

# Hashing

Análise de Algoritmos – Ciência da Computação



Prof. Daniel Saad Nogueira  
Nunes

Instituto Federal de Brasília,  
Câmpus Taguatinga



# Sumário

---

## 1 Introdução



# Introdução

---

- Em muitas aplicações, precisamos de um conjunto dinâmico, que suporte:
  - ▶ INSERT: inserção de um elemento pela sua chave.
  - ▶ DELETE: remoção de um elemento pela sua chave.
  - ▶ SEARCH: busca de um elemento pela sua chave.



# Estruturas de Dados

---

- Muitas estruturas suportam essas operações eficientemente.
  - ▶ Árvores Binárias de Pesquisa.
  - ▶ Árvores Rubro-Negras.
  - ▶ ...
- Geralmente, o custo de cada operação é  $\Theta(\log n)$ .



# Estruturas de Dados

---

- Muitas estruturas suportam essas operações eficientemente.
  - ▶ Árvores Binárias de Pesquisa.
  - ▶ Árvores Rubro-Negras.
  - ▶ ...
- Geralmente, o custo de cada operação é  $\Theta(\log n)$ .
- Será que podemos responder essas operações em  $\Theta(1)$ ?



# Estrutura de Dados

---

## Hashing

- Possibilita a resposta dessas consultas em  $O(1)$ . No **caso médio**.
- No pior caso, as consultas levam  $\Theta(n)$ .
- Com hashing perfeito, conseguimos  $O(1)$  até no **pior caso**!
- Alternativa eficiente na prática.
- Utilizado em Sistemas Operacionais para lookup rápido de tabelas.



# Motivação

---

- Antes de introduzir o conceito de hashing, vamos dar uma pequena motivação ao mostrar problemas existentes com outros métodos.
- Mostraremos como a técnica de Hashing soluciona estes problemas.



# Motivação

---

## Tabelas de Endereçamento Direto

- O uso de tabelas de endereçamento diretos proporciona consultas de inserção, remoção e busca em tempo  $O(1)$  no pior caso.
- Seja  $U = \{0, 1, 2, \dots, m - 1\}$  chaves do conjunto universo.
- **Premissa:** dois elementos distintos tem chaves distintas.
- Para cada chave, existe apenas uma entrada na tabela de endereçamento direto.
- Se o elemento não existe, então o conteúdo da tabela é preenchido com uma constante **NULL**.





## Tabelas de Endereçamento Direto

---

- As operações das operações usando tabelas de endereçamento direto são triviais de serem implementadas.



## Operações em Tabelas de Endereçamento Direto

---

---

### Algorithm 1: DAT-SEARCH

---

**Input:**  $T, k$

**Output:**  $x, \quad x.key = k$

1 **return**  $T[k]$

---

---

### Algorithm 2: DAT-INSERT

---

**Input:**  $T, x$

1  $T[x.key] \leftarrow x$

---

---

### Algorithm 3: DAT-DELETE

---

**Input:**  $T, x$

1  $T[x.key] \leftarrow \text{NULL}$

---



## Tabelas de Endereçamento Direto

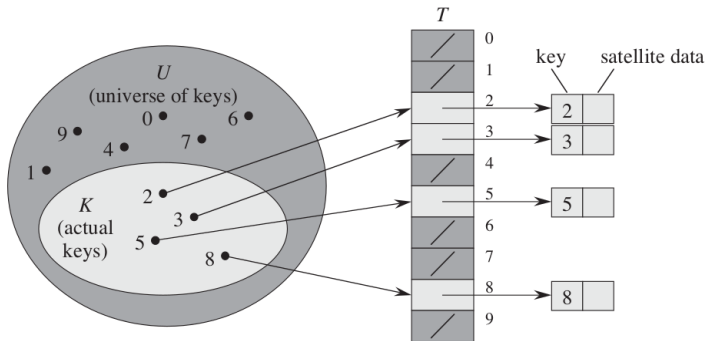


Figura: Tabelas de Endereçamento Direto.



## Tabelas de Endereçamento Direto

---

- Qual o problema de utiliza esta técnica?
- Quantas posições precisamos na nossa tabela?



## Tabelas de Endereçamento Direto

---

- Qual o problema de utiliza esta técnica?
- Quantas posições precisamos na nossa tabela?
- $|U|$ , o tamanho do espaço de chaves.



## Tabelas de Endereçamento Direto

---

- Qual o problema de utiliza esta técnica?
- Quantas posições precisamos na nossa tabela?
- $|U|$ , o tamanho do espaço de chaves.
- O que acontece se as chaves forem números de 64 bits?



## Tabelas de Endereçamento Direto

---

- Qual o problema de utiliza esta técnica?
- Quantas posições precisamos na nossa tabela?
- $|U|$ , o tamanho do espaço de chaves.
- O que acontece se as chaves forem números de 64 bits?
- $|U| = 2^{64}$ .



## Tabelas de Endereçamento Direto

---

- Qual o problema de utiliza esta técnica?
- Quantas posições precisamos na nossa tabela?
- $|U|$ , o tamanho do espaço de chaves.
- O que acontece se as chaves forem números de 64 bits?
- $|U| = 2^{64}$ .
- Além disso, o número de elementos inseridos pode ser muito menor que o espaço de chaves.





## Tabelas de Endereçamento Direto

---

- Qual o problema de utiliza esta técnica?
- Quantas posições precisamos na nossa tabela?
- $|U|$ , o tamanho do espaço de chaves.
- O que acontece se as chaves forem números de 64 bits?
- $|U| = 2^{64}$ .
- Além disso, o número de elementos inseridos pode ser muito menor que o espaço de chaves.
- **Problema:** muito espaço utilizado.



## Tabelas de Endereçamento Direto

---

- Qual o problema de utiliza esta técnica?
- Quantas posições precisamos na nossa tabela?
- $|U|$ , o tamanho do espaço de chaves.
- O que acontece se as chaves forem números de 64 bits?
- $|U| = 2^{64}$ .
- Além disso, o número de elementos inseridos pode ser muito menor que o espaço de chaves.
- **Problema:** muito espaço utilizado.
- **Solução:** hashing!



# Sumário

---

## 2 Hashing



# Hashing

---

## Ideia do Hashing

- Nas tabelas de endereçamento direto, o elemento de chave  $k$  é armazenado no  $k$ -ésimo slot.
- Utilizando a técnica de **hashing**, aplicamos uma função  $h : U \rightarrow \{0, 1, \dots, m - 1\}$ , de modo que  $h$  mapeie um elemento  $x$  e uma entrada da tabela de **hash**.
- **Problema:** nem sempre  $h$  é injetora.



# Hashing

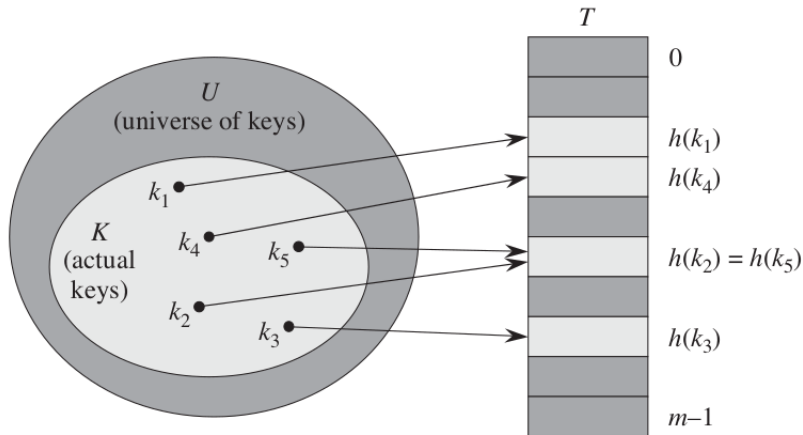


Figura: Técnica de hashing.



# Hashing

---

## Vantagens e Desvantagens

- Obviamente, a técnica de hashing tem uma vantagem enorme de espaço em relação à técnica de endereçamento direto.
- Em vez de utilizar  $|U|$ , utilizamos apenas  $m$  entradas da tabela hash.
- Problema: dois ou mais elementos podem ser mapeados na mesma posição da tabela.



# Hashing

---

## Vantagens e Desvantagens

- Obviamente, a técnica de hashing tem uma vantagem enorme de espaço em relação à técnica de endereçamento direto.
- Em vez de utilizar  $|U|$ , utilizamos apenas  $m$  entradas da tabela hash.
- Problema: dois ou mais elementos podem ser mapeados na mesma posição da tabela.
- Princípio da casa dos pombos!



# Hashing

---

## Vantagens e Desvantagens

- Obviamente, a técnica de hashing tem uma vantagem enorme de espaço em relação à técnica de endereçamento direto.
- Em vez de utilizar  $|U|$ , utilizamos apenas  $m$  entradas da tabela hash.
- Problema: dois ou mais elementos podem ser mapeados na mesma posição da tabela.
- Princípio da casa dos pombos!
- Chamamos isso de **colisão**.





# Hashing

---

## Tratamento de Colisões

- Obviamente, haverá colisões, uma vez que  $m < |U|$ .
- Temos que escolher  $h$  de maneira apropriada, de maneira que espalhe os elementos de maneira uniforme!
- $h$  tem que “parecer aleatória”.
- Além disso, temos que conseguir distinguir um elemento do outro, temos que tratar a colisão.



# Hashing

---

## Tratamento de Colisões

- Obviamente, haverá colisões, uma vez que  $m < |U|$ .
- Temos que escolher  $h$  de maneira apropriada, de maneira que espalhe os elementos de maneira uniforme!
- $h$  tem que “parecer aleatória”.
- Além disso, temos que conseguir distinguir um elemento do outro, temos que tratar a colisão.
- Agora sim, Luiz, alguns princípios da criptografia.



# Hashing

---

## Tratamento de Colisões

- Existem diversas maneiras de tratar colisão.
- Estudaremos o método de encadeamento.



# Encadeamento

---

## Encadeamento

- No encadeamento, colocamos todos os elementos mapeados em uma mesma entrada em uma **lista ligada**.
- Cada slot  $i$  tem um ponteiro para a cabeça da lista.



# Encadeamento

---

## Encadeamento

- No encadeamento, colocamos todos os elementos mapeados em uma mesma entrada em uma **lista ligada**.
- Cada slot  $i$  tem um ponteiro para a cabeça da lista.



## Operações em Hashing com Encadeamento

---

---

### Algorithm 4: CHAINED-HASH-SEARCH

---

**Input:**  $T, k$

**Output:**  $x, \quad x.key = k$

- 1 Procure o elemento  $x$  na lista apontada por  $h(x)$
  - 2 Retorne **NULL** caso  $x$  não ocorra na lista, e  $x$  caso contrário.
- 

---

### Algorithm 5: CHAINED-HASH-INSERT

---

**Input:**  $T, x$

- 1 Insira  $x$  no final da lista apontada por  $h(x)$
- 

---

### Algorithm 6: CHAINED-HASH-DELETE

---

**Input:**  $T, x$

- 1 Ache  $x$  na lista apontada por  $h(x)$
  - 2 Remova  $x$  da lista
-



# Operações em Hashing com Encadeamento

---

## Análise

- Inserção:  $O(1)$  de pior caso.
- Remoção:  $O(1)$  se as listas forem duplamente ligadas.
- Busca: tempo de pior caso proporcional ao tamanho da lista.



# Operações em Hashing de Encadeamento

---

## Análise

- O pior caso da técnica de hashing com encadeamento é muito ruim.  $\Theta(n)$ .
- Mas se escolhermos uma função de hashing adequada, o caso médio é bastante atraente na prática.





# Operações em Hashing de Encadeamento

---

## Definição (Fator de Carga)

- O fator de carga, definido como  $\alpha$ , é dado por  $n/m$ . Sendo  $n$  o número de elementos e  $m$  o tamanho da tabela hashing.



# Operações em Hashing de Encadeamento

---

## Análise

- Em uma função de hashing adequada, o tamanho de uma lista arbitrária é bem próxima de  $\alpha$ .
- Pior caso de busca:  $O(1 + \alpha)$ , desde que a função de hash espalhe de maneira uniforme.
- Quanto mais espaço, menos colisões.
- Quanto mais espaço, mais espaço!



# Sumário

---

- 2 Hashing
  - Funções de Hashing



# Funções de Hashing

---

- Vimos que o sucesso da técnica de hash se baseia na escolha da função de hash.
- A função tem que parecer uniforme.
- Caso contrário, haverão muitas colisões, e portanto, mais tempo será gasto procurando elementos.



# Funções de Hashing

---

## Projetando funções de Hashing

- Para projetar uma função de hashing adequada, temos que analisar uma série de fatores.
  - ▶ A distribuição das chaves é conhecida?
  - ▶ Queremos que valores próximos ocupem slots próximos?



# Funções de Hashing

---

## Projetando funções de Hashing

- É interessante interpretar as chaves como números naturais.
- Se as chaves não forem números naturais (ex: strings), é interessante interpretá-las como números naturais.
- Na prática: usamos o índice da tabela ascii.



# Funções de Hashing

---

## Método da Divisão

- O método da divisão mapeia uma chave  $k$  em um dos  $m$  slots ao pegar o resto da divisão de  $k$  por  $m$ .

$$h(k) = k \mod m$$

- Exemplo, se  $m = 12$  e  $k = 100$ , então  $h(k) = 4$ .
- Muito rápida na prática, precisamos apenas de uma instrução de divisão e um acesso à memória.



# Funções de Hashing

---

## Método da Divisão

- Temos que tentar evitar alguns valores de  $m$ .
- Por exemplo, se  $m$  for uma potência de 2, i.e,  $m = 2^p$ , então  $h(k)$  são os  $p$  bits menos significativos.
- O padrão dos bits menos significativos pode ser altamente repetitivo dependendo da aplicação.





# Funções de Hashing

---

## Método da Divisão

- Geralmente, um primo não muito próximo de uma potência de 2 é uma boa escolha.
- Por exemplo, para  $n = 2000$ ,  $m = 701$  é uma boa escolha:

$$h(k) = k \mod 701$$



# Funções de Hashing

---

## Método da Multiplicação

- O método da multiplicação para criar funções de hash opera em dois passos.
  - 1 Primeiro multiplicamos a chave  $k$  por uma constante  $A$ ,  $0 < A < 1$  e extraímos a parte fracionária de  $k$ .
  - 2 Depois, multiplicamos este valor por  $m$  e pegamos o chão do resultado.

$$h(k) = \lfloor m(k \cdot A \bmod 1) \rfloor$$



# Funções de Hashing

---

## Método da Multiplicação

- Uma vantagem deste método é que o valor de  $m$  não é crítico.
- Geralmente escolhemos  $m = 2^p$  para algum  $p$ .
  - ▶ Mais fácil de implementar.
- Seja  $w$  o tamanho da palavra do computador e suponha que as chaves caibam em uma palavra. Podemos restringir  $A$  como um número da forma  $s/2^w$ , onde  $0 < s < 2^w$ .
- Apesar deste método funcionar para qualquer valor de  $A$ , alguns valores se comportam melhor que outros.
- Sugestão do D. Knuth:  $A \approx (\sqrt{5} - 1)/2 = 0.6180339887\dots$