# Aula 11

# Estruturas de Dados

## Listas de pares chave-valor

*Programação II, 2015-2016*

*v0.9, 28-04-2015*

DETI, Universidade de Aveiro



**Objectivos:**
- Conceito de dicionário;
- Listas de pares chave-valor.

## Conteúdo





### As estruturas de dados que já vimos

- `LinkedList`
  - `addFirst()`,`addLast()`,`removeFirst()`,`first()`, ...
- `SortedList`
  - `insert()`,`remove()`,`first()`, ...
- `Stack`
  - `push()`,`pop()`,`top()`, ...
- `Queue`
  - `in()`,`out()`,`peek()`, ...



## 1 O conceito de dicionário

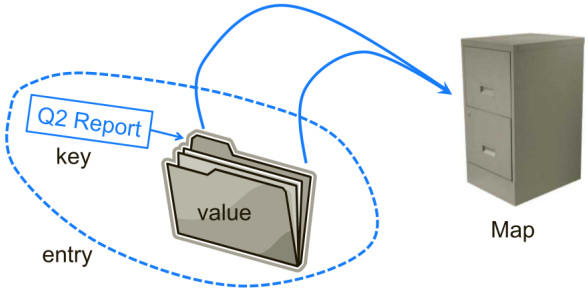
### Dicionários: o conceito

- Um **dicionário** é uma estrutura de dados que permite armazenar pares chave-valor
- Cada chave funciona como um identificador único, ou seja, no conjunto de todos os pares chave-valor, cada chave aparece apenas uma vez
- Assim, a associação de chaves a valores (ou elementos) define uma correspondência unívoca (*mapping*)
  - Por isso também se chama **mapa** (*map*)

- Tal como os índices dão acesso eficiente aos elementos armazenados num vector, também aqui as chaves permitem implementar um acesso eficiente aos elementos que lhes estão associados
  - Por isso, também se chama **vector associativo** (*associative array*)
  - A eficiência do acesso por chave depende da implementação

11.4

## Dicionários: ilustração




- Cada pasta contém uma determinada etiqueta no topo (chave) e determinado conteúdo no seu interior (o valor ou elemento associado)
- O conjunto de pastas com as respectivas etiquetas são arrumadas no armário (dicionário)

11.5

## Dicionários: exemplos de aplicação

- Um sistema de informação sobre estudantes da universidade usa o identificador único (nº mecanográfico) de cada estudante como chave de acesso à respectiva informação
- Um sistema de autenticação pode armazenar as credenciais dos utilizadores (nome e senha) na forma de um dicionário em que a chave é o nome e o elemento associado é a senha desse utilizador
- Um sistema de nomes (DNS) pode usar um dicionário para associar o nome de cada máquina ou servidor (www.ua.pt) ao respectivo endereço IP (193.136.173.81)
- Pode-se usar um dicionário para registar contagens de ocorrência de certos eventos, por exemplo palavras num texto
  - Neste caso o evento será a chave de acesso e a contagem será o elemento associado

11.6

## Dicionário: serviços públicos & semântica

- **get(k)** - devolve o elemento associado à chave dada
  - Pré-condição: `contains(k)`
- **set(k,e)** - actualizar o elemento associado à chave `k`, caso esta exista, ou inserir o novo para `(k,e)`
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove a chave dada bem como o elemento associado
  - Pré-condição: `contains(k)`
  - Pós-condição: `!contains(k)`
- **contains(k)** - devolve `true` se o dicionário contém a chave dada e `false` caso contrário
- **size()** - devolve o número de elementos no dicionário
- **isEmpty()** - devolve `true` se o dicionário está vazio e `false` caso contrário
- **keys()** - devolve um vector com todas as chaves existentes

11.7

### Dicionários: estratégias de implementação

- Existem muitas formas de implementar um dicionário, com simplicidade e eficiência variáveis
- Nesta unidade curricular, vamos ver as seguintes:
  - Listas ligadas de pares chave-valor, na presente aula
  - Tabelas de dispersão, na aula 12
  - Árvores binárias de procura, na aula 13

11.8

# 2 Listas ligadas de pares chave-valor

### Listas ligadas de pares chave-valor

- Segue a estrutura geral das listas ligadas
  - Ver aula 07
- No entanto:
  - Cada nó, além do **elemento** e da referência do nó **seguinte**, tem também a **chave** que dá acesso ao elemento
  - Não precisamos da referência do último nó, ou seja, trabalhamos apenas com a referência do primeiro (`first`)
- Vamos trabalhar com chaves que são cadeias de caracteres e elementos de tipo não especificado

11.9

### Listas ligadas de pares chave-valor

- A lista de pares chave-valor é uma implementação naive do conceito de dicionário
  - Em geral não é eficiente: o acesso a cada elemento demora um tempo proporcional ao número de elementos (complexidade $O(n)$)
- No entanto, a lista de pares chave-valor vai ser utilizada na tabela de dispersão, que é uma implementação muito eficiente do conceito de dicionário
  - Ver aula 12

11.10

## 2.1 Implementação

### Nós com chave e elemento

```
class KeyValueNode<E> {

   final String key;
   E elem;
   KeyValueNode<E> next;

   KeyValueNode(String k,E e,KeyValueNode<E> n) {
      key  = k;
      elem = e;
      next = n;
   }

   KeyValueNode(String k,E e) {
      key  = k;
      elem = e;
      next = null;
   }
}
```

11.11

## Lista de pares chave-valor: esqueleto de implementação

```
public class KeyValueList<E> {

   public KeyValueList() { }

   public E get(String k) {
      assert contains(k) : "Key does not exist";
      ... ...
   }
   public boolean set(String k,E e) {
      ... ...
      assert contains(k) && get(k).equals(e);
      return ...
   }
   public void remove(String k) {
      assert contains(k) : "Key does not exist";
      ... ...
      assert !contains(k) : "Key still exists";
   }
   public boolean contains(String k)
   { ... }
   public String[] keys()
   { ... }
   public int size() { return size; }
   { ... }
   public boolean isEmpty() { return size == 0; }
   { ... }

   private KeyValueNode<E> first = null;
   private int size = 0;
}
```



## Consulta: `get()`

```
public class KeyValueList<E> {
   ...
   public E get(String k) {
      assert contains(k) : "Key does not exist";
      return get(first,k);
   }
   private E get(KeyValueNode<E> n,String k) {
      if (n.key.equals(k)) return n.elem;
      return get(n.next,k);
   ...
}
```



## Actualização: `set()`

```
public class KeyValueList<E> {
   ...
   public boolean set(String k,E e) {
      int prev_size = size;
      first = set(first,k,e);
      assert contains(k) && get(k).equals(e);
      return size>prev_size;
   }
   private KeyValueNode<E> set(KeyValueNode<E> n,String k,E e) {
      if (n==null) {
         KeyValueNode<E> newnode = new KeyValueNode<E>(k,e);
         size++;
         return newnode;
      }
      if (n.key.equals(k)) {
         n.elem = e;
      }
      else n.next = set(n.next,k,e);
      return n;
   }
   ...
}
```