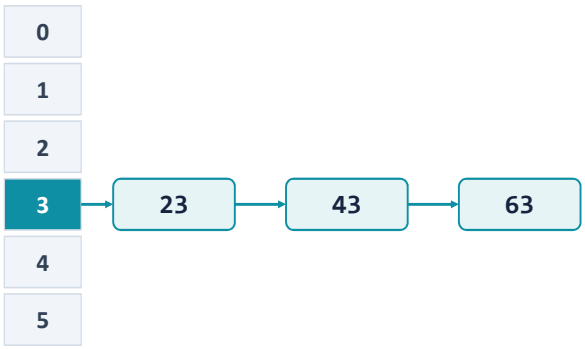
Em caso de colisão, procura-se a próxima posição livre na própria tabela.

Encadeamento Separado

Cada posição da tabela contém uma lista encadeada

Como funciona o encadeamento

Cada posição da tabela aponta para uma lista encadeada. Colisões são adicionadas à lista daquele índice.



As chaves 23, 43 e 63 colidem no índice 3 → todas ficam na mesma lista.

Implementação em Python

```
tabela = [[] for _ in range(10)]

def inserir(chave):
    indice = chave % 10
    tabela[indice].append(chave)

def buscar(chave):
    indice = chave % 10
    return chave in tabela[indice]
```

Vantagens

- ✓ Simples de implementar
- ✓ Fácil de entender
- ✓ Muitas colisões não degradam rapidamente

Endereçamento Aberto

Procura-se outra posição livre na própria tabela

Sondagem linear

Quando ocorre colisão, avança-se para a próxima posição livre da tabela.

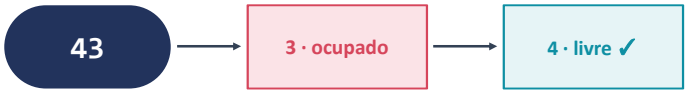


Tabela após inserir 23 e 43

0	1	2	3	4	5	6	7	8	9
			23	43					

Código e comparação

```
tabela = [None] * 10

def inserir(valor):
    indice = valor % 10
    while tabela[indice] is not None:
        indice = (indice + 1) % 10
    tabela[indice] = valor
```

Encadeamento

índice 3 → 23 → 43 → 63

uma lista por posição

Sondagem linear

3 → 23
4 → 43
5 → 63

posições livres na tabela

Dicionários em Python

A tabela hash embutida na linguagem

Dicionários

- No Python as tabelas hash são implementadas na estrutura de dados nativa dict
- dict usa uma função "hash" como função de hash
- dict utiliza endereçamento aberto para tratar colisões
- Possui uma função de segurança chamada "hashing randomization"
  - Valor de hash(x) se altera entre diferentes execuções do Python para o mesmo x

Definição

- Um dicionário é uma coleção não ordenada (até o Python 3.6), mutável e indexada por chaves
- Representado por chaves { }
- **Usa o formato chave: valor**

```
pessoa = {
    "nome": "João",
    "idade": 30,
    "profissao": "Engenheiro"
}
print(pessoa)
```

```
{'nome': 'João', 'idade': 30, 'profissao': 'Engenheiro'}
```

Acessando e modificando dados

- Os itens são acessados e modificados com base na sua chave, que não pode ser alterada
- Para adicionar um item, basta utilizar uma chave não existente no dicionário
- Para remover um item podemos utilizar o "del" (obs: Dicionários também possuem o método pop)

```
# Acesso
print(pessoa["nome"])

# Modificação
pessoa["idade"] = 29

# Adição
pessoa["cidade"] = "São Paulo"

# Remoção
del pessoa["profissao"]

print(pessoa)
```

```
João
{'nome': 'João', 'idade': 29, 'cidade': 'São Paulo'}
```


Alguns métodos úteis

- Dicionários possuem muitos métodos
- Alguns deles são:
  - keys(): Retorna as chaves do dicionário
  - values(): Retorna os valores
  - items(): Retorna pares (chave, valor)

```
# Mostrando todas as chaves do dicionário com keys()
print("🔑 Chaves do dicionário:")
print(pessoa.keys())
# Mostrando todos os valores do dicionário com values()
print("\n📋 Valores do dicionário:")
print(pessoa.values())
# Mostrando os pares chave-valor com items()
print("\n🧩 Itens (chave, valor) do dicionário:")
print(pessoa.items())
# Percorrendo os itens do dicionário com for
print("\n📝 Percorrendo chave e valor com for:")
for chave, valor in pessoa.items():
    print(f"{chave}: {valor}")
```

```
🔑 Chaves do dicionário:
dict_keys(['nome', 'idade', 'cidade'])

📋 Valores do dicionário:
dict_values(['João', 29, 'São Paulo'])

🧩 Itens (chave, valor) do dicionário:
dict_items([('nome', 'João'), ('idade', 29), ('cidade', 'São Paulo')])

📝 Percorrendo chave e valor com for:
nome: João
idade: 29
cidade: São Paulo
```

Hash table nativa: dict

Python implementa tabelas hash por meio do tipo dict (chave → valor).

```
alunos = {
    "Ana": 9.5,
    "Carlos": 8.2,
    "Maria": 10
}
print(alunos["Maria"])   # 10
alunos["Pedro"] = 7.8    # inserção
```

Acesso direto

A busca e a inserção usam a chave diretamente — sem percorrer a estrutura inteira.

`acesso médio O(1)`

Busca e conjuntos (set)

```
if "Ana" in alunos:
    print(alunos["Ana"])
```

Verificar existência de uma chave é imediato.

```
numeros = {10, 20, 30}
print(20 in numeros)   # True
```

O tipo set também usa tabelas hash internamente — por isso a verificação de pertencimento é tão rápida.

Complexidade

O custo das operações em uma Tabela Hash

Complexidade das operações

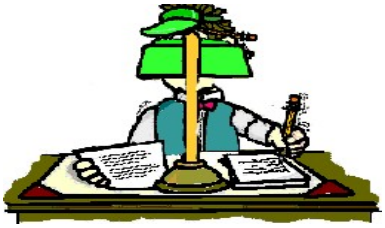
Operação	Caso médio	Pior caso
Inserção	O(1)	O(n)
Busca	O(1)	O(n)
Remoção	O(1)	O(n)

Na média as operações são O(1); o pior caso O(n) ocorre com muitas colisões concentradas.

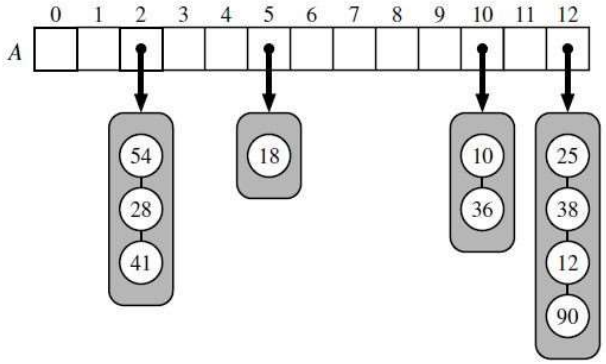
Comparação entre estruturas

Estrutura	Organização	Busca média
Vetor	Sequencial	O(n)
Lista Encadeada	Sequencial	O(n)
BST Balanceada	Hierárquica	O(log n)
AVL	Hierárquica	O(log n)
Tabela Hash	Dispersão por índice	O(1)

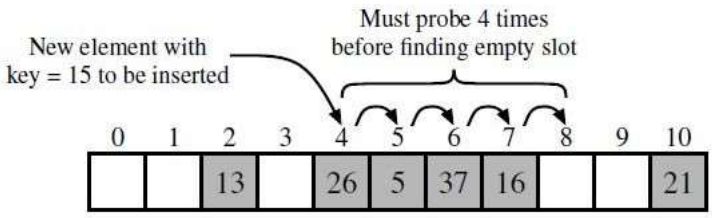
Exercícios



1) Qual método de tratamento de colisões é representado na figura abaixo? Explique.



2) Qual método de tratamento de colisões é representado na figura abaixo? Explique.



3) Implemente no Python uma tabela de hash utilizando o método da divisão e uma lista de tamanho 10. Obs: Não é necessário realizar o tratamento de colisões.

4) Teste a função hash do Python para os seguintes valores: "Python", 33, 34, 3333333333333333333, "python". Algum dos resultados da função se repetiu? Qual característica da função hash esse resultado exemplifica.

5) Teste dois valores em sua implementação do exercício 3 que vão gerar uma colisão

6) Crie um dicionário com os alunos e notas:

"Maria": 8.5, "João": 7.0, "Pedro": 6.0

a) Adicione "Ana": 6.5.

b) Acesse e imprima a nota de "João".

c) Liste todos os alunos e suas respectivas notas.

7) Crie uma agenda telefônica utilizando um dicionário. A agenda deve possuir as seguintes funcionalidades:

- Adicionar contato
- Buscar contato
- Remover contato
- Listar contatos
- Sair

8) Solicite uma palavra ao usuário e conte quantas vezes cada letra aparece. O resultado deve ser salvo em um dicionário.
Exemplo:

Entrada:

"banana"

Saída:

```
{
     'b': 1,
     'a': 3,
     'n': 2
}
```

1 **Construa a tabela**

Tamanho 10, com h(x) = x mod 10, para: 15, 22, 35, 42, 55, 63, 74.

2 **Implemente em Python**

Funções inserir, buscar e imprimir a tabela.

3 **Desafio**

Refaça usando encadeamento separado e sondagem linear e compare os resultados.

Problema

Implemente uma classe HashTable em Python com tratamento de colisões.

```
class HashTable:
    def inserir(self, chave, valor): ...
    def buscar(self, chave): ...
    def remover(self, chave): ...
    def imprimir(self): ...
```

Requisitos

- Inserção de pares chave–valor
- Busca por chave
- Remoção de elementos
- Colisões por encadeamento separado
- Exibição do conteúdo da tabela

Desafio extra: faça uma 2ª versão com endereçamento aberto e compare colisões, tempos e taxa de ocupação.

Contato

eferlin@live.com

(BLOG) professorferlin.blogspot.com

(SITE) professorferlin.com.br

(YOUTUBE) ProfEdsonPedroFerlin