



# Estruturas de Dados

### Matrizes e Matrizes Esparsas

Professores: Anisio Lacerda

Wagner Meira Jr.

Matrizes são estruturas extremamente versáteis e amplamente utilizadas.

O objetivo deste módulo é entender um pouco mais desse tipo de dado e conhecer algumas maneiras de atacar um de seus maiores problemas: consumo de espaço.

Assim como nos outros TADs que estudamos, vamos ilustrar algumas operações, são elas:

- Construir uma matriz
- Obter e alterar elementos da matriz.
- Imprimir seus elementos
- Produto de uma matriz com um vetor denso.
- Soma de duas matrizes.

Antes de entendermos a diferença entre matrizes densas e esparsas, vamos criar um TAD básico de matrizes com essas operações.

#### Observação importante!

As implementações nesta seção não necessariamente são as mais otimizadas, o propósito delas é ilustrar e exemplificar o TAD.

```
class Matriz{
    public:
        Matriz (int linhas, int colunas, int leTeclado);
        ~Matriz();
        double Get(int linha, int coluna);
        void Set(int linha, int coluna, double valor);
        void Imprime();
        static Matriz Soma (Matriz A, Matriz B);
        static double* Produto (Matriz A, double* v, int n);
    private:
        double** M;
        int 1, c;
        void LeMatriz();
};
```

```
Matriz::Matriz(int linhas, int colunas, int leTeclado) {
    l = linhas,
    c = colunas;
    M = \text{new double*}[1];
    if(!leTeclado){
        for (int i = 0; i < 1; i++) {
             M[i] = new double[c];
             for (int j = 0; j < c; j++)
                 M[i][j] = 0;
     else {
       LeMatriz();
```

```
void Matriz::LeMatriz() {
    for (int i = 0; i < 1; i++) {
        M[i] = new double[c];
        for (int j = 0; j < c; j++) {
           cin >> M[i][j];
Matriz::~Matriz() {
    for(int i = 0; i < 1; i++){
        delete[](M[i]);
    delete[](M);
```

```
double Matriz::Get(int linha, int coluna) {
    if(linha < 0 | linha >= 1 | l coluna < 0 | l coluna >= c) {
        throw "Posicao invalida!!!\n";
    return M[linha][coluna];
void Matriz::Set(int linha, int coluna, double valor) {
    if (linha < 0 \mid \mid linha >= 1 \mid \mid coluna < 0 \mid \mid coluna >= c) {
        throw "Posicao invalida!!!\n";
    M[linha][coluna] = valor;
```

```
void Matriz::Imprime() {
    for(int i = 0; i < 1; i++) {
        for(int j = 0; j < c; j++) {
            cout << M[i][j] << " ";
        }
        cout << "\n";
    }
}</pre>
```

```
double* Matriz::Produto(Matriz A, double* v, int n) {
    if(A.c != n) {
        throw "Dimensoes das matrizes incompativeis\n";
    double* s = new double[n];
    for (int i = 0; i < n; i++) {
        s[i] = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            s[i] += A.M[i][j] * v[j];
    return s;
```

```
Matriz Matriz::Soma(Matriz A, Matriz B) {
    if (A.1 != B.1 || A.c != B.c) {
        throw "Dimensoes das matrizes incompativeis\n";
    Matriz C = Matriz(A.l, A.c, 0);
    for(int i = 0; i < A.l; i++){</pre>
        for (int j = 0; j < A.c; j++) {
            C.Set(i, j, A.M[i][j]+B.M[i][j]);
    return C;
```

### Matrizes densas vs esparsas

Uma matriz é densa quando a maior parte de seus elementos não são 0, e é esparsa quando a maior parte dos elementos é 0.

### Matrizes densas vs esparsas

- Uma matriz é densa quando a maior parte de seus elementos não são 0, e é esparsa quando a maior parte dos elementos é 0.
- Note que no TAD dos slides anteriores, caso a matriz seja esparsa, muita memória e computação nas operações serão gastos com os elementos nulos.

### Matrizes densas vs esparsas

- Uma matriz é densa quando a maior parte de seus elementos não são 0, e é esparsa quando a maior parte dos elementos é 0.
- Note que no TAD dos slides anteriores, caso a matriz seja esparsa, muita memória e computação nas operações serão gastos com os elementos nulos.
- Nosso objetivo é encontrar formas de representar matrizes as quais reduzimos os gasto de memória e computação com elementos nulos.

### Matrizes esparsas

 Nosso objetivo é encontrar uma representação que omita os elementos nulos.

### Matrizes esparsas

- Nosso objetivo é encontrar uma representação que omita os elementos nulos.
- Uma forma simples de se pensar é apenas endereçar os elementos de interesse, no nosso caso os não nulos.

### Matrizes esparsas

- Nosso objetivo é encontrar uma representação que omita os elementos nulos.
- Uma forma simples de se pensar é apenas endereçar os elementos de interesse, no nosso caso os não nulos.
- Chamamos essa forma de representar matrizes esparsas de Coordinate list (COO).

Nessa representação para cada elemento iremos armazenar três informações:

- Nessa representação para cada elemento iremos armazenar três informações:
  - A linha onde o elemento está.

- Nessa representação para cada elemento iremos armazenar três informações:
  - A linha onde o elemento está.
  - A coluna onde o elemento está.

- Nessa representação para cada elemento iremos armazenar três informações:
  - A linha onde o elemento está.
  - A coluna onde o elemento está.
  - O valor do elemento.

Dessa forma nossa estrutura será um arranjo em que cada "linha" armazena essas três informações.

Exemplo de uma matriz e sua representação

3	0	0	15
0	0	4	0
0	7	0	0
0	0	0	12

linhas:

0
---

3

2

3

3

colunas:

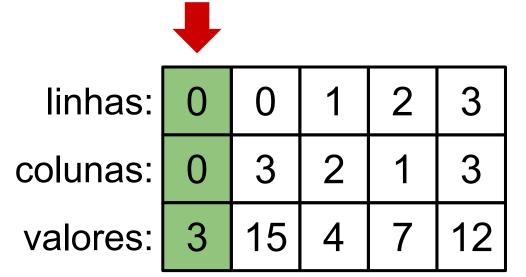
2 | 1

valores:

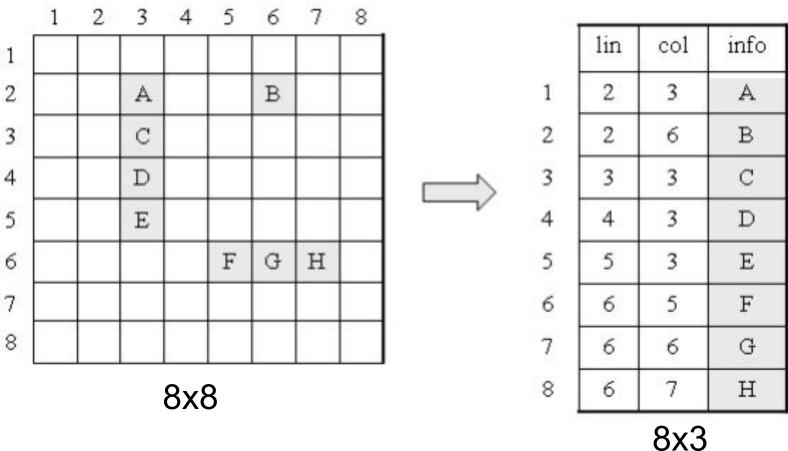
3 | 15 | 4 | 7 | 12

Apesar da apresentação visual, por questões de localidade de referência, cada entrada será representada como uma "linha" em nossa estrutura.

3	0	0	15
0	0	4	0
0	7	0	0
0	0	0	12



### Outro exemplo COO (um pouco maior)



```
class MatrizCOO{
   public:
        MatrizCOO(int linhas, int colunas, int leTeclado);
        ~MatrizCOO();
        double Get(int linha, int coluna);
        void Set(int linha, int coluna, double valor);
        void Imprime();
        static MatrizCOOSoma (MatrizCOO A, MatrizCOO B);
        static double* Produto (MatrizCOO A, double* v, int t);
    private:
        double** M;
        int tamanho, tamanhoMAX, 1, c;
        void LeMatriz();
};
```

Observação: Ao longo desta implementação faremos uso de algumas funções auxiliares. Suas implementações serão omitidas pois são simples e não tem relação direta com o conteúdo do módulo.

```
double** GetArranjoTamanhoN(int N);
```

Esta função retorna um arranjo com N linhas e 3 colunas com 0 em todos os elementos.

```
void InsereElemento(int 1, int c, int v);
```

Esta função insere um elemento de valor v na linha l e coluna c em nossa representação, aumentando o tamanho do arranjo caso necessário.

Além disso, após a inserção ela ordena os elementos por linha coluna.

```
void RemoveElemento(int p);
```

Esta função remove um elemento na posição *p* da nossa representação.

Além disso, após a remoção ela ordena os elementos por linha coluna.

- Construir uma matriz.
- Obter e alterar elementos da matriz.
- Imprimir seus elementos.
- Produto de uma matriz com um vetor denso.
- Soma de duas matrizes.

- Construir uma matriz.
- Obter e alterar elementos da matriz.
- Imprimir seus elementos.
- Produto de uma matriz com um vetor denso.
- Soma de duas matrizes.

Para ilustrar a construção, vamos supor que iremos receber os dados da matriz linha por linha, incluindo os elementos nulos.

```
MatrizCOO::MatrizCOO(int linhas, int colunas, int leTeclado) {
    l = linhas;
    c = colunas;
    tamanho = 0;
    if(!leTeclado) {
        tamanhoMAX = 1;
        M = GetArranjoTamanhoN(tamanhoMAX);
    else{
        LeMatriz();
```

```
MatrizCOO::LeMatriz() {
    tamanhoMAX = floor(0.3*1*c);
    M = GetArranjoTamanhoN(tamanhoMAX);
    double x;
    for(int i = 0; i < 1; i++) {
        for(int j = 0; j < c; j++) {
            cin >> x;
            InsereElemento(i, j, x);
        }
    }
}
```

Considere a matriz abaixo. Vamos construir sua representação COO.

3	0	0	15
0	0	4	0
0	7	0	0
0	0	0	12

Vamos construir nossa tabela que irá armazenar as triplas de cada elemento não nulo.

3	0	0	15
0	0	4	0
0	7	0	0
0	0	0	12

linhas:

colunas:

valores:

Agora iremos ler os dados da matriz, e considerar apenas os elementos não nulos.

3	0	0	15
0	0	4	0
0	7	0	0
0	0	0	12

linhas: 0

colunas: 0

valores: 3

Agora iremos ler os dados da matriz, e considerar apenas os elementos não nulos.

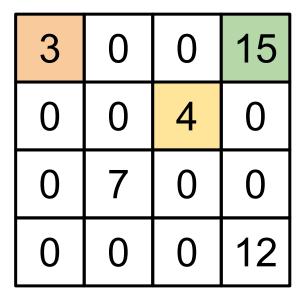
3	0	0	15
0	0	4	0
0	7	0	0
0	0	0	12

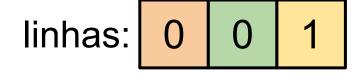
linhas: 0 0

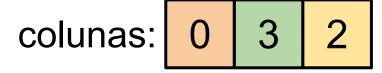
colunas: 0 3

valores: 3 15

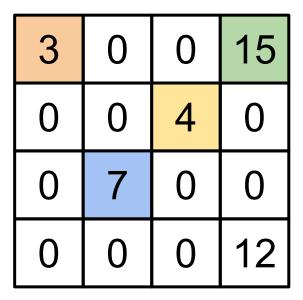
Agora iremos ler os dados da matriz, e considerar apenas os elementos não nulos.

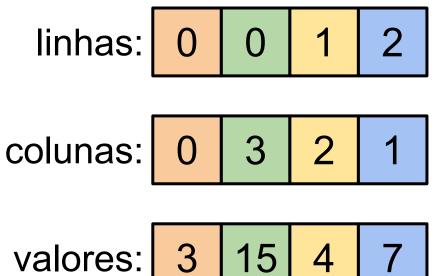




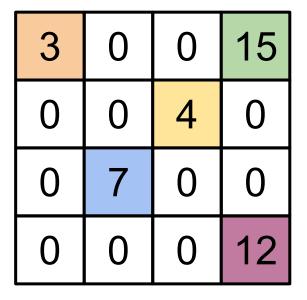


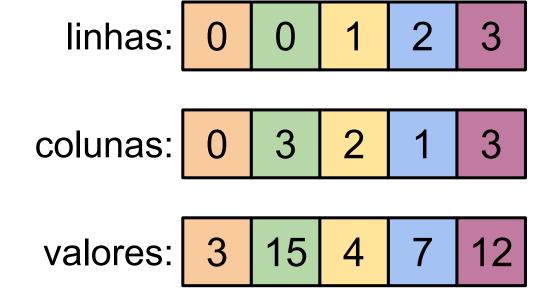
Agora iremos ler os dados da matriz, e considerar apenas os elementos não nulos.





Agora iremos ler os dados da matriz, e considerar apenas os elementos não nulos.





Note que da forma que a matriz foi lida no exemplo, os elementos estão ordenados por linha e por coluna, mas isso não é um requisito do formato.

- Note que da forma que a matriz foi lida no exemplo, os elementos estão ordenados por linha e por coluna, mas isso não é um requisito do formato.
- No entanto é ideal que os elementos sejam mantidos ordenados, para que as operações possam ser executadas de forma mais eficiente.

- Construir uma matriz.
- Obter e alterar elementos da matriz.
- Imprimir seus elementos.
- Soma de duas matrizes.
- Produto de uma matriz com um vetor denso.

```
double MatrizCoO::Get(int linha, int coluna) {
    if(linha < 0 || linha >= 1 || coluna < 0 || coluna >= c) {
        throw "Posicao inválida!!!\n";
    }
    for(int i = 0; i < tamanho; i++) {
        if(linha == M[i][0] && coluna == M[i][1])
            return M[i][2];
    }
    return 0;
}</pre>
```

Para obter um elemento na linha *i* coluna *j* vamos varrer nossa tabela comparando os valores de linha e coluna.

linhas:	0	0	1	2	2	3
colunas:	0	3	2	1	3	3
valores:	3	15	4	7	8	12

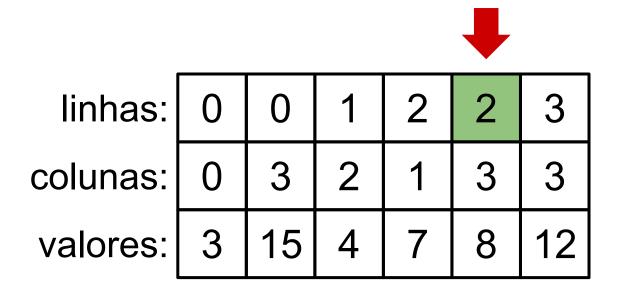
	1					
linhas:	0	0	1	2	2	3
colunas:	0	3	2	1	3	3
valores:	3	15	4	7	8	12

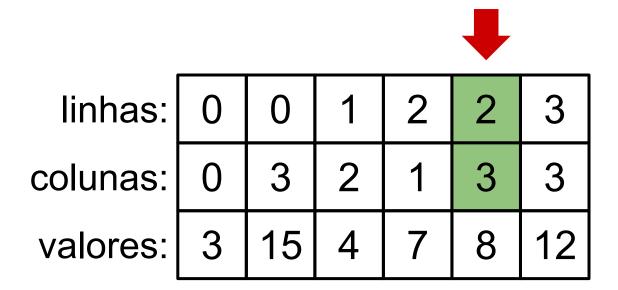
		1				
linhas:	0	0	1	2	2	3
colunas:	0	3	2	1	3	3
valores:	3	15	4	7	8	12

			1			
linhas:	0	0	1	2	2	3
colunas:	0	3	2	1	3	3
valores:	3	15	4	7	8	12

				<b>+</b>		
linhas:	0	0	1	2	2	3
colunas:	0	3	2	1	3	3
valores:	3	15	4	7	8	12

				1		
linhas:	0	0	1	2	2	3
colunas:	0	3	2	1	3	3
valores:	3	15	4	7	8	12





• Qual a complexidade para encontrar um elemento?

- Qual a complexidade para encontrar um elemento?
  - A quantidade de elementos que uma matriz n x n pode ter é n², no entanto esse limite seria muito "folgado".

- Qual a complexidade para encontrar um elemento?
  - A quantidade de elementos que uma matriz n x n pode ter é n², no entanto esse limite seria muito "folgado".
  - Como nós temos que por hipótese nossa matriz é esparsa, vamos chamar de k a quantidade de elementos não nulos presentes na matriz.

- Qual a complexidade para encontrar um elemento?
  - A quantidade de elementos que uma matriz n x n pode ter é n², no entanto esse limite seria muito "folgado".
  - Como nós temos que por hipótese nossa matriz é esparsa, vamos chamar de k a quantidade de elementos não nulos presentes na matriz.
  - Dessa forma a complexidade é O(k).

```
void MatrizCOO::Set(int linha, int coluna, double valor) {
    if(linha < 0 \mid | linha >= 1 \mid | coluna < 0 \mid | coluna >= c) {
        throw "Posicao inválida!!!\n";
    for (int i = 0; i < tamanho; i++) {
        if(linha == M[i][0] && coluna == M[i][1]){
            if(valor == 0) RemoveElemento(i);
            else M[i][2] = valor;
            return;
    if(valor == 0) return;
    InsereElemento(linha, coluna, valor);
```

Desejamos alterar o elemento na linha 0 coluna 1 para 20.

linhas:	0	0	1	2	2	3
colunas:	0	3	2	1	3	3
valores:	3	15	4	7	8	12

Desejamos alterar o elemento na linha 0 coluna 1 para 20.

Como o elemento nessa posição é 0, devemos incluir um novo elemento na representação

linhas:	0	0	1	2	2	3	0
colunas:	0	3	2	1	3	3	1
valores:	3	15	4	7	8	12	20

Observe que agora os elementos da nossa representação não estão mais ordenados.

linhas:	0	0	1	2	2	3	0
colunas:	0	3	2	1	3	3	1
valores:	3	15	4	7	8	12	20

Observe que agora os elementos da nossa representação não estão mais ordenados. Podemos utilizar o método da inserção para inserir ordenado em O(k).

linhas:	0	0	0	1	2	2	3
colunas:	0	1	3	2	1	3	3
valores:	3	20	15	4	7	8	12

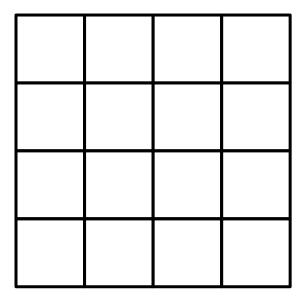
- Construir uma matriz.
- Obter e alterar elementos da matriz.
- Imprimir seus elementos.
- Soma de duas matrizes.
- Produto de uma matriz com um vetor denso.

Para as últimas três funções, iremos assumir que os elementos da nossa representação estão ordenados por linha e coluna.

- Construir uma matriz.
- Obter e alterar elementos da matriz.
- Imprimir seus elementos.
- Soma de duas matrizes.
- Produto de uma matriz com um vetor denso.

```
void MatrizCOO::Imprime() {
    int k = 0;
    for(int i = 0; i < 1; i++){
        for(int j = 0; j < c; j++) {
            if(M[k][0] == i \&\& M[k][1] == j){
                cout << M[k][2] << " ";
                k++;
            else cout << "0 ";
        cout << "\n";
```

Vamos executar o código de impressão neste exemplo.



linhas:

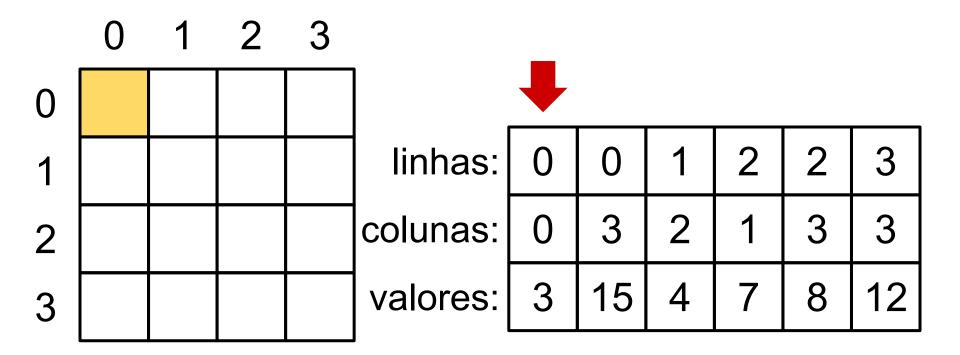
0	0	1	2	
---	---	---	---	--

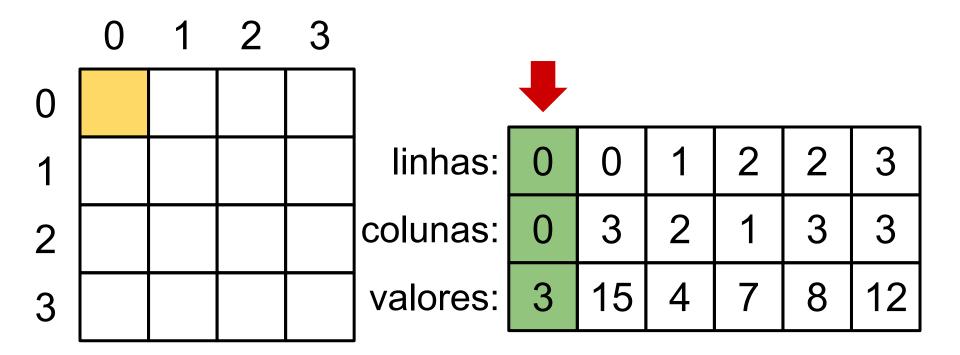
colunas:

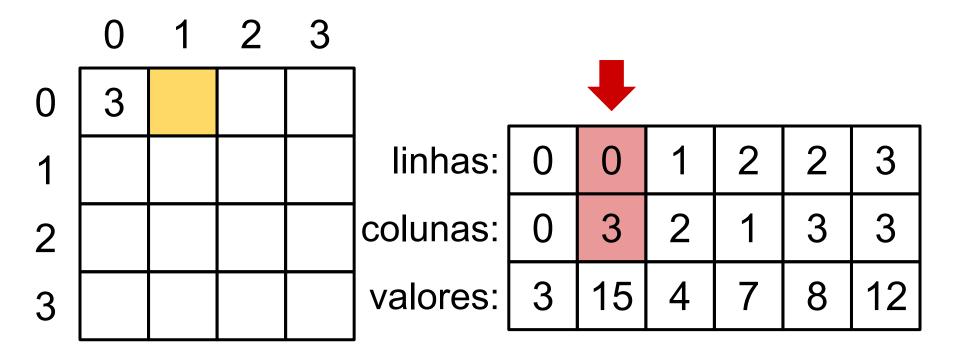
0 3 2 1 3

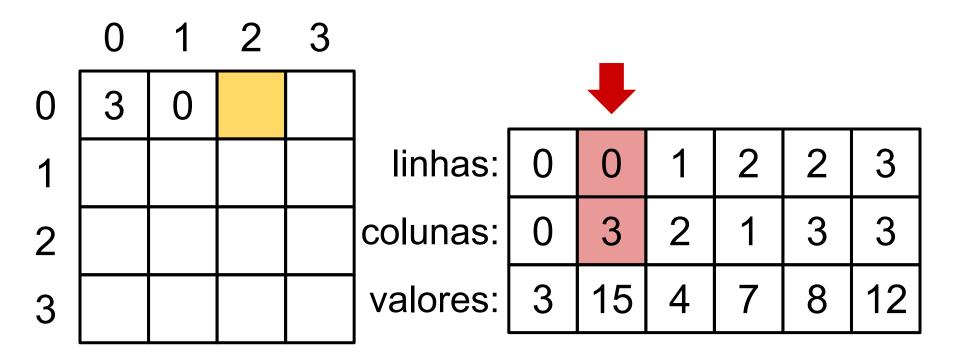
valores:

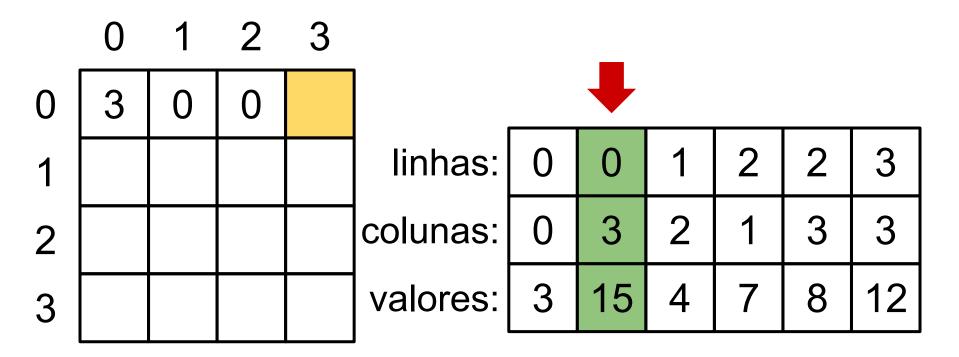
3   15   4   7	8 12
3   15   4   7	8   12

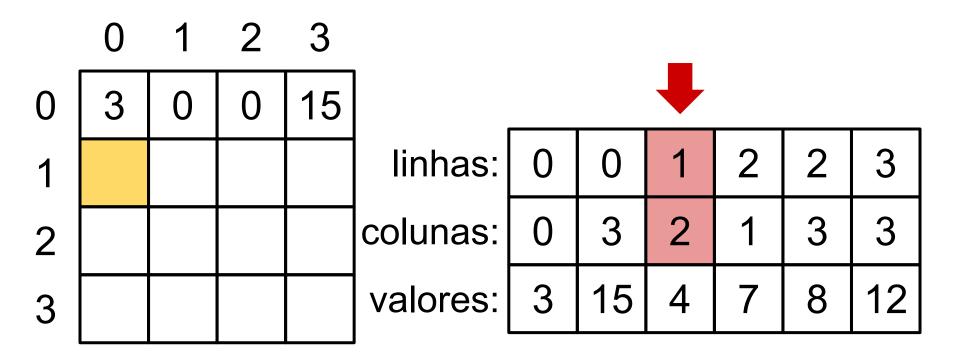


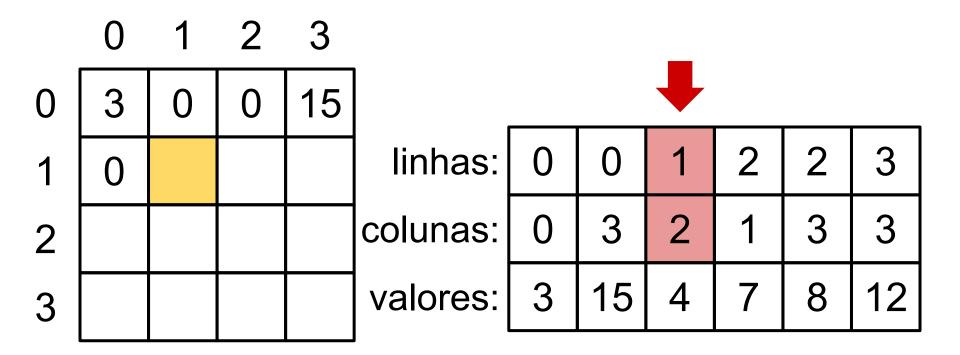


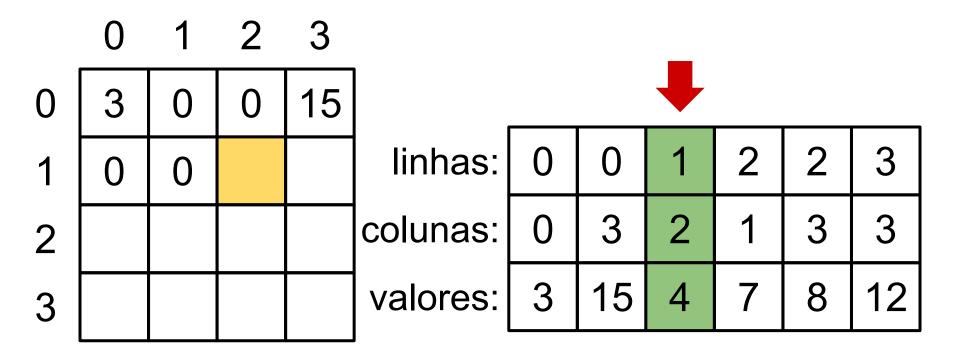


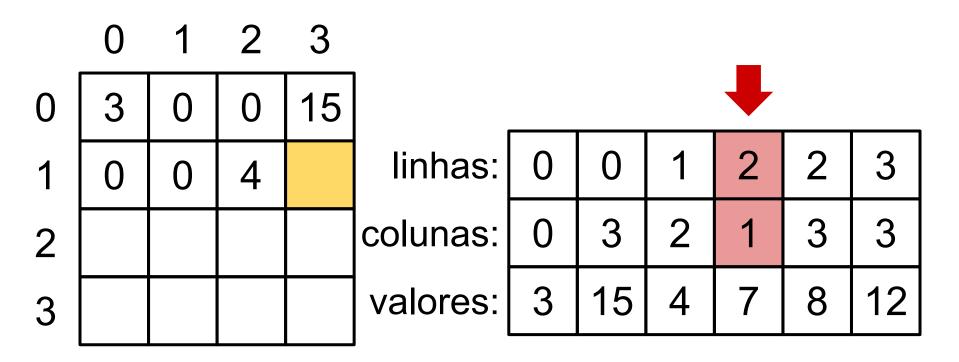


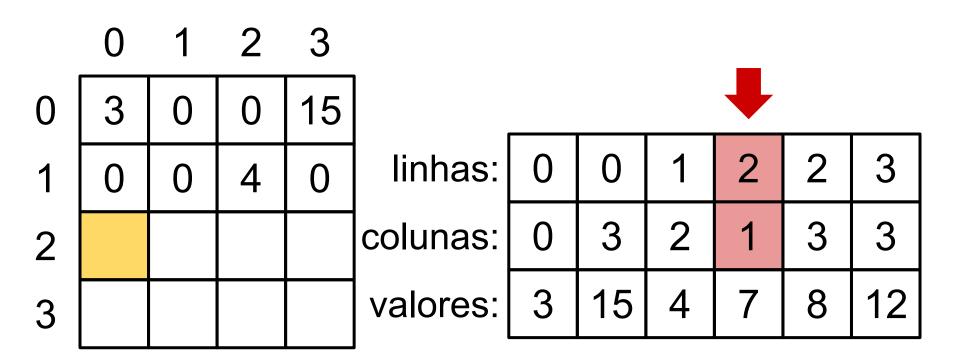


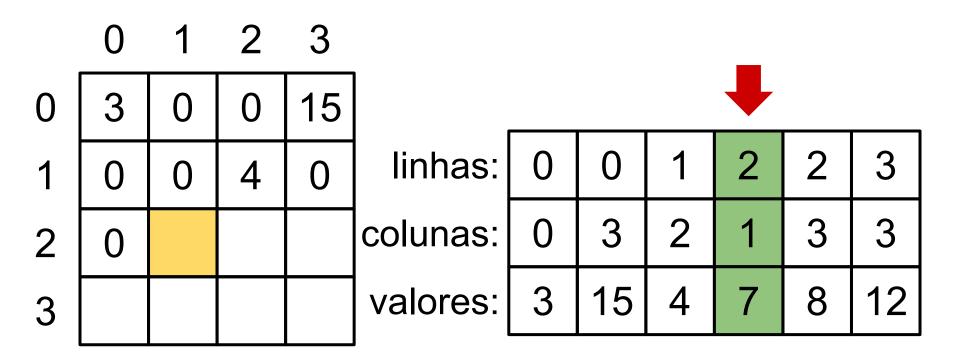


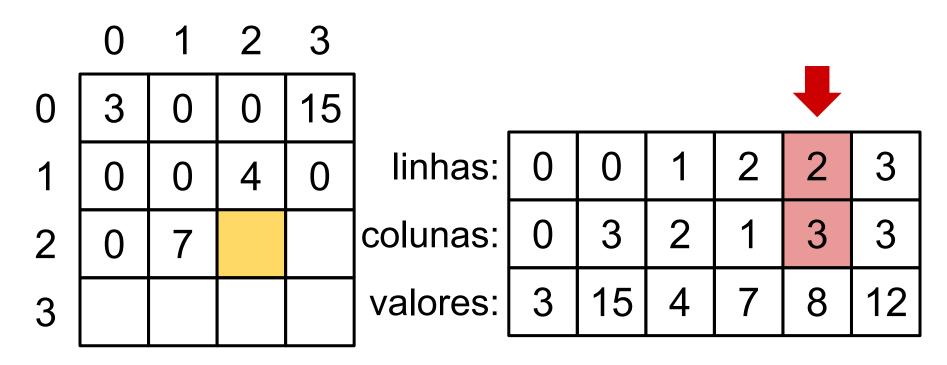


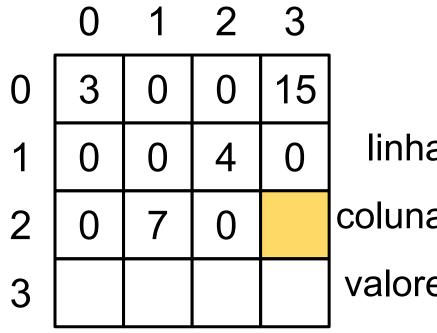




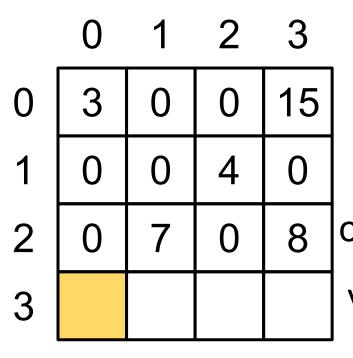


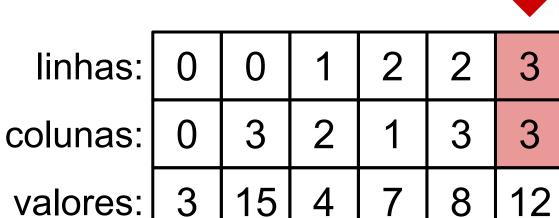


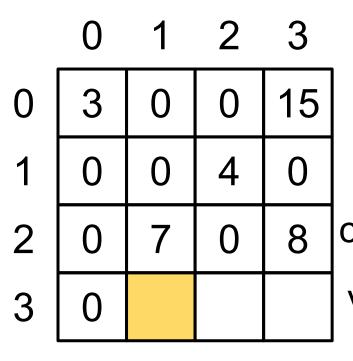


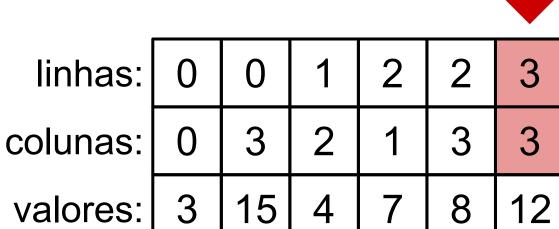


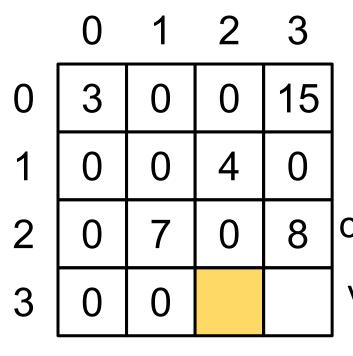
linhas:	0	0	1	2	2	3
colunas:	0	3	2	1	3	3
valores:	3	15	4	7	8	12

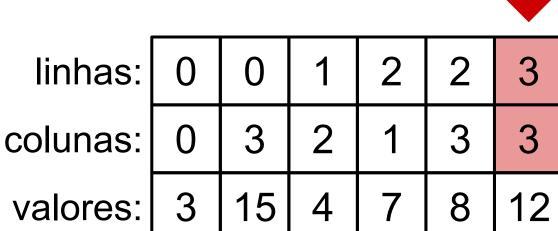


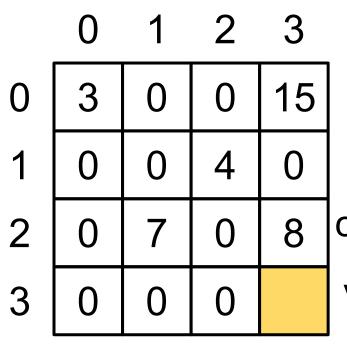












linhas:	0	0	1	2	2	3
colunas:	0	3	2	1	3	3
valores:	3	15	4	7	8	12

	0	1	2	3
0	3	0	0	15
1	0	0	4	0
2	0	7	0	8
3	0	0	0	12

linhas:	0	0	1	2	2	3
olunas:	0	3	2	1	3	3
alores:	3	15	4	7	8	12

• Qual a complexidade para imprimir os elementos?

- Qual a complexidade para imprimir os elementos?
  - Apesar de termos uma representação esparsa da nossa matriz, passamos por todos os indices.

- Qual a complexidade para imprimir os elementos?
  - Apesar de termos uma representação esparsa da nossa matriz, passamos por todos os indices.
  - Logo temos  $O(n^2)$ .

- Construir uma matriz.
- Obter e alterar elementos da matriz.
- Imprimir seus elementos.
- Soma de duas matrizes.
- Produto de uma matriz com um vetor denso.

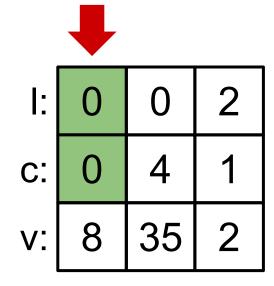
```
MatrizCOO MatrizCOO::Soma (MatrizCOO A, MatrizCOO B) {
    if (A.1 != B.1 || A.c != B.c) {
        throw "Dimensoes das matrizes incompativeis\n";
    MatrizCOO C = MatrizCOO(A.1, A.c, 0);
    int kA = 0, kB = 0;
    while(kA < A.tamanho && kB < B.tamanho) {</pre>
        //Parte 1
    while(kA < A.tamanho) {</pre>
        //Parte 2
    while(kB < B.tamanho) {</pre>
        //Parte 3
    return C;
```

```
//Parte 1
while(kA < A.tamanho && kB < B.tamanho) {</pre>
    int iA = A.M[kA][0], jA = A.M[kA][1];
    int iB = B.M[kB][0], jB = B.M[kB][1];
    if((iA == iB) && (jA == jB))
        C.InsereElemento(iA, jA, A.M[kA][2] + B.M[kB][2]);
        kA++;
        kB++;
    else if ((iA == iB \&\& jA < jB) \mid | (iA < iB)) {
        C.InsereElemento(iA, jA, A.M[kA][2]);
        kA++;
    else{
        C.InsereElemento(iB, jB, B.M[kB][2]);
        kB++;
```

```
//Parte 2
while(kA < A.tamanho) {</pre>
    C.InsereElemento (A.M[kA][0], A.M[kA][1], A.M[kA][2]);
    kA++;
//Parte 3
while(kB < B.tamanho) {</pre>
    C.InsereElemento(B.M[kB][0], B.M[kB][1], B.M[kB][2]);
    kA++;
```

Vamos efetuar a soma de duas representações.

	<b>1</b>					
l:	0	0	1	2	2	3
c:	0	3	2	1	3	3
v:	3	15	4	7	8	12



**!** 

C:

V:

Vamos efetuar a soma de duas representações.

		1				
l:	0	0	1	2	2	3
c:	0	3	2	1	3	3
v:	3	15	4	7	8	12

_		•	
l:	0	0	2
c:	0	4	1
v:	8	35	2
•			

I: 0c: 0v: 11

		1				
l:	0	0	1	2	2	3
c:	0	3	2	1	3	3
v:	3	15	4	7	8	12

l:	0	0
c:	0	3
v:	11	15

		1	
l:	0	0	2
C:	0	4	1
v:	8	35	2

			1			
l:	0	0	1	2	2	3
c:	0	3	2	1	3	3
v:	3	15	4	7	8	12

l:	0	0
C:	0	3
v:	11	15

		1	
l:	0	0	2
C:	0	4	1
v:	8	35	2

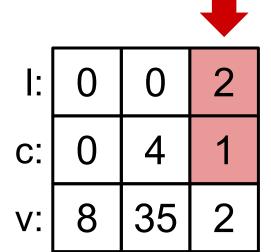
			1			
l:	0	0	1	2	2	3
c:	0	3	2	1	3	3
v:	3	15	4	7	8	12

l:	0	0	0
C:	0	3	4
V:	11	15	35

		1	
l:	0	0	2
C:	0	4	1
v:	8	35	2

			+			
l:	0	0	1	2	2	3
C:	0	3	2	1	3	3
v:	3	15	4	7	8	12

l:	0	0	0
C:	0	3	4
v:	11	15	35



			+			
1:	0	0	1	2	2	3
C:	0	3	2	1	3	3
v:	3	15	4	7	8	12

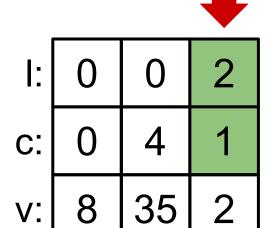
l:	0	0	0	1
C:	0	3	4	2
v:	11	15	35	4



l:	0	0	2
c:	0	4	1
			I

				<b>—</b>		
l:	0	0	1	2	2	3
C:	0	3	2	1	3	3
v:	3	15	4	7	8	12

l:	0	0	0	1
C:	0	3	4	2
v:	11	15	35	4



Vamos efetuar a soma de duas representações.









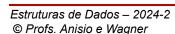
3 3 3

15 8

_	_	
<b> </b> •		9
1.		_

35 V:

l:	0	0	0	1	2
c:	0	3	4	2	1
v:	11	15	35	4	9









l:	0	0	1	2	2	3
<b>D</b> :	0	3	2	1	3	3
<b>/</b> :	3	15	4	7	8	12

l:	0	0	2
c:	0	4	1
v:	8	35	2

l:	0	0	0	1	2	2	3
C:	0	3	4	2	1	3	3
V:	11	15	35	4	9	8	12

1:	0	0	1	2	2	3
c:	0	3	2	1	3	3
v:	3	15	4	7	8	12

l:	0	0	2
c:	0	4	1
v:	8	35	2

l:	0	0	0	1	2	2	3
c:	0	3	4	2	1	3	3
v:	11	15	35	4	9	8	12

• Qual a complexidade para somar duas representações?

- Qual a complexidade para somar duas representações?
  - Assuma que as representações possuem respectivamente k<sub>1</sub> e k<sub>2</sub> elementos.

- Qual a complexidade para somar duas representações?
  - Assuma que as representações possuem respectivamente k<sub>1</sub> e k<sub>2</sub> elementos.
  - Nesse caso nossa complexidade é  $O(max(k_1,k_2))$ .

- Construir uma matriz.
- Obter e alterar elementos da matriz.
- Imprimir seus elementos.
- Soma de duas matrizes.
- Produto de uma matriz com um vetor denso.

```
double* MatrizCOO::Produto(MatrizCOO A, double* v, int n) {
    if(A.c != n) {
        throw "Dimensoes da entrada incompativeis\n";
    double* s = new double[n];
    for(int i = 0; i < n; i++) {
        s[i] = 0;
    for(int i = 0; i < A.tamanho; i++) {</pre>
        int linha = A.M[i][0];
        int coluna = A.M[i][1];
        s[linha] += A.M[i][2] * v[coluna];
    return s;
```

Vamos multiplicar uma representação por um vetor denso *u* e obter um novo vetor denso *w*.

1:	0	0	1	2	2	3
C:	0	3	2	1	3	3
v:	3	15	4	7	8	12

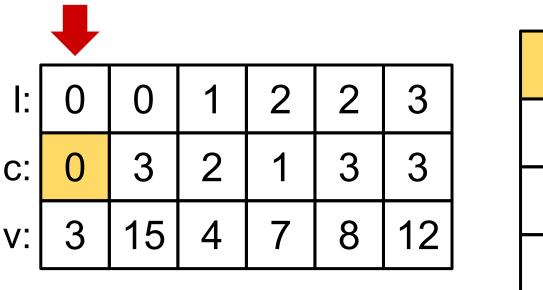
U	_	W
4		0
3		0
2		0
1		0

A linha indica em qual posição de w devemos somar nosso produto.

	<b>1</b>					
l:	0	0	1	2	2	3
C:	0	3	2	1	3	3
v:	3	15	4	7	8	12

U	W
4	0
3	0
2	0
1	0

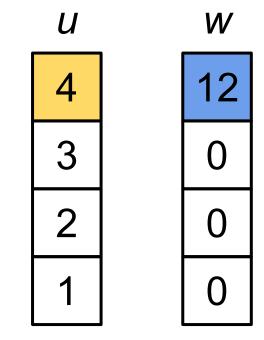
A coluna indica com qual elemento de *u* devemos multiplicar o elemento da nossa representação.



Ш	W
4	0
3	0
2	0
1	0

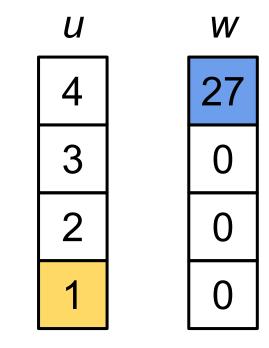
Então w[0] += 3\*4

	<b>1</b>					
l:	0	0	1	2	2	3
C:	0	3	2	1	3	3
v:	3	15	4	7	8	12



$$w[0] += 15*1$$

		<b>1</b>				
l:	0	0	1	2	2	3
c:	0	3	2	1	3	3
v:	3	15	4	7	8	12



$$w[1] += 4*2$$

			<b>1</b>				
l:	0	0	1	2	2	3	
c:	0	3	2	1	3	3	
v:	3	15	4	7	8	12	

И	_	W
4		27
3		8
2		0
1		0

$$w[2] += 7*3$$

								и	W	_
_ 1							]	4	27	
1:	0	0	1	2	2	3		3	Q	
C:	0	3	2	1	3	3		3	0	
		15	1	7	0	40		2	21	
V:	3	15	4	/	8	12		1	0	
								•		

$$w[2] += 8*2$$

								и	W	
_						T _	]	4	27	
1:	0	0	1	2	2	3		3	Ω	
C:	0	3	2	1	3	3		3	O	
\/'	3	15	1	7	8	12		2	37	
<b>V</b> :	3	13	4	/	0	12		1	0	
										i

$$w[3] += 12*1$$

							и	W	
_ 1							4	27	
1:	0	0	1	2	2	3	3	Ω	
c:	0	3	2	1	3	3	<u> </u>		
v:	3	15	1	7	8	12	2	37	
<b>v</b> .	<u> </u>	10		<i>'</i>		'   _	1	12	

E por fim, obtemos o vetor w.

1:	0	0	1	2	2	3
C:	0	3	2	1	3	3
v:	3	15	4	7	8	12

U	W
4	27
3	8
2	37
1	12

• Qual a complexidade para multiplicar uma representação por um vetor?

- Qual a complexidade para multiplicar uma representação por um vetor?
  - Note que passamos por cada elemento da representação apenas uma vez. Logo a operação não depende necessariamente do tamanho original da matriz.

- Qual a complexidade para multiplicar uma representação por um vetor?
  - Note que passamos por cada elemento da representação apenas uma vez. Logo a operação não depende necessariamente do tamanho original da matriz.
  - Portanto, podemos concluir que a complexidade é O(k).