ENGENHARIA INFORMÁTICA E DE COMPUTADORES

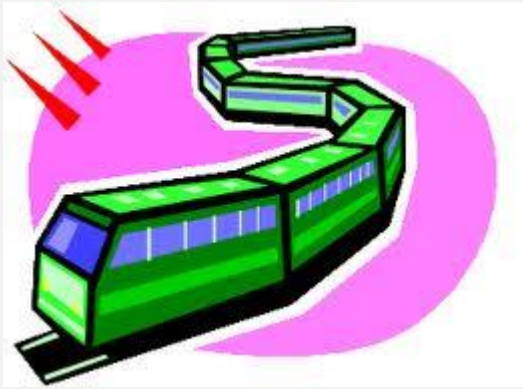
Algoritmos e Estruturas de Dados

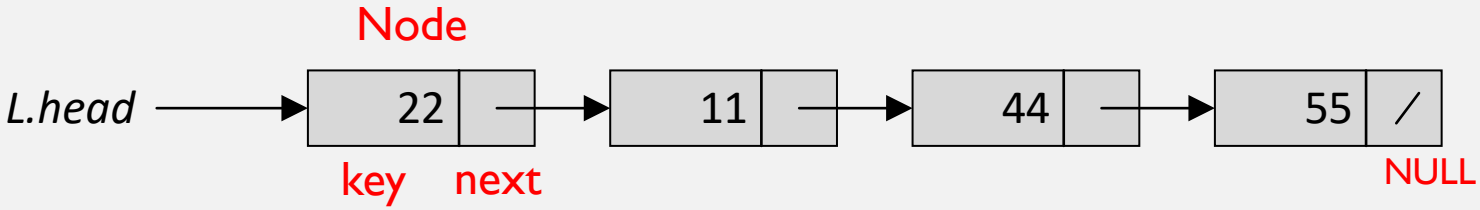
(parte 11 – Listas Ligadas)

2º Semestre 2022/2023
Instituto Superior de Engenharia de Lisboa
Paula Graça

LISTA SIMPLESMENTE LIGADA

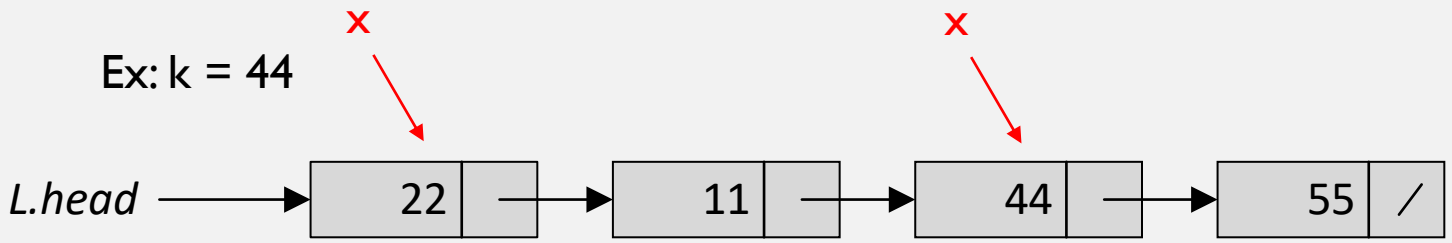
- Uma lista ligada (*linked list*) é uma sequência de zero ou mais elementos designados por nós
- Cada nó (*Node*) é composto por um valor (*key*) e uma referência para o nó seguinte da lista (*next*)
- A referência do primeiro nó é designada por cabeça da lista (*head*)
- Se a lista estiver vazia *head* = *NULL*





LISTAS SIMPLESMENTE LIGADAS

- Pesquisa numa Lista List-Contains
  - Procura através de uma pesquisa linear, o primeiro elemento com a chave k na lista L, devolvendo uma referência para esse elemento

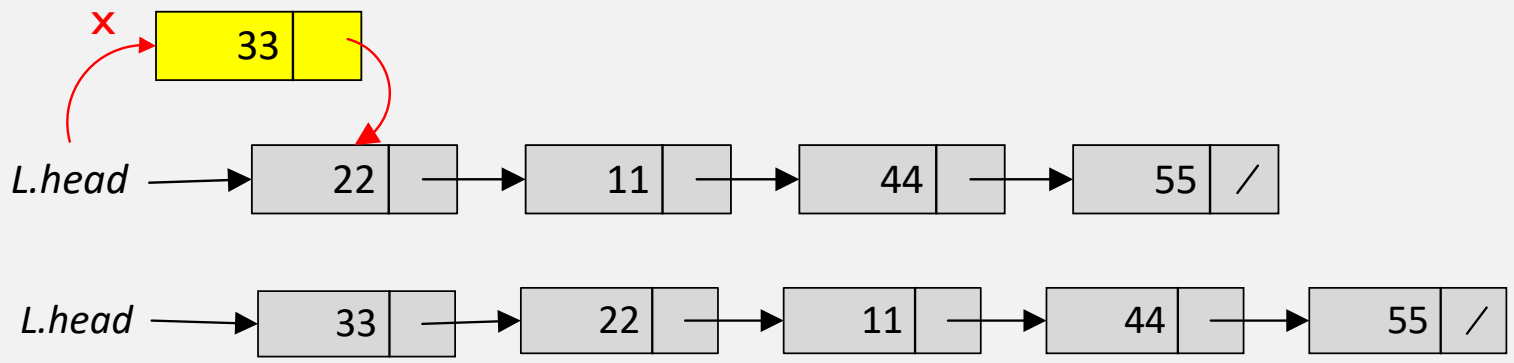


- A pesquisa numa lista de n elementos no pior caso, tem custo de *O* (n)

```
List-Contains(L, k)
      x = L.head
      while x ≠ NULL
            if k == x.key return true
            x = x.next
      return false
```

LISTAS SIMPLESMENTE LIGADAS

- Inserção numa Lista List-Add
  - Insere o novo elemento x à cabeça da lista (1ª posição) L

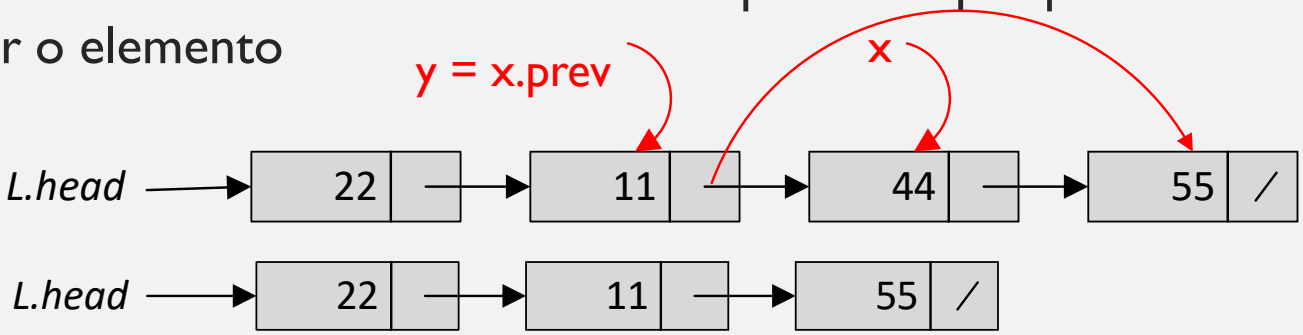


- A inserção numa lista de n elementos no pior caso, tem o custo de *O* (1)

```
List-Add(L, x)
    x.next = L.head
    L.head = x
    return L.head
```

LISTAS SIMPLESMENTE LIGADAS

- Remoção numa Lista *List-Remove*
  - Remove o elemento x da lista L com chave k. Implica uma pesquisa linear para encontrar o elemento

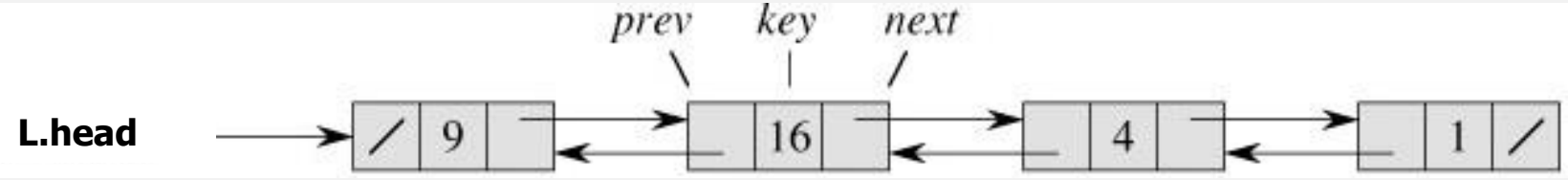


- A remoção numa lista de n elementos tem o custo de *O*(n) no pior caso, pois implica uma pesquisa linear para encontrar o elemento

```
List-Remove(L, k)
    x = L.head
    y = NULL
    while (x ≠ NULL)
        if k == x.key
            if y == NULL // remoção à cabeça
                L.head = x.next
            else y.next = x.next
            return L.head
        else
            y = x
            x = x.next
    return L.head
```

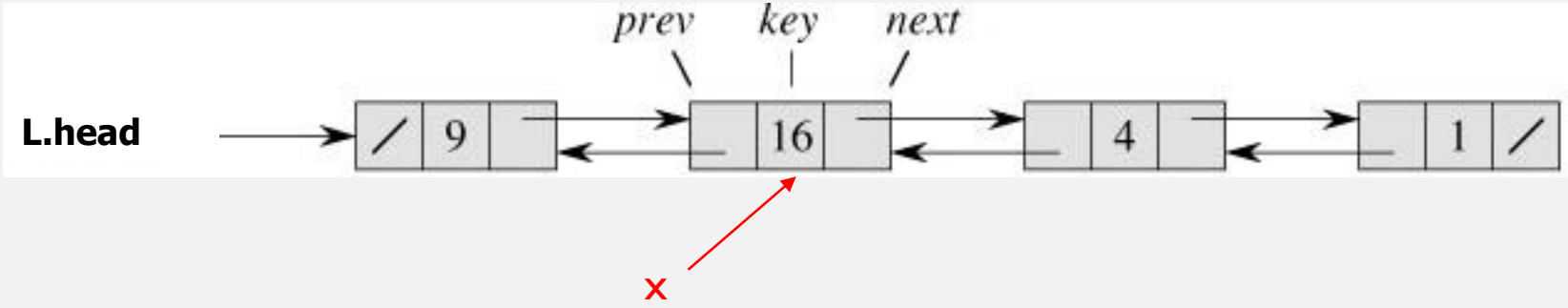
LISTAS DUPLAMENTE LIGADAS

- Numa lista duplamente ligada, cada elemento é composto por um atributo *key*, e dois atributos *next* e *prev* que são referências respetivamente para o sucessor e predecessor do elemento
- A lista é identificada através de uma referência para a cabeça da lista (*head*)



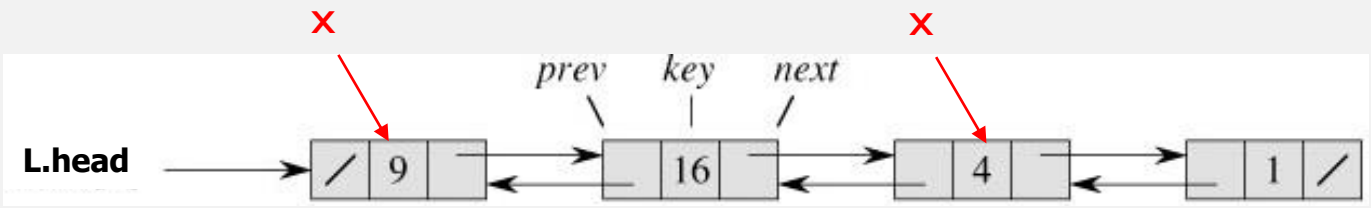
LISTAS DUPLAMENTE LIGADAS

- Dado um elemento x na lista,
  - x.next referencia o seu sucessor
  - Se x.next = NULL, o elemento não tem sucessor, sendo o último elemento ou cauda da lista (*tail*)
  - x.prev referencia o seu predecessor
  - Se x.prev = NULL, o elemento não tem predecessor, sendo o primeiro elemento ou cabeça da lista (*head*)



LISTAS DUPLAMENTE LIGADAS

- Pesquisa numa Lista List-Contains
  - Procura através de uma pesquisa linear, o primeiro elemento com a chave k na lista L

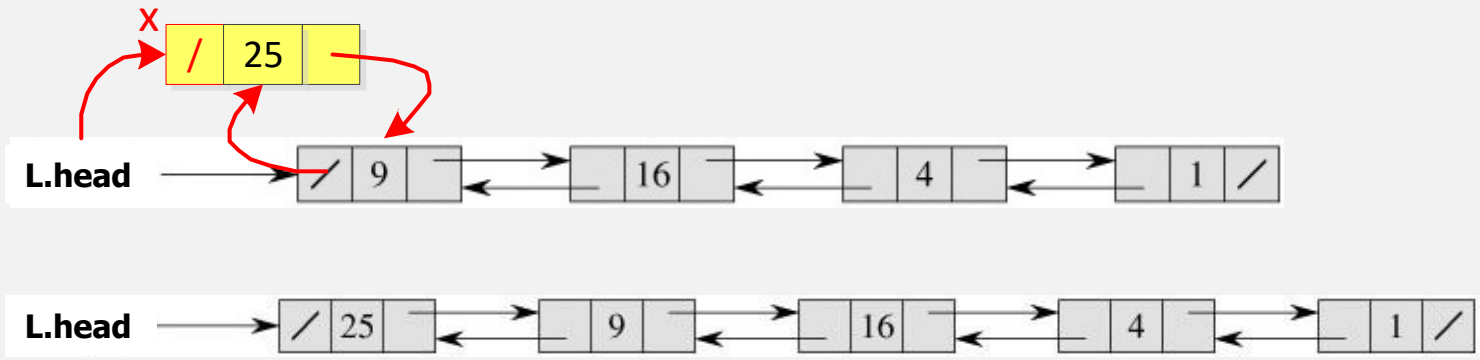


- A pesquisa numa lista de n elementos no pior caso, tem o custo de *O* (n)

```
List-Contains(L, k)
    x = L.head
    while x ≠ NULL
        if k == x.key
            return true
        x = x.next
    return false
```

LISTAS DUPLAMENTE LIGADAS

- Inserção numa Lista List-Add
  - Insere o novo elemento x na primeira posição da lista L

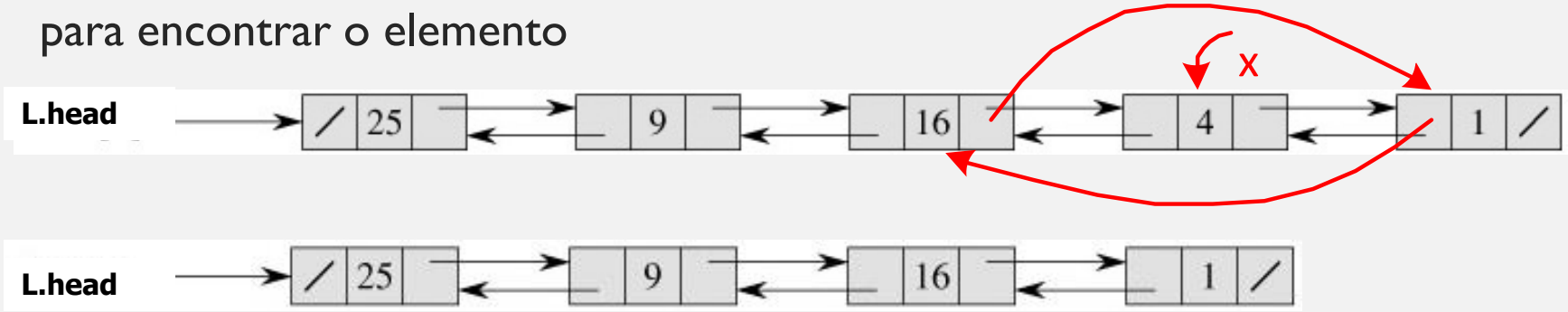


- A inserção numa lista de n elementos no pior caso, tem o custo de O (1)

```
List-Add(L, x)
    x.next = L.head
    if L.head ≠ NULL
        L.head.prev = x
    L.head = x
    x.prev = NULL
    return L.head
```

LISTAS DUPLAMENTE LIGADAS

- Remoção numa Lista *List-Remove*
  - Remove o elemento x com chave k da lista L. Implica uma pesquisa linear para encontrar o elemento



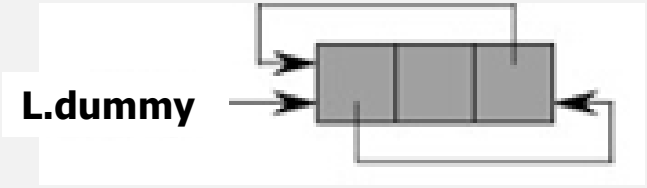
- A remoção numa lista de n elementos tem o custo de $O(n)$

```
List-Remove(L, k)
    x = L.head
    while x ≠ NULL
        if k == x.key
            if x.prev ≠ NULL
                x.prev.next = x.next
            else L.head = x.next
            if x.next ≠ NULL
                x.next.prev = x.prev
            return L.head
        else x = x.next
    return L.head
```

LISTAS CIRCULARES COM SENTINELA

- As listas circulares usam uma sentinela, ou seja, um elemento *dummy*, que permite simplificar as condições limite

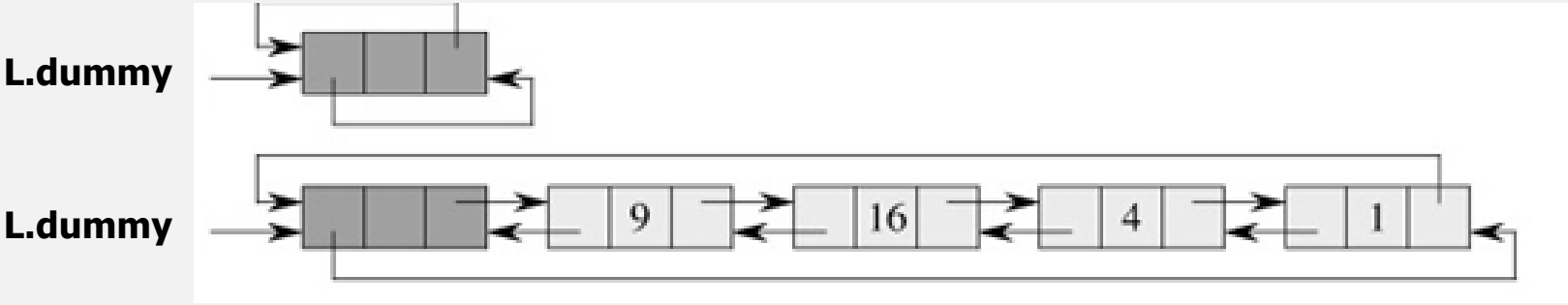
- A sentinela, é a cabeça da lista L.dummy que significa um elemento *dummy*, mas que tem todos os atributos dos outros elementos (nós)



LISTAS CIRCULARES COM SENTINELA

- Na lista circular, não existem referências a *NULL* pois são substituídas pela referência para a sentinela L.dummy, que tem uma ligação para a cabeça (L.dummy.next) e para a cauda da lista (L.dummy.prev)

- Desta forma, a lista é torna-se numa lista duplamente ligada circular



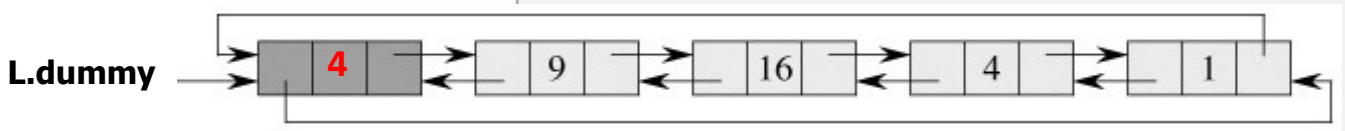
LISTAS CIRCULARES COM SENTINELA

- O algoritmo List-Contains mantém-se, ou é simplificado usando a pesquisa com sentinela.

```
List-Contains(L, k)
    x = L.dummy.next
    while x ≠ L.dummy
        if k == x.key
            return true
        x = x.next
    return false
```

```
List-Contains(L, k) // Usando a sentinela
    x = L.dummy.next
    L.dummy.key = k
    while k ≠ x.key
        x = x.next
    return x ≠ L.dummy
```

List-Contains(L, 4)

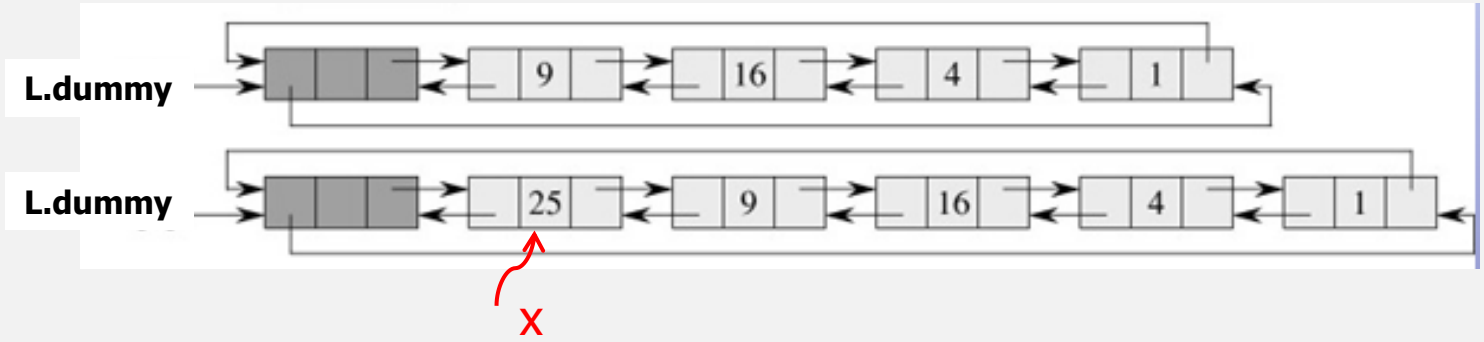


LISTAS CIRCULARES COM SENTINELA

- O algoritmo List-Add fica simplificado, pois não é necessário testar a condição limite no caso da lista estar vazia

```
List-Add(L, x)
    x.next = L.dummy.next
    L.dummy.next.prev = x
    L.dummy.next = x
    x.prev = L.dummy
```

List-Add(L, 25)



LISTAS CIRCULARES COM SENTINELA

- O algoritmo List-Remove fica mais simples pois podem ser ignoradas as condições limite no caso da remoção à cabeça ou cauda da lista

```
List-Remove(L, k)
    x = L.dummy.next
    while x ≠ L.dummy
        if k == x.key
            x.prev.next = x.next
            x.next.prev = x.prev
        else x = x.next
```