Algoritmos e Estruturas de Dados I

Profª Priscilla Abreu

priscilla.abreu@ime.uerj.br

2022.1

Algoritmos e Estruturas de Dados I

Roteiro da aula

- Listas lineares
  - Listas Sequenciais

Listas
Introdução

Algoritmos e Estruturas de Dados I

Listas

O QUE É UMA LISTA?

Consideraremos como listas conjuntos sem repetições!

Uma lista é um conjunto de dados relacionados, e de número variável de elementos.

- Exemplo:
  - Lista de alunos de uma turma;
  - Lista de aprovados em um concurso;
  - Lista de produtos de uma loja;
  - ...

Cada elemento da lista terá uma chave – identificador único.

Algoritmos e Estruturas de Dados I

Listas

Utilizaremos um elemento não-escalar para representar as listas:

estrutura Elemento:

Campo1: Tipo1

…

CampoK: TipoK

```
struct Elemento{
    tipo1 campo1;
     ...
    tipo campoK;
};
```

Algoritmos e Estruturas de Dados I

Listas – Exemplo:

estrutura funcionario:

matricula: inteiro

nome: texto

endereco: texto

salario: real

```
struct funcionario{
    int matricula;
    char nome[50];
    char endereço[30];
    float salario;
};
```

Algoritmos e Estruturas de Dados I

- Listas podem ser implementadas de diversas formas;
- A forma mais eficiente depende de cada contexto;
- Importância de conhecer as possibilidades.

Algoritmos e Estruturas de Dados I

Operações comuns em listas:

- **Busca;**
- **Inserção;**
- **Remoção.**

Outras operações:

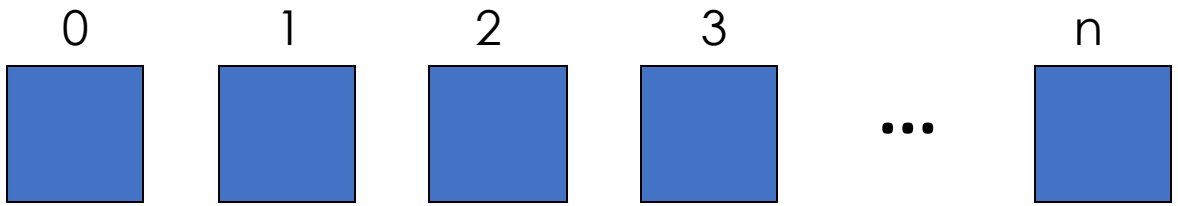
- Alteração de um elemento da lista;
- Ordenação dos elementos da lista segundo uma determinada chave;
- Determinação do primeiro elemento da lista, ...

Algoritmos e Estruturas de Dados I

Listas Linear

Estrutura que permite representar um conjunto de dados de forma a preservar a relação de ordem existente entre eles.

Temos um primeiro elemento, segundo elemento, ..., n-ésimo elemento.



Algoritmos e Estruturas de Dados I

Listas Linear

Listas lineares

Listas lineares gerais
SEM restrição de inserção e remoção de elementos

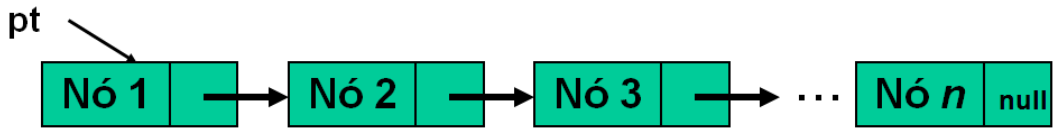
Listas particulares
COM restrição de inserção e remoção de elementos

Algoritmos e Estruturas de Dados I

O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa na memória (contínua ou não) de cada dois nós consecutivos na lista.

Existem dois tipos de alocação:

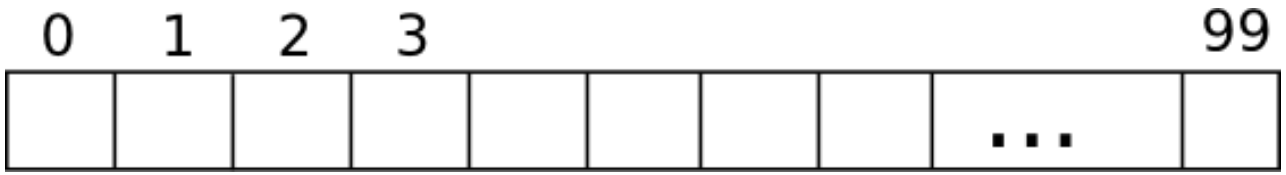
- Alocação sequencial
- Alocação encadeada



Algoritmos e Estruturas de Dados I

- **Alocação sequencial**

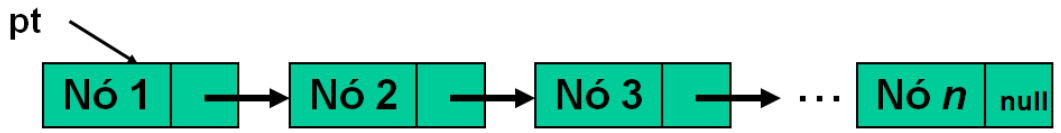
- Elementos ocupam memória em posições contíguas, em que cada elemento da lista ocupa posição física na memória sucessiva ao elemento anterior.
- Acesso aos elementos em tempo constante, já que a posição de cada elemento pode ser inferida.



Algoritmos e Estruturas de Dados I

- **Alocação encadeada**

- Elementos da lista encontram-se aleatoriamente dispostos na memória e não necessariamente ocupam posições contíguas em memória.
- São interligados por ponteiros, que indicam a localização do próximo nó.



Algoritmos e Estruturas de Dados I

- Quando os elementos se movem frequentemente entre as posições, o uso de ponteiros evita transferências de dados em memória;
- Elementos que participam em mais de uma lista linear podem manter seus dados em um único lugar com o uso de ponteiros, ao invés de criar duplicações de dados em memória;
- Quando a quantidade de elementos é relativamente alta, devido a organização do sistema operacional, **é necessário** o uso de **alocação dinâmica** ; para evitar o estouro de pilha;

Algoritmos e Estruturas de Dados I

Implementação de Listas Lineares varia também dependendo da ordem das chaves:

- **Não-Ordenada**: elementos são armazenados em ordem arbitrária, não tendo relação com os valores das respectivas chaves;

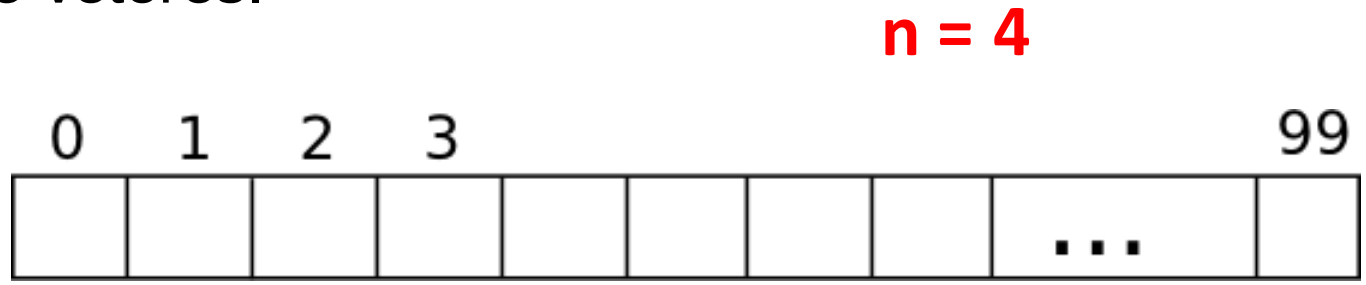
- **Ordenada**: elementos são armazenados de acordo com o valor da chave (em geral, ordem não decrescente).

Listas Lineares Sequenciais

Algoritmos e Estruturas de Dados I

Lista Linear Sequencial

- Uso de vetores.



  - MAX é a quantidade máxima de elementos que a lista poderá armazenar.
  - **n representa o número de elementos.**

Algoritmos e Estruturas de Dados I

Lista Linear Sequencial

```
estrutura no:
        chave: <inteiro>

estrutura listaSeq:
        valores [1..MAX]: no
        n: inteiro
```

```
const int MAX = 10
typedef struct no{
    int chave;
}no;

typedef struct listaSeq{
    no valores[MAX];
    int n;
} listaSeq;
```

Algoritmos e Estruturas de Dados I

Todas as estruturas de dados possuirão dois procedimentos especiais:

- Constrói: chamado antes do primeiro uso com o propósito de inicializar as variáveis com os valores adequados
- Destrói: chamado após o último uso com o propósito de desalocar recursos

Algoritmos e Estruturas de Dados I

Constrói:

procedimento Constroi(**ref** L: listaSeq)
    L.n ← 0

```
void constroi(listaSeq *L){
        L->n = 0;
}
```

Destrói:

Nada a ser feito quando temos uma lista sequencial

Algoritmos e Estruturas de Dados I

Operações:

- Tamanho da lista:

função Tamanho(ref L: ListaLinear): Inteiro

retornar (L.n)

Tempo:
θ(1)

```
int tamanhoLista( listaSeq *L){
        return L->n;
}
```

Algoritmos e Estruturas de Dados I

Operações:

- Exibir lista:

```
procedimento exibeLista(ref L: listaSeq)
        var i: Inteiro
        para i ← 1 até L.n faça
                escrever (L.valores[i].chave)
```

Tempo:
θ(n)

```
void imprimir(listaSeq *L){
    int i;
    if (L->n >=0){
        printf("\nImpressão da lista\n");
        for(i=0; i<L->n;i++)
            printf("%d ",L->valores[i].chave);
        printf("\n");
    }
    else
        printf("\nLista vazia!\n");
}
```

Algoritmos e Estruturas de Dados I

Operações:

- Busca:
  - Listas não-ordenadas

Algoritmos e Estruturas de Dados I

Operações:

- Busca – lista não-ordenada:

```
função BuscaPosicao(ref L: ListaSeq, c: inteiro): inteiro
    var i: Inteiro ← 1
    enquanto i ≤ L.n e L.valores[i].chave ≠ c faça
            i ← i + 1
    se i ≤ L.n então
            retornar (i)
    senão
            retornar (0)
```

Tempo:
Melhor Caso: θ(1)
Pior Caso: θ(N)

```
int buscaPosicao(listaSeq *L, int c) {
      int i=0;
      while (i < L->n){
            if (L->valores[i].chave== c) {
                  return i;
            }
            i++;
      }
      return -1;
}
```

Inserindo elementos...

Algoritmos e Estruturas de Dados I

Operações:

- Inserção:
  - Listas não-ordenadas
  - Listas ordenadas

Algoritmos e Estruturas de Dados I

Operações:

- Inserção:
  - Listas não-ordenadas

Valor a ser inserido

c

16

L.n : 4

1	2	3	4	5	6	7	8	9	10
15	52	36	48						

Algoritmos e Estruturas de Dados I

Operações:

- Inserção:
  - Listas não-ordenadas

Valor a ser inserido

c

16

Tem espaço disponível?

L.n : 4

1	2	3	4	5	6	7	8	9	10
15	52	36	48						

Algoritmos e Estruturas de Dados I

Operações:

- Inserção:
  - Listas não-ordenadas

Valor a ser inserido

L.n : 4

c

16

O elemento já está cadastrado?

1	2	3	4	5	6	7	8	9	10
15	52	36	48						

Algoritmos e Estruturas de Dados I

Operações:

- Inserção:
  - Listas não-ordenadas

L.valores[L.n + 1] = chave

c

16

1	2	3	4	5	6	7	8	9	10
15	52	36	48	16					

Algoritmos e Estruturas de Dados I

Operações:

- Inserção:
  - Listas não-ordenadas

L.n = L.n + 1

1	2	3	4	5	6	7	8	9	10
15	52	36	48	16					

Algoritmos e Estruturas de Dados I

Operações:

- Inserção – lista não-ordenada:

```
procedimento insere(ref L: ListaSeq, c: inteiro)
    se L.n < MAX então
        L.valores[(L.n)+1].chave ← c
        L.n ← L.n + 1
    senão
        escreva("Overflow")
```

Tempo:
θ(1)

Tempo:
θ(ln n)

```
void inserir(listaSeq *L, int c){
    if (L->n < MAX) {
        if (buscaPosicao(L, c)!=-1){
            L->valores[L->n].chave= c;
            L->n = L->n +1;
        }
        else
            printf("\nJá cadastrado!");
    }
    else
        printf("\nLista cheia!\n");
}
```

Removendo elementos...

Algoritmos e Estruturas de Dados I

Operações:

- Remoção:
  - Listas não-ordenadas
  - Listas ordenadas

Algoritmos e Estruturas de Dados I

Operações:

- Remoção:
  - Listas não-ordenadas

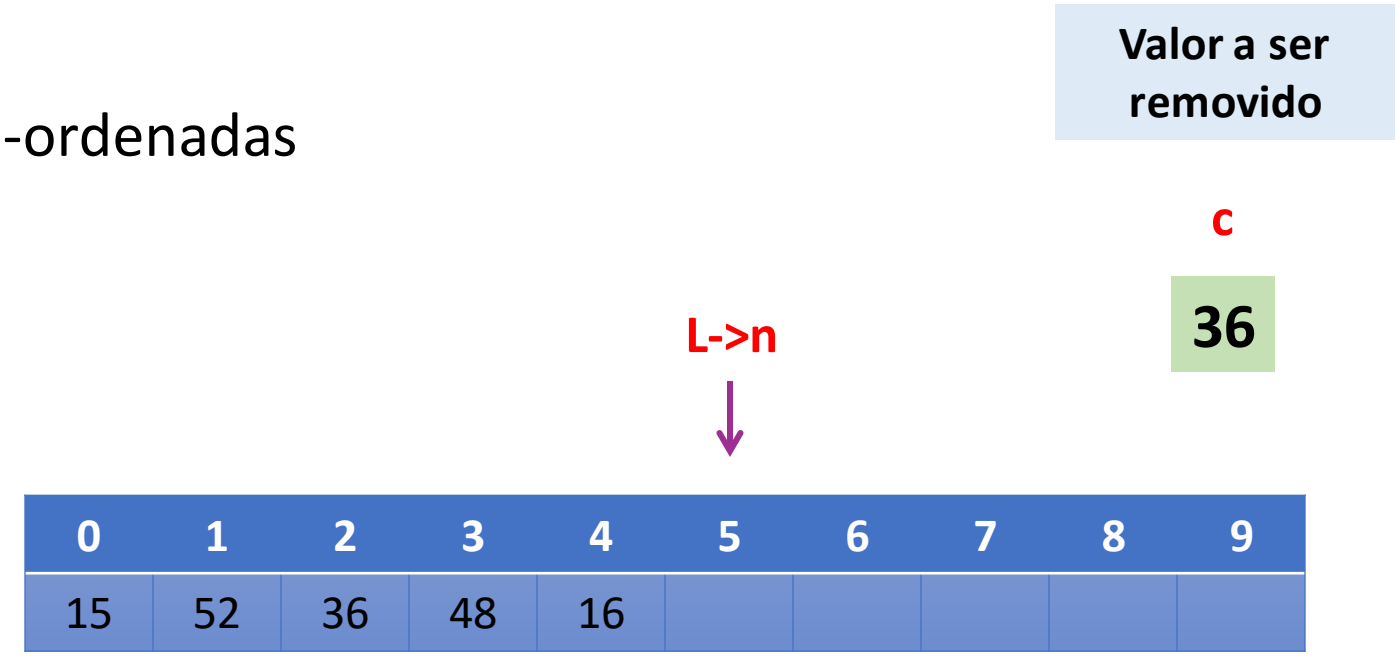
Remoção não ocorre de fato

simulação

Algoritmos e Estruturas de Dados I

Operações:

- Remoção:
  - Listas não-ordenadas



Algoritmos e Estruturas de Dados I

Operações:

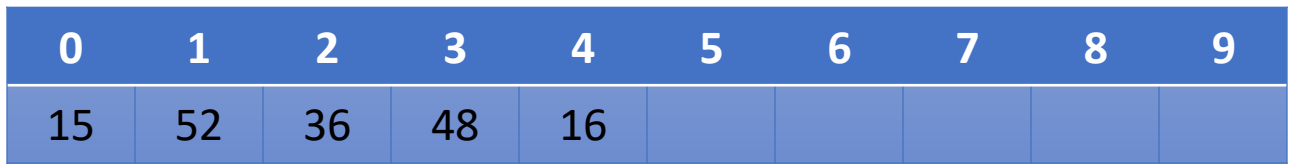
- Remoção:
  - Listas não-ordenadas

A lista está vazia?

c

36

L->n



Algoritmos e Estruturas de Dados I

Operações:

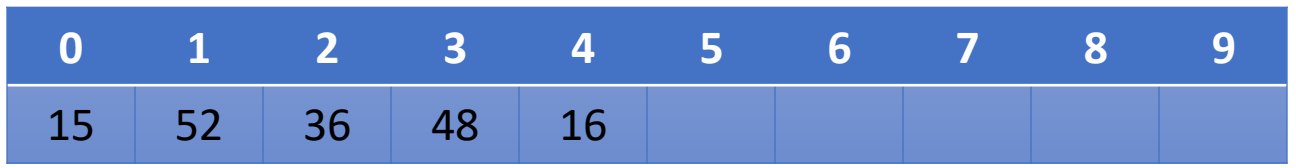
- Remoção:
  - Listas não-ordenadas

O valor a ser removido encontra-se na lista?

c

36

L->n



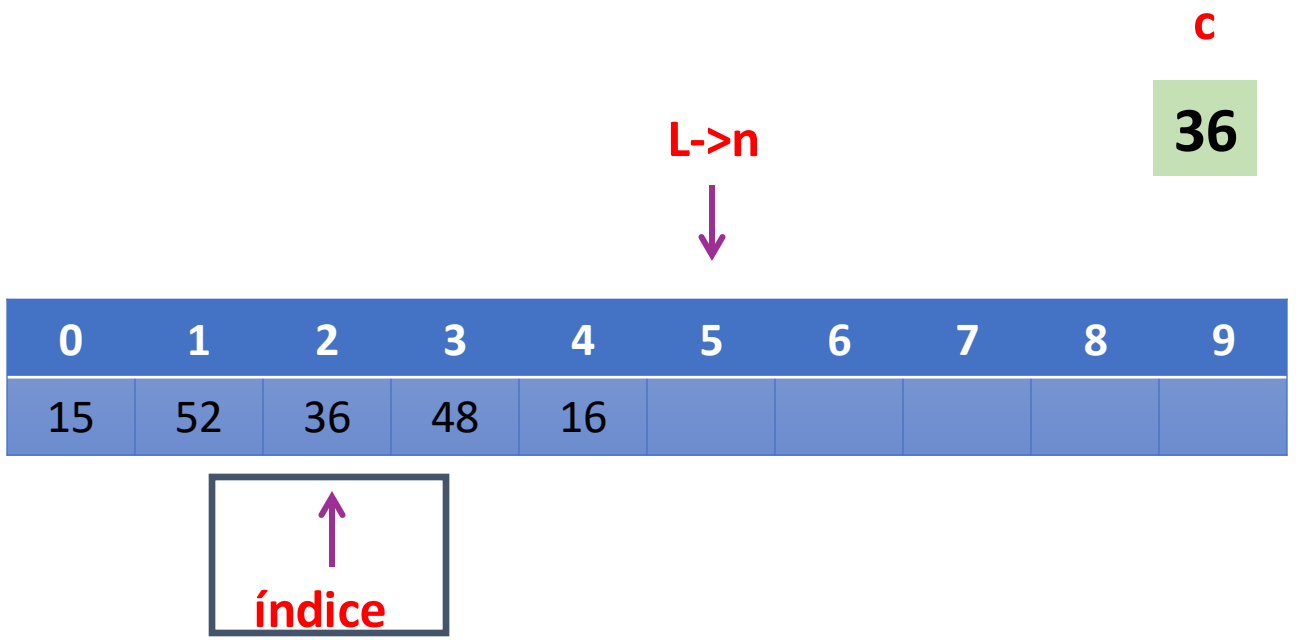
0	1	2	3	4	5	6	7	8	9
15	52	36	48	16					

Algoritmos e Estruturas de Dados I

Operações:

- Remoção:
  - Listas não-ordenadas

O valor a ser removido encontra-se na lista?

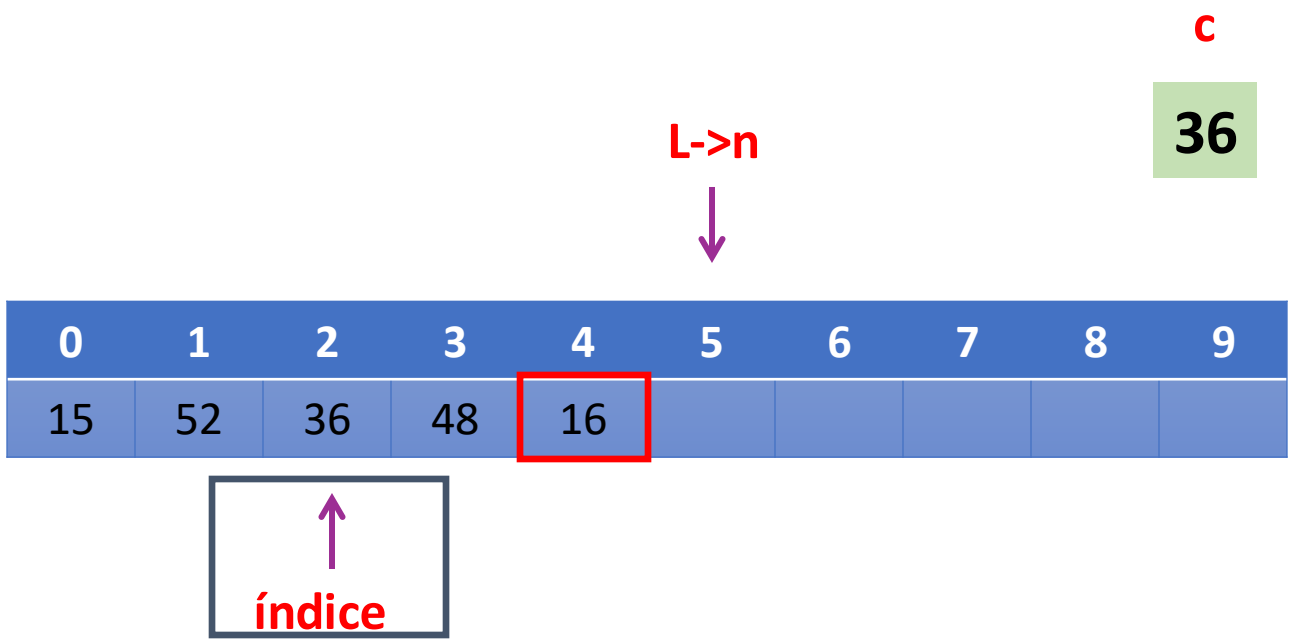


Algoritmos e Estruturas de Dados I

Operações:

- Remoção:
  - Listas não-ordenadas

Vamos copiar o último elemento para a posição do elemento a ser removido

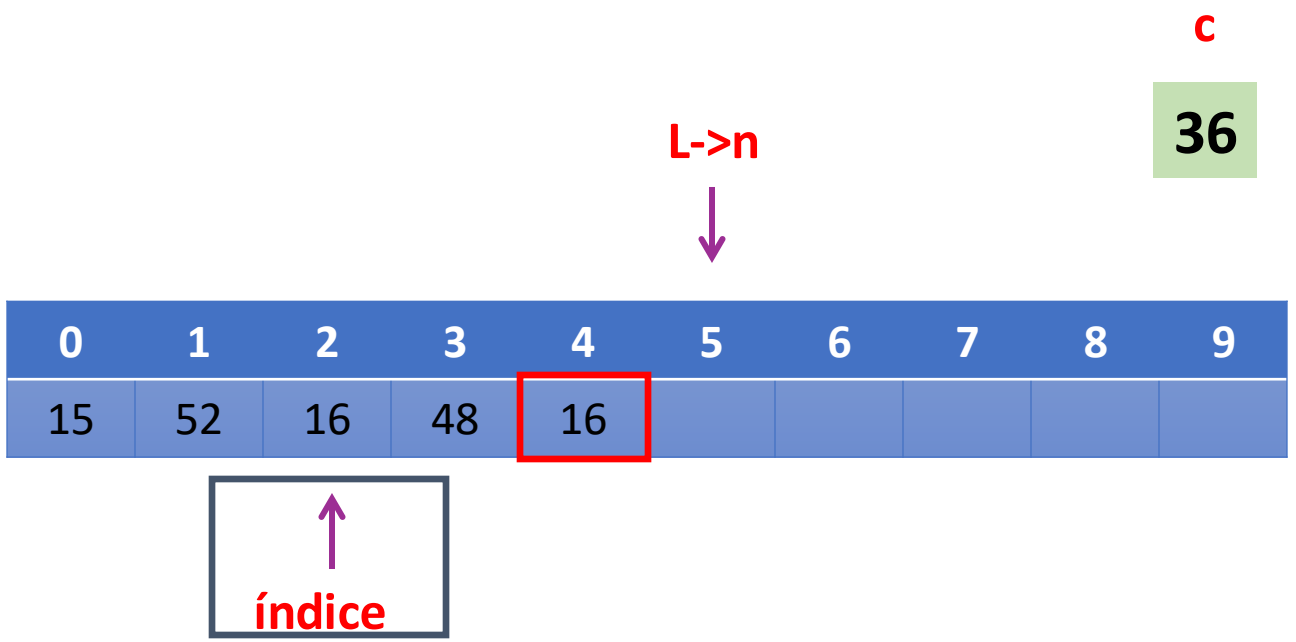


Algoritmos e Estruturas de Dados I

Operações:

- Remoção:
  - Listas não-ordenadas

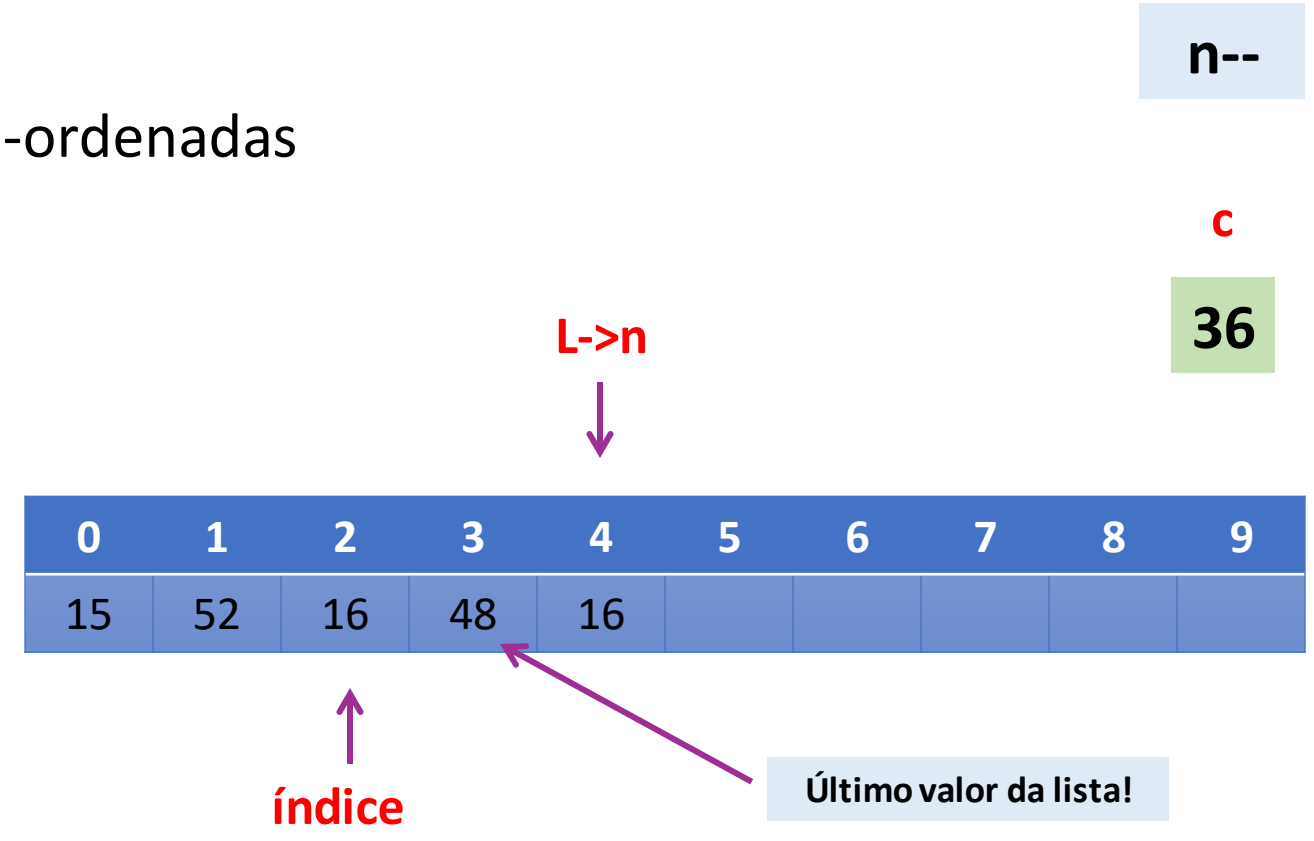
Vamos copiar o último elemento para a posição do elemento a ser removido



Algoritmos e Estruturas de Dados I

Operações:

- Remoção:
  - Listas não-ordenadas



Algoritmos e Estruturas de Dados I

Operações:

- Inserção – lista não-ordenada:

```
procedimento remove(ref L: ListaSeq, c: inteiro)
    i, índice: inteiro
    se L.n > 1 então
        indice ← buscaPosicao(L,c);
        se (indice ≠ 0) então
            L.valores[indice] ← L.valores[L.n]
            L.n ← L.n -1
        senão
            escreva(“Elemento não existente!”)
    senão
        escreva(“Lista vazia")
```

Tempo:
Melhor Caso: θ(1)
Pior Caso: θ(N)

```
void remover(listaSeq *L, int c){
    int i, indice;
    if (L->n > 0) {
            indice= buscaPosicao(L,c);
            if (indice != -1){
                L->valores[indice]=L->valores[L->n-1];
                (L->n)--;
            }
            else
                printf("\nElemento não existe\n");
    }
    else
            printf("\nLista vazia!\n");
}
```

FIM