

Aula 10: Tabela Hash



DCC405-Estrutura de Dados II
Prof. Me. Acauan C. Ribeiro

Roteiro da Aula

- **Contextualização**



- **Conceitos principais**

- **Função Hash**

- Escolha da Função
- Métodos para mapeamento de compressão

- **Evitando e tratando colisões**

- Hashing universal
- Fator de Carga
- Endereçamento aberto
- Hashing duplo
- Encadeamento

- **Análise da Busca usando Hashing**

Busca, inserção e remoção em vetores

- Métodos de busca estudados até agora: [Comparação de chaves](#)

Busca, inserção e remoção em vetores

- Métodos de busca estudados até agora: [Comparação de chaves](#)
- Tiramos proveito da ordenação

Busca, inserção e remoção em vetores

- Métodos de busca estudados até agora: Comparação de chaves
- Tiramos proveito da ordenação

Custo em vetores

Ordenação: $O(n \log_2 n)$ ou $O(n)$ em casos especiais

Busca: $O(\log_2 n)$ ou $O(\log_2(\log_2 n))$ em casos especiais (Ex.: Busca Interpolada)

Custo após realizar outras operações

Inserção: ?

Remoção: ?

Conceito Primordial: Estrutura Tabela → Mapa | Dicionário

Utiliza-se o conceito de **Chave → Valor**

Operações

- put(chave, valor)
- get(chave)
- remove(chave)

UF (chave)	Capital (valor)
“MT”	“Cuiabá”
“RR”	“Boa Vista”
“SP”	“São Paulo”
“PB”	

Conceito Primordial: Estrutura Tabela → Mapa | Dicionário

Utiliza-se o conceito de Chave → Valor

Operações

- put(chave, valor) : $t[ind_1] = \text{valor}$
- get(chave) : $t[ind_2] =$
- remove(chave) : $t[ind_3] = \text{null}$

UF (chave)	Capital (valor)
“MT”	“Cuiabá”
“RR”	“Boa Vista”
“SP”	“São Paulo”
“PB”	

Conceito Primordial: Estrutura Tabela → Mapa | Dicionário

UF (chave)	pos
“MT” →	0
“RR” →	1
“SP” →	2
“PB” →	3

	Capital (valor)
0	“Cuiabá”
1	“Boa Vista”
2	“São Paulo”
3	“João Pessoa”

Tabela Hash

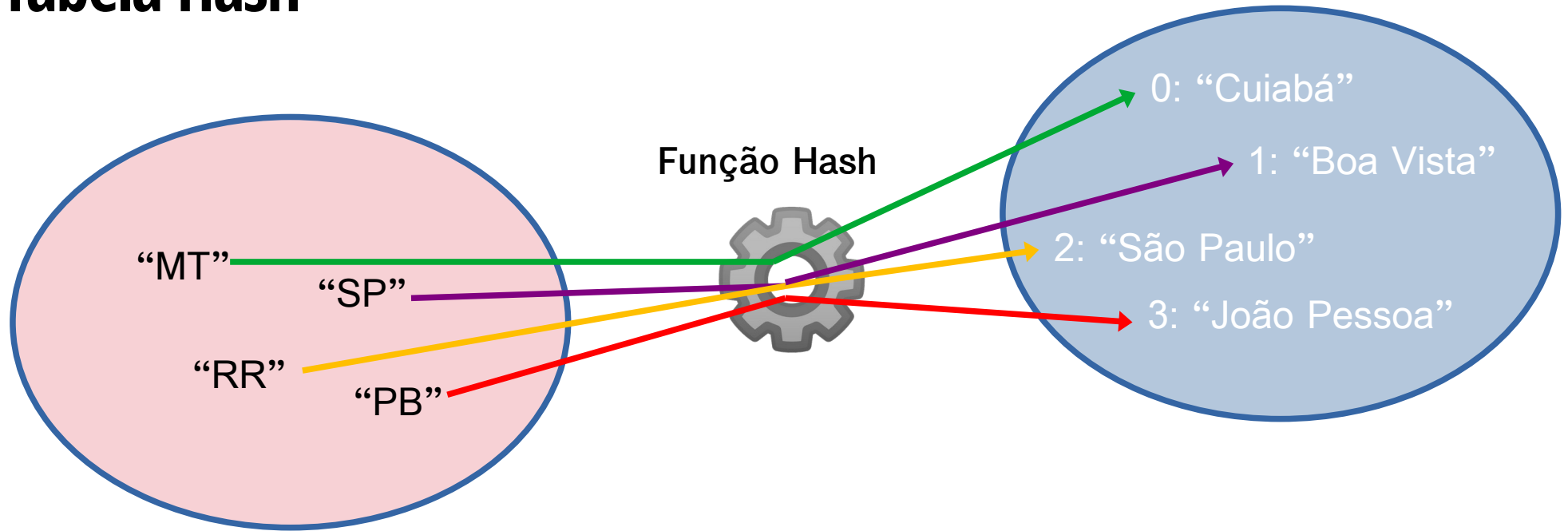


Tabela Hash

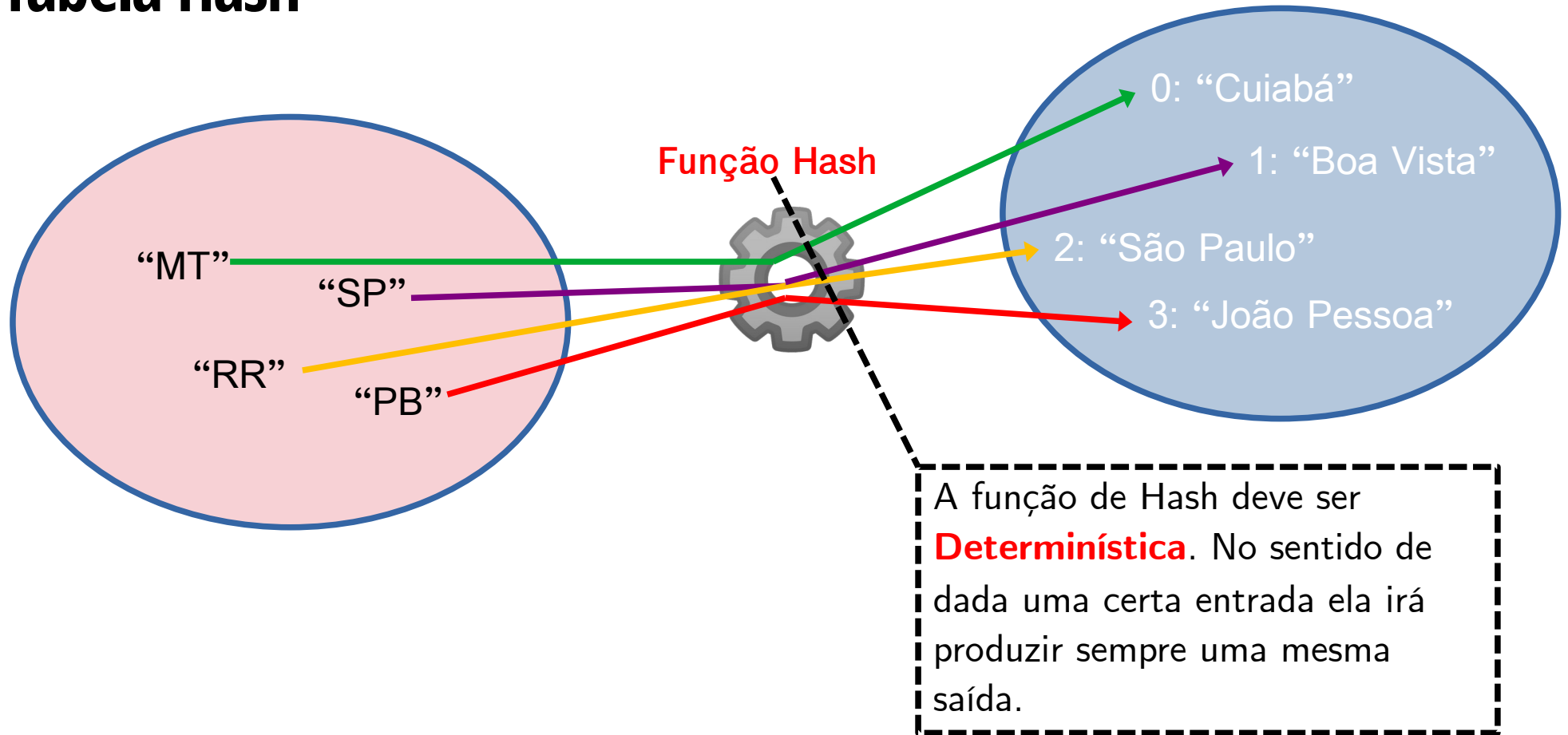


Tabela Hash – Técnica de Acesso Direto

Universo de chaves: $N = \{\text{"MS"}, \text{"PB"}, \text{"RR"}, \text{"SP"}, \text{"MT"}, \text{"AM"}, \dots\}$

Tabela: $|N| = |M| = m$

$\text{hash}(K \text{ key}) \rightarrow \{0, 1, 2, 3, \dots, m-1\}$

- **Sobrejetora:** toda chave é mapeada para um valor
- **Injetora:** cada valor é referente a apenas uma chave

Tabela Hash – Técnica de Acesso Direto

Universo de chaves: $N = \{\text{"MS"}, \text{"PB"}, \text{"RR"}, \text{"SP"}, \text{"MT"}, \text{"AM"}, \dots\}$

Tabela: $|N| = |M| = m$

$\text{hash}(K \text{ key}) \rightarrow \{0, 1, 2, 3, \dots, m-1\}$

$\text{put}(k, e) : T[\text{hash}(k)] = e$

$\text{remove}(k) : T[\text{hash}(k)] = \text{null}$

$\text{find}(k) : \text{return } T[\text{hash}(k)]$

- **Sobrejetora**: toda chave é mapeada para um valor
- **Injetora**: cada valor é referente a apenas uma chave

Tabela Hash

Universo de chaves: N

Tamanho da Tabela: M

E se $|N| \gg |M|$?

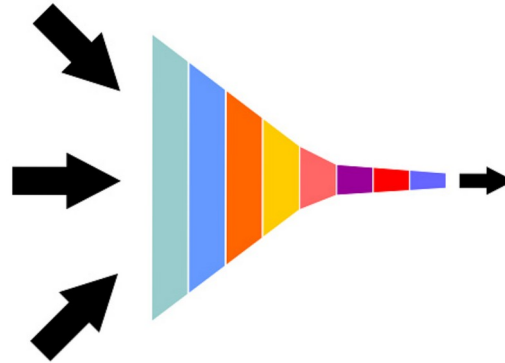


Tabela Hash – Técnica de Acesso Direto

Universo de chaves: N

Tamanho da Tabela: M

- Temos mais chaves que posições
- Permitir que mais de uma chave possa ser mapeada para um mesmo índice
- Função de Hash não é perfeita

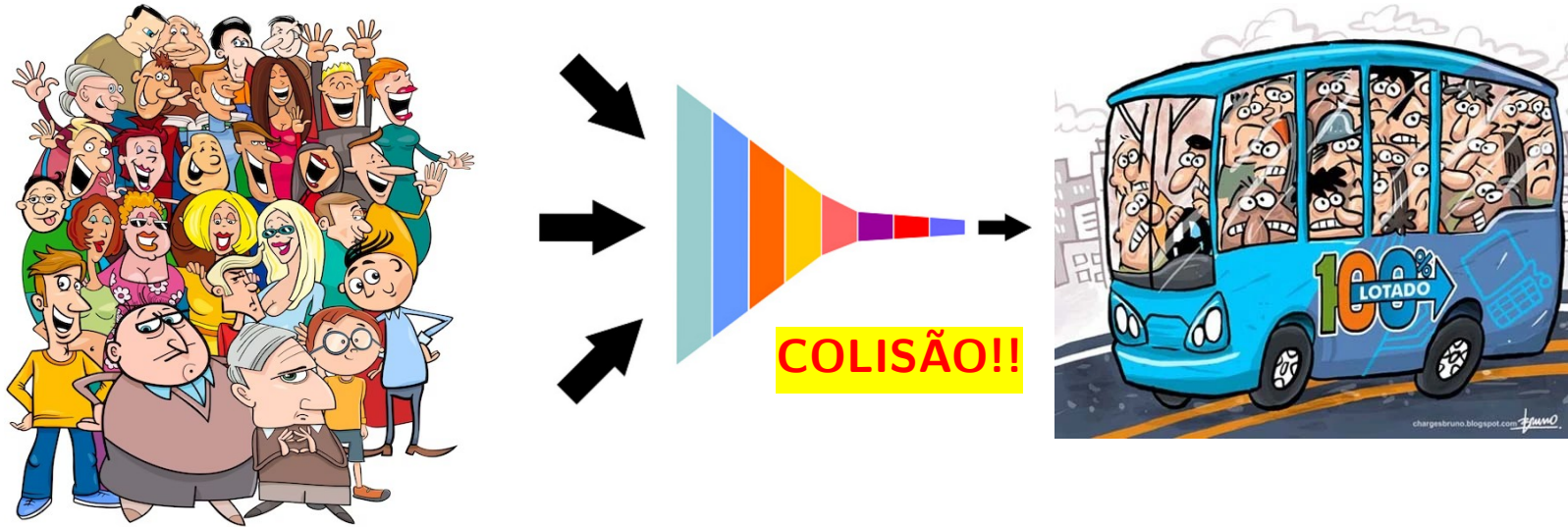


Tabela *Hash* ou de Espalhamento ou de Dispersão

- Função Hash + Tabela Hash redefine o problema de comparação de valor para um **mapa**;
- Atinge a posição da chave em tempo constante $\rightarrow O(1)$;

Tabela *Hash* ou de Espalhamento ou de Dispersão

- Função Hash + Tabela Hash redefine o problema de comparação de valor para um **mapa**;

- Atinge a posição da chave em tempo constante $\rightarrow O(1)$;

No sentido de mapear uma chave para encontrar seu lugar na tabela

Tabela *Hash* ou de Espalhamento ou de Dispersão

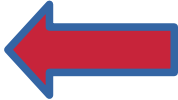
- **Função Hash + Tabela Hash** redefine o problema de comparação de valor para um **mapa**;
- Atinge a posição da chave em tempo **constante**;

No sentido de mapear uma chave para encontrar seu lugar na tabela

Em outras palavras...

Potencial para busca em $O(1)$

Roteiro da Aula

- Contextualização
- Conceitos principais 
- Função Hash
 - Escolha da Função
 - Métodos para mapeamento de compressão
- Evitando e tratando colisões
 - Hashing universal
 - Fator de Carga
 - Endereçamento aberto
 - Hashing duplo
 - Encadeamento
- Análise da Busca usando Hashing

Conceitos (1/4) – Tabela Hash

Ideia Central

Utilizar uma **função**, aplicada a informação (i.e. **chave**), que calcule o **índice** onde a informação deve (deveria) ser armazenada.

Conceitos (1/4) – Tabela Hash

Ideia Central

Utilizar uma **função**, aplicada a informação (i.e. **chave**), que calcule o **índice** onde a informação deve (deveria) ser armazenada.

- A função é chamada **função de espalhamento** (ou função hash)
- A estrutura de dados é a **tabela de espalhamento** (ou tabela hash)

Requer: projetar tabela e função adequada para aproximar $O(1)$

Conceitos (2/4) – Tabela Hash

Função Hash

Uma função de espalhamento $h(.)$ transforma uma **chave** k em um endereço-base $h(k)$ da **tabela hash**.

- **Função hash:** não há garantias que seja uma função **injetora**

Conceitos (2/4) – Tabela Hash

Função Hash

Uma função de espalhamento $h(.)$ transforma uma **chave** k em um endereço-base $h(k)$ da **tabela hash**.

- **Função hash:** não há garantias que seja uma função **injetora**
- Na prática:
 - pode existir $k \neq p$ tal que $h(k) = h(p)$
 - e se $h(k)$ estiver ocupado?
- O maior problema a lidar com Hashing: **colisão** de resultados

Colisão

Chaves

78 60 96 59 13

Função hash

$f(x) \rightarrow y$

$h(x) = \text{return } x \bmod 5$

Tabela hash

00	
01	
02	
03	
04	

Conceitos (3/4) – Função Hash

Função Hash

Formalmente, temos $h : X \rightarrow [0, \dots, m-1]$, sendo m posições possíveis

- É desejável que $\Pr_h (h(k) = i) = 1/m$,
- i.e., a função distribua de maneira uniforme as posições entre 0 e $m - 1$.

Conceitos (4/4) – Função Hash

Desejamos:

- Número baixo de colisões;
- Função facilmente computável;
- Produz distribuição uniforme.

Fator de Carga

- Fator de carga (número esperado de espaços ocupados):

$$\alpha = \frac{n}{m}$$

- Menor fator de carga \rightarrow Menor número de colisões
- Sempre pode haver uma colisão
- Previsão de tratamento de colisões
 - Endereçamento aberto
 - Encadeamento

Fator de Carga

O fator de carga de uma tabela hash não vazia é o número de itens armazenados na tabela dividido pelo tamanho da tabela. Este é o parâmetro de decisão usado quando queremos refazer o hash ou expandir as entradas da tabela de hash existentes.

O Fator de carga também nos ajuda a determinar a eficiência da função hash. Isso significa que informa se a função hash está distribuindo as chaves uniformemente ou não.

Tabela Hash (como tratar as colisões?)

- **Encadeamento Externo (Separate Chaining)**
 - Mais de um elemento no mesmo slot da tabela
- **Endereçamento aberto**
 - Um elemento por slot → busca por um slot vazio

Tabela Hash (como tratar as colisões?)

- Encadeamento Separado (Separate Chaining)

- Mais de um elemento no mesmo slot da tabela

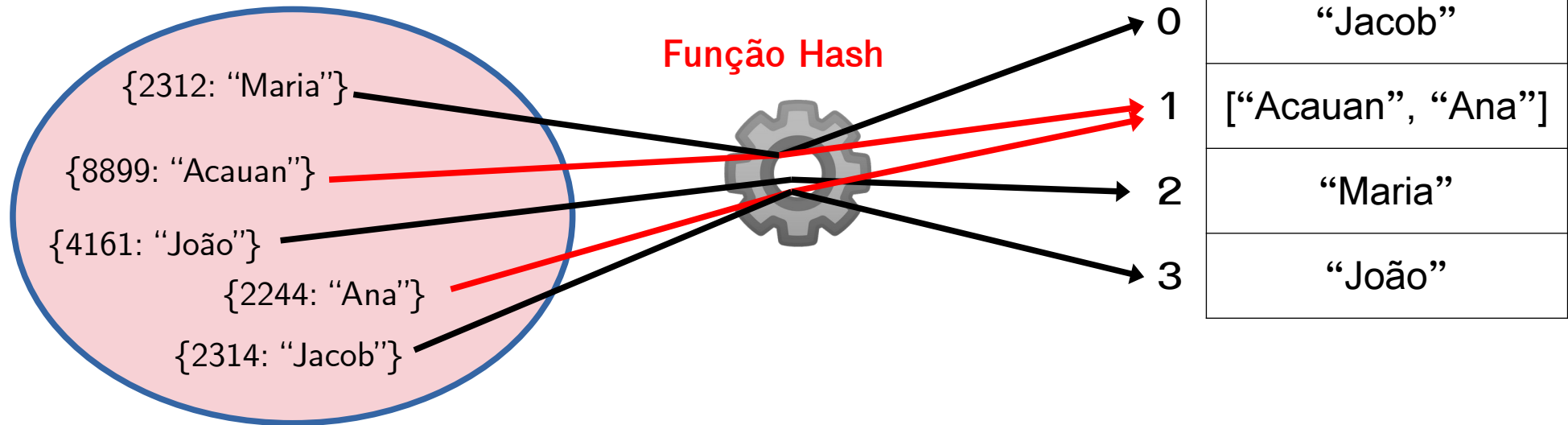
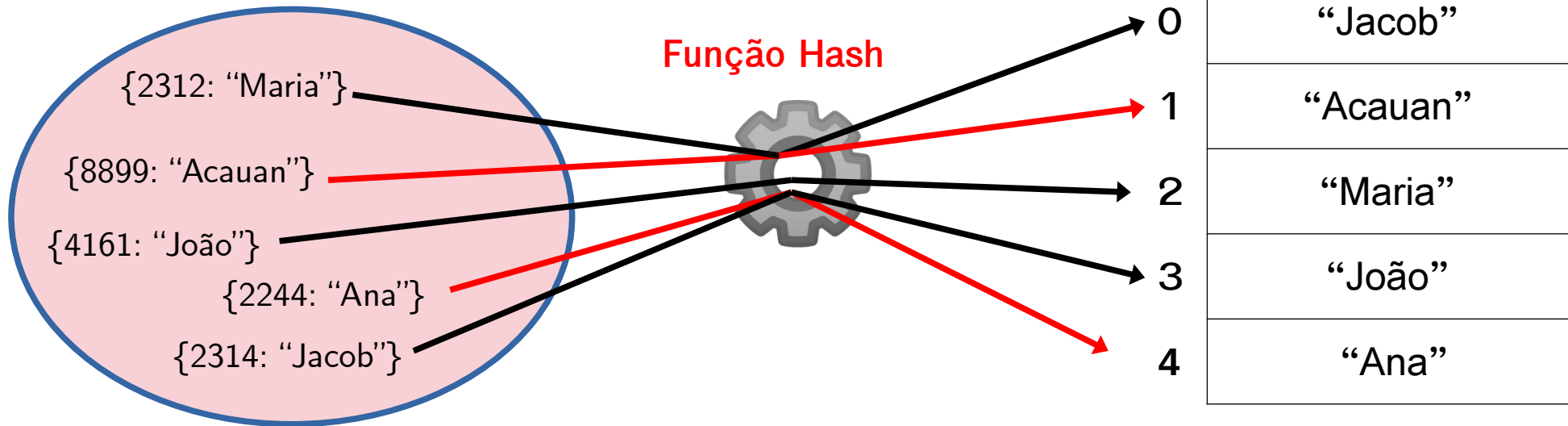


Tabela Hash (como tratar as colisões?)

- Endereçamento aberto

- Um elemento por slot → busca por um slot vazio



14.9 Collisions

Hash functions are used to map each key to a different address space, but practically it is not possible to create such a hash function and the problem is called *collision*. Collision is the condition where two records are stored in the same location.

14.10 Collision Resolution Techniques

The process of finding an alternate location is called *collision resolution*. Even though hash tables have collision problems, they are more efficient in many cases compared to all other data structures, like search trees. There are a number of collision resolution techniques, and the most popular are direct chaining and open addressing.

- **Direct Chaining:** An array of linked list application
 - Separate chaining
- **Open Addressing:** Array-based implementation
 - Linear probing (linear search)
 - Quadratic probing (nonlinear search)
 - Double hashing (use two hash functions)

Fonte: Livro – Data Structure and Algorithmic Thinking with Python: Data Structure and Algorithmic Puzzles – **Narasimha Karumanchi** – Cap 14

Tabela Hash (Encadeamento Externo)

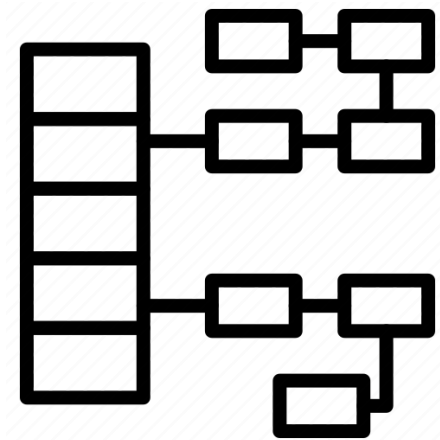


Tabela Hash (Encadeamento Externo)

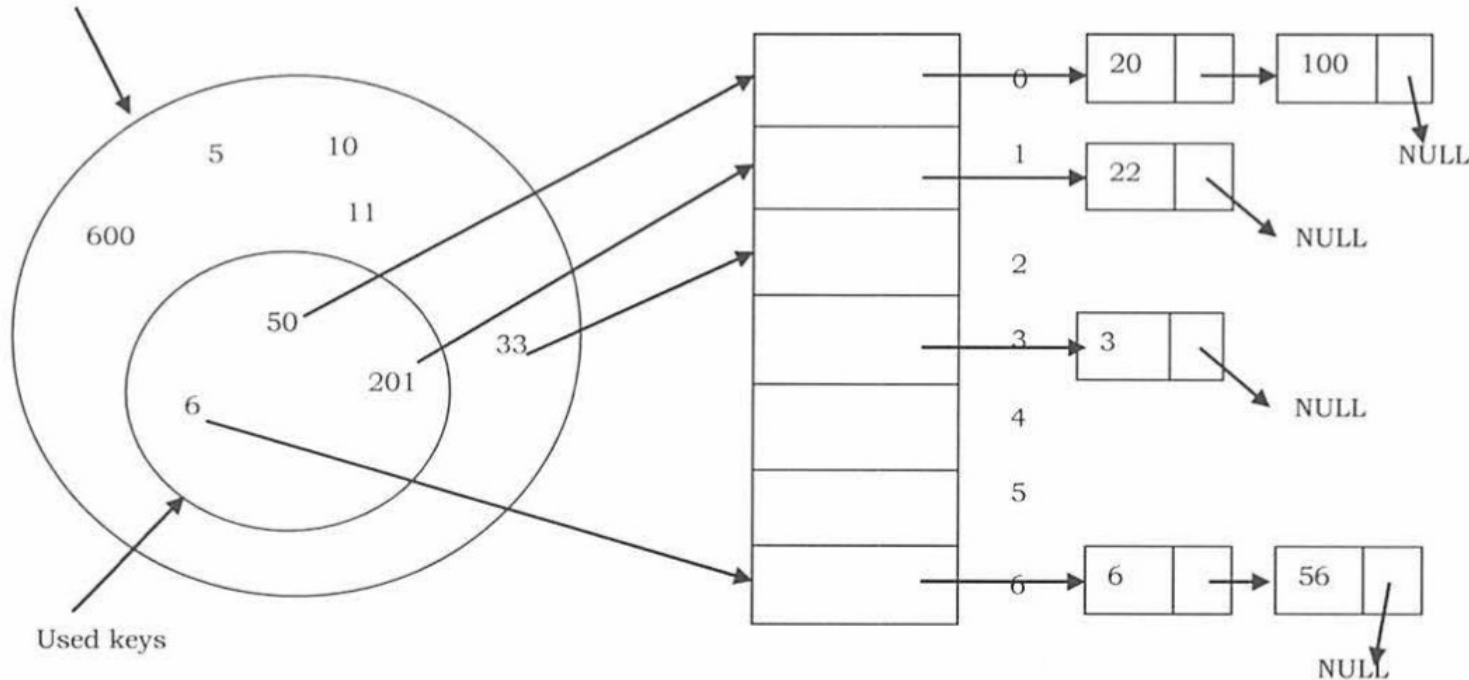
Ver uma simulação:

<https://www.cs.usfca.edu/~galles/visualization/OpenHash.html>

14.11 Separate Chaining

Collision resolution by chaining combines linked representation with hash table. When two or more records hash to the same location, these records are constituted into a singly-linked list called a *chain*.

Universe of possible keys



Fonte: Livro – Data Structure and Algorithmic Thinking with Python: Data Structure and Algorithmic Puzzles – **Narasimha Karumanchi** – Cap 14

Separate Chaining

- O **Encadeamento Externo**, ocorre quando a universo de chaves é muito maior que o tamanho máximo da tabela hash.

$$|N| \gg |M|$$

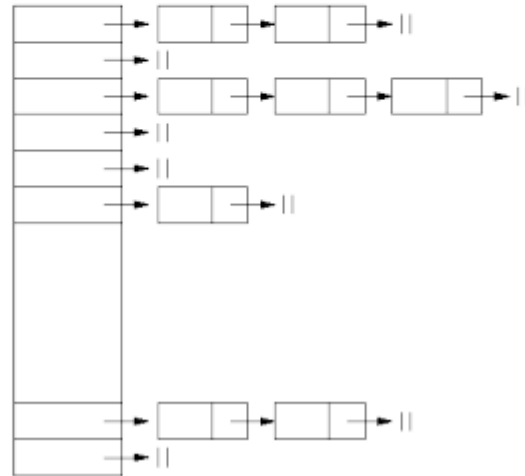
Separate Chaining

- A técnica de resolução de colisões de **Encadeamento Separado (SC)** é simples. Usamos **M** cópias de estruturas de dados auxiliares, geralmente **Listas Duplamente Ligadas**. Se duas chaves **a** e **b** tiverem o mesmo valor de hash **i**, ambas serão anexadas à (frente/trás) da Lista duplamente ligada **i**. É isso, onde as chaves serão inseridas é completamente dependente da própria função de hash, portanto, também chamamos de **Encadeamento Separado** como técnica de resolução de colisão de **Endereçamento Fechado**.

Separate Chaining

Armazena chaves sinônimas em listas encadeadas

- m listas encadeadas
- Endereço base: cabeça da lista
- Busca, inserção e remoção em listas encadeadas



Separate Chaining

- Melhor caso (inserir): $\mathcal{O}(1)$
- Pior caso (buscar): $\mathcal{O}(n)$
- Pode-se inserir ordenado para reduzir a busca
- **Deve-se questionar:** Qual operação será realizada mais vezes?

Implementação

- Let's code