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 $U=\{0,1,..., m-1\}$, 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*

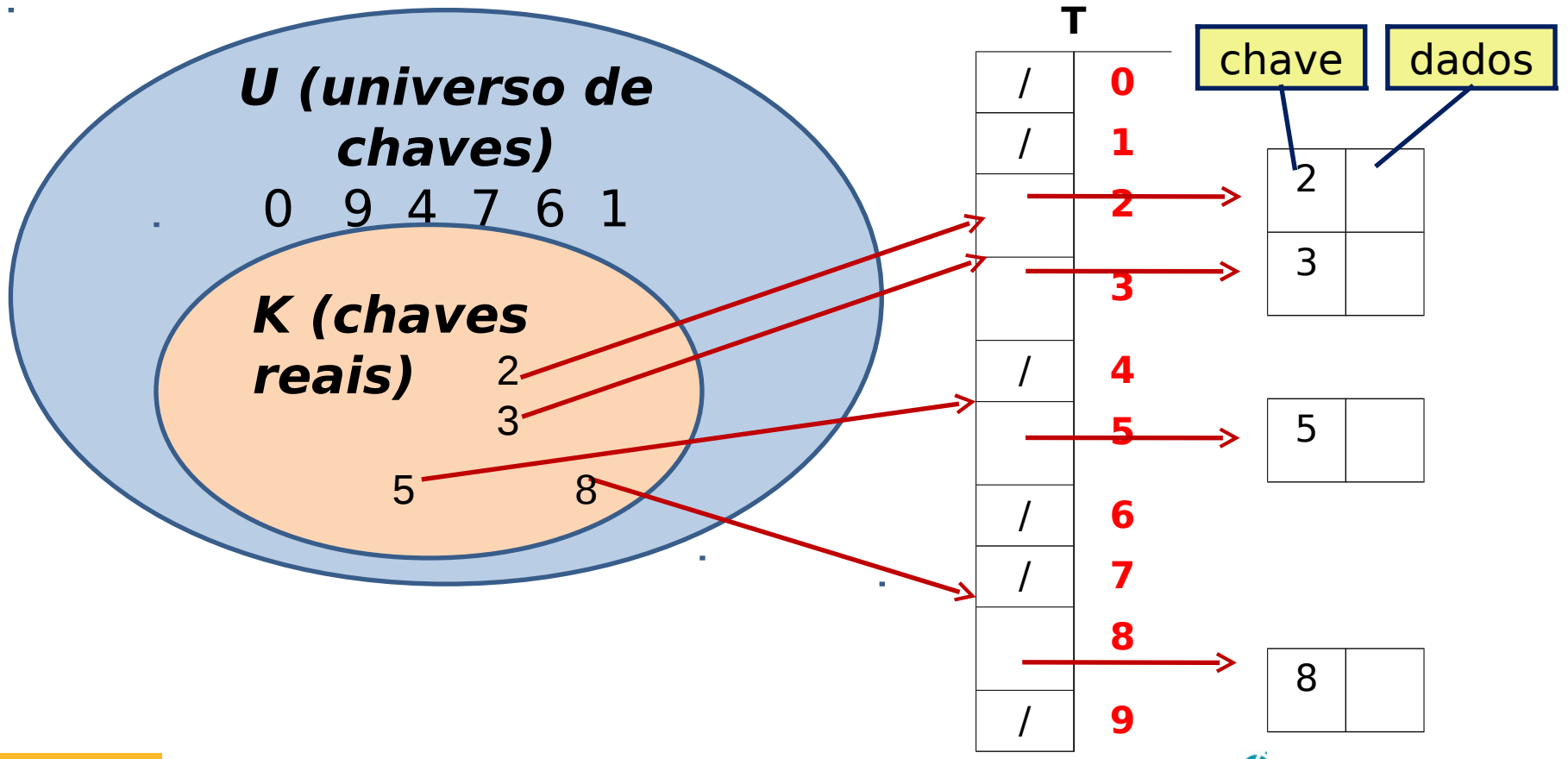
Endereçamento Direto

- 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 partir [illegible]iverso $U=\{0,1,..., m-1\}$, onde m [illegible]
  - não há elementos com a m[illegible]
  - para representar o conjunt[illegible] **endereçamento direto T[illegible]**
    - 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 conjunt 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?

Código	Ingresso	Curso	Nome	
8956495	2014/1	97001	Afonso Celso Penze Nunes da Cunha	Matriculado
9075205	2014/1	55051	Alair Jose de Souza Junior	Matriculado
9364819	2015/1	55051	Allan Ribeiro da Costa	Matriculado
9391831	2015/1	55051	Altair Fernando Pereira Junior	Matriculado
9313197	2015/1	18045	Anderson Hiroshi de Siqueira	Matriculado
8936694	2015/1	55051	Andre Merino Jorge	Matriculado
9292816	2015/1	55051	Andreia de Barros Carpi	Matriculado
8937510	2014/1	55041	Arnaldo Lopes Stanzani	Matriculado
7992894	2014/1	18045	Arthur Demarchi	Matriculado
8624525	2014/1	55051	Artur Artimonte	Matriculado
9292858	2015/1	55051	Bruno Bacelar Abe	Matriculado
9378996	2015/1	55051	Bruno Felipe Barbosa Pereira	Matriculado
9292910	2015/1	55051	Bruno Henrique Rasteiro	Matriculado
7656533	2011/1	55041	Bruno Molina Rosaboni	Matriculado
8957012	2014/1	18045	Cainã de Oliveira Figares	Matriculado

Endereçamento Direto

- Como endereçar outros tipos de chave?

Código	Ingresso	Curso	Nome	
8956495	2014/1	97001	Afonso Celso Penze Nunes da Cunha	Matriculado
9075205	2014/1	55051	Alair Jose de Souza Junior	Matriculado
9364819	2015/1	55051	Allan Ribeiro da Costa	Matriculado
9391831	2015/1	55051	Altair Fernando Pereira Junior	Matriculado
9313197	2015/1	18045	Anderson Hiroshi de Siqueira	Matriculado
8936694	2015/1	55051	Andre Merino Jorge	Matriculado
9292816	2015/1	55051	Andreia de Barros Carpi	Matriculado
8937510	2014/1	55041	Arnaldo Lopes Stanzani	Matriculado
7992894	2014/1	18045	Arthur Demarchi	Matriculado
8624525	2014/1	55051	Artur Artimonte	Matriculado
9292858	2015/1	55051	Bruno Bacelar Abe	Matriculado
9378996	2015/1	55051	Bruno Felipe Barbosa Pereira	Matriculado
9292910	2015/1	55051	Bruno Henrique Rasteiro	Matriculado
7656533	2011/1	55041	Bruno Molina Rosaboni	Matriculado
8957012	2014/1	18045	Cainã de Oliveira Figares	Matriculado

Valor máximo código: 9391831

Endereçamento Direto

- Como endereçar outros tipos de chave?

Código	Ingresso	Curso	Nome	
8956495	2014/1	97001	Afonso Celso Penze Nunes da Cunha	Matriculado
9075205	2014/1	55051	Alair Jose de Souza Junior	Matriculado
9364819	2015/1	55051	Allan Ribeiro da Costa	Matriculado
9391831	2015/1	55051	Altair Fernando Pereira Junior	Matriculado
9313197	2015/1	18045	Anderson Hiroshi de Siqueira	Matriculado
8936694	2015/1	55051	Andre Merino Jorge	Matriculado
9292816	2015/1	55051	Andreia de Barros Carpi	Matriculado
8937510	2014/1	55041	Arnaldo Lopes Stanzani	Matriculado
7992894	2014/1	18045	Arthur Demarchi	Matriculado
8624525	2014/1	55051	Artur Artimonte	Matriculado
9292858	2015/1	55051	Bruno Bacelar Abe	Matriculado
9378996	2015/1	55051	Bruno Felipe Barbosa Pereira	Matriculado
9292910	2015/1	55051	Bruno Henrique Rasteiro	Matriculado
7656533	2011/1	55041	Bruno Molina Rosaboni	Matriculado
8957012	2014/1	18045	Cainã de Oliveira Figares	Matriculado

Valor máximo código: 9391831

Endereçamento Direto

- Como endereçar outros tipos de chave?

Código	Ingresso	Curso	Nome	
8956495	2014/1	97001	Afonso Celso Penze Nunes da Cunha	Matriculado
9075205	2014/1	55051	Alair Jose de Souza Junior	Matriculado
9364819	2015/1	55051	Allan Ribeiro da Costa	Matriculado
9391831	2015/1	55051	Altair Fernando Pereira Junior	Matriculado
9313197	2015/1	18045	Anderson Hiroshi de Siqueira	Matriculado
8936694	2015/1	55051	Andre Merino Jorge	Matriculado
9292816	2015/1	55051	Andreia de Barros Carpi	Matriculado
8937510	2014/1	55041	Arnaldo Lopes Stanzani	Matriculado
7992894	2014/1	18045	Arthur Demarchi	Matriculado
8624525	2014/1	55051	Artur Artimonte	Matriculado
9292858	2015/1	55051	Bruno Bacelar Abe	Matriculado
9378996	2015/1	55051	Bruno Felipe Barbosa Pereira	Matriculado
9292910	2015/1	55051	Bruno Henrique Rasteiro	Matriculado
7656533	2011/1	55041	Bruno Molina Rosaboni	Matriculado
8957012	2014/1	18045	Cainã de Oliveira Figares	Matriculado

Valor máximo código: 9391831 Valor mínimo: 3144931
Diferença: 6246900

Endereçamento Direto

- Como endereçar outros tipos de chave?

Código	Ingresso	Curso	Nome	
8956495	2014/1	97001	Afonso Celso Penze Nunes da Cunha	Matriculado
9075205	2014/1	55051	Alair Jose de Souza Junior	Matriculado
9364819	2015/1	55051	Allan Ribeiro da Costa	Matriculado
9391831	2015/1	55051	Altair Fernando Pereira Junior	Matriculado
9313197	2015/1	18045	Anderson Hiroshi de Siqueira	Matriculado
8936694	2015/1	55051	Andre Merino Jorge	Matriculado
9292816	2015/1	55051	Andreia de Barros Carpi	Matriculado
8937510	2014/1	55041	Arnaldo Lopes Stanzani	Matriculado
7992894	2014/1	18045	Arthur Demarchi	Matriculado
8624525	2014/1	55051	Artur Artimonte	Matriculado
9292858	2015/1	55051	Bruno Bacelar Abe	Matriculado
9378996	2015/1	55051	Bruno Felipe Barbosa Pereira	Matriculado
9292910	2015/1	55051	Bruno Henrique Rasteiro	Matriculado
7656533	2011/1	55041	Bruno Molina Rosaboni	Matriculado
8957012	2014/1	18045	Cainã de Oliveira Figares	Matriculado

Valor máximo código: 9391831 Valor mínimo: 3144931
Diferença: 6246900 Número de matriculados: 71

0 6246900

Endereçamento Direto

- Como “mapear” o código para um intervalo pequeno de endereços?

Endereçamento Direto

- Como “mapear” o código para um intervalo pequeno de endereços?
- Por exemplo, vamos usar os dois últimos algarismos do código
- Mapeamos no intervalo 0 – 99

Mapeamento de endereço

- Como “mapear” o código para um intervalo pequeno de endereços?
- Por exemplo, vamos usar os dois últimos algarismos do código
- Mapeamos no intervalo 0 – 99

Mapeamento de endereço

8956495
9075205
9364819
9391831
9313197
8936694
9292816
8937510
7992894
8624525

5	9075205
10	8937510
16	9292816
19	9364819
25	8624525
31	9391831
94	8936694
95	8956495
97	9313197

Mapeamento de endereço

8956495
9075205
9364819
9391831
9313197
8936694
9292816
8937510
7992894
8624525

5	9075205
10	8937510
16	9292816
19	9364819
25	8624525
31	9391831
94	8936694
95	8956495
97	9313197

Qual o problema que temos aqui?

Mapeamento de endereço

8956495
9075205
9364819
9391831
9313197
8936694
9292816
8937510
7992894
8624525

5	9075205
10	8937510
16	9292816
19	9364819
25	8624525
31	9391831
94	8936694
95	8956495
97	9313197

Qual o problema que temos aqui?

Mapeamento de endereço

8956495
9075205
9364819
9391831
9313197
8936694
9292816
8937510
7992894
8624525

5	9075205
10	8937510
16	9292816
19	9364819
25	8624525
31	9391831
94	8936694
95	8956495
97	9313197

Qual o problema que temos aqui?

7992894

Colisões

Código	Hash
9364802	2
9075205	5
8936606	6
6908006	6
8626207	7
9292708	8
8937510	10
9292910	10
6767010	10
8957012	12
7656512	12
9292816	16
8910517	17
9364819	19
8956619	19
8936120	20

71 alunos

- 23 colisões
- nenhuma cadeia com mais de 3 colisões

Colisões

Código	Hash
9364802	2
9075205	5
8936606	6
6908006	6
8626207	7
9292708	8
8937510	10
9292910	10
6767010	10
8957012	12
7656512	12
9292816	16
8910517	17
9364819	19
8956619	19
8936120	20

71 alunos

- 23 colisões
- nenhuma cadeia com mais de 3 colisões

Mas como poderíamos reduzir as colisões?

Colisões

- Podemos, por exemplo, usar os 3 últimos dígitos
- E aumentar o tamanho da tabela

Código	Hash
6908006	6
6767010	10
8957012	12
8124062	62
8936120	120
8083126	126
8084127	127
8955132	132
8957155	155
9313197	197
9075205	205
8626207	207
8626228	228
8957263	263
6792263	263
8532280	280
6608332	332
8602357	357

Colisões

- Podemos, por exemplo, usar os 3 últimos dígitos
- E aumentar o tamanho da tabela

Código	Hash
6908006	6
6767010	10
8957012	12
8124062	62
8936120	120
8083126	126
8084127	127
8955132	132
8957155	155
9313197	197
9075205	205
8626207	207
8626228	228
8957263	263
6792263	263
8532280	280
6608332	332
8602357	357

Temos uma única colisão.
Mas temos uma tabela de tamanho 1000.

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 ***k***

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***
- 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]***

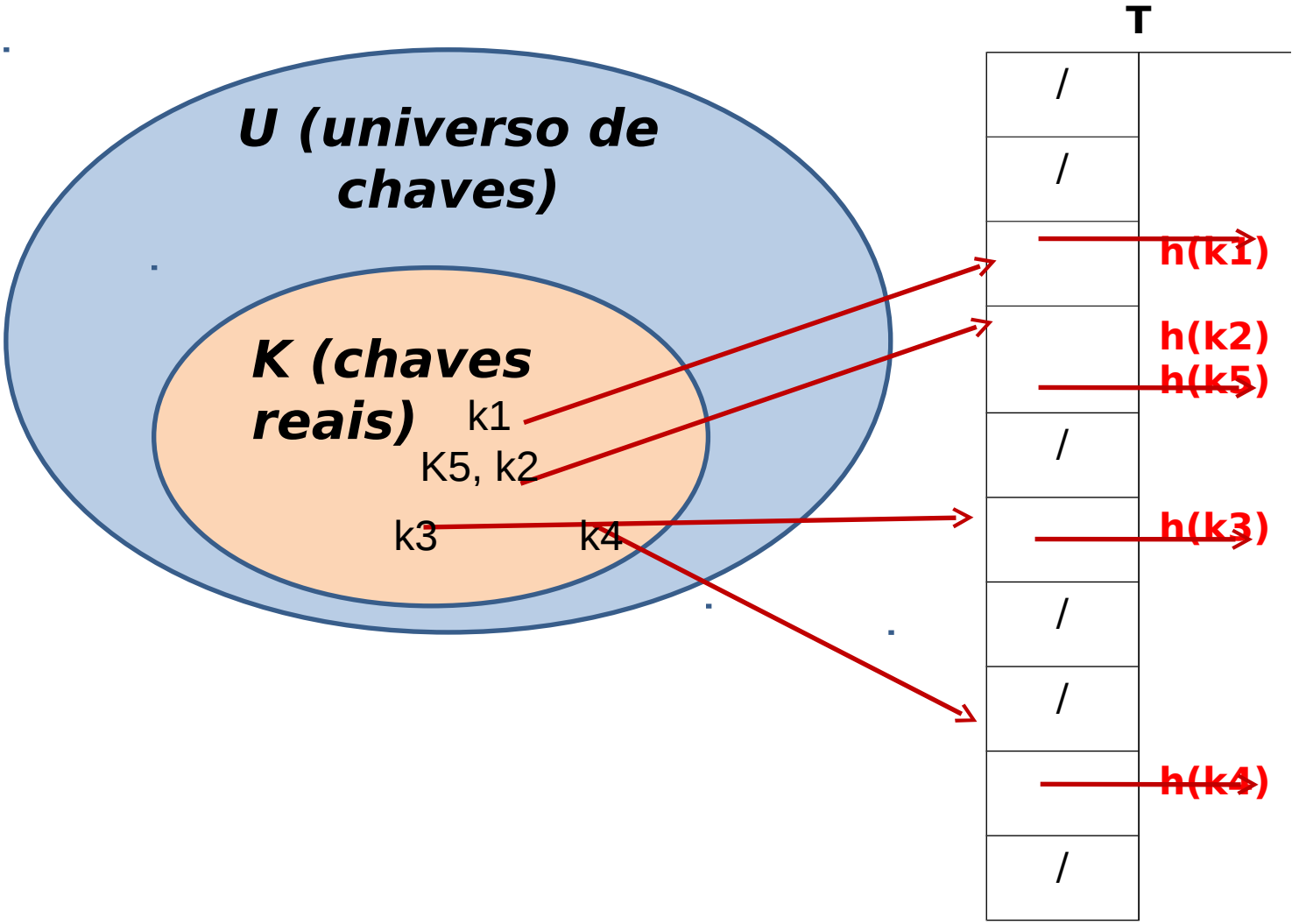
```
h: U X {0, 1,..., m-1}
```

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**.

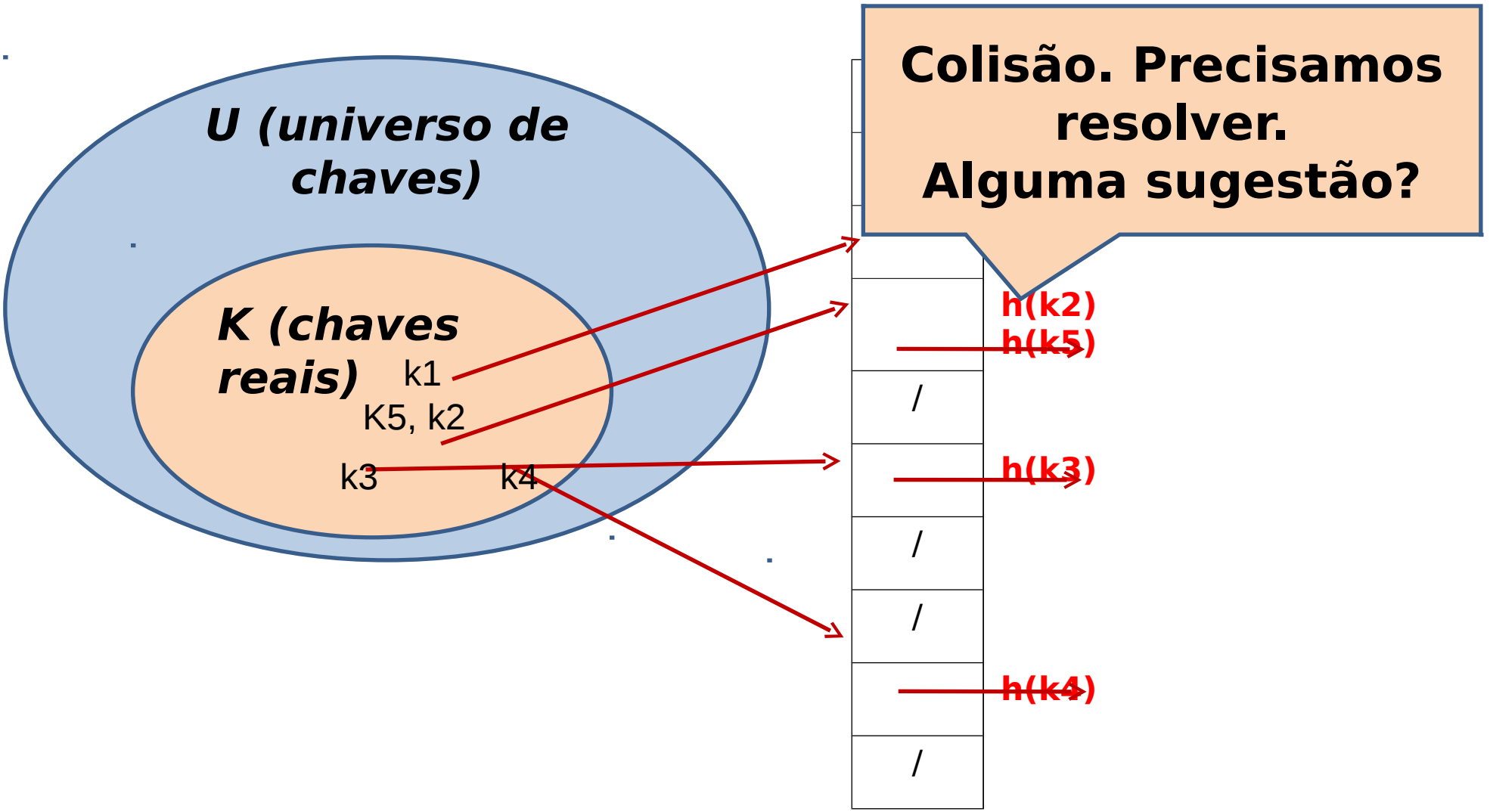
Tabelas *hash*

- 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 $x_1, x_2, \ldots, x_n$ 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 = 1, 2, 3, ..., n-1.
- Analogamente, x_i sucede x_{i-1} para i = 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.

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 elemento 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.
  - `nome` pode 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*:
- Vamos somar o valor dos caracteres do string
- A soma fazemos módulo 501
- Obtemos valor entre 0 e 500

Resolução de colisões por encadeamento

- Função *hash*:

```
int hash(char nome[]) {
  int soma = 0;
  int i, len = strlen(nome);

  // Soma os valores dos caracteres
 for (i = 0; i < len; i++)
   // nome contem somente letras e nome.length <= 8
   soma += nome[i];

  return (soma % 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.

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