Aula 11
Dicionários
Listas de pares chave-valor

Programação II, 2019-2020

v0.12, 17-05-2018

DETI, Universidade de Aveiro

Sumário

Sumário

As estruturas de dados que já vimos

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

Dicionários: o conceito

- Um **dicionário** é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor.
- Cada chave funciona como um identificador único do par, ou seja, num dicionário cada chave aparece apenas uma vez.
- Assim, um dicionário define uma *associação* ou correspondência unívoca (*mapping*) entre chaves e valores.
  - Por isso também se chama mapa (*map*).
- Tal como um índice dá acesso ao valor correspondente armazenado num vector, também aqui uma chave dá acesso ao valor que lhe está *associado* num dicionário.
  - Por isso, também se chama vector associativo (*associative array*).
  - A eficiência do acesso pela chave depende da implementação.

Dicionários: o conceito

- Um **dicionário** é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor.
- Cada chave funciona como um identificador único do par, ou seja, num dicionário cada chave aparece apenas uma vez.
- Assim, um dicionário define uma *associação* ou correspondência unívoca (*mapping*) entre chaves e valores.
  - Por isso também se chama **mapa** (*map*).
- Tal como um índice dá acesso ao valor correspondente armazenado num vector, também aqui uma chave dá acesso ao valor que lhe está *associado* num dicionário.
  - Por isso, também se chama **vector associativo** (*associative array*).
  - A eficiência do acesso por chave depende da implementação.

Dicionários: o conceito

- Um **dicionário** é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor.
- Cada chave funciona como um identificador único do par, ou seja, num dicionário cada chave aparece apenas uma vez.
- Assim, um dicionário define uma *associação* ou correspondência unívoca (*mapping*) entre chaves e valores.
  - Por isso também se chama **mapa** (*map*).
- Tal como um índice dá acesso ao valor correspondente armazenado num vector, também aqui uma chave dá acesso ao valor que lhe está *associado* num dicionário.
  - Por isso, também se chama **vector associativo** (*associative array*).
  - A eficiência do acesso por chave depende da implementação.

Dicionários: o conceito

- Um **dicionário** é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor.
- Cada chave funciona como um identificador único do par, ou seja, num dicionário cada chave aparece apenas uma vez.
- Assim, um dicionário define uma *associação* ou correspondência unívoca (*mapping*) entre chaves e valores.
  - Por isso também se chama **mapa** (*map*).
- Tal como um índice dá acesso ao valor correspondente armazenado num vector, também aqui uma chave dá acesso ao valor que lhe está *associado* num dicionário.
  - Por isso, também se chama **vector associativo** (*associative array*).
  - A eficiência do acesso por chave depende da implementação.

Dicionários: o conceito

- Um **dicionário** é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor.
- Cada chave funciona como um identificador único do par, ou seja, num dicionário cada chave aparece apenas uma vez.
- Assim, um dicionário define uma *associação* ou correspondência unívoca (*mapping*) entre chaves e valores.
  - Por isso também se chama **mapa** (*map*).
- Tal como um índice dá acesso ao valor correspondente armazenado num vector, também aqui uma chave dá acesso ao valor que lhe está *associado* num dicionário.
  - Por isso, também se chama **vector associativo** (*associative array*).
  - A eficiência do acesso por chave depende da implementação.

Dicionários: o conceito

- Um **dicionário** é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor.
- Cada chave funciona como um identificador único do par, ou seja, num dicionário cada chave aparece apenas uma vez.
- Assim, um dicionário define uma *associação* ou correspondência unívoca (*mapping*) entre chaves e valores.
  - Por isso também se chama **mapa** (*map*).
- Tal como um índice dá acesso ao valor correspondente armazenado num vector, também aqui uma chave dá acesso ao valor que lhe está *associado* num dicionário.
  - Por isso, também se chama **vector associativo** (*associative array*).
  - A eficiência do acesso por chave depende da implementação.

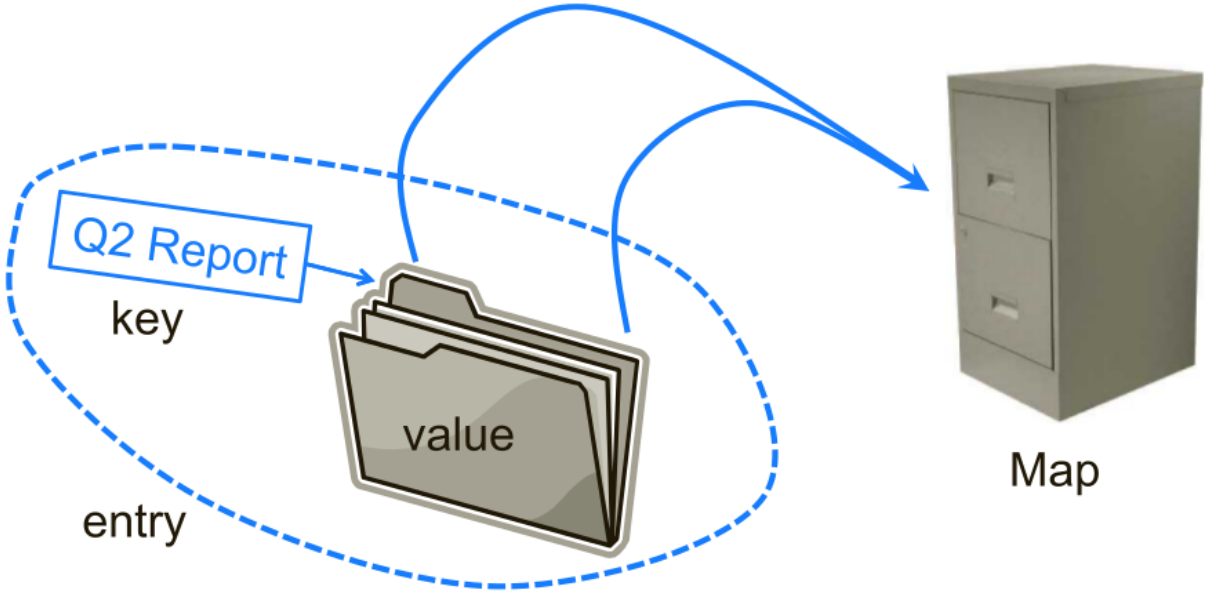
Dicionários: o conceito

- Um **dicionário** é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor.
- Cada chave funciona como um identificador único do par, ou seja, num dicionário cada chave aparece apenas uma vez.
- Assim, um dicionário define uma *associação* ou correspondência unívoca (*mapping*) entre chaves e valores.
  - Por isso também se chama **mapa** (*map*).
- Tal como um índice dá acesso ao valor correspondente armazenado num vector, também aqui uma chave dá acesso ao valor que lhe está *associado* num dicionário.
  - Por isso, também se chama **vector associativo** (*associative array*).
  - A eficiência do acesso por chave depende da implementação.

Dicionários: o conceito

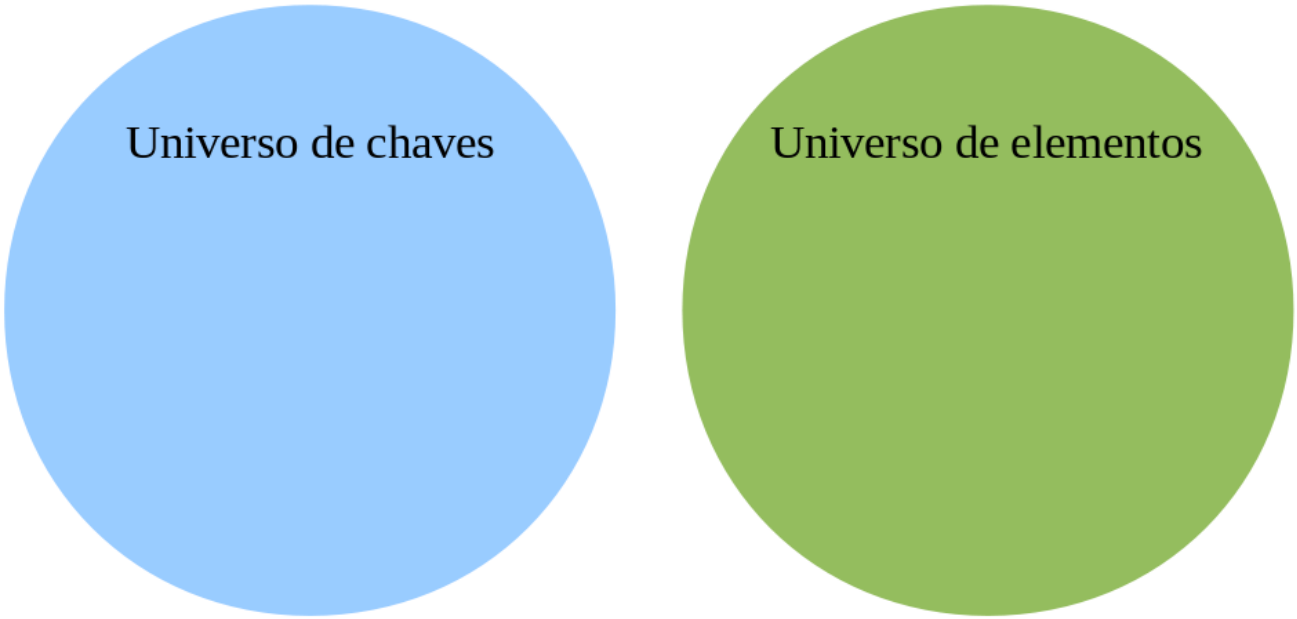
- Um **dicionário** é uma estrutura de dados que permite armazenar um conjunto de pares chave-valor.
- Cada chave funciona como um identificador único do par, ou seja, num dicionário cada chave aparece apenas uma vez.
- Assim, um dicionário define uma *associação* ou correspondência unívoca (*mapping*) entre chaves e valores.
  - Por isso também se chama **mapa** (*map*).
- Tal como um índice dá acesso ao valor correspondente armazenado num vector, também aqui uma chave dá acesso ao valor que lhe está *associado* num dicionário.
  - Por isso, também se chama **vector associativo** (*associative array*).
  - A eficiência do acesso por chave depende da implementação.

Dicionários: ilustração



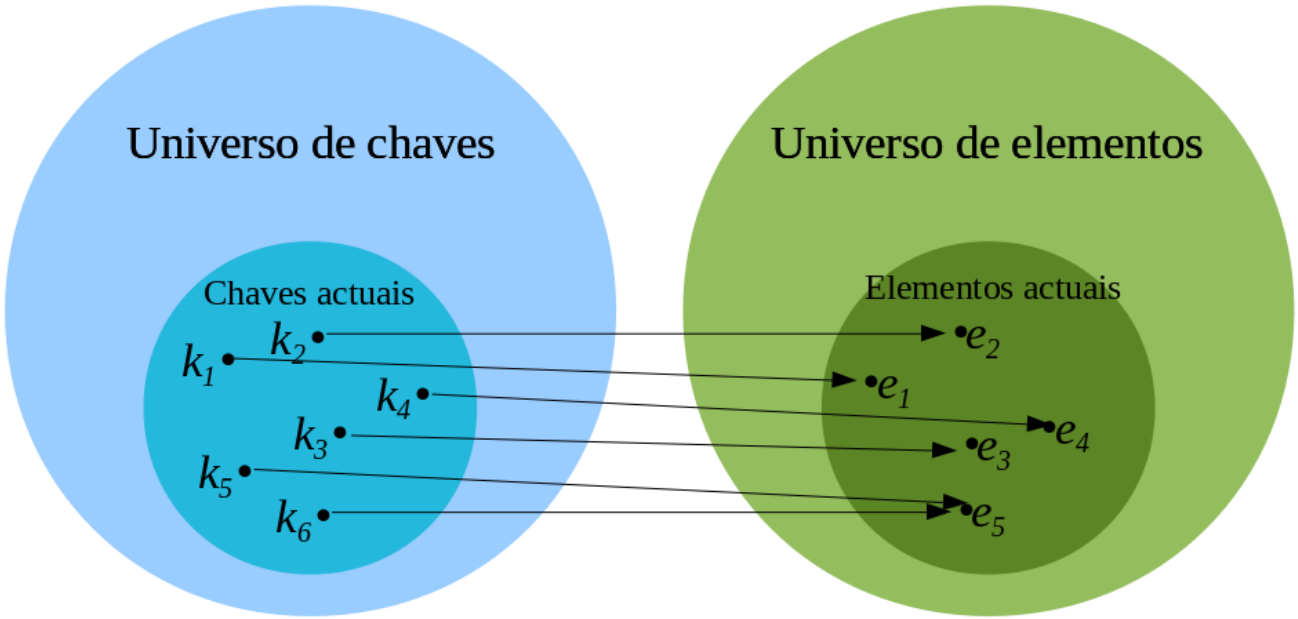
- Cada pasta contém uma determinada etiqueta (a chave) e determinado conteúdo no seu interior (o valor ou elemento associado).
- O conjunto de pastas com as respectivas etiquetas são arrumadas no armário (o dicionário).

Dicionários



$$\{(k_1, e_1), (k_2, e_2), (k_3, e_3), (k_4, e_4), (k_5, e_5), (k_6, e_5)\}$$

Dicionários



$$\{(k_1, e_1), (k_2, e_2), (k_3, e_3), (k_4, e_4), (k_5, e_5), (k_6, e_5)\}$$

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 valor associado é a senha desse utilizador.
- Uma tabela de símbolos (variáveis) num compilador ou interpretador de uma linguagem de programação.
- Um servidor de DNS pode usar um dicionário para associar o nome de cada computador (www.ua.pt) ao respectivo endereço IP (193.136.173.81).
- Pode-se usar um dicionário para registar contagens de ocorrências de certos eventos, por exemplo palavras num texto.
  - Neste caso a chave será a palavra de interesse e o valor será o valor associado.

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 valor associado é a senha desse utilizador.
- Uma tabela de símbolos (variáveis) num compilador ou interpretador de uma linguagem de programação.
- Um servidor de DNS pode usar um dicionário para associar o nome de cada computador (www.ua.pt) ao respectivo endereço IP (193.136.173.81).
- Pode-se usar um dicionário para registar contagens de ocorrências de certos eventos, por exemplo palavras num texto.
    - Neste caso o evento será a chave de acesso e a contagem será o valor associado.

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 valor associado é a senha desse utilizador.
- Uma tabela de símbolos (variáveis) num compilador ou interpretador de uma linguagem de programação.
- Um servidor de DNS pode usar um dicionário para associar o nome de cada computador (www.ua.pt) ao respectivo endereço IP (193.136.173.81).
- Pode-se usar um dicionário para registar contagens de ocorrências de certos eventos, por exemplo palavras num texto.
    - Neste caso o evento será a chave de acesso e a contagem será o valor associado.

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 valor associado é a senha desse utilizador.
- Uma tabela de símbolos (variáveis) num compilador ou interpretador de uma linguagem de programação.
- Um servidor de DNS pode usar um dicionário para associar o nome de cada computador (www.ua.pt) ao respectivo endereço IP (193.136.173.81).
- Pode-se usar um dicionário para registar contagens de ocorrências de certos eventos, por exemplo palavras num texto.
    - Neste caso o evento será a chave de acesso e a contagem será o valor associado.

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 valor associado é a senha desse utilizador.
- Uma tabela de símbolos (variáveis) num compilador ou interpretador de uma linguagem de programação.
- Um servidor de DNS pode usar um dicionário para associar o nome de cada computador (www.ua.pt) ao respectivo endereço IP (193.136.173.81).
- Pode-se usar um dicionário para registar contagens de ocorrências de certos eventos, por exemplo palavras num texto.
  - Neste caso o evento será a chave de acesso e a contagem será o valor associado.

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 valor associado é a senha desse utilizador.
- Uma tabela de símbolos (variáveis) num compilador ou interpretador de uma linguagem de programação.
- Um servidor de DNS pode usar um dicionário para associar o nome de cada computador (www.ua.pt) ao respectivo endereço IP (193.136.173.81).
- Pode-se usar um dicionário para registar contagens de ocorrências de certos eventos, por exemplo palavras num texto.
    - Neste caso o evento será a chave de acesso e a contagem será o valor associado.

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 valor associado é a senha desse utilizador.
- Uma tabela de símbolos (variáveis) num compilador ou interpretador de uma linguagem de programação.
- Um servidor de DNS pode usar um dicionário para associar o nome de cada computador (www.ua.pt) ao respectivo endereço IP (193.136.173.81).
- Pode-se usar um dicionário para registar contagens de ocorrências de certos eventos, por exemplo palavras num texto.
    - Neste caso o evento será a chave de acesso e a contagem será o valor associado.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

Dicionário: serviços públicos & semântica

- **get(k)** - devolve o valor associado à chave dada.
  - Pré-condição: `contains(k)`
- **set(k, e)** - actualiza o valor associado à chave `k`, caso esta exista, ou insere o novo par `(k, e)`.
  - Pós-condição: `contains(k) && get(k)==e`
- **remove(k)** - remove o par associado à chave `k`.
  - 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.

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, nesta aula;
  - Tabelas de dispersão, na aula 12;
  - Árvores binárias de pesquisa, na aula 13.

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, nesta aula;
  - Tabelas de dispersão, na aula 12;
  - Árvores binárias de pesquisa, na aula 13.

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, nesta aula;
  - Tabelas de dispersão, na aula 12;
  - Árvores binárias de pesquisa, na aula 13.

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, nesta aula;
  - Tabelas de dispersão, na aula 12;
  - Árvores binárias de pesquisa, na aula 13.

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, nesta aula;
  - Tabelas de dispersão, na aula 12;
  - Árvores binárias de pesquisa, na aula 13.

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, nesta aula;
  - Tabelas de dispersão, na aula 12;
  - Árvores binárias de pesquisa, na aula 13.

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 arbitrário.

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 arbitrário.

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 arbitrário.

Listas ligadas de pares chave-valor

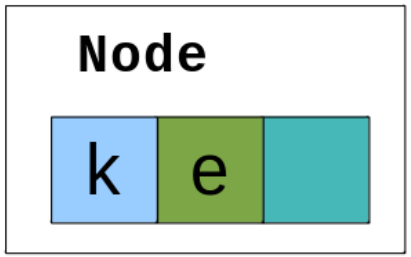
- Segue a estrutura geral das listas ligadas.
  - Ver aula 07.
- No entanto:
  - Cada nó, além do **elemento** e da referência do nó **seguinte**, tem também a **chave** que dá acesso ao elemento;
  - Não precisamos da referência do último nó, ou seja, trabalhamos apenas com a referência do primeiro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

Listas ligadas de pares chave-valor

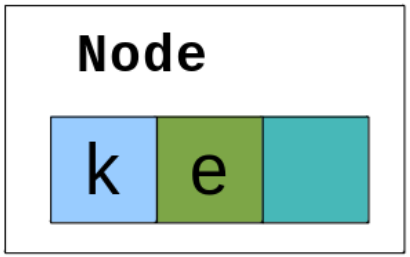
- Segue a estrutura geral das listas ligadas.
  - Ver aula 07.
- No entanto:
  - Cada nó, além do **elemento** e da referência do nó **seguinte**, tem também a **chave** que dá acesso ao elemento;
  - Não precisamos da referência do último nó, ou seja, trabalhamos apenas com a referência do primeiro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

Listas ligadas de pares chave-valor

Listas ligadas de pares chave-valor

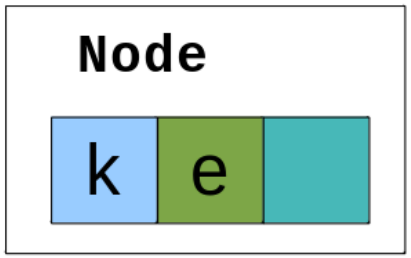


Listas ligadas de pares chave-valor

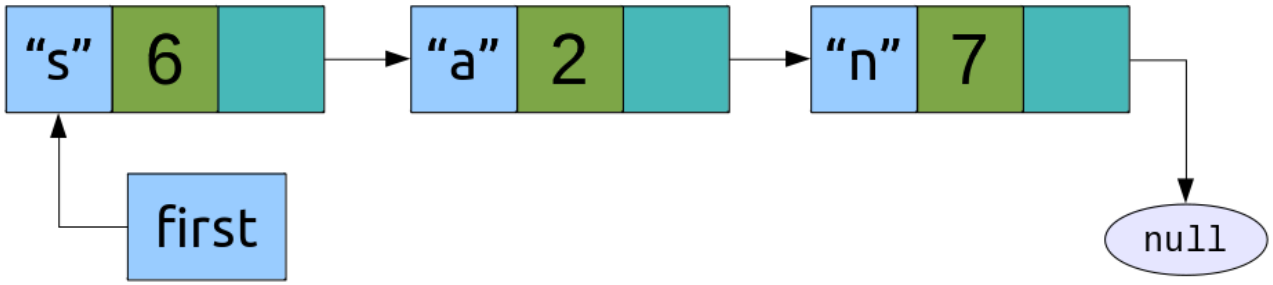



```
class KeyValueNode
{
   String key;
   int elem;
   KeyValueNode next;
}
```

Listas ligadas de pares chave-valor




```
class KeyValueNode
{
   String key;
   int elem;
   KeyValueNode next;
}
```



Listas ligadas de pares chave-valor

- A lista de pares chave-valor é uma implementação simplista 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.

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;
  }
}
```

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;
  }
}
```

Lista de pares chave-valor: esqueleto de implementação

```
public class KeyValueList<E> {

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

  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()
  { ... }
  public boolean isEmpty()
  { ... }
}
```

Lista de pares chave-valor: esqueleto de implementação

```
public class KeyValueList<E> {

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

  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()
  { ... }
  public boolean isEmpty()
  { ... }
}
```

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);
  }
  ...
}
```

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);
  }
  ...
}
```

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) {
      n = new KeyValueNode<E>(k, e);
      size++;
    }
    else if (n.key.equals(k)) {
      n.elem = e;
    }
    else n.next = set(n.next, k, e);
    return n;
  }
  ...
}
```

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) {
      n = new KeyValueNode<E>(k, e);
      size++;
    }
    else if (n.key.equals(k)) {
      n.elem = e;
    }
    else n.next = set(n.next, k, e);
    return n;
  }
  ...
}
```

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) {
      n = new KeyValueNode<E>(k, e);
      size++;
    }
    else if (n.key.equals(k)) {
      n.elem = e;
    }
    else n.next = set(n.next, k, e);
    return n;
  }
  ...
}
```