# Tabela Hash: Análise dos Algoritmos de Busca, Inserção e Remoção

# Tabela Hash: Análise dos Algoritmos de Busca, Inserção e Remoção

## 1. Introdução

Uma \*\*tabela hash\*\* é uma estrutura de dados que associa \*\*chaves\*\* a \*\*valores\*\* e permite a realização eficiente de operações como busca, inserção e remoção. Seu diferencial está na capacidade de acessar elementos quase instantaneamente, graças ao uso de uma \*\*função hash\*\*, que mapeia a chave para um índice de um array.

Essa abordagem é amplamente utilizada em aplicações onde o tempo de acesso é crítico, como em bancos de dados, caches e sistemas de indexação.

### Conceitos Fundamentais

- \*\*Função Hash\*\*: Função que converte uma chave em um índice de array.

- \*\*Colisão\*\*: Quando duas chaves diferentes resultam no mesmo índice hash.

- \*\*Resolução de colisão\*\*: Estratégia para lidar com colisões (endereçamento aberto ou encadeamento).

## 2. Funcionamento Geral de uma Tabela Hash

### Função Hash

A função hash recebe uma chave como entrada e retorna um número inteiro que representa a posição no array onde o valor correspondente será armazenado.

Exemplo simples em Python:

def hash\_function(key, table\_size):

return hash(key) % table\_size

### Resolução de Colisões

Existem duas abordagens principais:

- \*\*Encadeamento (Chaining)\*\*: Cada posição do array contém uma lista encadeada onde múltiplos elementos com o mesmo índice podem ser armazenados.

- \*\*Endereçamento Aberto (Open Addressing)\*\*: Quando ocorre colisão, procura-se a próxima posição livre no array segundo uma estratégia (linear, quadrática, dupla hashing).

## 3. Análise do Algoritmo de Busca

### Funcionamento

- \*\*Encadeamento\*\*: Localiza-se o índice via função hash e percorre-se a lista encadeada.

- \*\*Endereçamento Aberto\*\*: Procura-se sequencialmente (ou de forma mais complexa) até encontrar a chave ou uma célula vazia.

### Complexidade

- \*\*Melhor caso\*\*: O(1) — chave é encontrada na primeira tentativa.

- \*\*Pior caso\*\*: O(n) — todas as posições são examinadas (alta taxa de colisão).

- \*\*Caso médio\*\*: O(1) — com boa função hash e baixa taxa de ocupação (load factor).

## 4. Análise do Algoritmo de Inserção

### Funcionamento

- Aplica-se a função hash à chave.

- Se não houver colisão, insere-se diretamente.

- Se houver, resolve-se conforme a técnica adotada.

Exemplo de inserção com encadeamento em Python:

def insert(table, key, value):

index = hash\_function(key, len(table))

table[index].append((key, value))

### Complexidade

- \*\*Melhor caso\*\*: O(1)

- \*\*Pior caso\*\*: O(n) — várias colisões

- \*\*Caso médio\*\*: O(1)

## 5. Análise do Algoritmo de Remoção

### Funcionamento

- \*\*Encadeamento\*\*: A chave é localizada na lista encadeada e removida.

- \*\*Endereçamento Aberto\*\*: A chave é localizada via sondagem; deve-se garantir que a busca futura não seja interrompida pela ausência do item removido.

### Cuidados com Endereçamento Aberto

Deve-se usar marcadores especiais ("tombstones") para indicar que um item foi removido, evitando quebrar a cadeia de sondagem.

### Complexidade

- \*\*Melhor caso\*\*: O(1)

- \*\*Pior caso\*\*: O(n)

- \*\*Caso médio\*\*: O(1)

## 6. Conclusão

As tabelas hash são estruturas extremamente eficientes para operações de busca, inserção e remoção, com complexidade média constante. Seu desempenho, no entanto, depende da qualidade da função hash e da estratégia de resolução de colisão adotada.

Aplicações incluem dicionários, caches, compiladores, bancos de dados e muito mais.

\*Este material foi desenvolvido para fins educacionais na disciplina de Análise de Algoritmos.\*