Hashing
Endereçamento Direto
Tabelas Hash

Professora:

Fátima L. S. Nunes

Introdução

- Vimos até agora:
  - Conceitos e técnicas de Orientação a Objetos
  - Conceitos e aplicações de Complexidade Assintótica
  - Métodos de Ordenação
    - *Insertion Sort* (Ordenação por Inserção)
    - *Selection Sort* (Ordenação por Seleção)
    - *Bubble Sort* (Ordenação pelo método da Bolha)
    - *MergeSort* (Ordenação por intercalação)
    - *QuickSort* (Ordenação rápida)
    - *HeapSort* (Ordenação por monte)
    - *Counting Sort* (Ordenação por contagem)
    - *Radix Sort* (Ordenação da raiz)

Algoritmos de Ordenação

- Vimos até agora:
  - Métodos de Ordenação
    - *Insertion Sort* (Ordenação por Inserção)
    - *Selection Sort* (Ordenação por Seleção)
    - *Bubble Sort* (Ordenação pelo método da Bolha)
    - *MergeSort* (Ordenação por intercalação)
    - *QuickSort* (Ordenação rápida)
    - *HeapSort* (Ordenação por monte)
    - *Counting Sort* (Ordenação por contagem)
    - *Radix Sort* (Ordenação da raiz)

  - Qual a diferença do HeapSort para os demais métodos?

Dicionário

- *HeapSort* (Ordenação por monte)
  - Usa uma estrutura denominada ***heap*** para executar a ordenação.
  - Estruturas como o *heap* exigem diferentes operações a serem executados
    - inserir um elemento
    - eliminar um elemento
    - buscar um elemento
    - etc
  - Um conjunto que admite essas operações é chamado de ***dicionário***.
  - Muitas aplicações exigem um conjunto dinâmico que admita apenas as operações de dicionário ***insert***, ***search*** e ***delete***.

Definições

- O que é *Hash*?

Definições

- O que é *Hash*?
- Dicionário Michaelis:

hash

n **1** *Cook* prato feito de carne moída misturada com batata assada ou frita. **2** bagunça, confusão. **3** algo refeito ou reformado. • *vt* **1** cortar em pequenos pedaços. **2** misturar, confundir. **3** rever, fazer uma revisão. **to hash up an old story** reavivar uma velha história. **to make a hash of**estragar, complicar.

- Google:

noun

picado de carne
fricassé
confusão
erro grave
coisa refeita

verb

picar

Definições

- Tabela *hash*:
  - estrutura de dados eficiente para implementar dicionários
  - sob hipóteses razoáveis, apresenta tempo de busca de um elemento = O(1)
    - mas pode demorar O(n) no pior caso
  - generalização da noção mais simples de um arranjo comum
    - em um arranjo comum, para examinar uma posição arbitrária, basta ler esta posição ⇒ Tempo = O(1)
    - para isso,usa-se ***endereçamento direto***

Endereçamento Direto

- Técnica simples que funciona bem quando o universo ***U*** de chaves é razoavelmente pequeno
- ***Exemplo:***
  - uma aplicação que precisa de um conjunto dinâmico
  - cada elemento tem uma chave definida a partir do universo $U=\{0,1,\ldots, m-1\}$, onde ***m*** não é muito grande
  - não há elementos com a mesma chave
  - para representar o conjunto dinâmico, usamos uma **tabela de endereçamento direto T[0..m-1]**
    - cada posição da tabela corresponde a uma chave no universo *U*
    - a posição *k* aponta para um elemento no conjunto com chave *k*

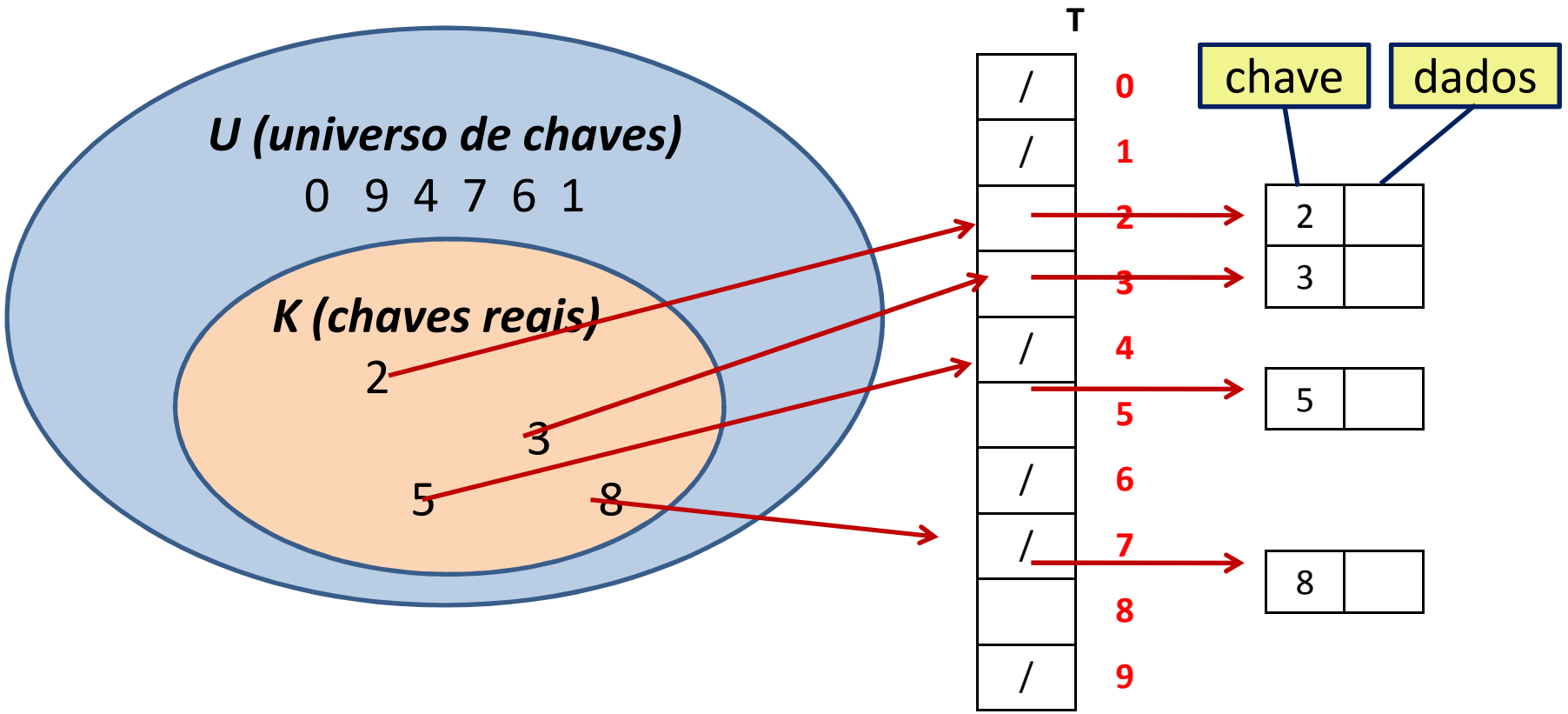
Endereçamento Direto

- Técnica simples que funciona bem quando o universo ***U*** de chaves é razoavelmente pequeno
- ***Exemplo:***
  - uma aplicação que precisa de um **conjunto dinâmico**
  - cada elemento tem uma chave definida a p do universo *U*={0,1,..., m-1
  - não há elementos com
  - para representar o conj **la de endereçamento dir**
    - cada posição da tabela corresponde a uma chave no universo *U*
    - a posição *k* aponta para um elemento no conjunto com chave *k*

Conjunto dinâmico: aumenta e diminui de acordo com a necessidade. Em um determinado momento pode estar vazio, completo ou parcialmente preenchido.

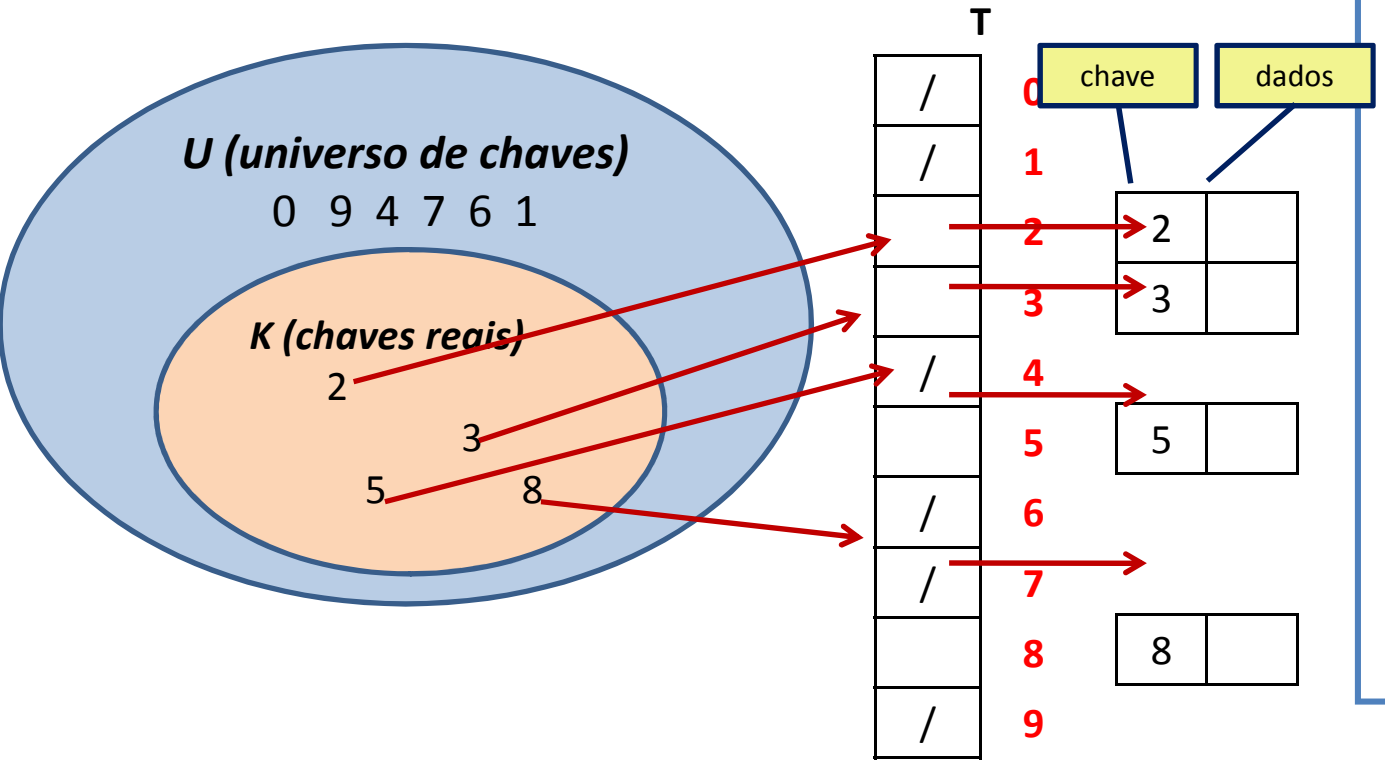
Endereçamento Direto

- ***Exemplo:***
  - para representar o conjunto dinâmico, usamos uma **tabela de endereçamento direto T[0..m-1]**
    - cada posição da tabela corresponde a uma chave no universo *U*
    - a posição *k* aponta para um elemento no conjunto com chave *k*



Endereçamento Direto

- Implementação das operações de dicionário:



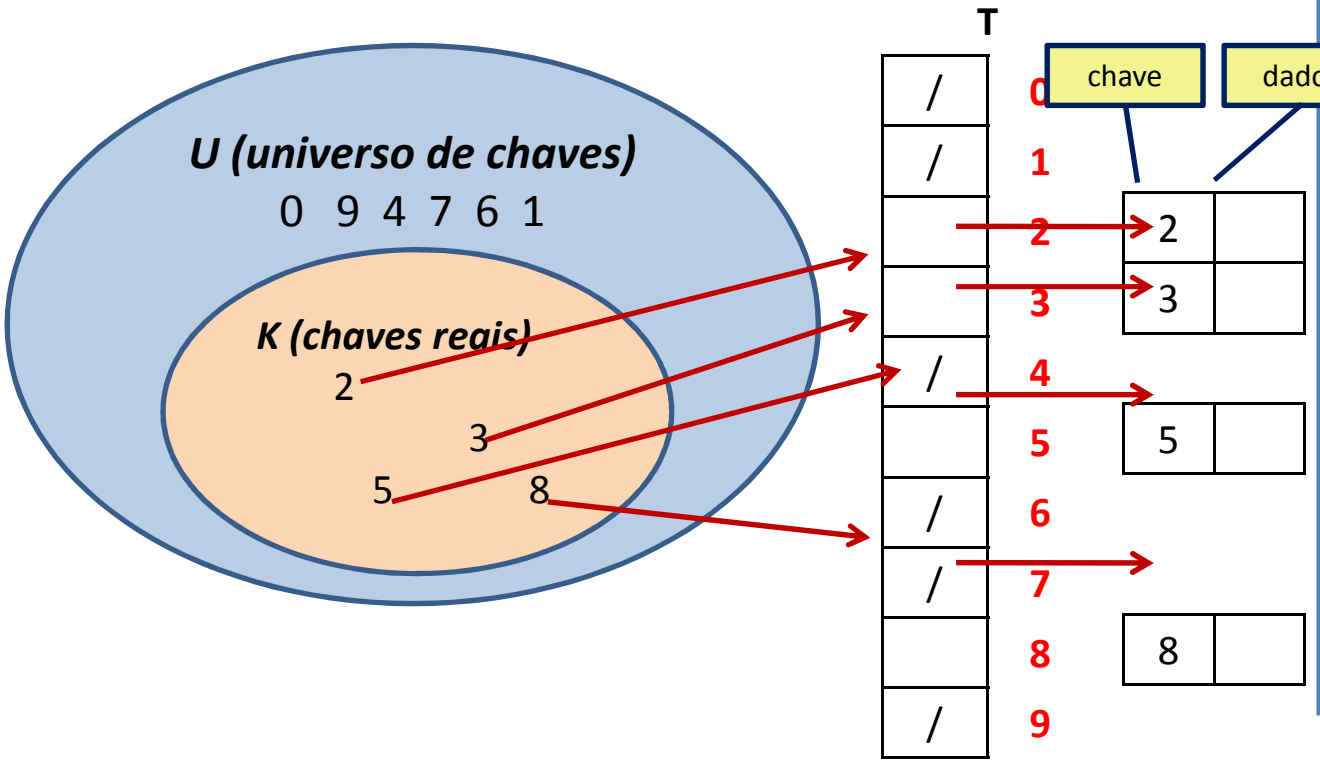

```
ED-search(T,k)
     return ???

ED-insert(T,x)
     T[???] ← x

ED-delete(T,x)
     T[???] ←
NULL
```

Endereçamento Direto

- Implementação das operações de dicionário:



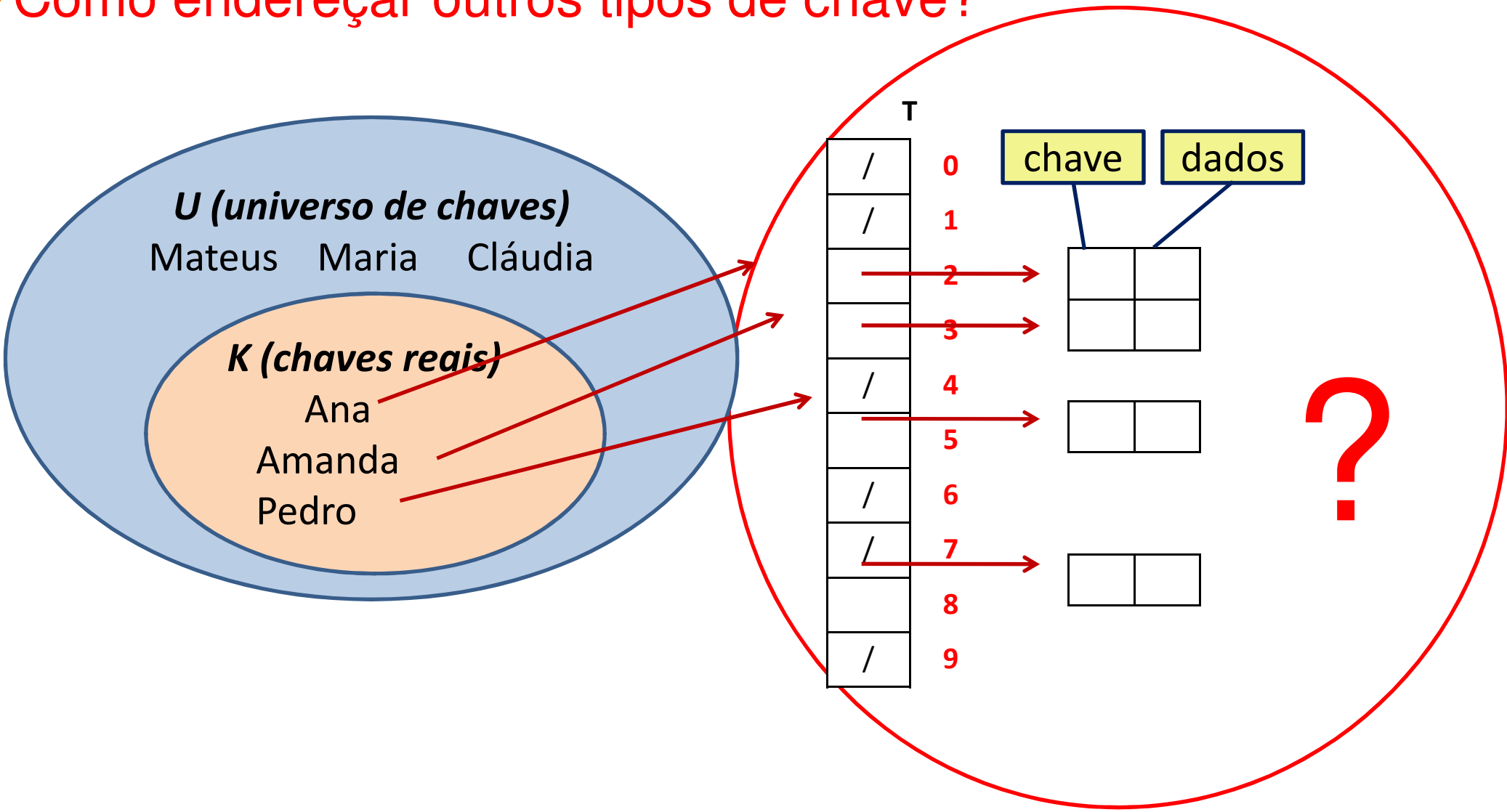

```
ED-search(T,k)
 return T[k]

ED-insert(T,x)
 T[chave[x]] ← x

ED-delete(T,x)
 T[chave[x]] ← NULL
```

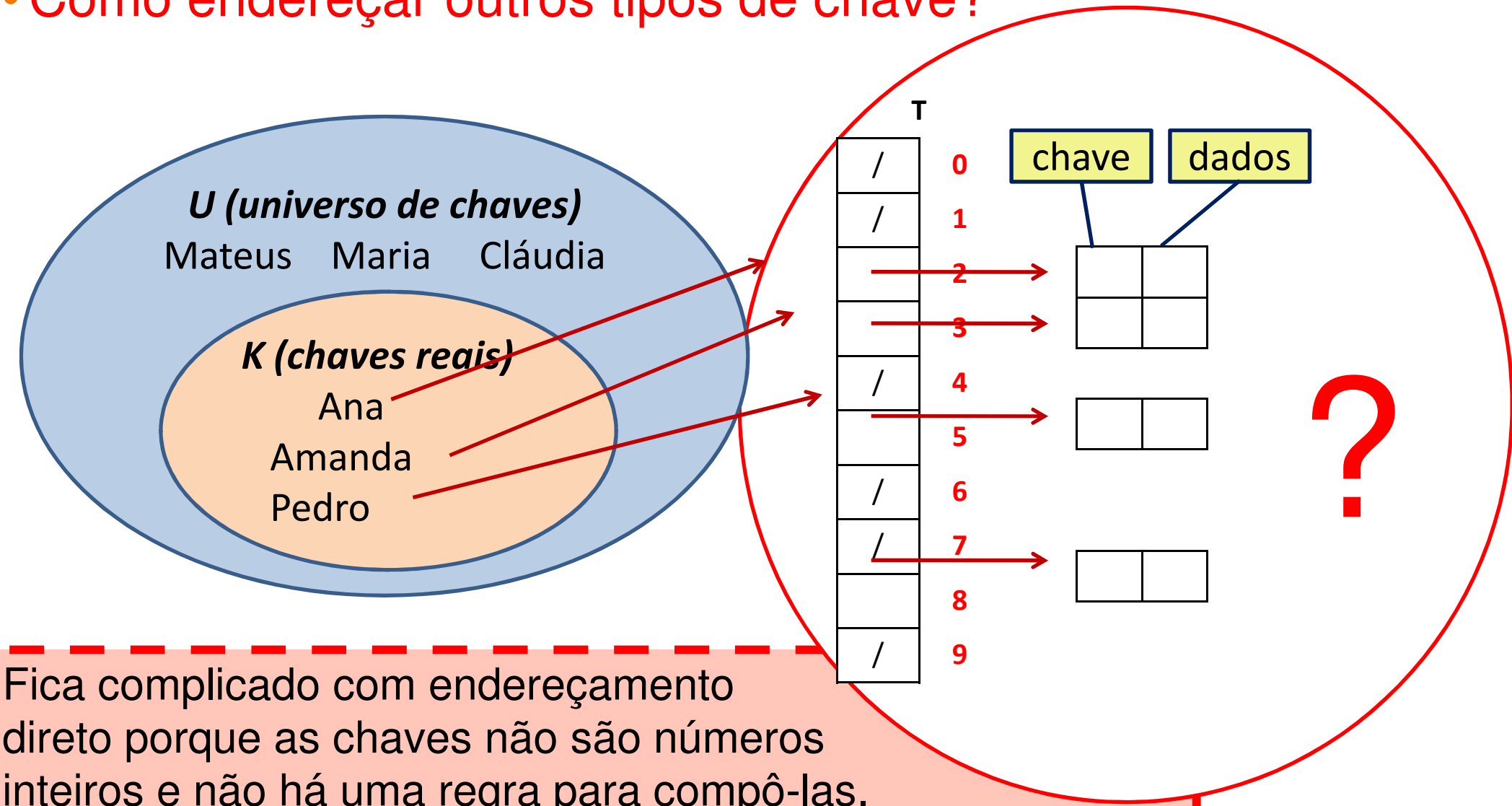
Endereçamento Direto

- Como endereçar outros tipos de chave?



Endereçamento Direto

- Como endereçar outros tipos de chave?



Fica complicado com endereçamento direto porque as chaves não são números inteiros e não há uma regra para compô-las. Teríamos que fazer busca sequencial para encontrar a localização de uma chave. **Tempo= O(n)**

Endereçamento Direto

- Tempo das operações *insert*, *delete* e *search*: **O(1)**

- Quando o endereçamento direto funciona bem?

Endereçamento Direto

- Tempo das operações *insert*, *delete* e *search*: **O(1)**

- Quando o endereçamento direto funciona bem?
  - quantidade de chaves do universo *U* é razoavelmente pequeno e facilmente mapeadas*;*
  - não há dois elementos com a mesma chave.

Endereçamento Direto

- Permite operações inserção, eliminação e buscas com tempo = O(1)

- Método limitado:
  - universos de chaves limitado
  - tabela de chaves de endereçamento tem tamanho do universo
  - exige chaves que possam ser mapeadas com custo O(1)

- Para resolver: estender conceito de endereçamento direto nas tabelas *hash*

Tabelas *hash*

- Grande problema do endereçamento direto:
  - se o universo ***U*** é grande: armazenamento de uma tabela **T** de tamanho [U] pode ser inviável e até mesmo impossível.
  - conjunto ***k*** de chaves realmente usada é muito menor que o conjunto possível de chaves do universo $U \Rightarrow$ maior parte do espaço alocado para **T** seria desperdiçada.
  - *Exemplo:*
    - Armazenar os nomes próprios dos meus clientes com no máximo 30 posições
    - Quantas combinações são possíveis obter com 30 posições, sendo que cada uma pode combinar 26 letras diferentes do alfabeto?

Tabelas *hash*

- *Exemplo:*
  - Quantas combinações são possíveis obter com 30 posições?
    - Se variarmos somente a primeira posição: 26 strings diferentes
    - Se variarmos a segunda posição: mais 26 strings
    - E assim por diante...
    - A quantidade de combinações é :
      $26 + 26^2 + 26^3 + ... + 26^{29} + 26^{30}$

Tabelas *hash*

- Quando o conjunto *k* de chaves é muito menor que o Universo, as tabelas *hash* exigem espaço de armazenamento muito menor que as tabelas de endereçamento direto.
- O tempo ***médio*** de pesquisa continua sendo O(1)
- Endereçamento direto: elemento com chave *k* é armazenado na posição ***???***

Tabelas *hash*

- Quando o conjunto *k* de chaves é muito menor que o Universo, as tabelas *hash* exigem espaço de armazenamento muito menor que as tabelas de endereçamento direto.
- O tempo ***médio*** de pesquisa continua sendo O(1)
- Endereçamento direto: elemento com chave *k* é armazenado na posição ***k***

Tabelas *hash*

- Quando o conjunto *k* de chaves é muito menor que o Universo, as tabelas *hash* exige espaço de armazenamento muito menor que as tabelas de endereçamento direto.
- O tempo ***médio*** de pesquisa continua sendo O(1)
- Endereçamento direto: elemento com chave *k* é armazenado na posição ***k***
- Tabela *hash*: elemento com chave *k* é armazenado na posição ***h(k).***
  - ***h(k)*** é chamada da função *hash* e é usada para calcular a posição da chave *k;*
  - ***h*** mapeia o universo *U* de chaves nas posições de uma *tabela hash* ***T[0..m-1]***

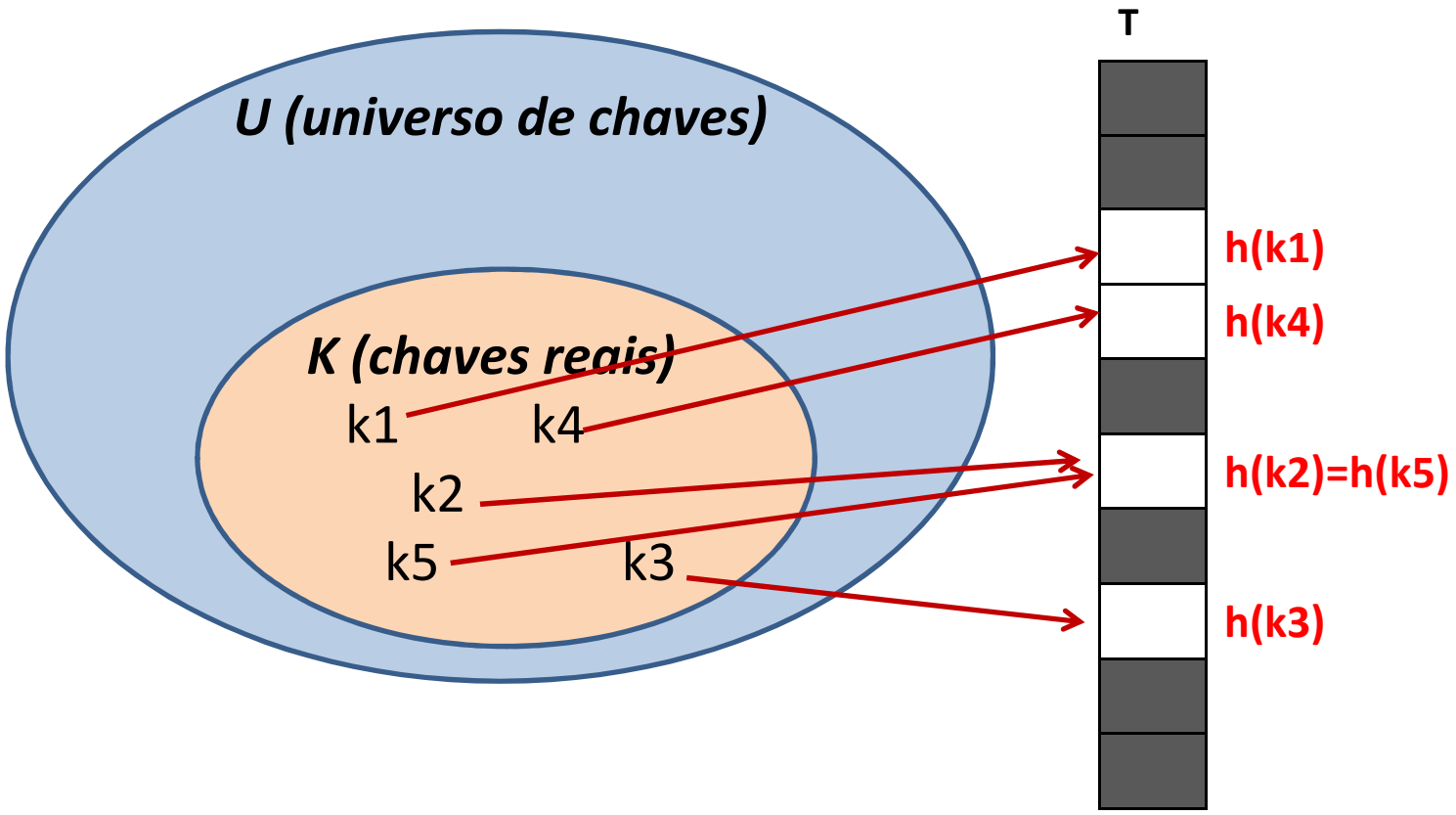
```
h: U X {0, 1,..., m–1}
```

Tabelas *hash*

- Termos comuns:
  - um elemento com chave *k* **efetua o *hash*** para a posição *h(k)*;
  - *h(k)* é o **valor *hash*** da chave *k.*
- Finalidade da função *hash:* reduzir o intervalo de índices de arranjos que precisam ser tratados.
- Função *h* deve ser determinística: uma dada entrada *k* deve sempre produzir a mesma saída *h(k).*
- Detalhe: duas chaves podem ter o *hash* na mesma posição**:** **colisão**.

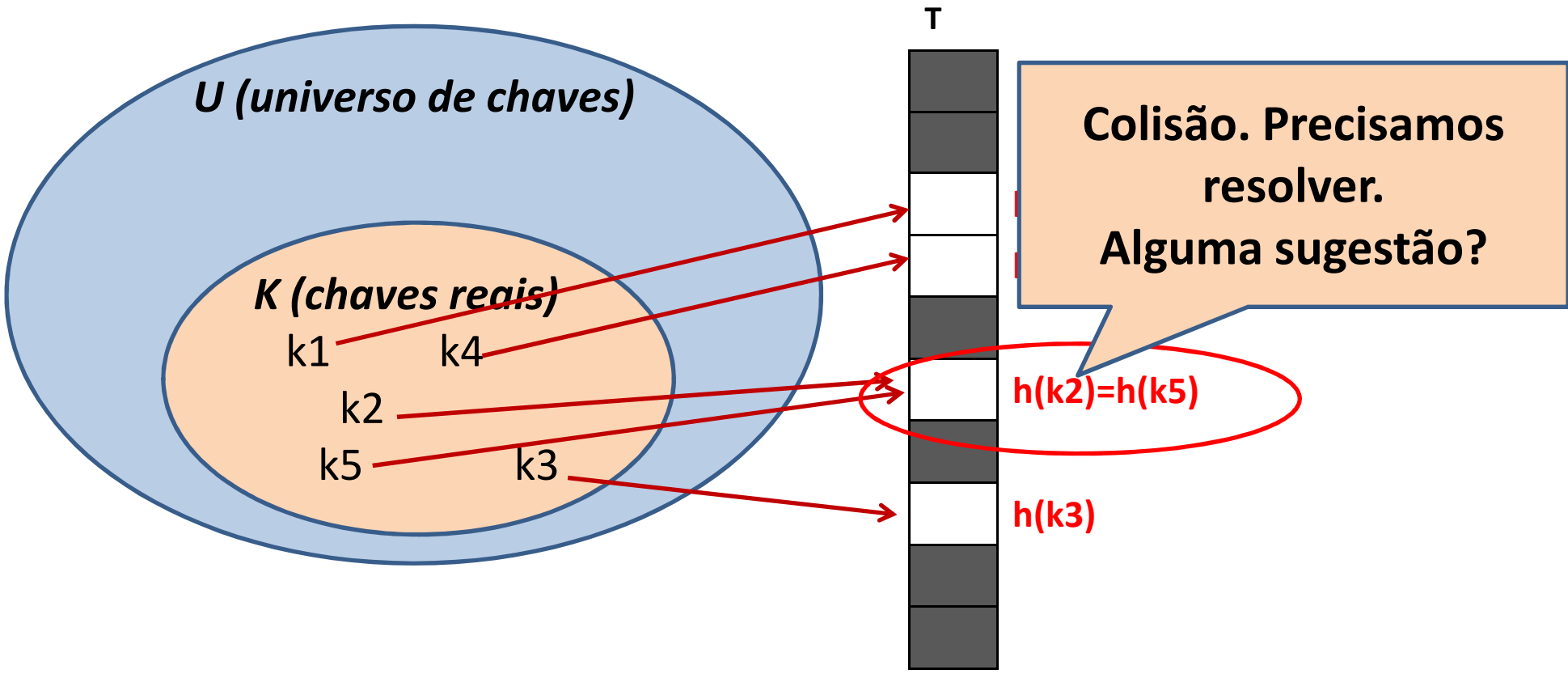
Tabelas *hash*

- Detalhe: duas chaves podem ter o *hash* na mesma posição: **colisão**.



Tabelas *hash*

- Detalhe: duas chaves podem ter o *hash* na mesma posição: **colisão**.



Resolução de colisões por encadeamento

- Todos os elementos que têm *hash* para a mesma posição são colocados em uma lista linear ligada.

- Lista linear:
  - estrutura de dados que implementa operações de inserir, eliminar e buscar;
  - estruturas dinâmicas e flexíveis: podem aumentar e diminuir de tamanho durante execução do programa;
  - estrutura muito útil para alocação dinâmica de memória - quando não é possível prever a quantidade necessária de memória para uma determinada aplicação.

Lista linear

- Sequência de zero ou mais elementos $x_1, x_2, ..., x_n$ na qual x_i é de um determinado tipo e n representa o tamanho da lista linear.
- Em geral, x_i precede x_{i+1} para i = 0, 1, 2, 3, ..., n-1.
- Analogamente, x_i sucede x_{i-1} para i = 1, 2, 3, ..., n.
- Elemento x_i pode ser de qualquer tipo – geralmente é um objeto.

Lista linear

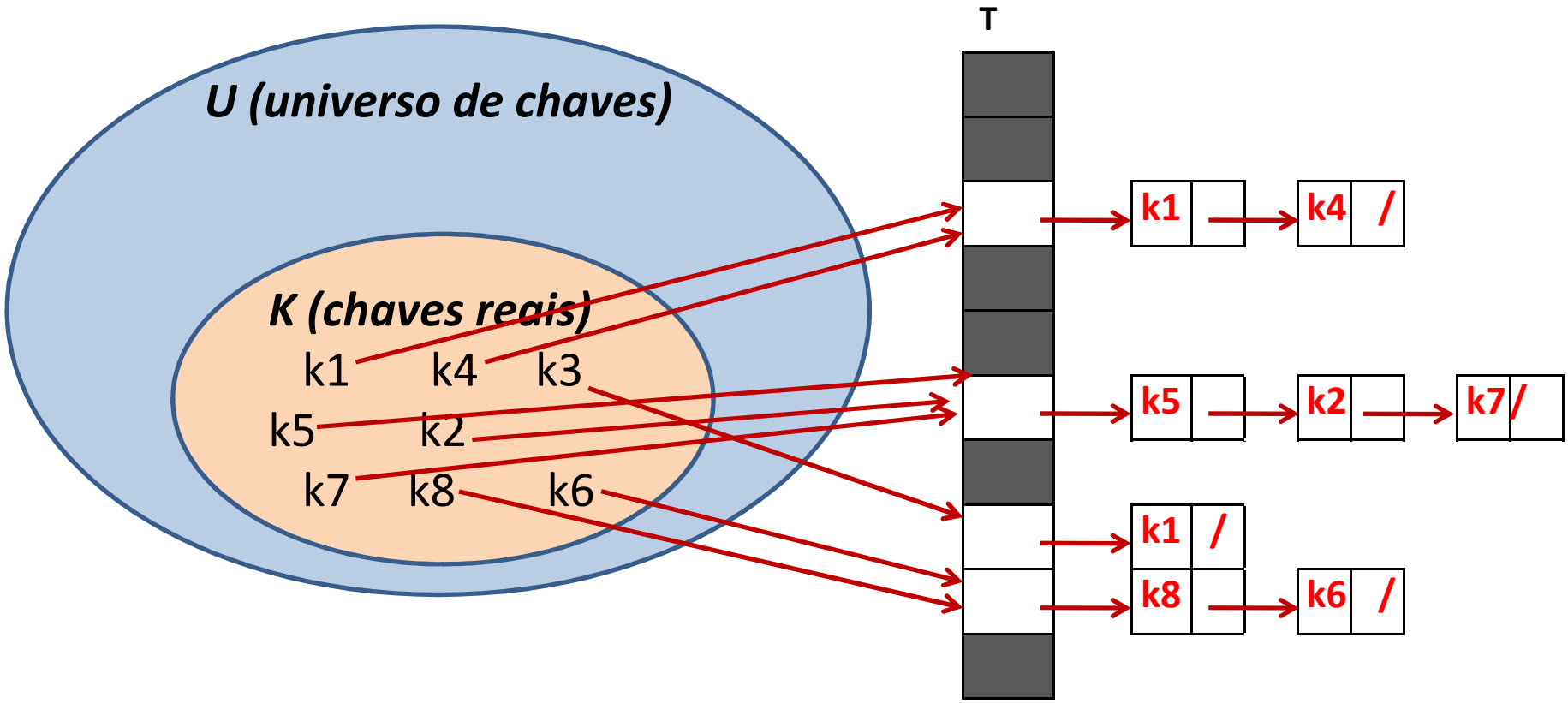
- São necessárias algumas operações sobre um objeto do tipo **Lista** a fim de permitir seu uso em tabelas *hash*:
  - insereNaLista(x): insere x no início da lista.
  - eliminaDaLista(x): verifica se x pertence à lista e retira-o da lista.
  - buscaDaLista(x): verifica se o elemento x pertence à lista.
  - estaVazia(): retorna verdadeiro ou falso dependendo se a lista está vazia ou não.
  - tamanhoDaLista(): retorna o número de elementos da lista.

- Java tem uma classe **LinkedList** que implementa esse conjunto de operações.

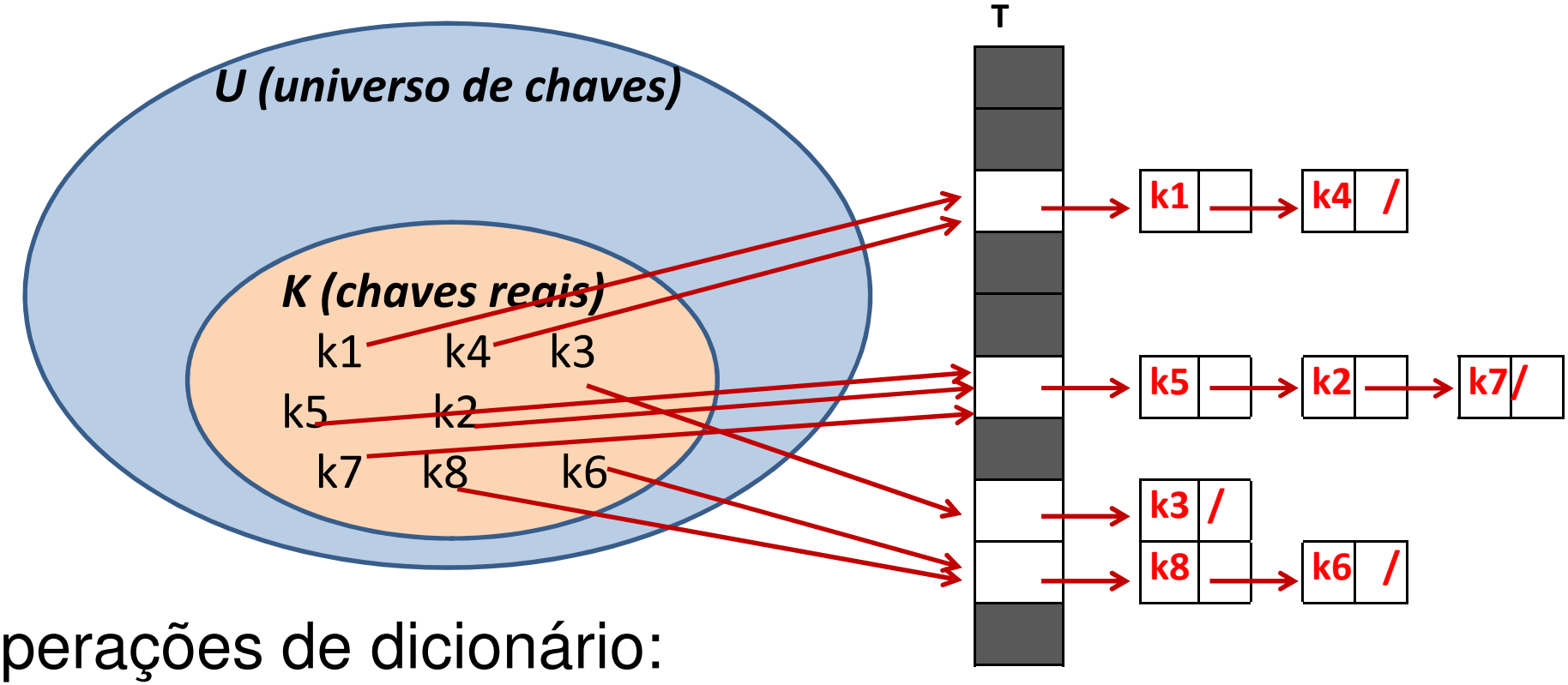
Lista linear

- Java tem uma classe **LinkedList** que implementa esse conjunto de operações.
  - insereNaLista(x): insere x no início da lista.
    - ✓ **void addFirst(Object o)**
  - eliminaDaLista(x): verifica se x pertence à lista e retira-o da lista.
    - ✓ **boolean remove(Object o)**
  - buscaDaLista(x): verifica se o elemento x pertence à lista.
    - ✓ **boolean contains(Object o)**
  - estaVazia(): retorna verdadeiro ou falso dependendo se a lista está vazia ou não.
    - ✓ **Implementar um método que faça "*return size() == 0*";**
  - tamanhoDaLista(): retorna o número de elementos da lista.
    - ✓ **int size()**

Resolução de colisões por encadeamento



Resolução de colisões por encadeamento



- Operações de dicionário:

```
LE-insert(T,x)
     insere x no início da lista T[h(chave[x])] ← x

LE-search(T,k)
     procura por um elmento com a chave k na lista T[h(k)]

LE-delete(T,x)
     elimina x da lista T[h(chave[x])]
```

Resolução de colisões por encadeamento

- Exemplo:
  - Construir um dicionário dinâmico que pode conter pares do seguinte tipo: (String nome, String significado) onde a chave é nome.
  - Strings podem ter no máximo 8 letras.
  - O dicionário dinâmico, apesar de possuir como chave qualquer String de 8 de letras, na prática vai armazenar em média uns 500 elementos.
  - Este dicionário deve ter as seguintes funções: insere, elimina e busca.

Resolução de colisões por encadeamento

- O que fazer:
  - Criar a tabela *hash.*
  - Encontrar uma função *hash* ***h(s)*** que mapeia qualquer String *s* para um número entre 0 e 500.
  - Esta função deve ter complexidade assintótica (1), como o endereçamento direto.
  - Tratar as possíveis colisões... porque o número de 500 elementos é uma média. Se houver, mais de 500 inevitavelmente haverá colisões.

Resolução de colisões por encadeamento

- Função *hash*:

```
private int hash(String nome) {
   int somaValCar = 0;
   int i;

   // Soma os valores dos caracteres
    for (i = 0; i < nome.length() && i < 8; i++)
    // nome contem somente letras e nome.length <= 8
    somaValCar += nome.charAt(i);

   return (somaValCar % 501);
  }
```

Resolução de colisões por encadeamento

- Tempo de execução no pior caso:
  - inserção = O(1)
  - pesquisa = proporcional ao tamanho da lista encadeada
  - eliminação = encontrar e eliminar o elemento = mesmo tempo da pesquisa

Resolução de colisões por encadeamento

- Qual a qualidade da execução de *hash* com encadeamento?
- Quanto tempo leva para procurar um elemento com uma determinada chave?
  - Depende da função *h.*
  - **Pior caso**: todas as chaves executam o *hash* na mesma posição: O(n) + tempo para calcular função *hash.*
  - **Caso médio**: depende de como a função *h* distribui o conjunto de chaves a serem armazenadas entre as *m* posições, em média.
  - Supondo que qualquer elemento tem igual probabilidade de efetuar o *hash* para qualquer uma das *m* posições, temos o chamado *hash uniforme simples.*

Resolução de colisões por encadeamento

- Dada uma tabela *T* com *m* posições que armazena *n* elementos, definimos o **fator de carga** α como: n/m
- α = número médio de elementos armazenados em uma cadeia.
- Considerando o resultado da pesquisa:
  - pesquisa não foi bem sucedida (se não houver elemento igual a chave): $\Theta(1+\alpha)$ considerando *hash uniforme simples;*
  - pesquisa foi bem sucedida (se houver elemento igual a chave): $\Theta(1+\alpha)$, na média, considerando *hash uniforme simples.*

***Provas: Livro do Cormen* et al. pág. 183.**

Resolução de colisões por encadeamento

- Como desenvolver uma função *hash* de boa qualidade?
  - Boa qualidade: satisfaz aproximadamente *hash uniforme simples.*
  - Há várias técnicas - não serão vistas neste curso.

Quem quiser aprofundar*: Livro do Cormen* et al. pág. 185.

Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Algoritmos - Tradução da 2a. Edição Americana. Editora Campus, 2002.
- Nota de aulas do professor Delano Beder (EACH-USP).

Hashing
Endereçamento Direto
Tabelas Hash

Professora:

Fátima L. S. Nunes