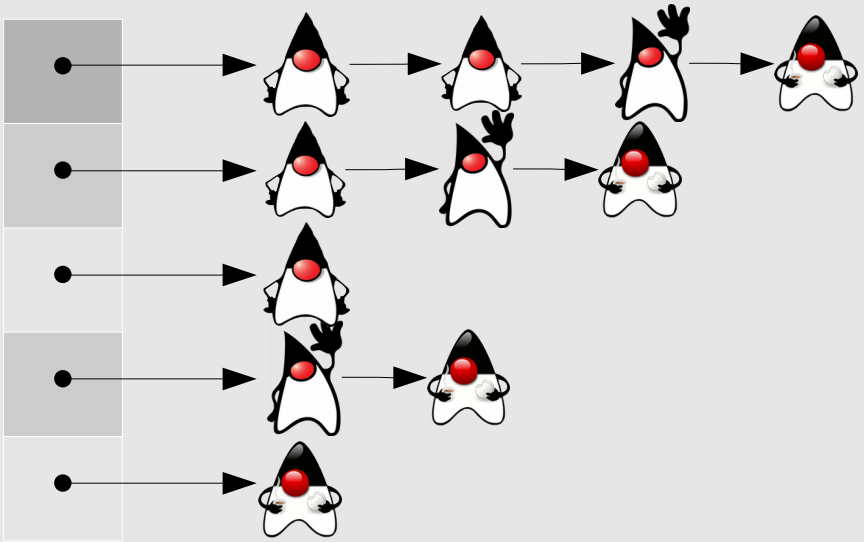


Tabelas Hash

ICOMP
Instituto de Computação

Horácio Fernandes
horacio@icomp.ufam.edu.br

Tabelas Hash
Avisos

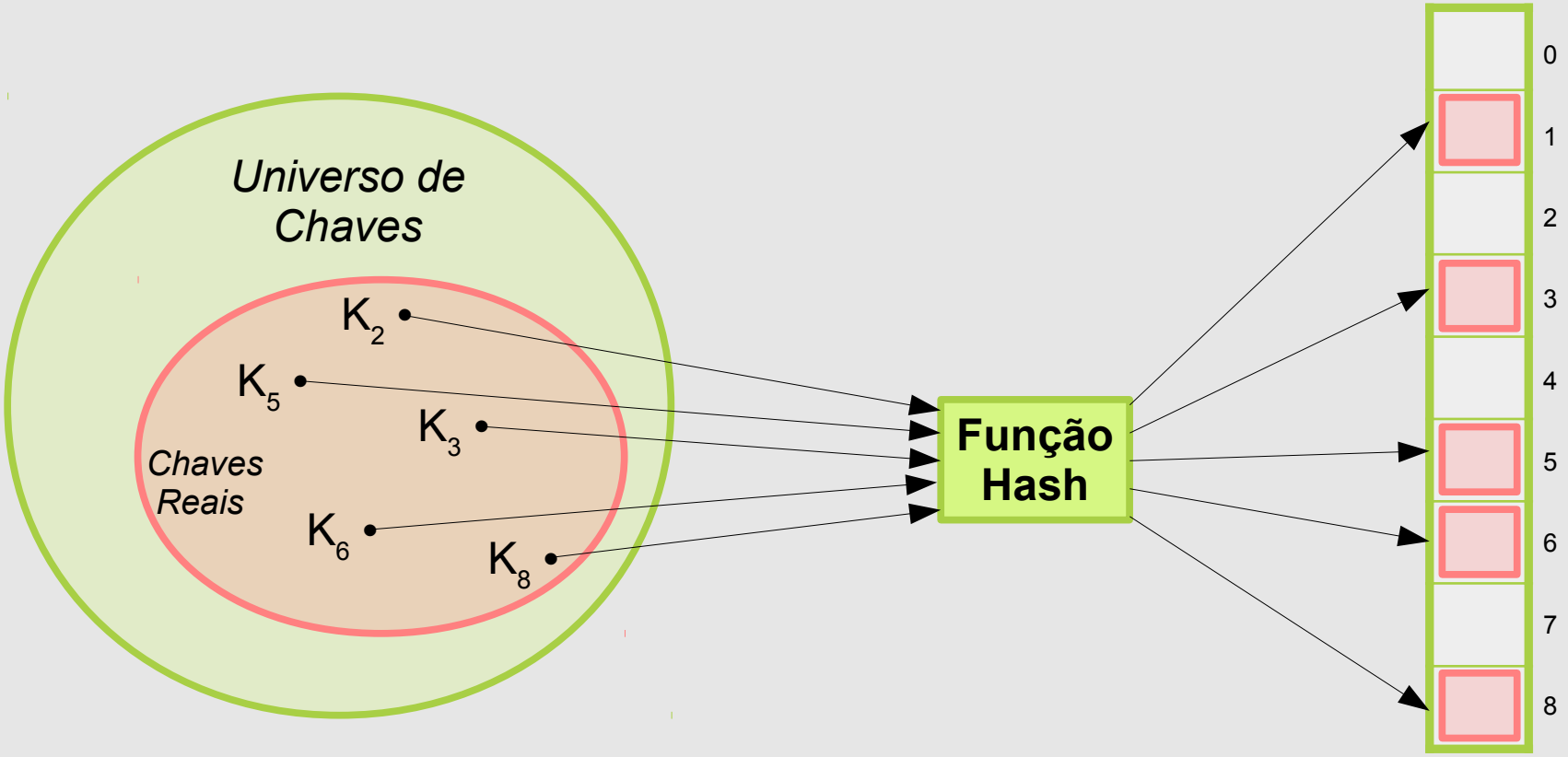
- Atenção #1
  - Estes slides estão em versão beta.
  - Apesar de terem sido revisados, alguns códigos podem ainda conter alguns bugs.
  - Caso encontre algum, envie e-mail para
    - *horacio@icomp.ufam.edu.br*
- Atenção #2
  - Cuidado ao copiar/colar códigos do formato PDF. Em geral as “aspas” ficam erradas e caracteres invisíveis são introduzidos em seu código, gerando erros de compilação quase impossíveis de serem detectados. Dê preferência para o formado ODP (LibreOffice).

Tabelas Hash
Introdução

- Tabelas *Hash* (tabelas de espalhamento, tabelas de dispersão):
  - Armazenam dados em uma tabela (vetor) de acordo com a chave de pesquisa do dado (identificador)
  - A chave de pesquisa pode ser de qualquer tipo
    - *inteiro, float, string, chave composta, etc*
  - Uma função *hash* (função de transformação) é usada para “converter” a chave de pesquisa em uma das posições do vetor (que é um inteiro)
  - Uma vez que podem haver colisões (duas ou mais chaves apontando para o mesmo índice do vetor), é necessário haver métodos para resolvê-los.

Tabelas Hash
Introdução

- Esquema de uma tabela hash:



Tabelas Hash
Aplicações

- Tabelas Hash são usadas em aplicações que exijam apenas operações “de dicionário” (Buscar, Inserir, Remover) e, em especial, precisam de um acesso rápido aos dados (busca):
  - Compilador armazenando a tabela de identificadores (variáveis) precisa acessar as informações uma variável rapidamente pelo nome dela
  - Tabelas Associativas
    - *Muitas linguagens de programação (PHP, Perl) permitem criar vetores cujos índices sejam strings (ou outros objetos)*
  - Índices de Banco de Dados
  - Torrents usam uma versão distribuída das tabelas hash
    - *Distributed Hash Tables (DHT)*
  - Coleções de produtos cuja chave seja o código de barra
  - A Máquina Virtual Java (HotSpot) usa uma tabela hash para implementar o seu “Repositório de Strings”
  - Armazenamento e acesso a dados de alunos via matrícula (próximo slide)

Tabelas Hash
Aplicações

- Imagine que uma aplicação precisa armazenar dados de alunos e acessá-los pela matrícula deles
  - Para ter um acesso rápido sem precisar fazer uma pesquisa, a solução mais eficiente, em termos de processamento, seria um vetor
  - Mas como a matrícula possui 8 dígitos, seria necessário um vetor de com 100.000.000 posições! Vamos dizer que a UFAM possui cerca de 100.000 alunos (chute), isso seria um desperdício, pois estaríamos usando apenas 0.1% do espaço alocado
  - Uma solução mais interessante seria uma *tabela hash* com 100.000 posições (ou menos) e um mapeamento da matrícula do aluno para uma dessas posições

Tabelas Hash
Sumário

- Introdução
- **Funções Hash**
- Tabelas Hash com Endereçamento Aberto
- Tabelas Hash com Listas Encadeadas
- Conclusão
  - Tabelas Hash em Java
  - Referências

Funções Hash (Funções de Dispersão)

- Uma função hash mapeia uma chave em um índice da tabela
- A eficiência de uma tabela hash dependerá em grande parte da função hash utilizada. Uma boa função hash deve:
  - Espalhar bem e de forma uniforme (igual probabilidade) as chaves na tabela, de modo a evitar colisões
  - Ser eficiente (fácil implementação)
- Funções hash são também conhecidas como
  - *funções de dispersão*
  - *funções de transformação*

Funções Hash (Funções de Dispersão)

- Algumas das principais técnicas para criação de funções hash incluem:
  - Método da divisão
    - *Usado principalmente quando a chave é um número*
  - Método da multiplicação
    - *Usado principalmente quando a chave é um número*
  - Método da adição
    - *Usado principalmente quando a chave é uma string*

Tabelas Hash
Funções Hash – Método da Divisão

- Nas funções hash baseadas no método da divisão:
  - Mapeamos uma chave k para um vetor de m posições tomando o resto de k dividido por m
    - $h(k) = k \bmod m$
  - Exemplo:
    - Para $m = 13$
      - $h(100) = 9$
      - $h(20) = 7$
      - $h(1) = 1$
      - $h(2) = 2$
      - $h(3) = 3$
  - A eficiência dependerá do valor de m
    - m não deve ser potência de 2
    - m sendo um primo grande não muito próximo de uma potência exata de 2 frequentemente é uma boa escolha
  - Este método é também conhecido como *Método da Congruência Linear*

Funções Hash – Método da Multiplicação

- Nas funções hash baseadas no método da multiplicação:
  - Multiplicamos a chave k por uma constante A entre 0 e 1;
  - Extraímos a parte fracionária de $k * A$;
  - Multiplicamos o valor por m; e
  - Pegamos o piso (*floor*) do valor retornado:
    - $h(k) = \lfloor m\ (kA\ mod\ 1) \rfloor$
  - Exemplo:
    - *Para* $m = 16\ e\ A = 0.543$
      - $h(81) = \lfloor 16*(81*0.543 \bmod 1) \rfloor = \lfloor 16*0.983 \rfloor = \lfloor 15.728 \rfloor = 15$
  - Neste método, o valor de m não é crítico (em geral é uma potência de 2)
  - Este método é também conhecido como *Método da Congruência Linear Multiplicativo*

Funções Hash – Método da Adição

- Nas funções hash baseadas no método da adição:
  - Os valores dos caracteres da string correspondentes na tabela ASCII são somados.
  - A chave é o resultado dessa soma, e podemos usar uma das técnicas anteriores para calcular a posição na tabela.
  - Exemplo:
    - *Para* $m = 13$ *e usando o método da divisão*
      - $h(\text{“AED2”}) = (65 + 69 + 68 + 50) \bmod 13 = 252 \bmod 13 = 5$
      - $h(\text{“UFAM”}) = (85 + 70 + 65 + 77) \bmod 13 = 297 \bmod 13 = 11$

Tabelas Hash
Colisões

- Por melhor que seja uma função hash, eventualmente ela gerará uma mesma posição na tabela para chaves diferentes
  - Exemplo:
    - *Para* $m = 13$ *e usando o método da divisão*
      - $h(8) = 8$
      - $h(21) = 8$
      - $h(34) = 8 \ldots$
- Quando isso acontece, dizemos que houve uma colisão
- Portanto, tabelas hash precisam de um método para tratamento de colisões. Os mais conhecidos são:
  - Endereçamento Aberto (*open addressing*)
  - Encadeamento (*separate chaining*)

Sumário

- Introdução
- Funções Hash
- **Tabelas Hash com Endereçamento Aberto**
- Tabelas Hash com Encadeamento
- Conclusão
  - Tabelas Hash em Java
  - Referências

Tabelas Hash com Endereçamento Aberto
Introdução

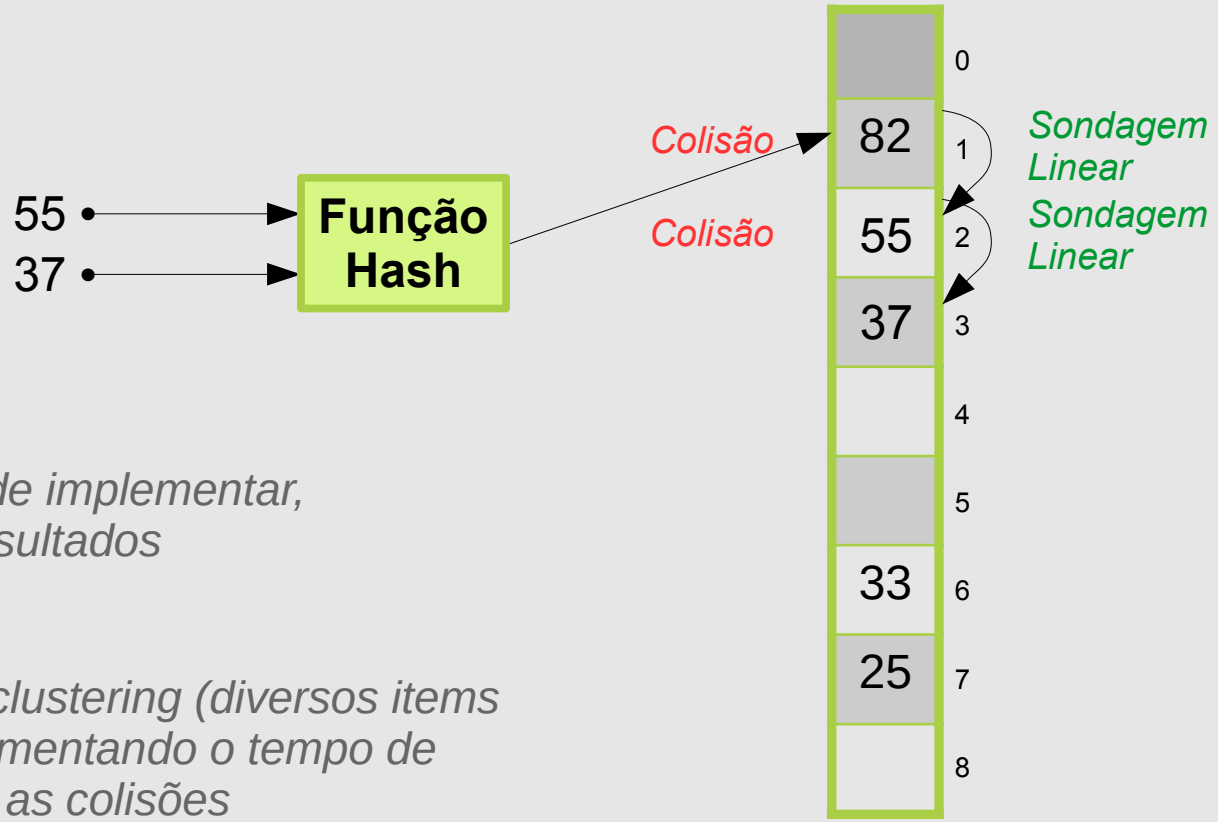
- Nas tabelas hash com endereçamento aberto (*open addressing*), quando há uma colisão, um segundo índice da tabela é gerado
  - Se houver outra colisão (conhecido como *colisão secundária*), um terceiro índice será gerado
  - Assim por diante, até que se encontre uma posição livre na tabela
  - Ao encontrar uma posição livre, o elemento é inserido naquela posição

Tabelas Hash com Endereçamento Aberto
Introdução

- Para gerar este próximo endereço na tabela, algumas técnicas são conhecidas:
  - Sondagem Linear *(Linear Probing)*
    - *O valor do índice que sofreu colisão é incrementado em 1 (um)*
  - Sondagem Quadrática *(Quadratic Probing)*
    - *Parecido com Linear Probing, só que agora, o incremento é quadrático*
  - Hash Duplo *(Double Hashing)*
    - *Utiliza uma segunda função hash para calcular o próximo índice*

Tabelas Hash com Endereçamento Aberto
Sondagem Linear

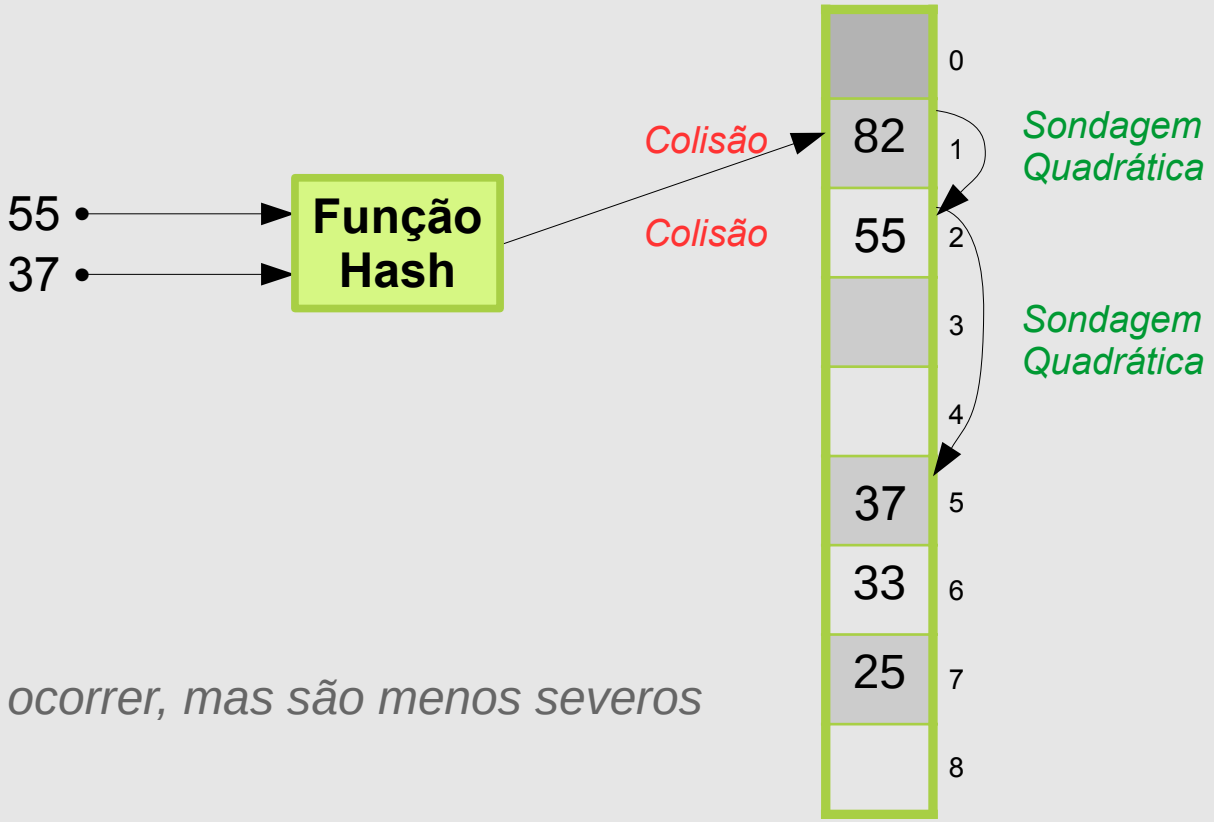
- O valor do índice correspondente à chave que sofreu colisão é incrementado até que uma posição disponível seja encontrada
  - `h(x, i) = ( h'(x) + i ) mod m`



  - Vantagem:
    - *Método simples e fácil de implementar, que pode gerar bons resultados*
  - Desvantagens:
    - *Faz surgir o fenômeno clustering (diversos items próximos na tabela), aumentando o tempo de consulta e aumentando as colisões*
  - Está técnica é também chamada de “teste linear”

Tabelas Hash com Endereçamento Aberto
Sondagem Quadrática

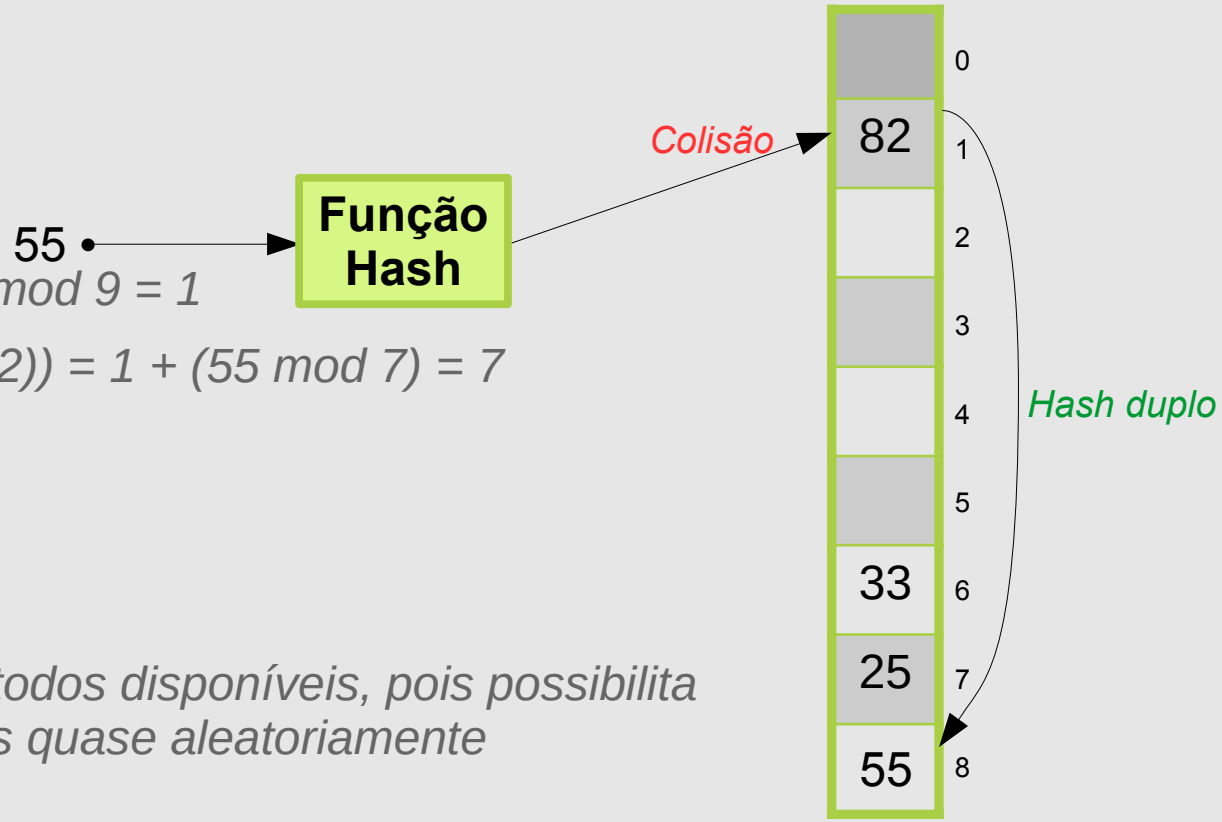
- Comportamento parecido com a sondagem linear, só que agora, o incremento é quadrático
  - $h(x, i) = (h'(x) + i^2) \bmod m$



  - Vantagem:
    - *Evita clusters*
  - Desvantagens:
    - *Clusters podem ainda ocorrer, mas são menos severos*

Tabelas Hash com Endereçamento Aberto
Hash Duplo

- Utiliza duas funções hash para tratar a colisão, a primeira é a mesma utilizada para o mapeamento da chave e a segunda é uma espécie de função incremento
  - `h(x, i) = ( h'(x) + i*g(x) ) mod m`
  - Exemplo:
    - *x = 55*
    - *m = 9*
    - *h'(x) = x mod m = 55 mod 9 = 1*
    - *g(x) = 1+(x mod (m – 2)) = 1 + (55 mod 7) = 7*
    - *h'(55) = 1*
    - *g(55) = 7*
  - Vantagem:
    - *Um dos melhores métodos disponíveis, pois possibilita distribuir os elementos quase aleatoriamente*
  - Desvantagens:
    - *Maior nível de complexidade*



Tabelas Hash com Endereçamento Aberto
Operações

- Principais operações em uma tabela hash:
  - Inicializa (construtor): `public TabelaHashEndAberto(int maxTam)`
  - Tamanho: `public int tamanho()`
  - Imprime: `public void imprime()`
  - Busca: `public Item busca(String chave)`
  - Função Hash: `private int funcaoHash(String chave)`
  - Insere: `public boolean insere(String chave, Object valor)`
  - Remove: `public boolean remove(String chave)`
- Implementação:
  - Um cuidado especial deve ser tomado com a remoção dos elementos. Se simplesmente removêssemos um elemento, isso poderia interferir na busca dos elementos que estão após ele.
    - *Solução: ao invés de remover um elemento, marque-o como removido*

Tabelas Hash com Endereçamento Aberto
Estrutura do Item

```
class Item {
  private String  chave;
  private Object  valor;
  private boolean removido;

  public Item(String chave, Object valor) {
    this.chave = chave;
    this.valor = valor;
    this.removido = false;
  }

  public String getChave() { return chave; }
  public void   setChave(String chave) { this.chave = chave; }

  public Object getValor() { return valor; }
  public void   setValor(Object valor) { this.valor = valor; }

  public boolean isRemovido() { return removido; }
  public void    setRemovido(boolean removido) {
    this.removido = removido;
  }
}
```

Chave do item

Valor do item

Se o item foi removido

Tabelas Hash com Endereçamento Aberto
Construtor, Tamanho, Impressão

```
public class TabelaHashEndAberto {

  private Item tabela[];
  private int tamanho;

  public TabelaHashEndAberto(int maxTam) {
    tabela = new Item[maxTam];
  }

  public int getTamanho() {
    return tamanho;
  }

  public void imprime() {
    for (int i=0; i<tabela.length; i++)
      if (tabela[i] != null && tabela[i].isRemovido() == false)
        System.out.println(tabela[i].getChave() + " = " +
                           tabela[i].getValor());
  }

  // Métodos ...
}
```

Tabela contendo os itens. Um item nulo indica que a posição está livre.

Inicializa a tabela

Para cada item da tabela ..

Se ele existe ..

E não foi apagado ..

Imprime a chave e o valor

Tabelas Hash com Endereçamento Aberto
Função Hash

- A função hash será calculada somando os valores ASCII dos caracteres da chave e pegando o resto da divisão pelo tamanho da tabela.

```
private int funcaoHash(String chave) {
  int soma = 0;

  for (int i=0; i<chave.length(); i++)
    soma += chave.charAt(i);

  return soma % tabela.length;
}
```

Soma os caracteres

Pega o resto com o tamanho máximo da tabela

Custo $O(m)^*$

** m = quantidade de caracteres da chave que é mínimo em relação à quantidade de elementos*

Tabelas Hash com Endereçamento Aberto
Busca

```
public Item busca(String chave) {
  if (chave == null) return null;

  int indice = funcaoHash(chave);
  int i = 0;
  while (tabela[indice] != null &&
         tabela[indice].getChave() != chave &&
         i < tabela.length) {
    i++;
    indice = (indice + 1) % tabela.length;
  }

  Item retorno = tabela[indice];

  if (retorno != null &&
      retorno.isRemovido() == false &&
      retorno.getChave() == chave)
    return retorno;
  else
    return null;
}
```

Posição inicial do item com base na função hash

Enquanto existir um item na posição mas a chave for diferente, tenta o próximo (sondagem linear)

Caso Médio	Pior Caso
$O(1)$	$O(n)$

Tabelas Hash com Endereçamento Aberto
Inserção de um Item

```
public boolean insere(String chave, Object valor) {
  if (tamanho == tabela.length || busca(chave) != null) return false;

  int indice = funcaoHash(chave);
  int i = 0;
  while (
      tabela[indice] != null &&
      tabela[indice].isRemovido() == false &&
      i < tabela.length
      ) {
    i++;
    indice = (indice + 1) % tabela.length;
  }

  tabela[indice] = new Item(chave, valor);

  tamanho++;
  return true;
}
```

Não aceita chaves repetidas

Tamanho máximo atingido

Enquanto existir um item na posição que não tenha sido removido (colisão), tenta o próximo

Insere o item

Caso Médio	Pior Caso
$O(1)$	$O(n)$

Tabelas Hash com Endereçamento Aberto
Remoção de um Item

- Para remover um item, basta setá-lo como apagado
  - Neste caso, setamos sua chave para nulo para indicar que o mesmo foi apagado

```
public boolean remove(String chave) {
  Item item = busca(chave);

  if (item == null) return false;

  item.setRemovido(true);
  item.setChave(null);
  item.setValor(null);

  tamanho--;
  return true;
}
```

Item não encontrado

Seta como removido

Remove referências

Caso Médio **O(1)**

Pior Caso **O(*n*)**

Tabelas Hash com Endereçamento Aberto
Conclusões

- A principal desvantagem das Tabelas Hash com Endereçamento Aberto é que tamanho da tabela é fixo e, portanto, limita a quantidade de elementos
  - Isso pode ser resolvido através de um *rehash* (será visto mais adiante)
- Devido a isso, em geral Tabelas Hash com Encadeamento (a seguir) são mais utilizadas

Sumário

Tabelas Hash com Encadeamento
Introdução

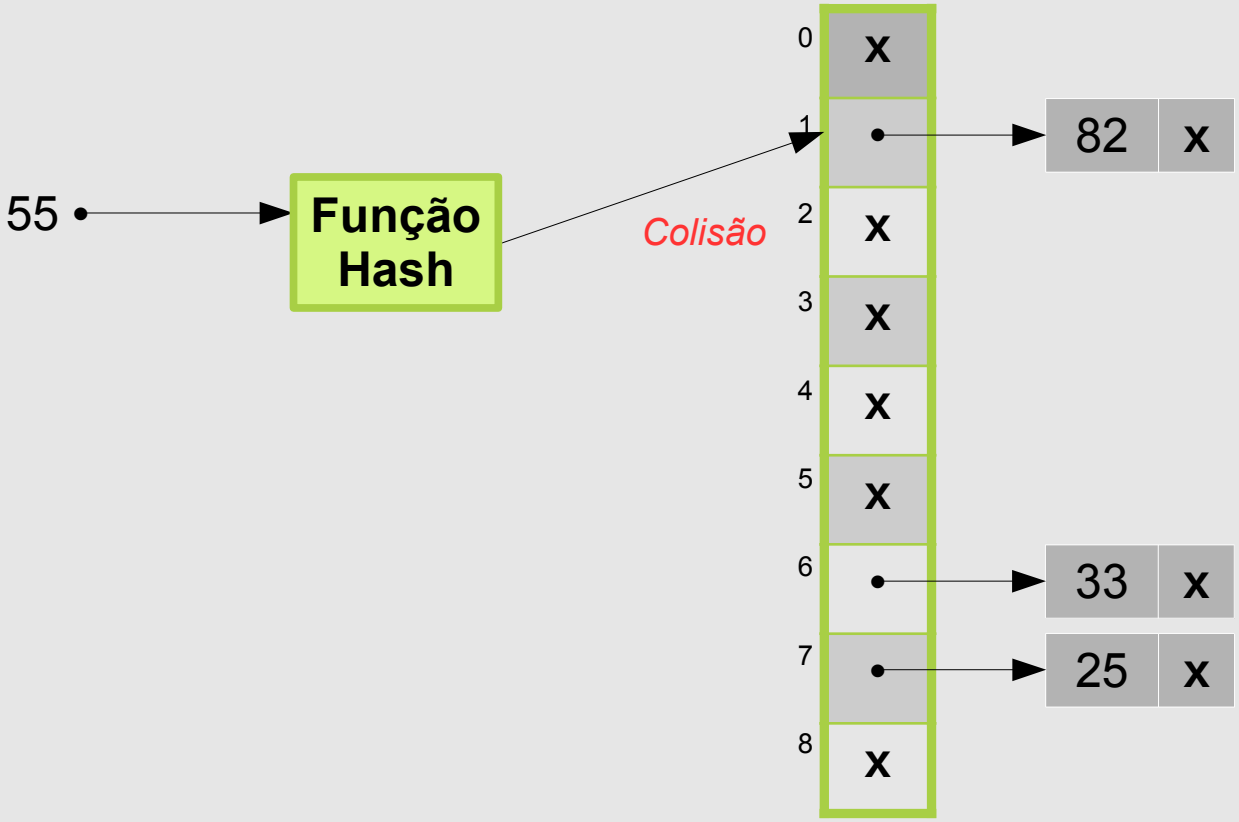
- Nas tabelas hash com encadeamento, quando há uma colisão, uma estrutura de dados separada (como uma lista) é usada para armazenar as chaves conflitantes no mesmo índice da tabela
  - Neste caso, ao invés de termos um vetor de elementos, temos um vetor de listas encadeadas
- Tabelas hash com encadeamento são também conhecidas como:
  - *Tabelas hash com endereçamento separado*
  - *Separate chaining (hashing)*
  - *Chained hash*

Tabelas Hash com Encadeamento
Introdução

- Na grande maioria das vezes, uma lista encadeada simples é usada para armazenar elementos com mesmo índice, mas outras estruturas podem também ser utilizadas:
  - Vetores (ou mesmo um vetor de vetores)
  - Árvores (próximo tópico da disciplina)
  - Outras tabelas hash, etc

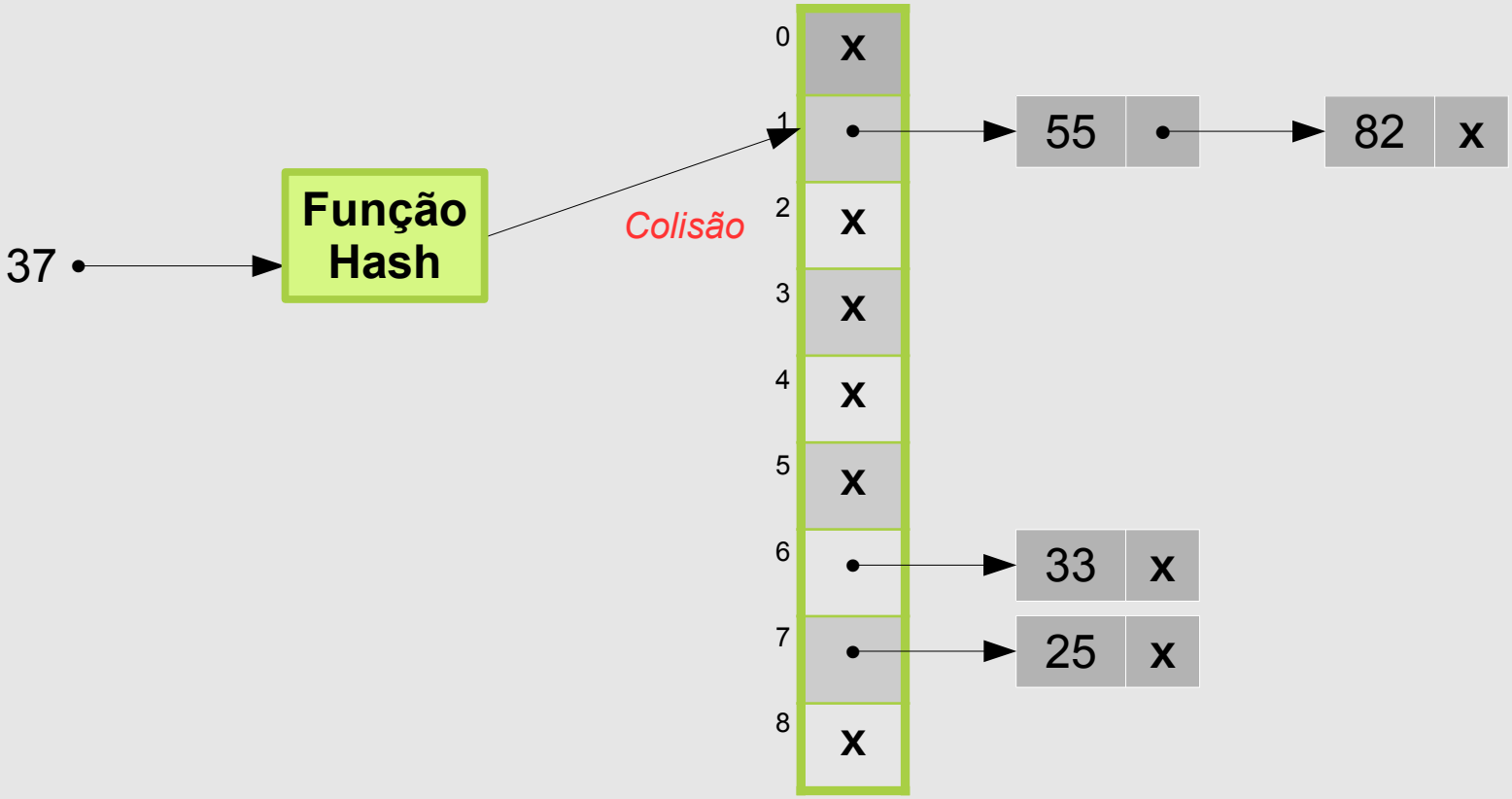
Tabelas Hash com Encadeamento
Introdução

- Esquema de funcionamento:



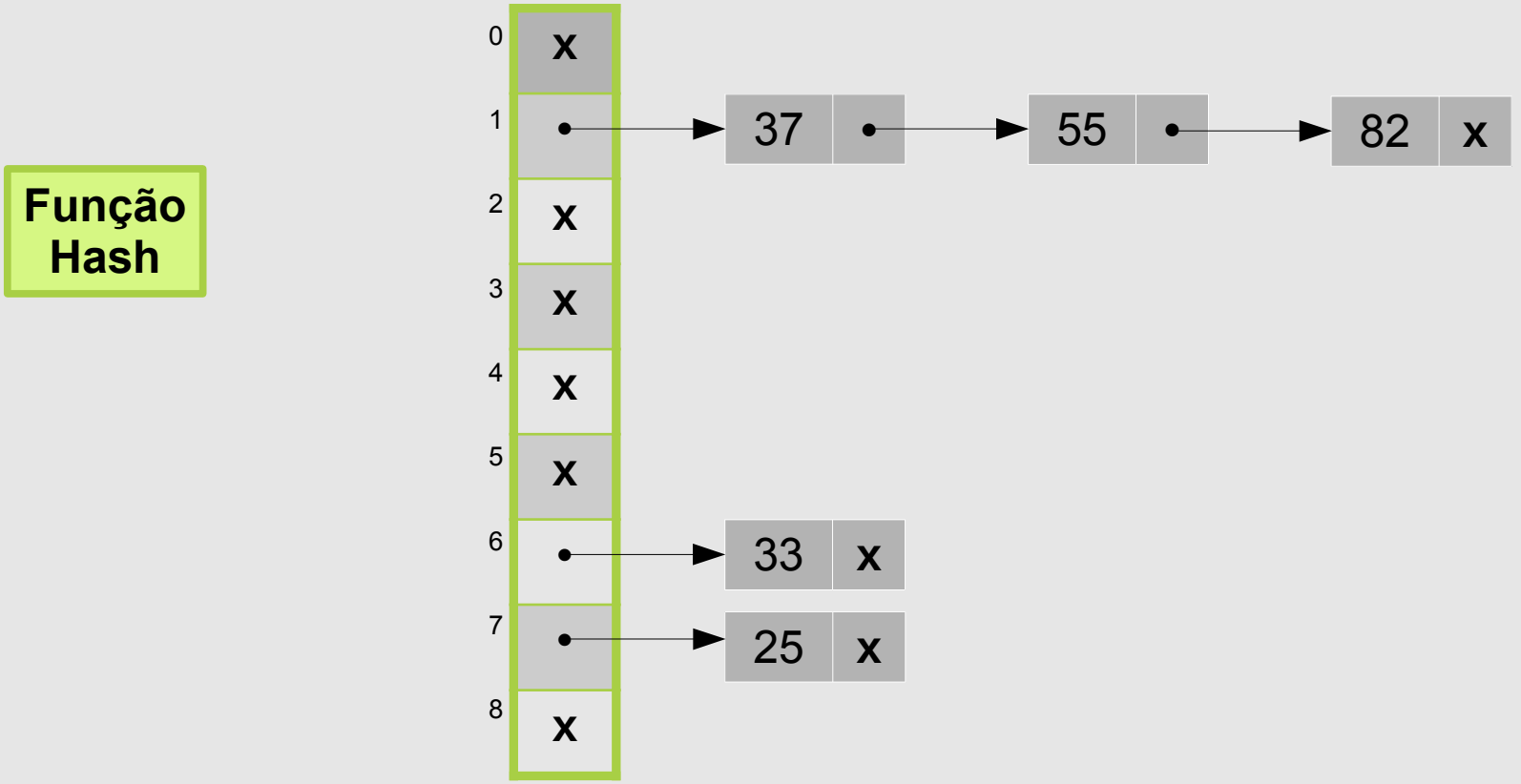
Tabelas Hash com Encadeamento
Introdução

- Esquema de funcionamento:



Tabelas Hash com Encadeamento
Introdução

- Esquema de funcionamento:



Tabelas Hash com Encadeamento
Tamanho das Listas

- Idealmente, o tamanho das listas encadeadas será bem pequeno, de modo que a pesquisa pode ser feita com custo linear
  - Isso nos casos em que você esteja usando uma função hash descente com um tamanho de vetor adequado
  - Entretanto, se o tamanho das listas encadeadas começar a aumentar, existem algumas alternativas:
    - *Utilizar outra estrutura além da lista encadeada. O Java, por exemplo, usa lista encadeada quando a lista é menor que 8. A partir disso, ele transforma a lista em uma árvore.*
    - *Fazer um rehash com um tamanho de tabela maior*
      - *Um rehash expande o vetor da tabela hash para ter mais elementos. Isso reduz, possivelmente, o tamanho das diversas listas encadeadas em cada posição do vetor. Cada elemento do vetor da tabela hash anterior, é adicionado no novo vetor. Note que os elementos na nova tabela ficarão em posições diferentes da tabela antiga, uma vez que a função hash usa o tamanho da tabela para calcular as posições dos elementos*
      - *Na implementação do Java, isso acontece sempre que o vetor da tabela hash fica 75% ocupado (carga de utilização >= 75%)*

Tabelas Hash com Encadeamento
Operações

- Principais operações em uma tabela hash:
  - Inicializa (construtor): `public TabelaHashEncadeamento(int maxTam)`
  - Tamanho: `public int tamanho()`
  - Imprime: `public void imprime()`
  - Busca: `public ItemNo busca(String chave)`
  - Insere: `public boolean insere(String chave, Object dado)`
  - Remove: `public boolean remove(String chave)`

Tabelas Hash com Encadeamento
Estrutura do ItemNo

```
class ItemNo {
  private String chave;
  private Object valor;
  private ItemNo prox;

  public ItemNo(String chave, Object valor) {
    this.chave = chave;
    this.valor = valor;
    this.prox = null;
  }

  // getChave, setChave, getValor, setValor ...

  public ItemNo getProx() {
    return prox;
  }
  public void setProx(ItemNo prox) {
    this.prox = prox;
  }
}
```

Endereço do próximo item (nó) na lista

Tabelas Hash com Encadeamento
Construtor, Tamanho, Impressão, Funcão Hash

```
public class TabelaHashEncadeamento {

  private ItemNo tabela[];
  private int tamanho;

  public TabelaHashEncadeamento(int maxTam) {
    tabela = new ItemNo[maxTam];
  }

  public int tamanho() {
    return tamanho;
  }

  public void imprime() {
    for (int i=0; i<tabela.length; i++)
      for (ItemNo atual = tabela[i]; atual != null; atual = atual.getProx())
        System.out.println(atual.getChave() + " = " + atual.getValor());
  }

  // A funcaoHash é a mesma da anterior
  // Métodos ...
}
```

Tabela contendo os itens

Inicializa a tabela

Para cada item da tabela ..

Percorre a lista encadeada

Imprime a chave e o valor

Tabelas Hash com Encadeamento
Busca de Elementos na Tabela

```
public ItemNo busca(String chave) {
  if (chave == null) return null;

  for (ItemNo retorno = tabela[funcaoHash(chave)];
       retorno != null; retorno = retorno.getProx()) {
    if (retorno.getChave().equals(chave))
      return retorno;
  }

  return null;
}
```

Percorre a lista encadeada referenciada pelo topo indicado na tabela

Se encontrar a chave ..

Retorna o item

Caso Médio	Pior Caso
$O(1)$	$O(n)$

Tabelas Hash com Encadeamento
Inserindo um Elemento na Tabela

```
public boolean insere(String chave, Object valor) {
  if (busca(chave) != null) return false;

  int indice = funcaoHash(chave);
  ItemNo topo = tabela[indice];
  tabela[indice] = new ItemNo(chave, valor);
  tabela[indice].setProx(topo);

  tamanho++;
  return true;
}
```

Encontra o topo da lista encadeada

Insere no topo

Caso Médio	Pior Caso
$O(1)$	$O(n)$

Tabelas Hash com Encadeamento
Removendo um Elemento da Tabela

```
public boolean remove(String chave) {
  int indice = funcaoHash(chave);

  ItemNo noAtual = tabela[indice];
  ItemNo noAnterior = null;

  while (noAtual != null && !noAtual.getChave().equals(chave)) {
    noAnterior = noAtual;
    noAtual = noAtual.getProx();
  }

  if (noAtual == null) return false;

  if (noAtual == tabela[indice])
    tabela[indice] = tabela[indice].getProx();
  else
    noAnterior.setProx(noAtual.getProx());

  tamanho--;
  return true;
}
```

Referências para o item atual e o anterior

Busca o nó com a chave

Se o nó encontrado for o topo

Remove do topo

Caso contrário, remove do meio

Caso Médio	Pior Caso
$O(1)$	$O(n)$

Tabelas Hash com Encadeamento
Conclusão

- Vantagens
  - Sem limites de tamanho
  - Sem problemas de clustering
- Desvantagens
  - Por usar referências:
    - *Há um acesso maior à memória nas operações de busca*
      - *seguir referências na lista que não foram pré-carregadas pelo cache*
    - *Há a necessidade de alocação de memória nas operações de inserção*

Tabelas Hash
Sumário

- Introdução
- Funções Hash
- Tabelas Hash com Endereçamento Aberto
- Tabelas Hash com Encadeamento
- **Conclusão**
  - Tabelas Hash em Java
  - Referências

Conclusão
Vantagens/Desvantagens

- Vantagens
  - Velocidade. O acesso aos dados de uma tabela hash, no caso médio, é extremamente rápido ($O(1)$), independente da quantidade de dados
  - Facilidade de implementação
- Desvantagens
  - O pior caso de busca em tabelas hash ainda é linear $O(n)$, quando todos os elementos da tabela hash caem no mesmo índice (colisão). Caso que pode ser facilmente evitado com uma boa função hash
  - Tabelas hash usam vetores, que podem ter espaços vazios, desperdiçando um pouco de memória
  - Não há uma ordem sequencial nos elementos, portanto a impressão e iteração pelos elementos da lista é feita de forma aleatória
    - *Listar os elementos em ordem seria uma operação muito custosa e, se esta operação for necessária, Árvores (próximo assunto) certamente seriam mais interessantes*

Conclusão
Custos

- Custo das Operações

Operação	**Pior Caso**	**Melhor Caso**	**Caso Médio**
busca	$O(n)$	$O(1)$	$O(1)$
insere	$O(n)$	$O(1)$	$O(1)$
remove	$O(n)$	$O(1)$	$O(1)$

Conclusão
Tabelas Hash em Java

- `Hashtable`
  - Em Java, Tabelas Hash são implementadas através da classe `Hashtable`
    - *Por ser uma coleção genérica (generic collection), é necessário especificar os tipos de dados (da chave e dos dados) que serão utilizados*
    - *Principais métodos:*
      - *`V put(K key, V value)` – insere um valor com uma determinada chave*
      - *`V get(Object key)` – busca um elemento pela chave*
      - *`V remove(Object key)` – remove um elemento com determinada chave*
      - *`int size()` – quantidade de elementos*
      - *`Enumeration<V> elements()` – retorna uma enumeração dos elementos*

Conclusão
Tabelas Hash em Java

- `Hashtable`
  - A classe `Hashtable` usa *Tabelas Hash com Encadeamento* (listas encadeadas) para armazenar os valores
    - *Entretanto, quando a tabela atinge um certo fator de uso (loadFactor), indicando que a tabela está ficando cheia (e muitas colisões irão ocorrer), o tamanho da tabela é automaticamente incrementado e todos os elementos são reajustados na tabela. Isso é conhecido como rehash e o fator de uso normalmente é de 75%*
    - *Além disso, quando o tamanho de uma lista encadeada passa de um determinado limite (8 elementos por padrão), esta é transformada em uma árvore (próximo assunto) para agilizar a busca por elementos*

Conclusão
Tabelas Hash em Java

```
import java.util.*;

public class HashJava {

  public static void main(String args[]) {

    Hashtable<String, Integer> mestres = new Hashtable<String, Integer>();

    mestres.put("Obi-Wan Kenobi", 57);
    mestres.put("Qui-Gon Jinn", 92);
    mestres.put("Yoda", 896);

    Integer n = mestres.get("Yoda");

    if (n != null)
      System.out.println("Nascimento de Yoda: " + n + " BBY");

  }

}
```

Tabela Hash em que as chaves são strings e valores são inteiros

Referências

- Slides do Prof. Barreto (AED2)
- ZIVIANI, Nivio. Projeto de Algoritmos com Implementação em Pascal e C. Cengage Learning, 2011. ISBN 9788522110506.
- CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L. STEIN, Clifford. Algoritmos □ Teoria e Prática, 3ª edição. Campus Editora, 2012. ISBN 978□85□352□3699□6.
- DEITEL, Paul; DEITEL, Harvey. *Java - How to Program*, 10ª edição. Pearson, 2015. ISBN 978-0-13-380780-6.
- NECAISE, Rance D. *Data Structures and Algorithms using Python*. Wiley, 2011. ISBN 978-0-470-61829-5.