



UNIFACS
LAUREATE INTERNATIONAL UNIVERSITIES

Pesquisa, Ordenação e Técnicas de Armazenamento

Aulas nº X: Tabela Hash

Prof. Luis Gustavo Araujo
2018

Objetivo

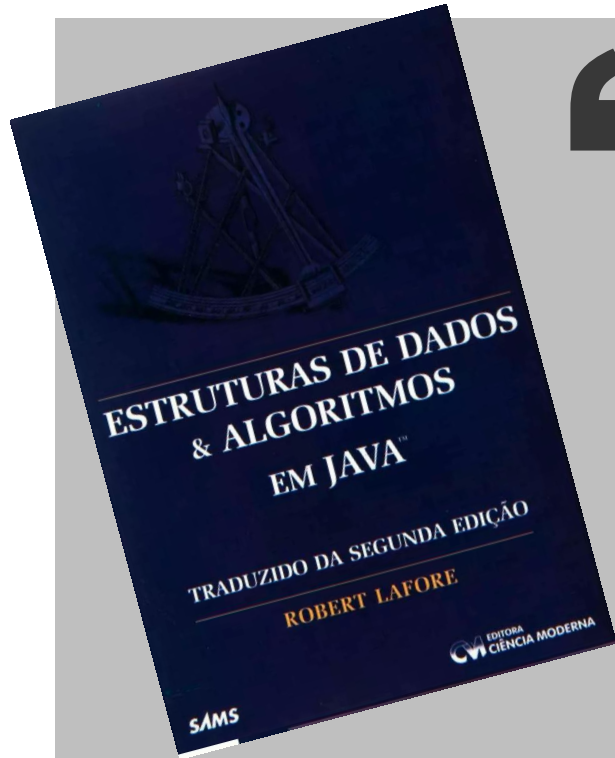
Entender o conceito de Tabela Hash, construindo e avaliando exemplos de implementação, bem como aplicar os conhecimentos em linguagem Java.

Tabela Hash

Est. Dados

Java

Tabela Hash



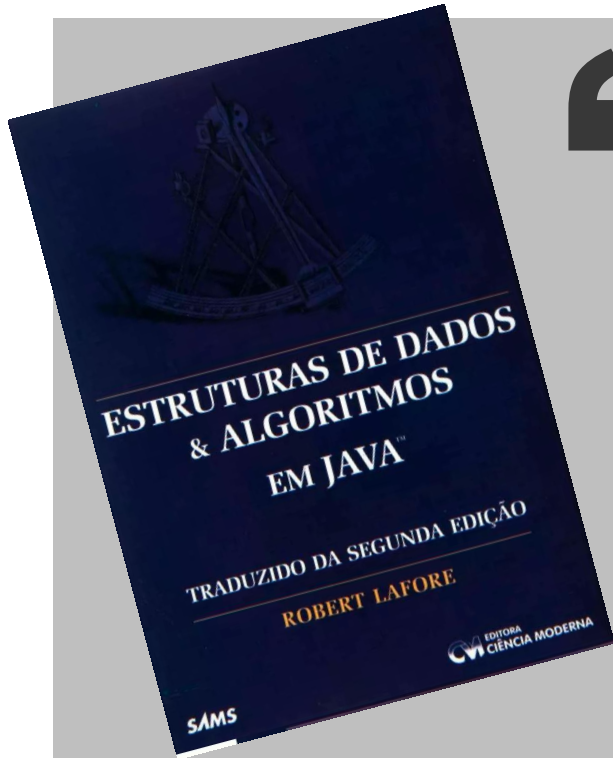
“

Tabela Hash é uma estrutura de dados que oferece inserção e busca muito rápidas. [...] Não importa quando itens de dados existem, inserção e busca (e algumas vezes eliminação) podem chegar perto do tempo constante $O(1)$ na notação Big O.

**Robert
Lafore**

Estrutura de Dados e Algoritmos em Java

Tabela Hash



“

Não são apenas rápidas, tabelas hash são relativamente fáceis de programar.

**Robert
Lafore**

Estrutura de Dados e Algoritmos em Java

Desvantagem da Tabela Hash

- 1 Baseada em vetores
- 2 Desempenho pode cair, quando cheias
- 3 Não há modo conveniente de visitar os itens

Quando usar Tabela Hash?

Sabe o tamanho da base de dados?

Não precisa visitar itens em ordem?



Sim!

Não!

Introdução a Hash e Hasing


Uma tabela hash transforma valores em índices de vetor. Mas um conceito importante é: *como faixa de valores são convertida em faixa de índices de um vetor.* Na tabela hash isso é feito como uma **função hash**.

Introdução a Hash e Hasing

Em alguns caso, não é necessário haver transformações, algum dado pode ser usado como índice. Vejamos:

Implementando uma Tabela Hash (caso 1)

Você foi designado para construir um software de controle de funcionários. O cliente pediu que os dados fossem acessados o **mais rápido possível**. A empresa tem 1000 funcionários. *Cada funcionário recebeu um número (1 para o primeiro funcionário e 1000 para o último contratado).*



Podemos usar isso como índice.

Implementando uma Tabela Hash (caso 1)

0	Funcionário (1, Abel, ...)
1	Funcionário (2, Laeb, ...)
2	...
3	...
4	...

996	...
997	...
998	...
999	Funcionário (1000, Arnaldo, ...)

id do Funcionário - 1 = Índice.

Vantagem do caso 1

Acessar o elemento é muito rápido, basta saber o id do usuário!

$u = \text{Funcionario}[id-1]$

Vantagem do caso 1

Inserir o elemento é muito rápido, basta seguir a numeração.

`u.id = ultimoRegistro;`

`Funcionario[ultimoRegistro++] = u`

Desvantagem do caso 1

Caso o usuário não saiba o número, buscá-lo torna-se uma tarefa custosa.

** Realizar um loop do 0 à 999 e compará-lo com o nome.*

Implementando uma Tabela Hash (caso 2)

Talvez usar o **nome** como índice seja uma opção melhor, mas nome é string... como converter em número?



Implementando uma Tabela Hash (caso 2)

Podemos usar número para cada letra, assim como a tabela ASCII.

Número	Letra
A	1
B	2
C	3
D	4
E	5
F	6

•
•
•

Implementando uma Tabela Hash (caso 2)

0	...
1	...
2	...
3	...
4	...

soma(letras) = indice

A B E L

↑ ↑ ↑ ↑

$$1 + 2 + 5 + 12 = 20$$

20 Funcionário (1, Abel, ...)

997	...
998	...
999	...



Vantagem do caso 2

Buscar por nome é muito rápido, basta passar o nome do usuário!

$u = \text{Funcionario}[\text{hash}(\text{Nome})]$

Vantagem do caso 2

Inserir o elemento é muito rápido, basta passar o nome do usuário!

$\text{Funcionario}[\text{hash}(\text{Nome})] = u$

Desvantagem do caso 2

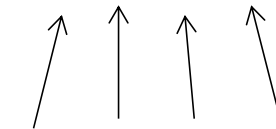
Muitos nomes podem conter a mesma soma, gerando muitas colisões.

Implementando uma Tabela Hash (caso 2)

0	...
1	...
2	...
3	...
4	...

soma(letras) = indice

L A E B



$$12 + 1 + 5 + 2 = 20$$

20 Funcionário (1, Abel, ...)

997	...
998	...
999	...



Implementando uma Tabela Hash (caso 3)

Então, podemos adicionar também algo relacionado à posição da letra!



Implementando uma Tabela Hash (caso 3)

0	...
1	...
2	...
3	...
4	...

$$nLetra * totalLetras^{Posicao} = indice$$

19.070

Funcionário (1, Abel, ...)

19.998

...

19.999

...

20.000

...

A B E L

$$1 * 26^3 + 2 * 26^2 + 5 * 26^1 + 12 * 26^0 = ?$$

$$1 * 17.576 + 2 * 676 + 5 * 26 + 12 * 1 = ?$$

$$17.576 + 1.352 + 130 + 12 = ?$$

$$17.576 + 1.352 + 130 + 12 = \mathbf{19.070}$$

19.070

Implementando uma Tabela Hash (caso 3)

$$nLetra * totalLetras^{Posicao} = indice$$

0 ...

19.069 ...

19.070 Funcionário (1, Abel, ...)

4 ...

214.320 Funcionário (1, Leab, ...)

214321 ...

214322 ...

214323 ...



L E A B

$12 * 26^3 + 5 * 26^2 + 1 * 26^1 + 2 * 26^0 = ?$

$12 * 17.576 + 5 * 676 + 1 * 26 + 2 * 1 = ?$

$210912 + 3380 + 26 + 2 = ?$

$210912 + 3380 + 26 + 2 = \mathbf{214320}$

214320

Implementando uma Tabela Hash (caso 3)

0

...

19.069

...

19.070

Funcionário (1, Abel, ...)

4

...

214.320

Funcionário (1, Leab, ...)

214321

...

214322

...

214323

...

$$n\text{Letra} * total\text{Letras}^{Posicao} = indice$$

Mas não eram apenas
1000 Funcionários?



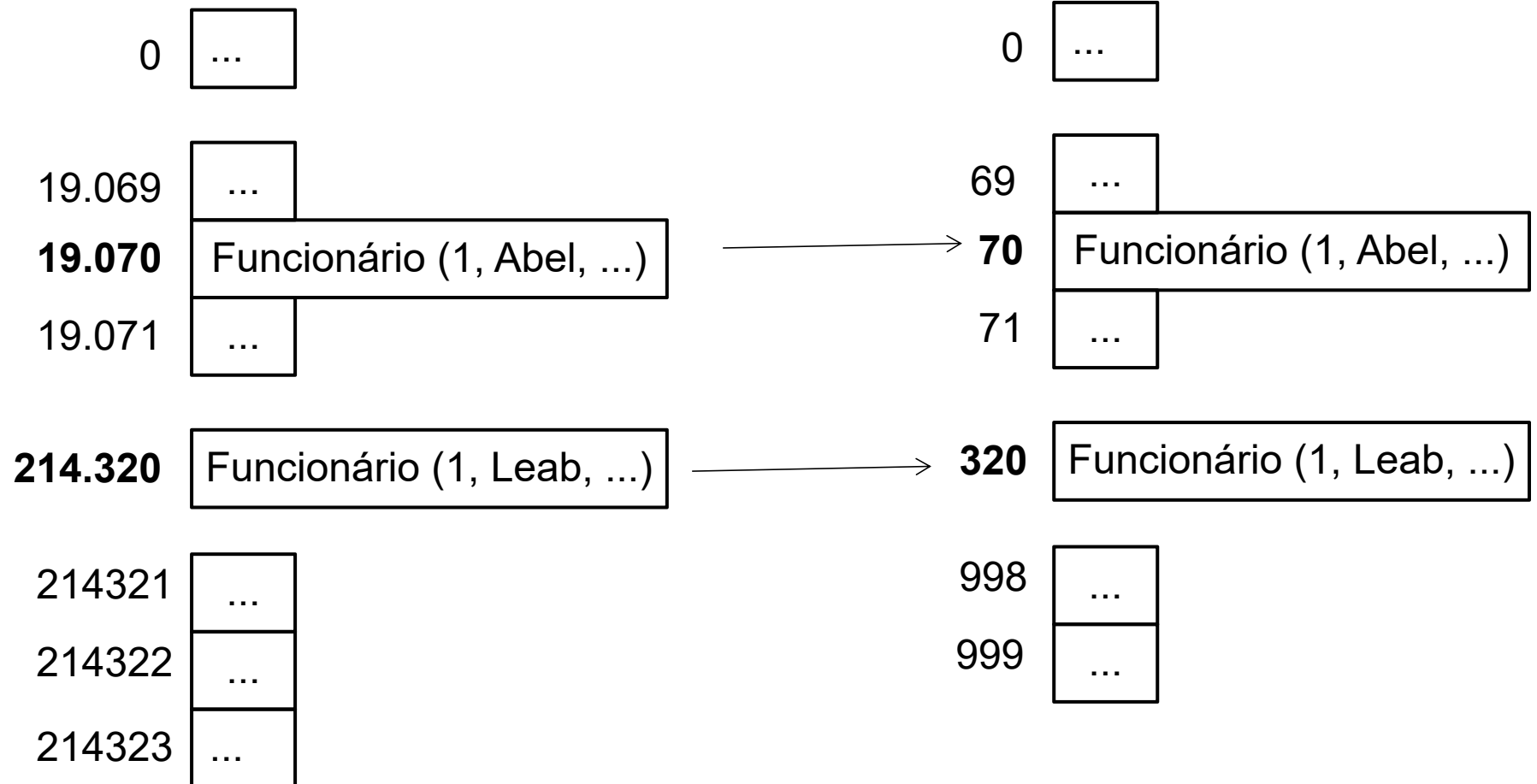
Hasing

Precisamos converter uma faixa enorme em uma faixa menor, que esteja dentro do tamanho do nosso vetor. Uma abordagem simples é usar %;

numeroLargo % limite = numeroDentroFaixa

Implementando uma Tabela Hash (caso 4)

índice % limite = índice



Tratando Colisões

Existem três formas de tratar colisões com Endereçamento Aberto (Linear, Quadrática e Duplo).

Tratando colisões: Exploração Linear

0	...
1	...
2	...
3	...
4	...

20	Funcionário (1, Abel, ...)
21	Funcionário (2, Laeb, ...)

997	...
998	...
999	...

Exploração Linear: Procura a próxima posição livre.

Desvantagens

Criação de *clustering*, ou seja os elementos vão se agrupando (seguidos e fora do seu índice).

Tratando colisões: Exploração Quadrática

0	...
1	...
2	...
3	...
4	...

20	Funcionário (1, Abel, ...)
21	Funcionário (3, Gaea, ...)

24	...
998	...
999	...

Funcionário (2, Laeb, ...)

*Exploração Quadrática:
Procura a posição $x + n^2$ livre.*

*$n = \{1, 2, 3, \dots\}$, sendo as
tentativas.*

se $x = 20$

Laeb tenta = $20 + 1^2 = 21$ (**cheio**)

Laeb tenta = $20 + 2^2 = 24$ (**ok**)

Desvantagens

Criação de *clustering* secundário, ou seja os elementos com o mesmo índice vão percorrer o mesmo trajeto.

Solução

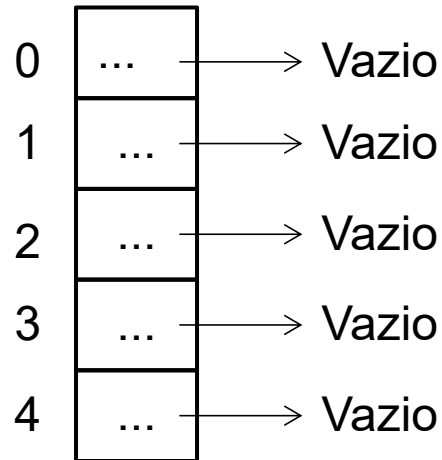
Usar um Hash Duplo, com base em casa chave, assim gera-se diferentes caminhos.

Ex: contante - (cheve % consante)

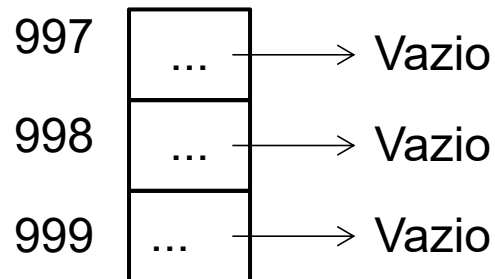
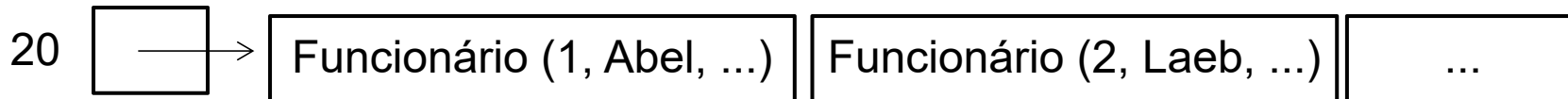
Tratando Colisões

Além do Endereçamento Aberto, podemos usar o Endereçamento Separado.

Tratando colisões: Endereçamento Aberto



*Endereçamento Separado:
Adiciona uma lista para cada
índice.*



Atividade

- a) Implemente, em Java, o caso 4, visto em sala (crie uma função hash própria);
- b) Implemente os métodos de tratamento de colisão:
 - i) Linear;
 - ii) Quadrática;
 - ii) Duplo;
 - iii) Separado.
- c) Faça inserções na tabela usando os 4 métodos e verifique quantas colisões ocorreram (*Você precisa criar um contador*) e quantos espaços vazios há na tabela

Atividade

d) Crie um relatório simplificado descrevendo a função hash utilizada e os métodos utilizados. No relatório deve haver uma sessão para apresentar os dados de colisões e espaços, para cada método.

Atividade



No site da disciplina:

http://luisaraujo.github.io/aulas/unifacs/disciplinas/pesq_orde/2018/home/



Enviado por e-mail (@unifacs)

Referências Técnicas

LAFORE, R. **Estruturas de Dados e Algoritmos em Java**. 1. Ed. São Paulo: Ciência Moderna, 2005.

ECKEL, B. ***Pensando em Java*** (tradução de Thinking in Java. 3. ed. Prentice-Hall, Dezembro 2002.) - Online.

CAELUM. Curso CS14: **Algoritmo e Estrutura de Dados em Java**. Online.



UNIFACS
LAUREATE INTERNATIONAL UNIVERSITIES

Pesquisa, Ordenação e Técnicas de Armazenamento

Aulas nº X: Tabela Hash

Prof. Luis Gustavo Araujo
2018