



Estruturas de Dados

Matrizes Esparsas (cont)

Professores: Anisio Lacerda

Wagner Meira Jr.

 Na última aula começamos a estudar matrizes esparsas.

- Na última aula começamos a estudar matrizes esparsas.
- Nosso objetivo é encontrar uma representação que omita os elementos nulos.

- Na última aula começamos a estudar matrizes esparsas.
- Nosso objetivo é encontrar uma representação que omita os elementos nulos.
- Vamos pensar em uma possível otimização para o formato COO.

- Na última aula começamos a estudar matrizes esparsas.
- Nosso objetivo é encontrar uma representação que omita os elementos nulos.
- Vamos pensar em uma possível otimização para o formato COO.
- A ideia será comprimir o vetor que armazena as linhas dos elementos não nulos.

- Na última aula começamos a estudar matrizes esparsas.
- Nosso objetivo é encontrar uma representação que omita os elementos nulos.
- Vamos pensar em uma possível otimização para o formato COO.
- A ideia será comprimir o vetor que armazena as linhas dos elementos não nulos.
- Este novo formato é o Compressed Sparse Row (CSR).

- Na estrutura da representação por coordenadas tínhamos:
 - A linha onde o elemento está.
 - A coluna onde o elemento está.
 - O valor do elemento.

- Na estrutura da representação por coordenadas tínhamos:
 - A linha onde o elemento está.
 - A coluna onde o elemento está.
 - O valor do elemento.

Agora vamos armazenar apenas "ponteiros" para os índices onde cada linha começa no vetor de colunas.

Observe um exemplo da representação CSR.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

O vetor linhas indica os índices em que cada linha começa nos vetores colunas e valores.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

Por exemplo note que os elementos da linha 1 começam a partir da posição 2 em ambos os vetores.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas: 0 2 4 5 7

colunas: 0 3 1 2 1 0

valores: 3 15 2 4 7 6

Conseguimos também saber onde os elementos da linha terminam, simplesmente consultando onde os elementos da próxima linha começam.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas: 0 2 4 5 7 colunas: 0 3 1 2 1 0 3 valores: 3 15 2 4 7 6 12

E para simplificar casos de borda, inserimos uma entrada a mais no vetor de linhas, para sabermos onde a última linha acaba.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas: 0 2 4 5 7

colunas: 0 3 1 2 1 0 3

valores: 3 15 2 4 7 6 12

Note que este valor também computa quantos elementos não nulos existem na matriz original.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas: colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

Comparando com o modelo COO, os ganhos em termos de memória só se tornam palpáveis quando nossa matriz é muito grande.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

Exemplo COO vs CSR

COO

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:	0	0	1	1	2	3	3
colunas:	0	3	1	2	1	0	3
valores:	3	15	2	4	7	6	12

CSR

linhas: 0 2 4 5 7

colunas: 0 3 1 2 1 0 3

valores: 3 | 15 | 2 | 4 | 7 | 6 | 12

Estudaremos as mesmas operações que vimos no modelo COO.

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

Observação importante!

As implementações nesta seção são apenas ilustrativas. Não necessariamente são as mais otimizadas, o objetivo é apenas exemplificar o TAD.

```
class MatrizCSR{
    public:
        MatrizCSR(int linhas, int colunas, int leTeclado);
        ~MatrizCSR();
        double Get(int linha, int coluna);
        void Set(int linha, int coluna, double valor);
        void Imprime();
        static MatrizCSR Soma (MatrizCSR A, MatrizCSR B);
        static double * Produto (MatrizCSR A, double * v, int t);
    private:
        void InsereElemento(int linha, int coluna, int posicao, int v);
        void RemoveElemento(int linha, int posicao);
        void AumentaTamanhoVetores();
        int* linhas;
        int* colunas;
        double* valores;
        int tamanho, tamanhoMAX, 1, c;
};
```

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

Vamos construir a representação CSR a partir de uma matriz.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

O vetor de linhas será inicializado com -1, e a matriz será lida sequencialmente por linhas.



3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

-1	-1	-1	-1	-1
----	----	----	----	----

colunas:

Ao encontrar um elemento não nulo devemos inseri-lo na estrutura.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

-1	-1	-1	-1	-1
----	----	----	----	----

colunas:

Nosso elemento está na linha 0, e o "ponteiro" desta linha está com -1, o que significa que este é o primeiro elemento não nulo da linha 0 que foi lido.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:



colunas:

Devemos então gravar nesta posição o tamanho atual dos nossos vetores de colunas e valores, que é precisamente o índice que este elemento irá aparecer nestes vetores.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

0 -1 -1 -1 -

colunas:

valores:

3

Apenas os elementos não nulos nos interessam, então podemos ignorar todos os elementos nulos lidos.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

0 -1

-1 -

-1 | -1

valores:

colunas:

Note que ao ler mais elementos na mesma linha nosso "ponteiro" já não é mais -1, então apenas inserimos o novo elemento no fim dos vetores.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

0 -	1 -1	-1	-1
-----	------	----	----

colunas: valores:

3	15

É importante lembrar que ao trocar de linha devemos atualizar o "ponteiro" referente àquela linha ao ler o primeiro elemento não nulo.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas: valores:

3	15

É importante lembrar que ao trocar de linha devemos atualizar o "ponteiro" referente àquela linha ao ler o primeiro elemento não nulo.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

0 2 -1 -1 -1

colunas:

U)	
3	15	2

Terminando a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

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

colunas:

3 15 2 4

Terminando a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

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

colunas:

3 | 15 | 2 | 4

3

Terminando a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

valores:

colunas:

3

Terminando a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	-1	-1
0	3	1	2	1
3	15	2	4	7

Terminando a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

	0	2	4	5	-1	
	0	3	1	2	1	0
,	3	15	2	4	7	6

Terminando a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	-1		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

Após terminar de ler a matriz, o último elemento deve receber a quantidade de elementos não nulos. Em outras palavras ele recebe o tamanho dos vetores de colunas e valores.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	-1		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

E assim encerra o algoritmo e obtemos nossa representação CSR.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

• Qual a complexidade de criar uma representação lendo a matriz da entrada padrão?

- Qual a complexidade de criar uma representação lendo a matriz da entrada padrão?
 - A quantidade de elementos que uma matriz n x n pode ter é n², e como temos que ler todos os elementos da matriz obtemos um limite "folgado" de O(n²).

```
MatrizCSR::MatrizCSR(int m, int n, int leTeclado) {
    1 = m; c = n; tamanho = 0;
    linhas = new int[l+1];
    if (!leTeclado) {
        for (int i = 0; i <= 1; i++)</pre>
            linhas[i] = -1;
        tamanhoMAX = 10;
        colunas = new int[tamanhoMAX];
        valores = new double[tamanhoMAX];
    else{
        //Le a matriz do teclado
```

```
//Le a matriz do teclado
tamanhoMAX = floor(0.3*1*c);
tamanho = 0;
colunas = new int[tamanhoMAX];
valores = new double[tamanhoMAX];
for (int i = 0; i \le 1; i++) linhas[i] = -1;
for (int i = 0; i < 1; i++) {
     linhas[i] = tamanho;
     for (int j = 0; j < c; j++) {
         double x; cin >> x;
        if(x != 0) {
             if (tamanho == tamanhoMAX) AumentaTamanhoVetores();
             colunas[tamanho] = j;
             valores[tamanho] = x;
             tamanho++;
linhas[l] = tamanho;
```

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

Dadas as coordenadas *i* e *j*, retornar o elemento na linha *i* coluna *j*.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

Dados coordenadas *i* e *j*, retornar o elemento na linha *i* coluna *j*.

Suponha que recebemos de entrada linha 1 e coluna 2.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

Primeiro consultamos no vetor das linhas em qual posição os elementos da linha 1 começam e terminam.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

Agora temos um subvetor de candidatos. Basta executarmos uma busca nesse subvetor.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

• Qual a complexidade para encontrar um elemento?

- Qual a complexidade para encontrar um elemento?
 - Conseguimos acessar a linha em tempo constante.

- Qual a complexidade para encontrar um elemento?
 - Conseguimos acessar a linha em tempo constante.
 - Nosso pior caso é quando toda a linha possui elementos não nulos, ou seja O(n).

- Qual a complexidade para encontrar um elemento?
 - Conseguimos acessar a linha em tempo constante.
 - Nosso pior caso é quando toda a linha possui elementos não nulos, ou seja O(n).
 - Considerando que a matriz é esparsa e que tem k elementos não nulos, temos O(k).

É possível fazer melhor?

- É possível fazer melhor?
 - Neste caso possuímos um intervalo de candidatos a resposta que estão ordenados.

- É possível fazer melhor?
 - Neste caso possuímos um intervalo de candidatos a resposta que estão ordenados.
 - Diferente do formato COO também conseguimos obter rapidamente (em tempo O(1)) os limites deste intervalo.

- É possível fazer melhor?
 - Neste caso possuímos um intervalo de candidatos a resposta que estão ordenados.
 - Diferente do formato COO também conseguimos obter rapidamente (em tempo O(1)) os limites deste intervalo.
 - Então poderíamos ao invés de fazer uma busca sequencial, utilizar a busca binária e obter uma complexidade de tempo O(log(k)).

- É possível fazer melhor?
 - Mas é importante ressaltar que como estamos tratando de matrizes esparsas, não esperamos que o tamanho do intervalo seja muito grande.

- É possível fazer melhor?
 - Mas é importante ressaltar que como estamos tratando de matrizes esparsas, não esperamos que o tamanho do intervalo seja muito grande.
 - O que significa que o ganho computacional de tempo pode na prática não ser tão grande assim.

```
double MatrizCSR::Get(int i, int j) {
    if(i < 0 || i >= 1 || j < 0 || j >= c) {
        throw "Posicao inválida!!!\n";
    }

    for(int k = linhas[i]; k < linhas[i+1]; k++) {
        if(colunas[k] == j)
            return valores[k];
    }
    return 0;
}</pre>
```

Dados coordenadas *i* e *j* e um valor *v*, alterar o valor do elemento na linha *i* coluna *j* para *v*.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

Dados coordenadas *i* e *j* e um valor *v*, alterar o valor do elemento na linha *i* coluna *j* para *v*. Suponha que nossa entrada seja linha 2, coluna 3 e o valor seja 8.

3	0	0	15
0	2	4	0
0	7	0	8
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

O primeiro passo é verificar se a linha 2 coluna 3 está na estrutura, de forma semelhante a função get.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

Como a posição não está presente, devemos adicioná-la.

linhas:	0	2	4	5	7		
colunas:	0	3	1	2	1	0	3
valores:	3	15	2	4	7	6	12

Uma vez que haverá um novo elemento na linha 2, a posição do primeiro elemento das próximas linhas é incrementada em 1.

linhas:	0	2	4	6	8		
colunas:	0	3	1	2	1	0	3
valores:	3	15	2	4	7	6	12

Agora basta inserir de forma a manter a ordenação o novo elemento.

linhas:	0	2	4	6	8			
colunas:	0	3	1	2	1	3	0	3
valores:	3	15	2	4	7	8	6	12

• Qual a complexidade para alterar um elemento?

- Qual a complexidade para alterar um elemento?
 - O procedimento é bem semelhante ao de encontrar um elemento.

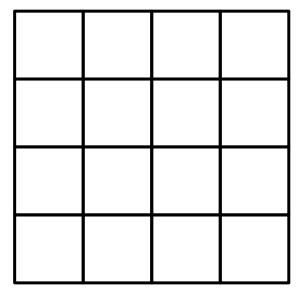
- Qual a complexidade para alterar um elemento?
 - O procedimento é bem semelhante ao de encontrar um elemento.
 - O pior dos casos é quando a posição em questão não está na estrutura e o valor é não nulo.

- Qual a complexidade para alterar um elemento?
 - O procedimento é bem semelhante ao de encontrar um elemento.
 - O pior dos casos é quando a posição em questão não está na estrutura e o valor é não nulo.
 - Utilizando um método de ordenação eficaz para a situação (como o método da inserção) conseguimos inserir ordenado em tempo O(k).

```
void MatrizCSR::Set(int i, int j, double valor) {
    if(i < 0 | | i >= 1 | | j < 0 | | j >= c)
        throw "Posicao inválida!!!\n";
    for (int k = linhas[i]; k < linhas[i+1]; k++) {
        if (colunas[k] == j) {
            if (valor == 0) RemoveElemento(i, k);
            else
                      valores[k] = valor;
            return;
        else if (colunas[k] > j) {
            InsereElemento(i,j,k,valor);
            return;
    if(valor == 0) return;
    InsereElemento(i, j, tamanho, valor);
```

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

Dada representação CSR de uma matriz *M*, imprimir *M*.



linhas:

0 2 4 5 7

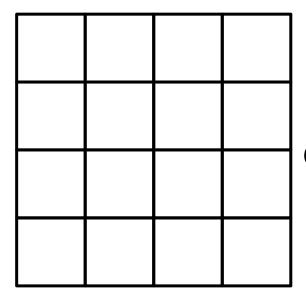
colunas:

0 | 3 | 1 | 2 | 1 | 0 | 3

valores:

2 4 7 6 12

O procedimento é bem semelhante ao do formato COO. Vamos simplesmente varrer a matriz e verificar se o próximo valor a ser impresso na representação possui as coordenadas desejadas.



linhas:

colunas: 0

4

0

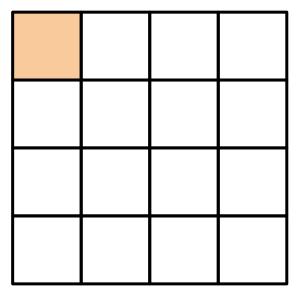
valores:

3 | 15 | 2

7

6 12

Temos um contador indicando qual a próxima posição da estrutura que deve ser impressa.



linhas:

0 2 4 5 7

colunas:

) | 3 | 1 | 2 | 1

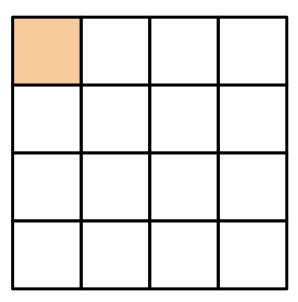
valores:

3 15 2 4 7 6 12



0

Temos um contador indicando qual a próxima posição da estrutura que deve ser impressa. Caso as coordenadas coincidam, basta imprimir o valor e incrementar o contador.



linhas:

0 2 4 5 7

colunas:

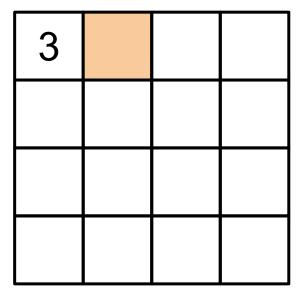
3 | 1 | 2 | 1 | 0 | 3

valores:

3 | 15 | 2 | 4 | 7 | 6 | 12



Caso as coordenadas não coincidam é porque a entrada da matriz naquela posição é 0.

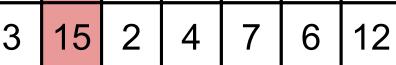


linhas:

0 2 4 5 7

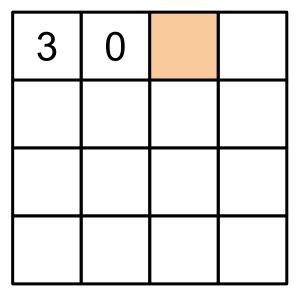
colunas:

0 3 1 2 1 0





Segue a execução do algoritmo no exemplo:



linhas:

0 2 4 5 7

colunas:

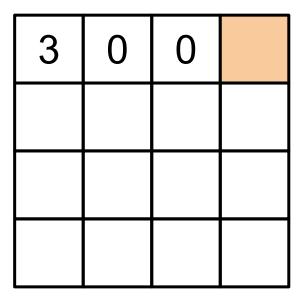
0 3 1 2 1 0 3

valores:

3 15 2 4 7 6 12



Segue a execução do algoritmo no exemplo:



linhas:

0 2 4 5 7

colunas:

 0
 3
 1
 2
 1
 0
 3

 3
 15
 2
 4
 7
 6
 1



Segue a execução do algoritmo no exemplo:

3	0	0	15

linhas:

0 2 4 5 7

colunas:

) 3 1 2

valores

3 15 2 4 7 6 12



Segue a execução do algoritmo no exemplo:

3	0	0	15
0			

linhas:

0 2 4 5 7

colunas:

) 3 1 2 1 0

valores

3 15 2 4 7 6 12



Segue a execução do algoritmo no exemplo:

3	0	0	15
0	2		

linhas:

colunas:

4

6





Sendo assim até o fim da linha podemos apenas imprimir valores 0!

3	0	0	15
0	2	4	

linhas:

olunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12



Este formato requer alguns cuidados especiais. Note que nosso contador está agora em uma posição da linha 2!

linhas:	0	2	4	5	7		
colunas:	0	3	1	2	1	0	3
valores:	3	15	2	4	7	6	12

Conseguimos identificar se este é o caso verificando a posição do primeiro elemento da próxima linha!

linhas:	0	2	4	5	7		
colunas:	0	3	1	2	1	0	3
valores:	3	15	2	4	7	6	12



Segue a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12



Segue a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0			

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12



O marcador já está em elementos da próxima linha, então os elementos restantes desta linha são todos 0.

3	0	0	15
0	2	4	0
0	7		

linhas:

colunas:

0		2	4	5	7		
0		3	1	2	1	0	3
3	•	15	2	4	7	6	12



Segue a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0

linhas:

colunas:

l	0	2	4	5	7		
	0	3	1	2	1	0	3
	3	15	2	4	7	6	12



Segue a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0
6			

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12



Segue a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0
6	0		

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12



Segue a execução do algoritmo no exemplo:

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12



Após iterar por todas as posições possíveis, o algoritmo encerra.

3	0	0	15
0	2	4	0
0	7	0	0
6	0	0	12

linhas:

colunas:

0	2	4	5	7		
0	3	1	2	1	0	3
3	15	2	4	7	6	12

• Qual a complexidade para imprimir os elementos da matriz?

- Qual a complexidade para imprimir os elementos da matriz?
 - Devemos passar por todos os elementos da matriz, até mesmo os nulos. Portanto a complexidade é O(n²).

Matrizes esparsas (CSR)

```
void MatrizCSR::Imprime() {
    int k = 0;
    for(int i = 0; i < 1; i++){
        for (int j = 0; j < c; j++) {
            if(k < linhas[i+1] && colunas[k] == j){
                cout << valores[k] << " ";</pre>
                k++;
            else cout << "0 ";
        cout << "\n";
```

Matrizes esparsas (CSR)

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

Dado duas representações CSR de matrizes A e B, retornar a representação CSR da matriz A+B.

1:	0	1	4	5	6	
C:	1	0	1	2	1	3
v:	1	1	1	1	1	1

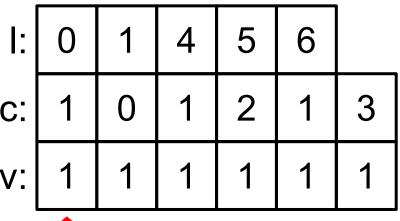
1:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
V:	3	15	2	4	7	6	12

A execução é semelhante a da representação COO. Vamos varrer as representações e combinar os elementos caso as coordenadas sejam iguais.

1:	0	1	4	5	6	
C:	1	0	1	2	1	3
v:	1	1	1	1	1	1

1:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
V:	3	15	2	4	7	6	12

Vamos executar o algoritmo no exemplo:



l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12





C.

V:

Vamos executar o algoritmo no exemplo:

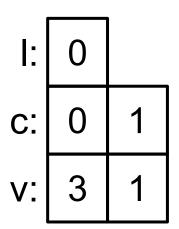
 :	0	1	4	5	6	
C:	1	0	1	2	1	3
v:	1	1	1	1	1	1
,						

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

I: 0 c: 0 v: 3

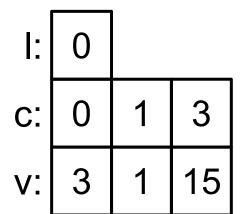
l:	0	1	4	5	6	
C:	1	0	1	2	1	3
v:	1	1	1	1	1	1

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12



1:	0	1	4	5	6	
c:	1	0	1	2	1	3
v:[1	1	1	1	1	1

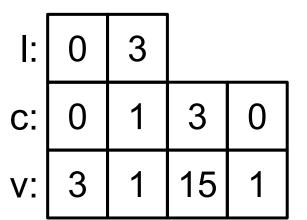
1:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12



1:	0	1	4	5	6	
C:	1	0	1	2	1	3
v:	1	1	1	1	1	1

l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

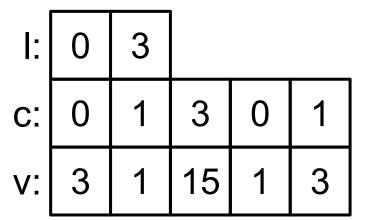




l:	0	1	4	5	6	
C:	1	0	1	2	1	3
v:	1	1	1	1	1	1

1:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

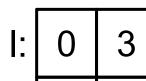




Vamos executar o algoritmo no exemplo:

c:	1	0	1	2	1	3
v:	1	1	1	1	1	1

1:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12



c: 0 1 3 0 1 2 v: 3 1 15 1 3 5

l :	0	1	4	5	6	
C:	1	0	1	2	1	3
v:	1	1	1	1	1	1

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12





l:	0	3	6				
c:	0	1	3	0	1	2	1
v:	3	1	15	1	3	5	8

1:	0	1	4	5	6	
C:	1	0	1	2	1	3
v:	1	1	1	1	1	1

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12





l :	0	3	6	7				
c:	0	1	3	0	1	2	1	0
v:	3	1	15	1	3	5	8	6

l :	0	1	4	5	6	
C:	1	0	1	2	1	3
V:	1	1	1	1	1	1

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

l:	0	3	6	7					
c:	0	1	3	0	1	2	1	0	3
v:	3	1	15	1	3	5	8	6	13

1:	0	1	4	5	6	
C:	1	0	1	2	1	3
V:	1	1	1	1	1	1

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v :	3	15	2	4	7	6	12

l:	0	3	6	7	9				
C:	0	1	3	0	1	2	1	0	3
v:	3	1	15	1	3	5	8	6	13

1:	0	1	4	5	6	
C:	1	0	1	2	1	3
V:	1	1	1	1	1	1

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

l:	0	3	6	7	9				
C:	0	1	3	0	1	2	1	0	3
v:	3	1	15	1	3	5	8	6	13

• Qual a complexidade para somar duas representações?

- Qual a complexidade para somar duas representações?
 - Assuma que as representações possuem respectivamente k₁ e k₂ elementos.

- Qual a complexidade para somar duas representações?
 - Assuma que as representações possuem respectivamente k₁ e k₂ elementos.
 - Nesse caso nossa complexidade é $O(max(k_1,k_2))$.

```
MatrizCSR MatrizCSR::Soma (MatrizCSR A, MatrizCSR B) {
    if (A.l != B.l || A.c != B.c) {
        throw "Dimensoes das matrizes incompativeis\n";
    MatrizCSR C = MatrizCSR(A.1, A.c, 0);
    for (int i = 0; i < A.1; i++) {
        int cA = A.linhas[i]; int lA = A.linhas[i+1];
        int cB = B.linhas[i]; int lB = B.linhas[i+1];
        while (cA < lA \&\& cB < lB)
            // Parte 1
        while (cA < lA)
            // Parte 2
        while (cB < lB)
            // Parte 2
    return C;
```

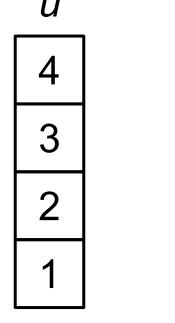
```
// Parte 1
while (cA < lA \&\& cB < lB) {
   if (A.colunas[cA] == B.colunas[cB]) {
      C.InsereElemento(i, A.colunas[cA], C.tamanho, A.valores[cA] +
                        B.valores[cB]);
      cA++; cB++;
   } else if(A.colunas[cA] < B.colunas[cB]){</pre>
      C.InsereElemento(i, A.colunas[cA], C.tamanho, A.valores[cA]);
      cA++;
   } else {
      C.InsereElemento(i, B.colunas[cB], C.tamanho, B.valores[cB]);
      cB++;
```

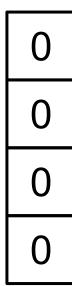
```
// Parte 2
while (cA < lA) {
    C.InsereElemento(i, A.colunas[cA], C.tamanho, A.valores[cA]);
    cA++;
}
while (cB < lB) {
    C.InsereElemento(i, B.colunas[cB], C.tamanho, B.valores[cB]);
    cB++;
}</pre>
```

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

Dado uma representação CSR de uma matriz *M* e um vetor *u*, calcule *Mu*.

l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

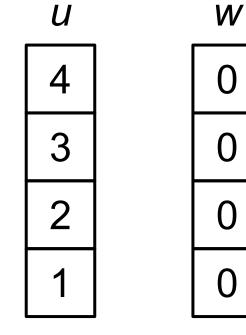




W

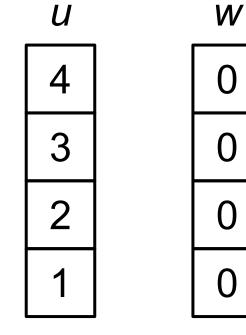
Este algoritmo também segue uma linha similar ao do COO.

l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12



Este algoritmo também segue uma linha similar ao do COO.

l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12



Dado um elemento na posição *i j* da matriz *M*, a linha indica a posição em *w* que o resultado do produto deste elemento com o vetor *u* deve ser gravado.

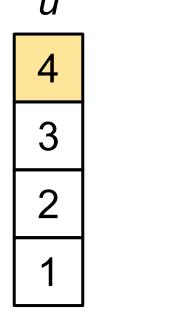
l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

<u>u</u>	
4	
3	
2	
1	

W

Enquanto a coluna indica por qual elemento de *u* ele deve ser multiplicado.

l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12



W

Agora basta iterar pela estrutura efetuando as multiplicações e somando-as nas posições apropriadas de w.

l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

u	
4	
3	
2	
1	

$$w[0] += v[0] * u[0]$$

l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

u	N
4	0
3	0
2	0
1	0

$$w[0] += v[1] * u[3]$$

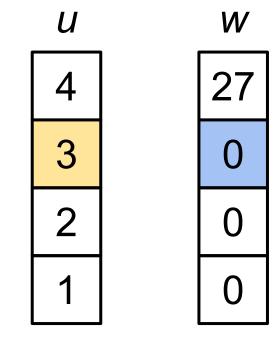
l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12

<u>u</u>	_
4	
3	
2	
1	

W

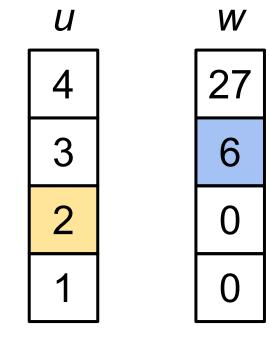
$$w[1] += v[2] * u[1]$$

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12



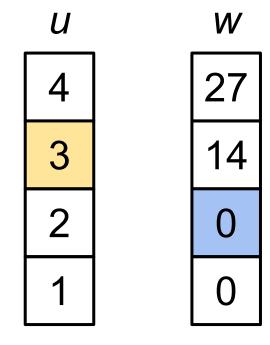
$$w[1] += v[3] * u[2]$$

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12



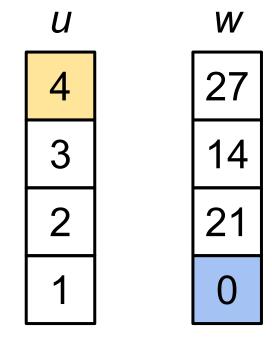
$$w[2] += v[4] * u[1]$$

l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
V:	3	15	2	4	7	6	12



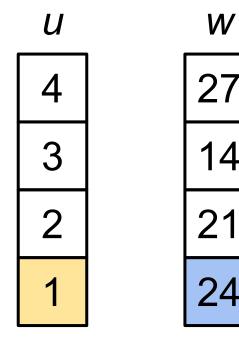
$$w[3] += v[5] * u[0]$$

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	12



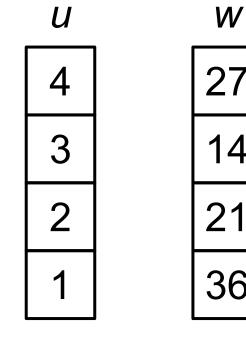
$$w[3] += v[6] * u[3]$$

l:	0	2	4	5	7		
c:	0	3	1	2	1	0	3
V:	3	15	2	4	7	6	12



Após iterar pela estrutura o vetor resultante *w* é precisamente *Mu*.

l:	0	2	4	5	7		
C:	0	3	1	2	1	0	3
v:	3	15	2	4	7	6	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).

```
double* MatrizCSR::Produto(MatrizCSR 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++) {</pre>
        s[i] = 0;
    for (int i = 0; i < A.1; i++) {
        for (int j = A.linhas[i]; j < A.linhas[i+1]; j++) {
            int coluna = A.colunas[j];
            s[i] += A.valores[j] * v[coluna];
    return s;
```

Outros modelos

- Dentre os modelos de esparsificação mais utilizados ainda existem mais alguns dignos de nota.
 - Compressed Sparse Column(CSC)
 - List of Lists (LiL)
 - Dictionary of Keys (DOK)

 Análogo ao CSR, no entanto as colunas são comprimidas ao invés das linhas.

- Análogo ao CSR, no entanto as colunas são comprimidas ao invés das linhas.
 - Possui vantagens e desvantagens análogas ao CSR.

- Análogo ao CSR, no entanto as colunas são comprimidas ao invés das linhas.
 - Possui vantagens e desvantagens análogas ao CSR.
 - Ao contrário de navegar em uma matriz convencional por colunas, esta estrutura não perde em termos de localidade de referência, uma vez que sua construção é feita por vetores separados.

- Análogo ao CSR, no entanto as colunas são comprimidas ao invés das linhas.
 - Possui vantagens e desvantagens análogas ao CSR.
 - Ao contrário de navegar em uma matriz convencional por colunas, esta estrutura não perde em termos de localidade de referência, uma vez que sua construção é feita por vetores separados.
 - Curiosidade: é o formato utilizado pelo MATLAB em sua função sparse.

Essa estrutura consiste em:

- Essa estrutura consiste em:
 - Uma lista por linha.

- Essa estrutura consiste em:
 - Uma lista por linha.
 - Cada uma das listas possui as entradas não nulas e suas respectivas colunas.

- Essa estrutura consiste em:
 - Uma lista por linha.
 - Cada uma das listas possui as entradas não nulas e suas respectivas colunas.
 - Assim como COO, é recomendado que estes valores estejam ordenados pela coluna.

- Essa estrutura consiste em:
 - Uma lista por linha.
 - Cada uma das listas possui as entradas não nulas e suas respectivas colunas.
 - Assim como COO, é recomendado que estes valores estejam ordenados pela coluna.
 - Similar a uma lista de adjacência em grafos.

- Essa estrutura consiste em:
 - Uma lista por linha.
 - Cada uma das listas possui as entradas não nulas e suas respectivas colunas.
 - Assim como COO, é recomendado que estes valores estejam ordenados pela coluna.
 - Similar a uma lista de adjacência em grafos.

Proposta de exercíco:

Elabore uma estrutura de esparsificação utilizando a ideia de LiL para linhas e colunas simultaneamente!

- Essa estrutura consiste em:
 - Um dicionário que mapeia duplas linha e coluna para seus respectivos valores.

- Essa estrutura consiste em:
 - Um dicionário que mapeia duplas linha e coluna para seus respectivos valores.
 - Eficiente para construir a matriz em uma ordem arbitrária.

- Essa estrutura consiste em:
 - Um dicionário que mapeia duplas linha e coluna para seus respectivos valores.
 - Eficiente para construir a matriz em uma ordem arbitrária.
 - Ineficiente para percorrer os elementos por linha/coluna.

- Essa estrutura consiste em:
 - Um dicionário que mapeia duplas linha e coluna para seus respectivos valores.
 - Eficiente para construir a matriz em uma ordem arbitrária.
 - Ineficiente para percorrer os elementos por linha/coluna.
 - Assim como LiL são formatos bons para construir a matriz, e depois transformá-lo em um formato CSR ou CSC para fazer operações que requerem iterações lexicográficas.