

# **AULA 05**

# **ESTRUTURA DE DADOS**

---

**Lista ligada (implementação estática)**

**Norton T. Roman & Luciano A. Digiampietri**

# Lista linear sequencial

Na última aula aprendemos listas lineares sequenciais.

Realizamos a **inserção ordenada pela chave**:

# Lista linear sequencial

Na última aula aprendemos listas lineares sequenciais.

Realizamos a **inserção ordenada pela chave**:

- A **busca** era feita de maneira **eficiente** (busca binária);

# Lista linear sequencial

Na última aula aprendemos listas lineares sequenciais.

Realizamos a **inserção ordenada pela chave**:

- A **busca** era feita de maneira **eficiente** (busca binária);
- Porém **a inserção e a exclusão** eram **custosas**, pois potencialmente precisariam deslocar vários elementos.

# Lista ligada

Para evitar o deslocamento de elementos durante a inserção e a exclusão utilizaremos uma **lista ligada**:

# Lista ligada

Para evitar o deslocamento de elementos durante a inserção e a exclusão utilizaremos uma **lista ligada**:

- É uma **estrutura linear** (cada elemento possui no máximo um predecessor e um sucessor);

# Lista ligada

Para evitar o deslocamento de elementos durante a inserção e a exclusão utilizaremos uma **lista ligada**:

- É uma **estrutura linear** (cada elemento possui no máximo um predecessor e um sucessor);
- A **ordem lógica** dos elementos (a ordem “vista” pelo usuário) **não é a mesma ordem física** (em memória principal) dos elementos.

# Lista ligada

Para evitar o deslocamento de elementos durante a inserção e a exclusão utilizaremos uma **lista ligada**:

- É uma **estrutura linear** (cada elemento possui no máximo um predecessor e um sucessor);
- A **ordem lógica** dos elementos (a ordem “vista” pelo usuário) **não é a mesma ordem física** (em memória principal) dos elementos.
- Cada elemento precisa **indicar** quem é **o seu sucessor**.



# Lista ligada (implementação estática)

Chamaremos de lista ligada **implementação estática**, porque nossos registros serão armazenados em um **arranjo** criado inicialmente. Adicionalmente, cada elemento da nossa lista terá um campo para indicar a **posição** (no arranjo) **de seu sucessor**.

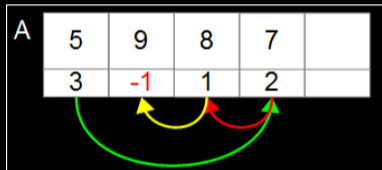
# Lista ligada (ideia geral)

Temos um arranjo de elementos  
Cada elemento indica seu sucessor

A	5	9	8	7	
	3	-1	1	2	

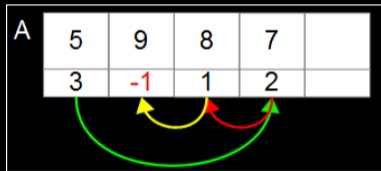
# Lista ligada (ideia geral)

Temos um arranjo de elementos  
Cada elemento indica seu sucessor



# Lista ligada (ideia geral)

Temos um arranjo de elementos  
Cada elemento indica seu sucessor  
Como excluimos o elemento 8?

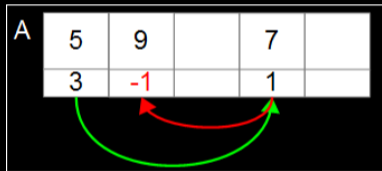


# Lista ligada (ideia geral)

Temos um arranjo de elementos

Cada elemento indica seu sucessor

Como excluimos o elemento 8?



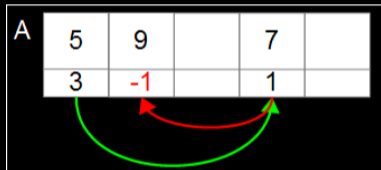
# Lista ligada (ideia geral)

Temos um arranjo de elementos

Cada elemento indica seu sucessor

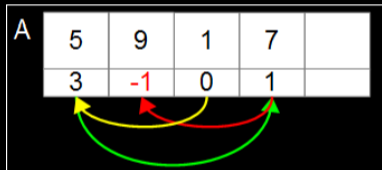
Como excluimos o elemento 8?

Como inserimos o elemento 1?



# Lista ligada (ideia geral)

- Temos um arranjo de elementos
- Cada elemento indica seu sucessor
- Como excluimos o elemento 8?
- Como inserimos o elemento 1?



# Lista ligada (ideia geral)

Precisamos tomar alguns **cuidados**:



# Lista ligada (ideia geral)

Precisamos tomar alguns **cuidados**:

Precisamos saber onde está o **primeiro elemento**;

# Lista ligada (ideia geral)

Precisamos tomar alguns **cuidados**:

Precisamos saber onde está o **primeiro elemento**;

Precisamos saber quais são os **elementos disponíveis**.

# Modelagem

```
#define MAX 50
#define INVALIDO -1

typedef int TIPOCHAVE;

typedef struct{
    TIPOCHAVE chave;
    // outros campos...
} REGISTRO;
```

```
typedef struct{
    REGISTRO reg;
    int prox;
} ELEMENTO;
```

```
typedef struct {
    ELEMENTO A[MAX];
    int inicio;
    int dispo;
} LISTA;
```

# Modelagem

```
#define MAX 50
#define INVALIDO -1

typedef int TIPOCHAVE;

typedef struct{
    TIPOCHAVE chave;
    // outros campos...
} REGISTRO;
```

```
typedef struct{
    REGISTRO reg;
    int prox;
} ELEMENTO;
```

```
typedef struct {
    ELEMENTO A[MAX];
    int inicio;
    int dispo;
} LISTA;
```

# Modelagem

```
#define MAX 50
#define INVALIDO -1

typedef int TIPOCHAVE;

typedef struct{
    TIPOCHAVE chave;
    // outros campos...
} REGISTRO;
```

```
typedef struct{
    REGISTRO reg;
    int prox;
} ELEMENTO;
```

```
typedef struct {
    ELEMENTO A[MAX];
    int inicio;
    int dispo;
} LISTA;
```

# Modelagem

```
#define MAX 50
#define INVALIDO -1

typedef int TIPOCHAVE;

typedef struct{
    TIPOCHAVE chave;
    // outros campos...
} REGISTRO;
```

```
typedef struct{
    REGISTRO reg;
    int prox;
} ELEMENTO;
```

```
typedef struct {
    ELEMENTO A[MAX];
    int inicio;
    int dispo;
} LISTA;
```

# Modelagem

```
#define MAX 50
#define INVALIDO -1

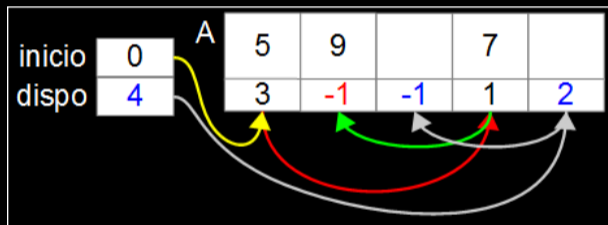
typedef int TIPOCHAVE;

typedef struct{
    TIPOCHAVE chave;
    // outros campos...
} REGISTRO;
```

```
typedef struct{
    REGISTRO reg;
    int prox;
} ELEMENTO;
```

```
typedef struct {
    ELEMENTO A[MAX];
    int inicio;
    int dispo;
} LISTA;
```

# Modelagem





# Funções de gerenciamento

Implementaremos funções para:

**Inicializar a estrutura**

**Retornar a quantidade de elementos válidos**

**Exibir os elementos da estrutura**

**Buscar por um elemento na estrutura**

**Inserir elementos na estrutura**

**Excluir elementos da estrutura**

**Reinicializar a estrutura**

# Inicialização

Para inicializarmos nossa lista ligada, **precisamos:**

- Colocar todos os elementos na “**lista**” de **disponíveis**;
- Acertar a variável **dispo** (primeiro item disponível);
- Acertar a variável **inicio** (para indicar que não há nenhum item válido);

# Inicialização

```
void inicializarLista(LISTA* l) {
```

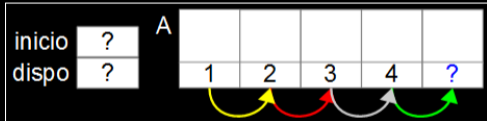
```
}
```

# Inicialização

```
void inicializarLista(LISTA* l) {  
    int i;  
    for (i=0; i<MAX-1; i++)  
        l->A[i].prox = i + 1;  
  
}
```

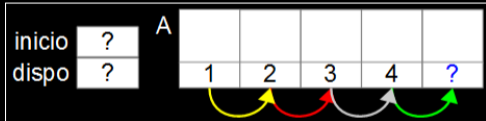
# Inicialização

```
void inicializarLista(LISTA* l) {  
    int i;  
    for (i=0; i<MAX-1; i++)  
        l->A[i].prox = i + 1;  
  
}
```



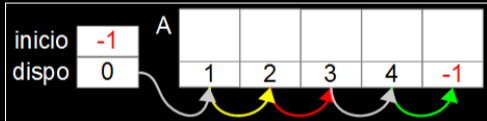
# Inicialização

```
void inicializarLista(LISTA* l) {  
    int i;  
    for (i=0; i<MAX-1; i++)  
        l->A[i].prox = i + 1;  
    l->A[MAX-1].prox=INVALIDO;  
    l->inicio=INVALIDO;  
    l->dispo=0;  
}
```



# Inicialização

```
void inicializarLista(LISTA* l) {  
    int i;  
    for (i=0; i<MAX-1; i++)  
        l->A[i].prox = i + 1;  
    l->A[MAX-1].prox=INVALIDO;  
    l->inicio=INVALIDO;  
    l->dispo=0;  
}
```



# Retornar número de elementos

Já que optamos por não criar um campo com o número de elementos na lista, precisaremos **percorrer todos os elementos válidos** para contar quantos são.



# Retornar número de elementos

```
int tamanho(LISTA* l) {
```

```
}
```

# Retornar número de elementos

```
int tamanho(LISTA* l) {  
    int i = l->inicio;  
    int tam = 0;  
  
}
```

# Retornar número de elementos

```
int tamanho(LISTA* l) {  
    int i = l->inicio;  
    int tam = 0;  
    while (i != INVALIDO) {  
        tam++;  
        i = l->A[i].prox;  
    }  
  
}
```

# Retornar número de elementos

```
int tamanho(LISTA* l) {  
    int i = l->inicio;  
    int tam = 0;  
    while (i != INVALIDO) {  
        tam++;  
        i = l->A[i].prox;  
    }  
    return tam;  
}
```

# Exibição/Impressão

Para exibir os elementos da estrutura precisaremos iterar pelos **elementos** válidos e, por exemplo, **imprimir suas chaves**.

# Exibição/Impressão

```
void exibirLista(LISTA* l) {
    int i = l->inicio;
    printf("Lista: \n ");
    while (i != INVALIDO) {
        printf("%i ", l->A[i].reg.chave);
        i = l->A[i].prox;
    }
    printf("\n\n");
}
```

# Buscar por elemento

A função de busca deverá:

**Receber uma chave** do usuário

**Retornar a posição** em que este elemento se encontra no arranjo (caso seja encontrado)

**Retornar *INVALIDO*** caso não haja um registro com essa chave na lista

# Busca secuencial (lista ordenada)

```
int buscaSecuencialOrd(LISTA* l, TIPOCHAVE ch) {
```

```
}
```



# Busca secuencial (lista ordenada)

```
int buscaSecuencialOrd(LISTA* l, TIPOCHAVE ch) {
    int i = l->inicio;

}
}
```

# Busca sequencial (lista ordenada)

```
int buscaSequencialOrd(LISTA* l, TIPOCHAVE ch) {  
    int i = l->inicio;  
    while (i != INVALIDO && l->A[i].reg.chave < ch)  
        i = l->A[i].prox;  
  
}
```

# Busca secuencial (lista ordenada)

```
int buscaSecuencialOrd(LISTA* l, TIPOCHAVE ch) {  
    int i = l->inicio;  
    while (i != INVALIDO && l->A[i].reg.chave < ch)  
        i = l->A[i].prox;  
    if (i != INVALIDO && l->A[i].reg.chave == ch)  
        return i;  
}
```

# Busca sequencial (lista ordenada)

```
int buscaSequencialOrd(LISTA* l, TIPOCHAVE ch) {
    int i = l->inicio;
    while (i != INVALIDO && l->A[i].reg.chave < ch)
        i = l->A[i].prox;
    if (i != INVALIDO && l->A[i].reg.chave == ch)
        return i;
    else return INVALIDO;
}
```

# Inserção de um elemento

O usuário passa como parâmetro um registro a ser inserido na lista

Realizaremos a inserção **ordenada pelo valor da chave** do registro passado e não permitiremos a inserção de **elementos repetidos**;

# Inserção de um elemento

O usuário passa como parâmetro um registro a ser inserido na lista

Realizaremos a inserção **ordenada pelo valor da chave** do registro passado e não permitiremos a inserção de **elementos repetidos**;

Na inserção precisamos identificar **entre quais elementos** o novo elemento será inserido;

# Inserção de um elemento

O usuário passa como parâmetro um registro a ser inserido na lista

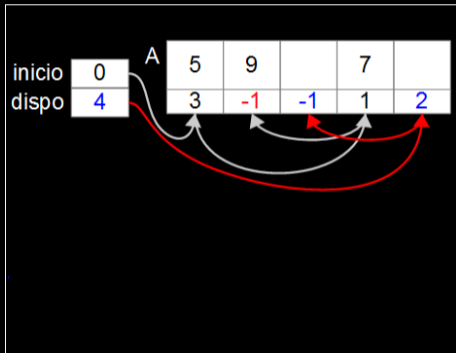
Realizaremos a inserção **ordenada pelo valor da chave** do registro passado e não permitiremos a inserção de **elementos repetidos**;

Na inserção precisamos identificar **entre quais elementos** o novo elemento será inserido;

O novo elemento será inserido no lugar do primeiro que estiver na **lista de disponíveis**.

# Inserção ordenada

Queremos inserir o 8

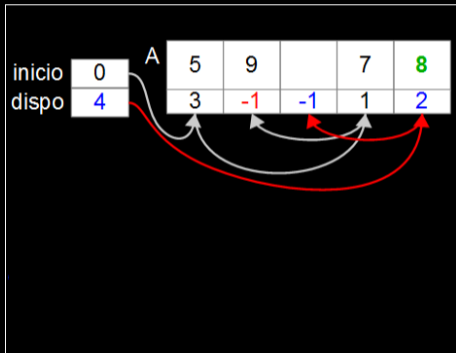




# Inserção ordenada

Queremos inserir o 8

Ele será inserido na **primeira posição disponível** (posição física)

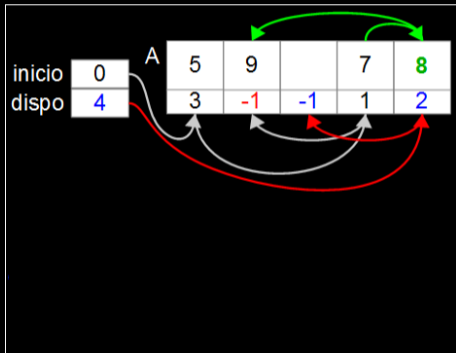


# Inserção ordenada

Queremos inserir o 8

Ele será inserido na **primeira posição disponível** (posição física)

Temos que descobrir **entre quais elementos** será inserido (ordem lógica)



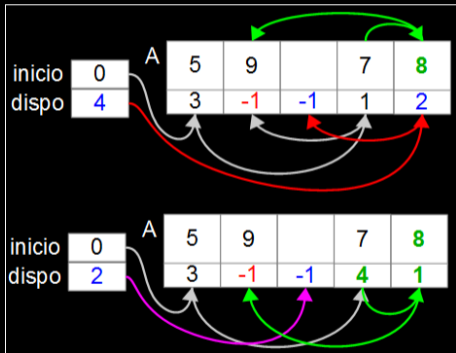
# Inserção ordenada

Queremos inserir o 8

Ele será inserido na **primeira posição disponível** (posição física)

Temos que descobrir **entre quais elementos** será inserido (ordem lógica)

Temos que **acertar os “ponteiros”**



# Inserção ordenada

Iniciaremos por uma **função auxiliar** que **retira** o primeiro elemento da **lista de disponíveis** e retorna sua posição no arranjo.

# Inserção ordenada - função auxiliar

```
int obterNo(LISTA* l) {  
    int resultado = l->dispo;  
    if (l->dispo != INVALIDO)  
        l->dispo = l->A[l->dispo].prox;  
    return resultado;  
}
```

# Inserção ordenada - parte 1

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {
```

# Inserção ordenada - parte 1

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {  
    if (l->dispo == INVALIDO) return false;
```

# Inserção ordenada - parte 1

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {  
    if (l->dispo == INVALIDO) return false;  
    int ant = INVALIDO;  
    int i = l->inicio;  
    TIPOCHAVE ch = reg.chave;
```



# Inserção ordenada - parte 1

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {
    if (l->dispo == INVALIDO) return false;
    int ant = INVALIDO;
    int i = l->inicio;
    TIPOCHAVE ch = reg.chave;
    while ((i != INVALIDO) && (l->A[i].reg.chave < ch)){
        ant = i;
        i = l->A[i].prox;
    }
}
```

# Inserção ordenada - parte 1

```
bool inserirElemListaOrd(LISTA* l, REGISTRO reg) {
    if (l->dispo == INVALIDO) return false;
    int ant = INVALIDO;
    int i = l->inicio;
    TIPOCHAVE ch = reg.chave;
    while ((i != INVALIDO) && (l->A[i].reg.chave < ch)){
        ant = i;
        i = l->A[i].prox;
    }
    if (i != INVALIDO && l->A[i].reg.chave == ch) return false;
```

# Inserção ordenada - parte 2

```
i = obterNo(l);  
l->A[i].reg = reg;
```

```
}
```

# Inserção ordenada - parte 2

```
i = obterNo(l);  
l->A[i].reg = reg;  
if (ant == INVALIDO) {  
    l->A[i].prox = l->inicio;  
    l->inicio = i;  
}
```

```
}
```

# Inserção ordenada - parte 2

```
i = obterNo(l);
l->A[i].reg = reg;
if (ant == INVALIDO) {
    l->A[i].prox = l->inicio;
    l->inicio = i;
} else {
    l->A[i].prox = l->A[ant].prox;
    l->A[ant].prox = i;
}
}
```

# Inserção ordenada - parte 2

```
i = obterNo(l);  
l->A[i].reg = reg;  
if (ant == INVALIDO) {  
    l->A[i].prox = l->inicio;  
    l->inicio = i;  
} else {  
    l->A[i].prox = l->A[ant].prox;  
    l->A[ant].prox = i;  
}  
return true;  
}
```

# Exclusão de um elemento

O usuário passa a chave do elemento que ele quer excluir

Se houver um elemento com esta chave na lista, **“exclui este elemento”** da lista de elementos válidos e o **insere** na lista de disponíveis.

Adicionalmente, **acerta os ponteiros** envolvidos e retorna *true*.

Caso contrário, retorna *false*

# Exclusão de um elemento

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {
```



# Exclusão de um elemento

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {
    int ant = INVALIDO;
    int i = l->inicio;
    while ((i != INVALIDO) && (l->A[i].reg.chave<ch)) {
        ant = i;
        i = l->A[i].prox;
    }
}
```

# Exclusão de um elemento

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {
    int ant = INVALIDO;
    int i = l->inicio;
    while ((i != INVALIDO) && (l->A[i].reg.chave<ch)) {
        ant = i;
        i = l->A[i].prox;
    }
    if (i==INVALIDO || l->A[i].reg.chave!=ch) return false;
```

# Exclusão de um elemento

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {
    int ant = INVALIDO;
    int i = l->inicio;
    while ((i != INVALIDO) && (l->A[i].reg.chave<ch)) {
        ant = i;
        i = l->A[i].prox;
    }
    if (i==INVALIDO || l->A[i].reg.chave!=ch) return false;
    if (ant == INVALIDO) l->inicio = l->A[i].prox;
```

# Exclusão de um elemento

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {
    int ant = INVALIDO;
    int i = l->inicio;
    while ((i != INVALIDO) && (l->A[i].reg.chave<ch)) {
        ant = i;
        i = l->A[i].prox;
    }
    if (i==INVALIDO || l->A[i].reg.chave!=ch) return false;
    if (ant == INVALIDO) l->inicio = l->A[i].prox;
    else l->A[ant].prox = l->A[i].prox;
}
```

# Exclusão de um elemento

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {
    int ant = INVALIDO;
    int i = l->inicio;
    while ((i != INVALIDO) && (l->A[i].reg.chave<ch)) {
        ant = i;
        i = l->A[i].prox;
    }
    if (i==INVALIDO || l->A[i].reg.chave!=ch) return false;
    if (ant == INVALIDO) l->inicio = l->A[i].prox;
    else l->A[ant].prox = l->A[i].prox;
    devolverNo(l,i);
}
```

# Exclusão de um elemento

```
bool excluirElemLista(LISTA* l, TIPOCHAVE ch) {
    int ant = INVALIDO;
    int i = l->inicio;
    while ((i != INVALIDO) && (l->A[i].reg.chave<ch)) {
        ant = i;
        i = l->A[i].prox;
    }
    if (i==INVALIDO || l->A[i].reg.chave!=ch) return false;
    if (ant == INVALIDO) l->inicio = l->A[i].prox;
    else l->A[ant].prox = l->A[i].prox;
    devolverNo(l,i);
    return true;
}
```

# Exclusão de um elemento - função auxiliar

Desejamos devolver um elemento para a **lista de disponíveis**

# Exclusão de um elemento - função auxiliar

Desejamos devolver um elemento para a **lista de disponíveis**

Em que **posição** da lista iremos colocar esse elemento?



# Exclusão de um elemento - função auxiliar

Desejamos devolver um elemento para a **lista de disponíveis**

Em que **posição** da lista iremos colocar esse elemento?

Na que é mais fácil de inserir: no **início**

# Exclusão de um elemento - função auxiliar

```
void devolverNo(LISTA* l, int j) {  
    l->A[j].prox = l->dispo;  
    l->dispo = j;  
}
```

# Reinicialização da lista

Para reinicializar a estrutura basta chamarmos a **função de inicialização**.

# Reinicialização da lista

```
void reinicializarLista(LISTA* l) {  
    inicializarLista(l);  
}
```

# **AULA 05**

# **ESTRUTURA DE DADOS**

---

**Lista ligada (implementação estática)**

**Norton T. Roman & Luciano A. Digiampietri**