

# **AULA 13**

# **ESTRUTURA DE DADOS**

---

**Matriz esparsa**

# Matriz

Uma **matriz bidimensional** é um conjunto de elementos (ou tabela) composta por  **$m$  linhas** e  **$n$  colunas**.

# Matriz

Uma **matriz bidimensional** é um conjunto de elementos (ou tabela) composta por  **$m$  linhas** e  **$n$  colunas**.

- Em computação utilizamos matrizes para **representar diferentes tipos de dados** (dados numéricos, imagens, etc.)

# Matriz esparsa

# Matriz esparsa

É uma matriz na qual a **grande maioria de seus elementos possui um valor padrão** (por exemplo zero) ou são nulos ou faltantes.

# Matriz esparsa

É uma matriz na qual a **grande maioria de seus elementos possui um valor padrão** (por exemplo zero) ou são nulos ou faltantes.

- Por exemplo, uma matriz que representa o **contorno de uma imagem em preto e branco**

# Matriz esparsa

É uma matriz na qual a **grande maioria de seus elementos possui um valor padrão** (por exemplo zero) ou são nulos ou faltantes.

- Por exemplo, uma matriz que representa o **contorno de uma imagem em preto e branco**
- Seria um **desperdício gastar  $m \times n$  posições de memória** sendo que apenas uma pequena parcela dos elementos tem valor diferente de zero

# Matriz esparsa

Para **evitar o desperdício** de memória (e eventualmente de processamento) criamos uma **estrutura específica** para gerenciar matrizes esparsas.



# Matriz esparsa

Para **evitar o desperdício** de memória (e eventualmente de processamento) criamos uma **estrutura específica** para gerenciar matrizes esparsas.

- Só serão alocados **elementos com valor diferente de zero** e alguma estrutura adicional de controle.

# Matriz esparsa - implementação

# Matriz esparsa - implementação

Cada linha da matriz será representada por uma lista ligada que só conterá **elementos com valores diferentes de zero**;

# Matriz esparsa - implementação

Cada linha da matriz será representada por uma lista ligada que só conterá **elementos com valores diferentes de zero**;

Teremos um **arranjo de listas ligadas**.

# Matriz esparsa - implementação

	0	1	2	3
0	0,0	0,0	5,2	0,0
1	0,0	0,0	0,0	0,0
2	0,0	3,0	0,0	4,0

**Temos uma matriz com muitos zeros.**

# Matriz esparsa - implementação

	0	1	2	3
0	0,0	0,0	5,2	0,0
1	0,0	0,0	0,0	0,0
2	0,0	3,0	0,0	4,0

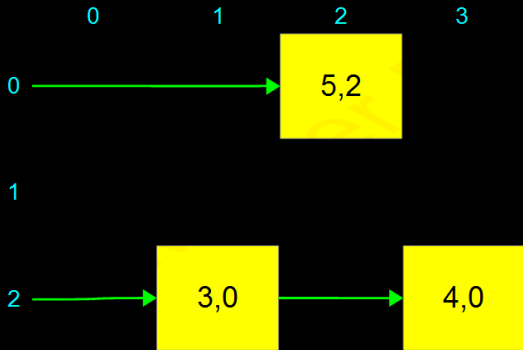
Queremos armazenar apenas os valores **diferentes de zero**.

# Matriz esparsa - implementação

	0	1	2	3
0			5,2	
1				
2		3,0		4,0

Queremos armazenar apenas os valores **diferentes de zero**.

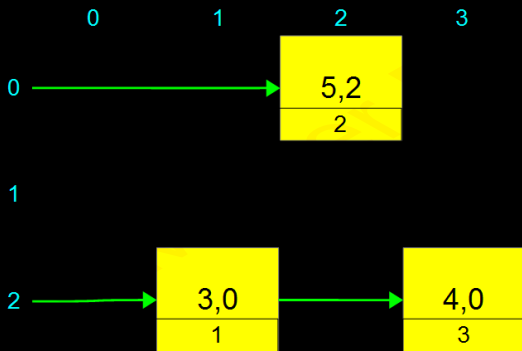
# Matriz esparsa - implementação



Criaremos listas de elementos **diferentes de zero**.



# Matriz esparsa - implementação



Cada elemento apontará para seu **vizinho** e saberá sua **coluna**.

# Modelagem

```
#include <stdio.h>
#include <malloc.h>

typedef struct tempNo {
    float valor;
    int coluna;
    struct tempNo* prox;
} NO;
```

```
typedef NO* PONT;

typedef struct {
    PONT* A;
    int linhas;
    int colunas;
} MATRIZ;
```

# Modelagem

```
#include <stdio.h>
#include <malloc.h>
```

```
typedef struct tempNo {
    float valor;
    int coluna;
    struct tempNo* prox;
} NO;
```

```
typedef NO* PONT;
```

```
typedef struct {
    PONT* A;
    int linhas;
    int colunas;
} MATRIZ;
```

# Modelagem

```
#include <stdio.h>
#include <malloc.h>
```

```
typedef struct tempNo {
    float valor;
    int coluna;
    struct tempNo* prox;
} NO;
```

```
typedef NO* PONT;
```

```
typedef struct {
    PONT* A;
    int linhas;
    int colunas;
} MATRIZ;
```

# Modelagem

```
#include <stdio.h>
#include <malloc.h>

typedef struct tempNo {
    float valor;
    int coluna;
    struct tempNo* prox;
} NO;
```

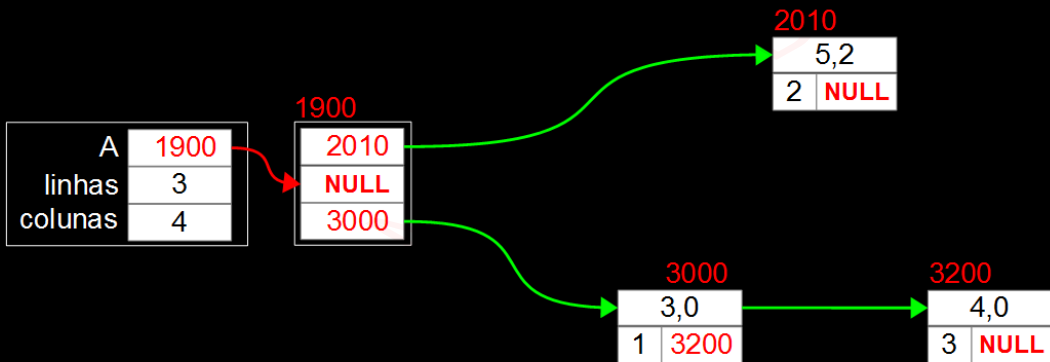
```
typedef NO* PONT;

typedef struct {
    PONT* A;
    int linhas;
    int colunas;
} MATRIZ;
```

# Modelagem

	0	1	2	3
0	0,0	0,0	5,2	0,0
1	0,0	0,0	0,0	0,0
2	0,0	3,0	0,0	4,0

# Modelagem



# Funções de gerenciamento

Implementaremos funções para:

**Inicializar a matriz** (“*new matriz[m][n]*”)

**Atribuir um valor** (*matriz[x][y] = valor*)

**Acessar valor** (*valor = matriz[x][y]*)



# Inicialização

Para inicializar nossa matriz esparsa, nós precisamos:

# Inicialização

Para inicializar nossa matriz esparsa, nós precisamos:

Acertar o valor dos campos *linhas* e *colunas* (isto é, a ordem da matriz passada pelo usuário).

# Inicialização

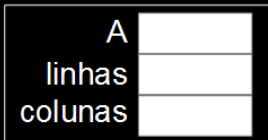
Para inicializar nossa matriz esparsa, nós precisamos:

Acertar o valor dos campos **linhas** e **colunas** (isto é, a ordem da matriz passada pelo usuário).  
Precisamos também **criar o arranjo de listas ligadas** e iniciar cada posição do arranjo com o valor **NULL** (indicando que **cada lista está vazia**).

# Inicialização

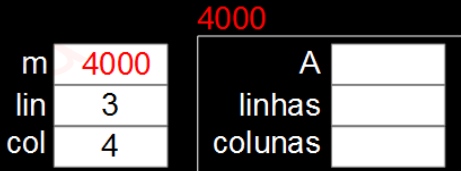
```
void inicializarMatriz(MATRIZ* m, int lin, int col) {  
    int i;  
    m->linhas = lin;  
    m->colunas = col;  
    m->A = (PONT*) malloc(lin*sizeof(PONT));  
    for (i=0;i<lin;i++) m->A[i] = NULL;  
}
```

4000



# Inicialização

```
void inicializarMatriz(MATRIZ* m, int lin, int col) {  
    int i;  
    m->linhas = lin;  
    m->colunas = col;  
    m->A = (PONT*) malloc(lin*sizeof(PONT));  
    for (i=0;i<lin;i++) m->A[i] = NULL;  
}
```



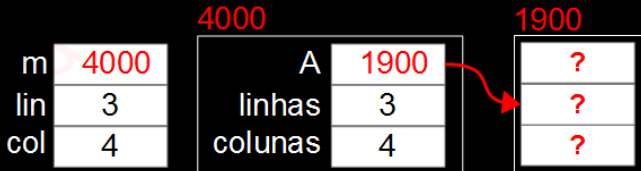
# Inicialização

```
void inicializarMatriz(MATRIZ* m, int lin, int col) {  
    int i;  
    m->linhas = lin;  
    m->colunas = col;  
    m->A = (PONT*) malloc(lin*sizeof(PONT));  
    for (i=0;i<lin;i++) m->A[i] = NULL;  
}
```

		4000	
m			
lin	3	A	
col	4	linhas	3
		colunas	4

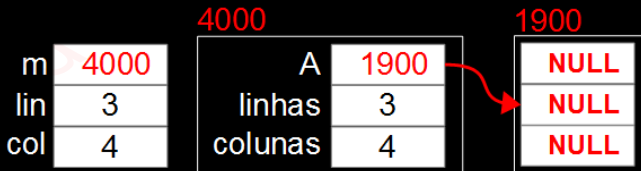
# Inicialização

```
void inicializarMatriz(MATRIZ* m, int lin, int col) {  
    int i;  
    m->linhas = lin;  
    m->colunas = col;  
    m->A = (PONT*) malloc(lin*sizeof(PONT));  
    for (i=0;i<lin;i++) m->A[i] = NULL;  
}
```



# Inicialização

```
void inicializarMatriz(MATRIZ* m, int lin, int col) {  
    int i;  
    m->linhas = lin;  
    m->colunas = col;  
    m->A = (PONT*) malloc(lin*sizeof(PONT));  
    for (i=0;i<lin;i++) m->A[i] = NULL;  
}
```





# Atribuir valor

O usuário nos passa: o endereço da matriz, a **linha**, a **coluna** e o **valor** a ser colocado na respectiva posição da matriz:

# Atribuir valor

O usuário nos passa: o endereço da matriz, a **linha**, a **coluna** e o **valor** a ser colocado na respectiva posição da matriz:

Se **não houver nenhum nó na posição** e o valor for **diferente de zero** temos que **inserir um novo nó na respectiva lista ligada**.

# Atribuir valor

O usuário nos passa: o endereço da matriz, a **linha**, a **coluna** e o **valor** a ser colocado na respectiva posição da matriz:

Se **não houver nenhum nó na posição** e o valor for **diferente de zero** temos que **inserir um novo nó na respectiva lista ligada**.

Se **já existir um nó na posição** e o valor for **diferente de zero** temos que **substituir o valor do nó**.

# Atribuir valor

O usuário nos passa: o endereço da matriz, a **linha**, a **coluna** e o **valor** a ser colocado na respectiva posição da matriz:

Se **não houver nenhum nó na posição** e o valor for **diferente de zero** temos que **inserir um novo nó na respectiva lista ligada**.

Se **já existir um nó na posição** e o valor for **diferente de zero** temos que **substituir o valor do nó**.

Se **já existir um nó na posição** e o valor for **igual a zero** temos que **excluir o nó de sua lista**.

# Atribuir valor - **busca**

```
bool AtribuiMatriz(MATRIZ* m,int lin, int col,  
                  float val) {
```

# Atribuir valor - **busca**

```
bool AtribuiMatriz(MATRIZ* m,int lin, int col,  
                  float val) {  
    if (lin<0 || lin >= m->linhas ||  
        col<0 || col >= m->colunas) return false;
```

# Atribuir valor - **busca**

```
bool AtribuiMatriz(MATRIZ* m,int lin, int col,
                  float val) {
    if (lin<0 || lin >= m->linhas ||
        col<0 || col >= m->colunas) return false;
    PONT ant = NULL;
    PONT atual = m->A[lin];
```

# Atribuir valor - busca

```
bool AtribuiMatriz(MATRIZ* m,int lin, int col,
                  float val) {
    if (lin<0 || lin >= m->linhas ||
        col<0 || col >= m->colunas) return false;
    PONT ant = NULL;
    PONT atual = m->A[lin];
    while (atual != NULL && atual->coluna<col) {
        ant = atual;
        atual = atual->prox;
    }
```



# Atribuir valor - **nó existe**

```
if (atual != NULL && atual->coluna == col) {
```

```
}
```

# Atribuir valor - **nó existe**

```
if (atual != NULL && atual->coluna == col) {  
    if (val == 0) {  
  
    }  
  
}
```

# Atribuir valor - **nó existe**

```
if (atual != NULL && atual->coluna == col) {  
    if (val == 0) {  
  
        free(atual);  
    }  
  
}
```

# Atribuir valor - **nó existe**

```
if (atual != NULL && atual->coluna == col) {  
    if (val == 0) {  
        if (ant==NULL) m->A[lin] = atual->prox;  
  
        free(atual);  
    }  
  
}
```

# Atribuir valor - **nó existe**

```
if (atual != NULL && atual->coluna == col) {  
    if (val == 0) {  
        if (ant==NULL) m->A[lin] = atual->prox;  
        else ant->prox = atual->prox;  
        free(atual);  
    }  
  
}
```

# Atribuir valor - **nó existe**

```
if (atual != NULL && atual->coluna == col) {  
    if (val == 0) {  
        if (ant==NULL) m->A[lin] = atual->prox;  
        else ant->prox = atual->prox;  
        free(atual);  
    }  
    else atual->valor = val;  
}
```

# Atribuir valor - **nó não existe**

```
else if (val != 0) {
```

```
}
```

```
}
```

# Atribuir valor - **nó não existe**

```
else if (val != 0) {  
    PONT novo = (PONT) malloc(sizeof(NO));
```

```
}
```

```
}
```



# Atribuir valor - **nó não existe**

```
else if (val != 0) {  
    PONT novo = (PONT) malloc(sizeof(NO));  
    novo->coluna = col;  
    novo->valor = val;
```

```
}
```

```
}
```

# Atribuir valor - **nó não existe**

```
else if (val != 0) {  
    PONT novo = (PONT) malloc(sizeof(NO));  
    novo->coluna = col;  
    novo->valor = val;  
    novo->prox = atual;  
  
}  
  
}
```

# Atribuir valor - **nó não existe**

```
else if (val != 0) {  
    PONT novo = (PONT) malloc(sizeof(NO));  
    novo->coluna = col;  
    novo->valor = val;  
    novo->prox = atual;  
    if (ant == NULL) m->A[lin] = novo;  
  
}  
  
}
```

# Atribuir valor - **nó não existe**

```
else if (val != 0) {  
    PONT novo = (PONT) malloc(sizeof(NO));  
    novo->coluna = col;  
    novo->valor = val;  
    novo->prox = atual;  
    if (ant == NULL) m->A[lin] = novo;  
    else ant->prox = novo;  
}  
  
}
```

# Atribuir valor - **nó não existe**

```
else if (val != 0) {  
    PONT novo = (PONT) malloc(sizeof(NO));  
    novo->coluna = col;  
    novo->valor = val;  
    novo->prox = atual;  
    if (ant == NULL) m->A[lin] = novo;  
    else ant->prox = novo;  
}  
return true;  
}
```

# Acessar valor

O usuário nos passa: o endereço da matriz, a **linha** e a **coluna** e a função deve **retornar o valor da respectiva posição**:

# Acessar valor

O usuário nos passa: o endereço da matriz, a **linha** e a **coluna** e a função deve **retornar o valor da respectiva posição**:

**Se não houver um nó na posição** então **retornar zero**;

# Acessar valor

O usuário nos passa: o endereço da matriz, a **linha** e a **coluna** e a função deve **retornar o valor da respectiva posição**:

**Se não houver um nó na posição** então **retornar zero**;  
**Caso contrário**, **retornar o valor do nó**.



# Acessar valor

```
float ValorMatriz(MATRIZ* m, int lin, int col) {
```

```
}
```

# Acessar valor

```
float ValorMatriz(MATRIZ* m, int lin, int col) {  
    if (lin<0 || lin>=m->linhas || col<0 ||  
        col >= m->colunas) return 0;  
  
}
```

# Acessar valor

```
float ValorMatriz(MATRIZ* m, int lin, int col) {  
    if (lin<0 || lin>=m->linhas || col<0 ||  
        col >= m->colunas) return 0;  
    PONT atual = m->A[lin];  
    while (atual != NULL && atual->coluna < col)  
        atual = atual->prox;  
  
}
```

# Acessar valor

```
float ValorMatriz(MATRIZ* m, int lin, int col) {  
    if (lin<0 || lin>=m->linhas || col<0 ||  
        col >= m->colunas) return 0;  
    PONT atual = m->A[lin];  
    while (atual != NULL && atual->coluna < col)  
        atual = atual->prox;  
    if (atual !=NULL && atual->coluna == col)  
        return atual->valor;  
}
```

# Acessar valor

```
float ValorMatriz(MATRIZ* m, int lin, int col) {  
    if (lin<0 || lin>=m->linhas || col<0 ||  
        col >= m->colunas) return 0;  
    PONT atual = m->A[lin];  
    while (atual != NULL && atual->coluna < col)  
        atual = atual->prox;  
    if (atual !=NULL && atual->coluna == col)  
        return atual->valor;  
    return 0;  
}
```

# **AULA 13**

# **ESTRUTURA DE DADOS**

---

**Matriz esparsa**