

# ALGORITMOS E ESTRUTURAS DE DADOS 2023/2024 DICIONÁRIO

---

Armanda Rodrigues

18 de outubro de 2023

# Guardar todos os documentos da biblioteca

- Vamos voltar ao nosso exemplo da biblioteca
- Os utilizadores do sistema da biblioteca devem poder executar as seguintes tarefas:
  - Inserir um novo documento na biblioteca – esta operação recebe toda a informação relativa a um documento e só deve ter sucesso se a **cota** do documento não existe já na biblioteca  
se o objeto existe
  - Remoção de um documento da biblioteca – esta operação só deve ter sucesso se a cota do documento existe na biblioteca e deve devolver o documento removido
  - Aceder à informação associada a um documento da biblioteca - esta operação só deve ter sucesso se a cota do documento existe na biblioteca



---

TAD Biblioteca

# Interface Library – incompleto (1)

```
package library;
```

```
public interface Library {
```

```
.....
```

```
    // Add new Book to library
```

```
    // Requires: documentCode does not identify an existing document
```

```
    void addNewBook(String title, String subject, String documentCode,  
                    String publisher, String author, long ISBN)
```

```
        throws ExistingDocException;
```

```
    // Add new Journal to library
```

```
    // Requires: documentCode does not identify an existing document
```

```
    void addNewJournal(String title, String subject, String documentCode,  
                      String publisher, int ISSN, String URL)
```

```
        throws ExistingDocException;
```

```
}
```

# Interface Library – incompleto (2)

[illegible]

# Library – perguntas

- O que representa a String documentCode neste sistema ? identificador (único) do documento
  - Que características deve ter? Título autor
- O que representa o tipo Document neste Sistema ? o objeto
- Quais as características que gostaríamos de ver associadas às operações apresentadas no TAD ? fácil pesquisa
- Qual deveria ser a complexidade temporal das operações associadas a implementações do TAD ? Será possível cumprir esse requisito ?

# Respostas

- A String documentCode é uma chave de acesso a um documento
- O TAD Dicionário reflete os serviços de acesso a objetos genéricos com uma chave associada
  - Uma implementação de dicionário permitirá guardar a informação relativa a todos os documentos da biblioteca
- Seria aconselhável um acesso rápido a um objeto, a partir da sua chave

---

TAD Dicionário – Acesso por chave



# TAD Dicionário

## Chaves do Tipo K e Valores do Tipo V

```
// Se existir uma entrada no dicionário cuja chave é a especificada,  
// retorna o seu valor; no caso contrário, retorna NIL.
```

```
V ∪ {NIL} pesquisa( K chave );
```

```
// Se existir uma entrada no dicionário cuja chave é a especificada,  
// substitui o seu valor pelo valor especificado e retorna o valor antigo;  
// no caso contrário, insere a entrada (chave, valor) e retorna NIL.
```

```
V ∪ {NIL} insere( K chave, V valor );
```

```
// Se existir uma entrada no dicionário cuja chave é a especificada,  
// remove-a do dicionário e retorna o seu valor;  
// no caso contrário, retorna NIL.
```

```
V ∪ {NIL} remove( K chave );
```

# Interface Dicionário (K,V) (1)

```
package dataStructures;
```

```
public interface Dictionary<K,V>{
```

```
    // Returns true iff the dictionary contains no entries.
```

```
    boolean isEmpty( );
```

```
    // Returns the number of entries in the dictionary.
```

```
    int size( );
```

```
    // Returns an iterator of the entries in the dictionary.
```

```
    Iterator<Entry<K,V>> iterator( );
```



- O dicionário guarda valores deste tipo: K refere o tipo que permite fazer o acesso (a chave), V o tipo do valor identificado pela chave
- Quando criamos um dicionário temos de saber qual o tipo da chave e qual o tipo do valor

# Interface Dicionário (K,V) (2)

```
// If there is an entry in the dictionary whose key is the specified key,  
// returns its value; otherwise, returns null.  
V find( K key );  
  
// If there is an entry in the dictionary whose key is the specified key,  
// replaces its value by the specified value and returns the old value;  
// otherwise, inserts the entry (key, value) and returns null.  
V insert( K key, V value );  
  
// If there is an entry in the dictionary whose key is the specified key,  
// removes it from the dictionary and returns its value;  
// otherwise, returns null.  
V remove( K key );  
  
} // End of Dictionary
```

---

TAD Entrada

Acesso aos campos Chave e Valor

# Interface Entrada (K,V)

interface mais perto do objeto guardado no dicionário

```
package dataStructures;

public interface Entry<K,V>{

    // Returns the key in the entry.
    K getKey( );

    // Returns the value in the entry.
    V getValue( );
}
```

---


TAD Objecto Comparável  
Comparação com outro Objecto

# Interface Objeto Comparável

## Objetos do Tipo T

```
package java.lang;
```

Comparable é um interface do pacote java.lang



```
public interface Comparable<T>{
```

```
    // Compares this object with the specified object for order.  
    // Returns a negative integer, zero, or a positive integer  
    // as this object is less than, equal to, or greater than  
    // the specified object.
```

```
    int compareTo( T object );
```

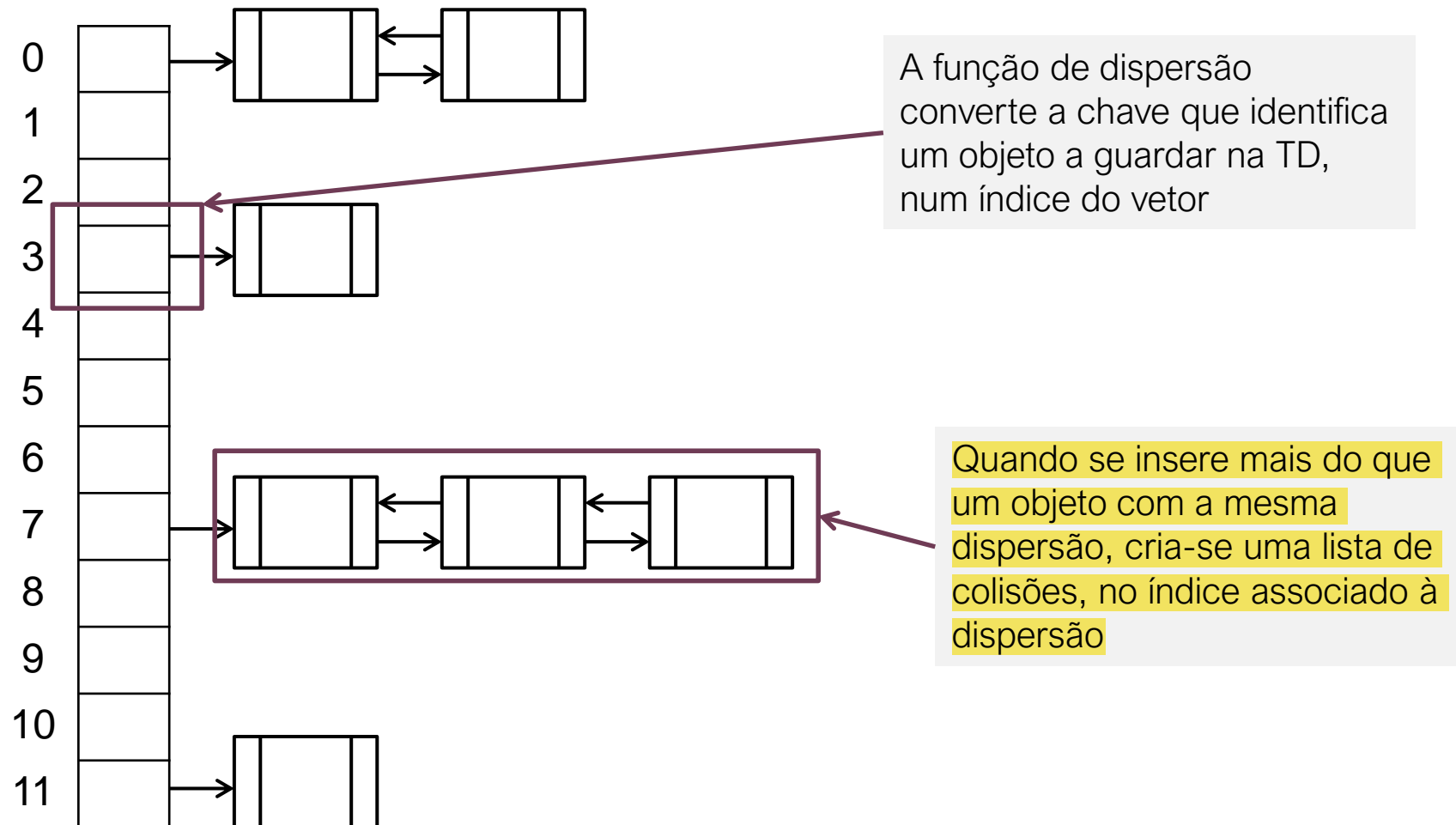
```
}
```

# Tabela de Dispersão

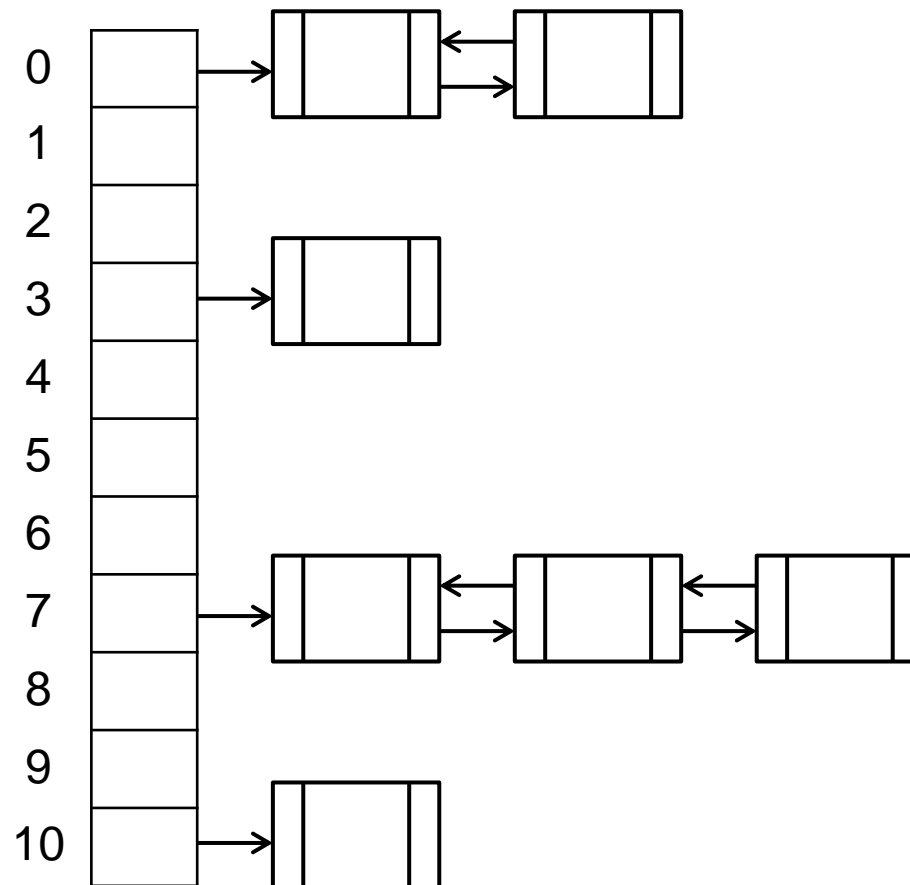
- A estrutura de dados Tabela de Dispersão é suportada pelo conceito de função de dispersão
- Dada uma chave que identifica univocamente um objeto, a função de dispersão converte a mesma chave num número inteiro dentro de um intervalo determinado
- A utilização da função de dispersão permite aceder ao objeto que a chave identifica de uma forma expedita
- A utilização da função de dispersão para localizar um objeto comporta a possibilidade de se darem colisões
  - Dois objetos diferentes podem ter a mesma dispersão, o que implica que estes dois objetos poderão ser armazenados na mesma zona
  - As colisões têm de ser tratadas e resolvidas



# Tabela de dispersão aberta

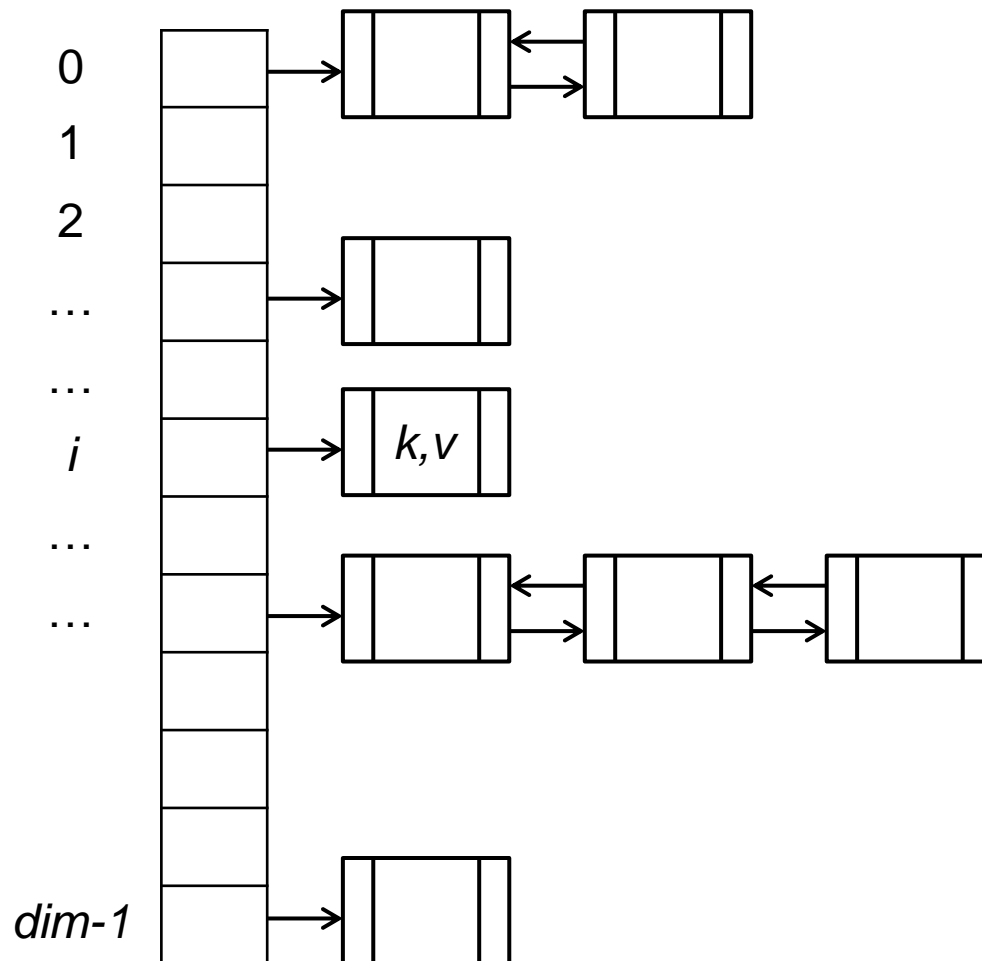


# Tabela de dispersão aberta



- **dim (11)** – é a dimensão do vetor
- **K** – é o conjunto (domínio) das chaves que identificam os objetos a guardar
- **n (7)** – número de objetos guardados na tabela
- $\lambda = n / \text{dim}$  deve ser sempre inferior a 1
- Complexidade ideal  $O(1)$
- Se  $\lambda$  for inferior a 1 e a função de dispersão for apropriada, o preenchimento da tabela será esparso e a dimensão das listas curta.
- O acesso a um elemento estará perto de  $O(1)$

# Função de Dispersão



$dispers\tilde{a}o : K \rightarrow \{ 0, 1, 2, \dots, dim-1 \}$

$dispers\tilde{a}o(k) = i$

Colisões:

Se  $\#K > dim$ ,

existem  $k1, k2 \in K$  tais que:

$k1 \neq k2$  e  $dispers\tilde{a}o(k1) = dispers\tilde{a}o(k2)$

# Função de Dispersão

- Objetivos
  - Deve ser eficiente.
  - Deve distribuir as chaves uniformemente por todas as posições da tabela.
- Regras Práticas
  - A função de dispersão deve ser simples de calcular.
  - A dimensão da tabela (*dim*) deve ser um número primo.
  - Se as chaves forem grandes, deve-se considerar apenas uma parte, oriunda de vários pontos.

# Cálculo da função de dispersão (2 passos)

- Passo 1: Cada chave sabe calcular o seu código de dispersão.

```
public int hashCode( );
```

- Passo 2: A tabela de dispersão converte o código de dispersão da chave num índice da tabela.

```
Math.abs( key.hashCode() ) % table.length
```

# Exemplos de códigos de dispersão

- Se a chave é um número inteiro  $n$  o código será o próprio  $n$ .
- Se a chave é uma cadeia de caracteres  $s_0 s_1 \dots s_{n-1}$  :
  - $s_0 + s_1 + \dots + s_{n-1}$ ;
  - $(s_0 a^{n-1} + s_1 a^{n-2} + \dots + s_{n-1}) \% b$  (onde  $a$  e  $b$  são primos)

# Regra de Horner (1)

$$\text{códigoDisp}(s_0 s_1 \dots s_{n-1}) = (s_0 a^{n-1} + s_1 a^{n-2} + \dots + s_{n-1}) \% b$$

$$v \leftarrow 0;$$

$$v \leftarrow (v * a + s_0) \% b = (s_0) \% b$$

$$v \leftarrow (v * a + s_1) \% b = (s_0 a + s_1) \% b$$

$$v \leftarrow (v * a + s_2) \% b = (s_0 a^2 + s_1 a + s_2) \% b$$

$$v \leftarrow (v * a + s_3) \% b = (s_0 a^3 + s_1 a^2 + s_2 a + s_3) \% b$$

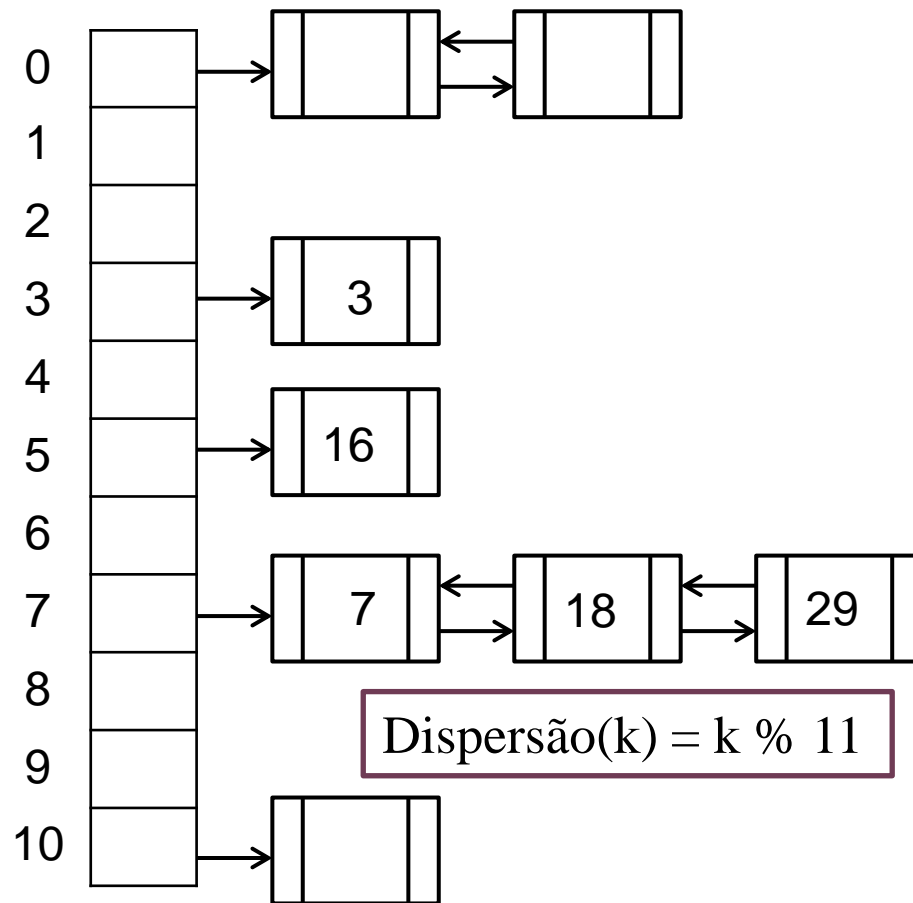
## Regra de Horner (2)

$$\text{códigoDisp}(s_0 s_1 \dots s_{n-1}) = (s_0 a^{n-1} + s_1 a^{n-2} + \dots + s_{n-1}) \% b$$

```
public static int hash( String key ){  
  
    int a = 127; // a is a prime number.  
    int b = 2147483647; // b is a prime number.  
    int hashCode = 0;  
  
    for ( int i = 0; i < key.length(); i++ )  
        hashCode = ( hashCode * a + key.charAt(i) ) % b;  
    return hashCode;  
}
```



# Tabela de Dispersão Aberta (Separate Chaining)



- **Criar a TD:** Criar o vetor de listas ligadas (do tipo dicionário) e criar todas as listas de colisões (que podem ser simples ou duplas, ordenadas ou desordenadas).
- **Pesquisar k:** Calcular  $\text{dispersão}(k)$  e pesquisar k no dicionário associado à  $\text{dispersão}(k)$
- **Inserir k,v:** Calcular  $\text{dispersão}(k)$ , inserir k,v no dicionário associado à  $\text{dispersão}(k)$
- **Remover k:** Calcular  $\text{dispersão}(k)$ , remover k do dicionário associado à  $\text{dispersão}(k)$

# Fator de Ocupação da tabela de dispersão

- Sejam:

- $n$  - o número de entradas na tabela
- $dim$  – a dimensão da tabela

- Fator de Ocupação da tabela :  $\lambda = \frac{n}{dim}$

( $\lambda$  é o comprimento médio das listas de colisões)

- $\lambda$  deve ser sempre inferior a 1

# Complexidades da Dispersão Aberta

Pesquisa	Melhor Caso	Pior Caso	Caso Esperado
Com sucesso	$O(1)$	$O(n)$	$O(1+\lambda)$
Sem sucesso	$O(1)$	$O(n)$	$O(1+\lambda)$