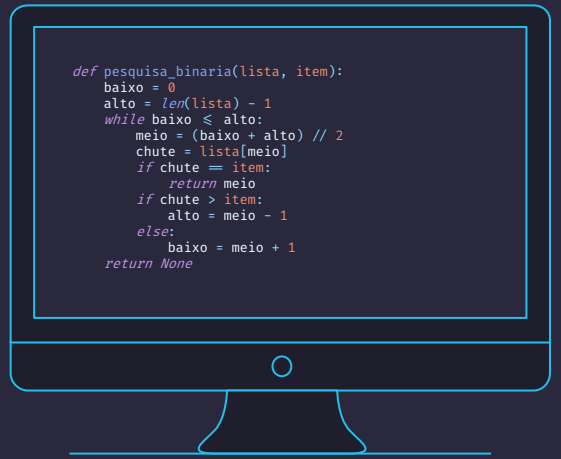
# Tabelas Hash

Professor Roberto Cândido



# Hashing

Hashing é uma técnica que transforma uma entrada (dados de qualquer tamanho) em um valor fixo, geralmente um número ou uma string curta. Essa transformação é feita por uma função hash, que gera um valor chamado hash code ou digest.

Hashing é usado para:

Busca rápida: Indexação eficiente em bancos de dados e estruturas de dados como tabelas hash.
Segurança: Armazenamento de senhas (com criptografia).

Integridade de dados: Verificação de arquivos e mensagens (exemplo: checksums).

# Tabelas Hash

Uma tabela hash (ou hash table) é uma estrutura de dados que usa hashing para armazenar e recuperar valores rapidamente. Funciona assim:

Cálculo do hash: Uma função hash gera um índice baseado na chave.

Armazenamento: O valor é armazenado nesse índice na tabela.

Busca eficiente: Para encontrar um valor, basta calcular o hash da chave e acessar diretamente a posição correspondente.

## Exemplo em Python

```python
# Criando uma tabela hash com um dicionário Python
tabela_hash = {}

# Adicionando valores
tabela_hash["chave1"] = "valor1"
tabela_hash["chave2"] = "valor2"

# Acessando valores
print(tabela_hash["chave1"])  # Saída: valor1
```

## Como o Hashing é Feito?

O hashing é realizado por uma função hash, que recebe uma entrada (string, número, arquivo, etc.) e gera um valor fixo, chamado hash code ou digest.

Uma função hash deve ser:

Determinística: A mesma entrada sempre gera o mesmo hash.

Rápida: Deve calcular o hash rapidamente.
Uniforme: Distribui os valores de forma uniforme para evitar colisões.

Irreversível (em alguns casos, como criptografia): Não é possível obter a entrada original a partir do hash.

## Exemplo de SHA-256 em Python

Você pode calcular o hash de uma string usando o módulo hashlib:

```
import hashlib

texto = "Olá, mundo!"
hash_sha256 = hashlib.sha256(texto.encode()).hexdigest()

print(hash_sha256)
```

SHA-256

SHA-256 (Secure Hash Algorithm 256-bit) é uma função de hash criptográfica que transforma qualquer entrada de dados em um valor fixo de 256 bits (32 bytes). Faz parte da família SHA-2, desenvolvida pela NSA (National Security Agency) e publicada pelo NIST (National Institute of Standards and Technology).

Usos do SHA-256

Segurança de Senhas: Armazena senhas de forma segura (geralmente com salt para maior proteção).

Blockchain e Criptomoedas: Bitcoin e outras criptos usam SHA-256 para proteger transações.

Verificação de Integridade: Garante que arquivos e mensagens não foram alterados.

Assinaturas Digitais e Certificados SSL/TLS

## Salt

Em criptografia, um salt é um valor aleatório adicionado a um dado (geralmente uma senha) antes de aplicar uma função de hash. O objetivo do salt é tornar os hashes mais seguros, especialmente contra ataques de rainbow tables e hashes duplicados.

Sem um salt, senhas iguais geram hashes idênticos, o que facilita ataques. Com um salt único para cada senha, mesmo senhas idênticas terão hashes diferentes.

```python
import hashlib

senha = "123456"
senha_2 = "123456"
hash1 = hashlib.sha256(senha.encode()).hexdigest()
hash2 = hashlib.sha256(senha_2.encode()).hexdigest()

print(hash1 == hash2)  # True (hashes iguais)
```

```python
import hashlib
import os

senha = "123456"
senha_2 = "123456"

salt = os.urandom(16)  # Gera 16 bytes aleatórios
salt_2 = os.urandom(16)  # Gera 16 bytes aleatórios

# Combina o salt com a senha antes do hash
hash1 = hashlib.sha256(salt + senha.encode()).hexdigest()
hash2 = hashlib.sha256(salt_2 + senha_2.encode()).hexdigest()

print(hash1)
print(hash2)
```

Em tabelas hash, o hashing não é diretamente o índice, mas sim um valor que precisa ser convertido em um índice válido.

Por quê?

Uma função hash pode gerar valores muito grandes, então precisamos reduzir esse valor para caber no tamanho da tabela.
Isso é feito com uma operação como módulo (%), para garantir que o índice esteja dentro dos limites da tabela.

O que acontece se duas chaves diferentes gerarem o mesmo índice?

Quando duas chaves diferentes geram o mesmo índice em uma tabela hash, ocorre um problema chamado colisão. Como a tabela hash deve armazenar cada valor em um único índice, precisamos de estratégias para resolver essa colisão.

# Métodos para Resolver Colisões

Existem várias estratégias para lidar com colisões, mas as mais comuns são:

1 Encadeamento Separado (Chaining)

Nesse método, cada índice da tabela contém uma lista (ou estrutura similar) onde armazenamos todas as chaves que colidirem no mesmo índice.

Como funciona?

Se duas chaves tiverem o mesmo índice, armazenamos ambas em uma lista encadeada nesse índice.
Na busca, percorremos a lista para encontrar a chave correta.

Endereçamento Aberto (Open Addressing)

Nesse método, quando ocorre uma colisão, procuramos outro espaço disponível na tabela hash.

Estratégias comuns:

Linear Probing: Se o índice estiver ocupado, procuramos o próximo índice disponível (indice + 1).

Quadratic Probing: Procuramos posições cada vez mais distantes (indice + 1^2, indice + 2^2, etc.).

Double Hashing: Aplicamos uma segunda função hash para encontrar um novo índice.

```python
import hashlib

def gerar_indice_hash(valor, tamanho_tabela):
    hash_obj = hashlib.sha256(valor.encode())
    hash_int = int(hash_obj.hexdigest(), 16)
    return hash_int % tamanho_tabela

valores = ["banana", "cenoura", "uva", "laranja", "batata"]
tamanho_tabela = 10

for valor in valores:
    indice = gerar_indice_hash(valor, tamanho_tabela)
    print(f"'{valor}' → Índice: {indice}")
```

# JWT (JSON Web Token)

JWT é um formato compacto e seguro de token usado para transmitir informações entre duas partes — normalmente, entre cliente e servidor — de forma autenticada e opcionalmente cifrada.

## 📦 Estrutura de um JWT

Um JWT é composto por 3 partes, separadas por ponto:

xxxxx.yyyyy.zzzzz

1 - Header (cabeçalho):
Informa o algoritmo de assinatura (ex: HS256) e o tipo do token (JWT).

2 - Payload (dados):
Contém as informações (claims) como sub (usuário), email, exp (expiração), etc.

3 - Signature (assinatura):
Protege o token contra alterações usando uma chave secreta (ou chave pública/privada).