Aula 2a

Breve reflexão sobre AED 1

Profa. Ariane Machado Lima

Lista linear sequencial

Lista linear sequencial

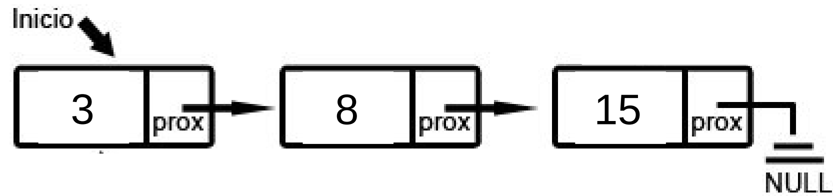
	0	1	2	3	4	5	6
V	-8	4	21	23	54	67	90

Lista linear ligada

EACH USP
Profa. Ariane Machado Lima

Lista linear ligada

Lista Linear Encadeada (alocação **dinâmica**)



Lista Linear Encadeada (alocação **estática**)

Início = 2

dispo = 1

	info	prox
0	8	6
1		3
2	3	0
3		4
4		5
5		8
6	15	-1
7		-1
8		7

Árvores

Profa. Ariane Machado Lima

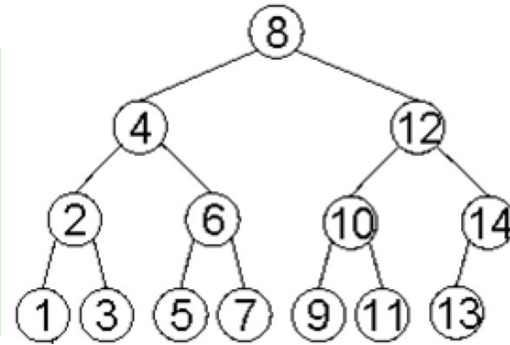
Árvores

Uma árvore é AVL se ela é ABB e para cada nó

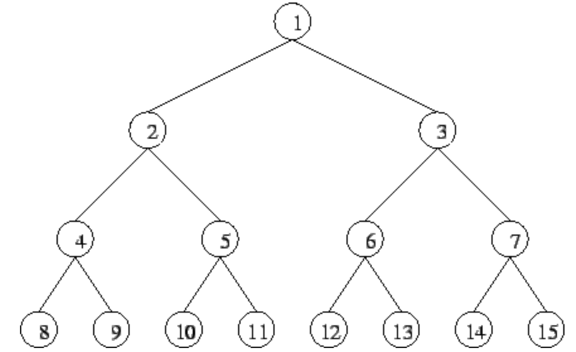
$$| h_D - h_E | \leq 1$$

h_D: altura da subárvore direita

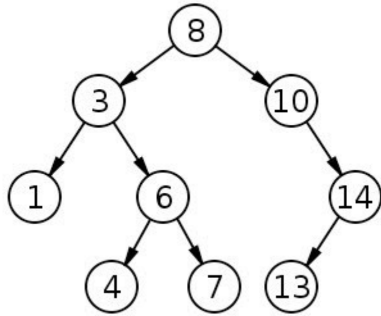
h_E: altura da subárvore esquerda



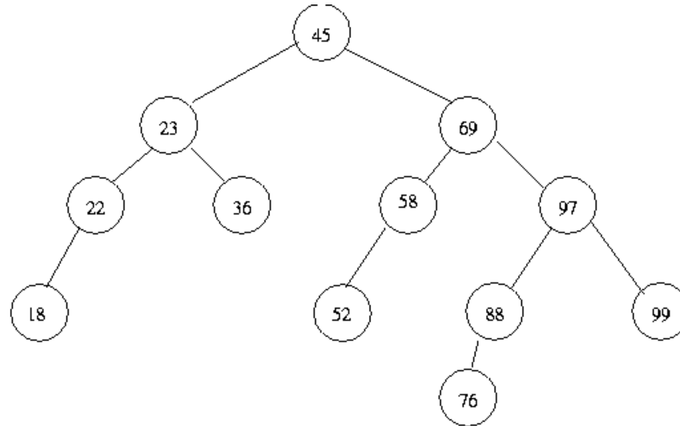
AVL (completa)



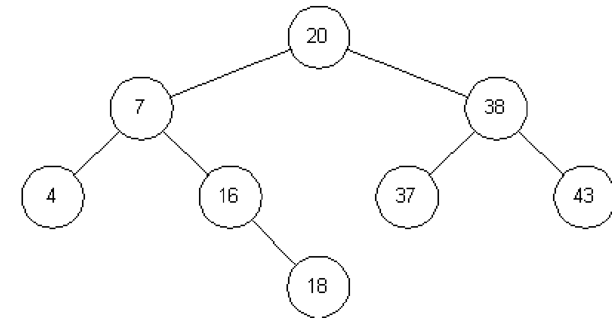
É completa, cheia, porém não é uma ABB, portanto não é AVL



Somente ABB



AVL



AVL completa

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)			
Facilidade de implementacao			
Busca			
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção			
min/max			

EACH USP

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)		
Facilidade de implementacao			
Busca			
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção			
min/max			

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	
Facilidade de implementacao			
Busca			
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção			
min/max			

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca			
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção			
min/max			

EACH USP

Profa. Ariane Machado Lima

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca	O(lg n)		
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção			
min/max			

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	**Ligada**	**Árvores Balanceadas**
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca	O(lg n)	O(n)	
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção			
min/max			

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção			
min/max			

EACH USP

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção	O(n)		
min/max			

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	**Ligada**	**Árvores Balanceadas**
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	
min/max			

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max			

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)		

EACH USP

Profa. Ariane Machado Lima

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	

EACH USP

Profa. Ariane Machado Lima

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao			
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço			
Gerenciamento de memória			
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço	Sim quando você sabe o tamanho exato Não, quando não sabe o tamanho pois mesmo com realloc há um momento em que precisa dos dois vetores (origem e destino), o que pode ser ruim em sistemas críticos de memória (Ex: micro-ondas)		
Gerenciamento de memória			
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço	Sim quando você sabe o tamanho exato	Sim, quando você tem variabilidade de tamanho	
	Não, quando não sabe o tamanho pois mesmo com realloc há um momento em que precisa dos dois vetores (origem e destino), o que pode ser ruim em sistemas críticos de memória (Ex: micro-ondas)	Por outro lado, gasto com ponteiros	
Gerenciamento de memória			
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço	Sim quando você sabe o tamanho exato	Sim, quando você tem variabilidade de tamanho	Sim, quando você tem variabilidade de tamanho
	Não, quando não sabe o tamanho pois mesmo com realloc há um momento em que precisa dos dois vetores (origem e destino), o que pode ser ruim em sistemas críticos de memória (Ex: micro-ondas)	Por outro lado, gasto com ponteiros	Gasto com ponteiros (2 por nos)
Gerenciamento de memória			
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

EACH USP

Profa. Ariane Machado Lima

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço	Sim quando você sabe o tamanho exato	Sim, quando você tem variabilidade de tamanho	Sim, quando você tem variabilidade de tamanho
	Não, quando não sabe o tamanho pois mesmo com realloc há um momento em que precisa dos dois vetores (origem e destino), o que pode ser ruim em sistemas críticos de memória (Ex: micro-ondas)	Por outro lado, gasto com ponteiros	Gasto com ponteiros (2 por nos)
Gerenciamento de memória	automático		
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

EACH USP

Profa. Ariane Machado Lima

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço	Sim quando você sabe o tamanho exato	Sim, quando você tem variabilidade de tamanho	Sim, quando você tem variabilidade de tamanho
	Não, quando não sabe o tamanho pois mesmo com realloc há um momento em que precisa dos dois vetores (origem e destino), o que pode ser ruim em sistemas críticos de memória (Ex: micro-ondas)	Por outro lado, gasto com ponteiros	Gasto com ponteiros (2 por nos)
Gerenciamento de memória	automático	controle	
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

EACH USP
Profa. Ariane Machado Lima

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão			
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço	Sim quando você sabe o tamanho exato	Sim, quando você tem variabilidade de tamanho	Sim, quando você tem variabilidade de tamanho
	Não, quando não sabe o tamanho pois mesmo com realloc há um momento em que precisa dos dois vetores (origem e destino), o que pode ser ruim em sistemas críticos de memória (Ex: micro-ondas)	Por outro lado, gasto com ponteiros	Gasto com ponteiros (2 por nos)
Gerenciamento de memória	automático	controle	controle
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

EACH USP

Profa. Ariane Machado Lima

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão	Bom para quando o nr de buscas é bem maior que de inserções e remoções, quando eu sei o nr de elemento		
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço	Sim quando você sabe o tamanho exato	Sim, quando você tem variabilidade de tamanho	Sim, quando você tem variabilidade de tamanho
	Não, quando não sabe o tamanho pois mesmo com realloc há um momento em que precisa dos dois vetores (origem e destino), o que pode ser ruim em sistemas críticos de memória (Ex: micro-ondas)	Por outro lado, gasto com ponteiros	Gasto com ponteiros (2 por nos)
Gerenciamento de memória	automático	controle	controle
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	Ligada	Árvores Balanceadas
Conclusão	Bom para quando o nr de buscas é bem maior que de inserções e remoções, quando eu sei o nr de elemento	Bom para quando o nr de inserções e remoções é menor que o nr de buscas; Muito eficiente para filas, pilhas e deques	
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço	Sim quando você sabe o tamanho exato	Sim, quando você tem variabilidade de tamanho	Sim, quando você tem variabilidade de tamanho
	Não, quando não sabe o tamanho pois mesmo com realloc há um momento em que precisa dos dois vetores (origem e destino), o que pode ser ruim em sistemas críticos de memória (Ex: micro-ondas)	Por outro lado, gasto com ponteiros	Gasto com ponteiros (2 por nos)
Gerenciamento de memória	automático	controle	controle
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

Comparação

	Listas Lineares		Listas Não Lineares
	Sequencial (vetor)	**Ligada**	**Árvores Balanceadas**
Conclusão	Bom para quando o nr de buscas é bem maior que de inserções e remoções, quando eu sei o nr de elemento	Bom para quando o nr de inserções e remoções é menor que o nr de buscas; Muito eficiente para filas, pilhas e deques	Bom para quando tenho tanto buscas, inserções e remoções e a perda de espaco com ponteiros não e crítica
Tempo de acesso aleatório (n-esimo elemento)	O(1)	O(n)	O(n)
Facilidade de implementacao	sim	mais difícil que vetor, mais simples que árvores	mais difícil que as outras duas
Busca	O(lg n)	O(n)	O(log n)
Eficiência de espaço	Sim quando você sabe o tamanho exato	Sim, quando você tem variabilidade de tamanho	Sim, quando você tem variabilidade de tamanho
	Não, quando não sabe o tamanho pois mesmo com realloc há um momento em que precisa dos dois vetores (origem e destino), o que pode ser ruim em sistemas críticos de memória (Ex: micro-ondas)	Por outro lado, gasto com ponteiros	Gasto com ponteiros (2 por nos)
Gerenciamento de memória	automático	controle	controle
Inserção / remoção	O(n)	O(1) – só quando souber a posicao	O(log n)
min/max	O(1)	O(1) (com cabeca apontando para ultimo)	O(log n)

Matrizes esparsas

Matrizes

- Para que usamos mesmo?

Matrizes

- Ex: imagens (matriz de pixels)
  - PB: 2 dimensões
  - Colorida: 3 dimensões (sistema RGB)

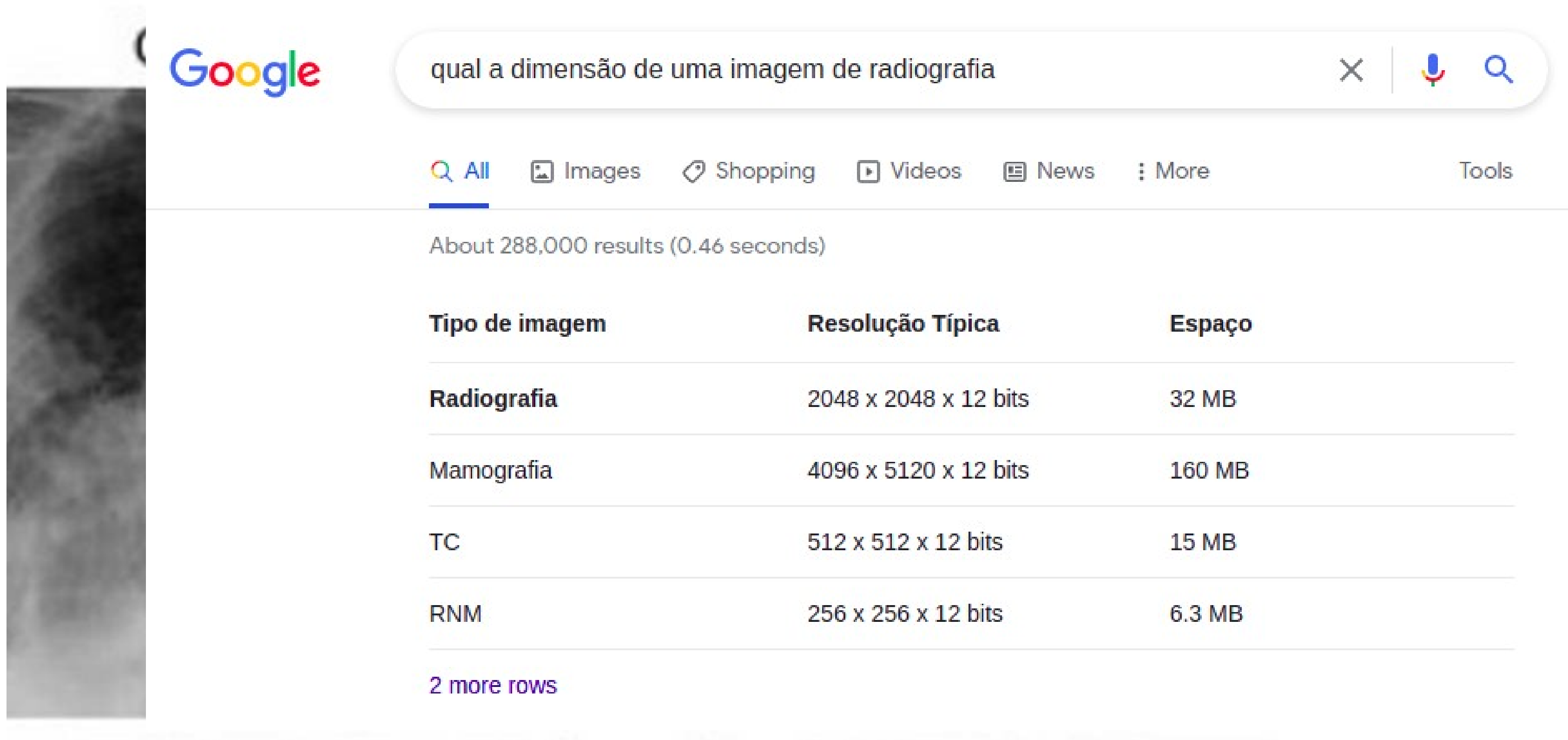
Matrizes

- Aplicações em saúde

https://oglobo.globo.com/epoca/brasil/pesquisadores-criam-aplicativo-para-diagnostico-da-covid-19-partir-de-raio-dos-pulmoes-24550251

Matrizes

- Ex: imagens



Tipo de imagem	Resolução Típica	Espaço
Radiografia	2048 x 2048 x 12 bits	32 MB
Mamografia	4096 x 5120 x 12 bits	160 MB
TC	512 x 512 x 12 bits	15 MB
RNM	256 x 256 x 12 bits	6.3 MB

2 more rows

Matrizes

- Algumas imagens possuem menos “informação”

t.ly/kbEc

Matrizes

- Algumas imagens possuem menos “informação”

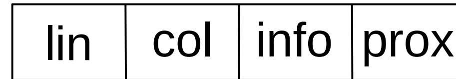
t.ly/kbEc

Matrizes esparsas !!!

- Uma matriz é esparsa quando ela possui mais zeros (ou elementos nulos) do que não-zeros (valores válidos)
- Uma representação enxuta: representamos só o que for diferente de zero!
- Há algumas de representar, ex:
  - Linhas (lista ligada de células por linha)
  - Listas cruzadas (listas ligadas cruzadas de linhas e colunas)

Implementação de matrizes esparsas por linhas (lista ligada organizada por linha)

Cada célula válida é um nó:

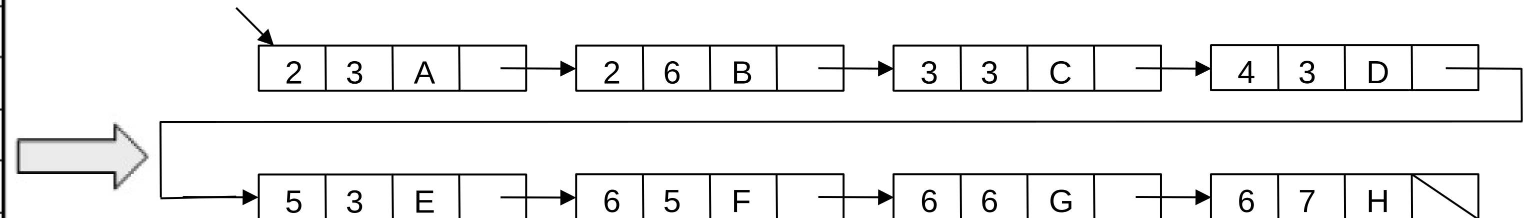



```
typedef struct estrutura
{
        int lin;
        int col;
        TIPOINFO info;
        struct estrutura *prox;
} NO;

typedef struct
{
        NO* inicio;
} MATRIZ;
```

Nós ordenados primeiro pela linha, depois pela coluna:

	1	2	3	4	5	6	7	8
1								
2			A			B		
3			C					
4			D					
5			E					
6					F	G	H	
7								
8								



Quando representação por linhas ocupa menos espaço

Matriz de MAX_LIN linhas e MAX_COL colunas, S = sizeof(TIPOINFO), N = número de elementos não nulos

Espaços ocupados:

Matriz tradicional: MAX_LIN * MAX_COL * S

Matriz por Linhas: N * (2*sizeof(int) + S + sizeof(ponteiro))

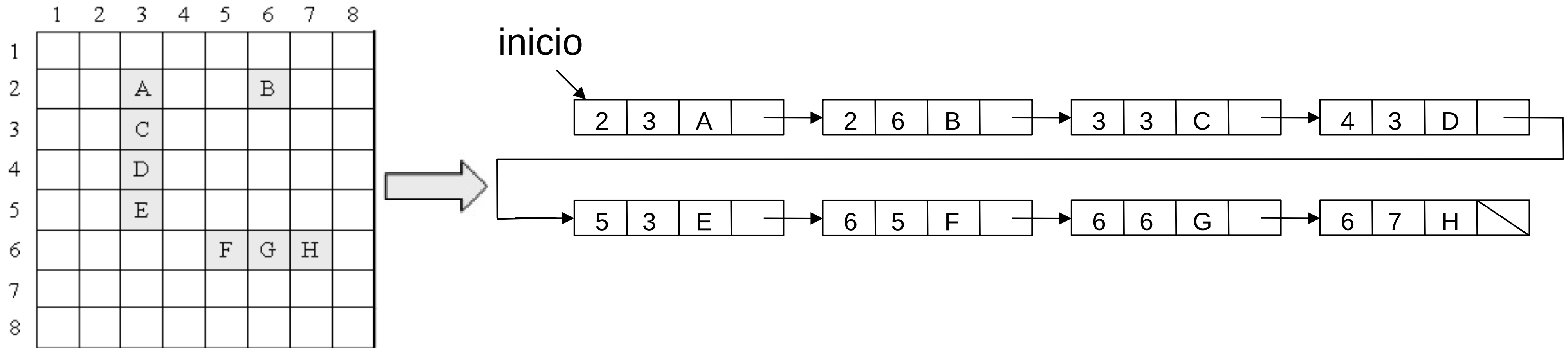
Vantajoso por Linhas quando:

N * (2*sizeof(int) + S + sizeof(ponteiro)) < MAX_LIN * MAX_COL * S

$$N < \frac{MAX_LIN * MAX_COL * S}{2*sizeof(int) + S + sizeof(ponteiro)}$$

Profa. Ariane Machado Lima

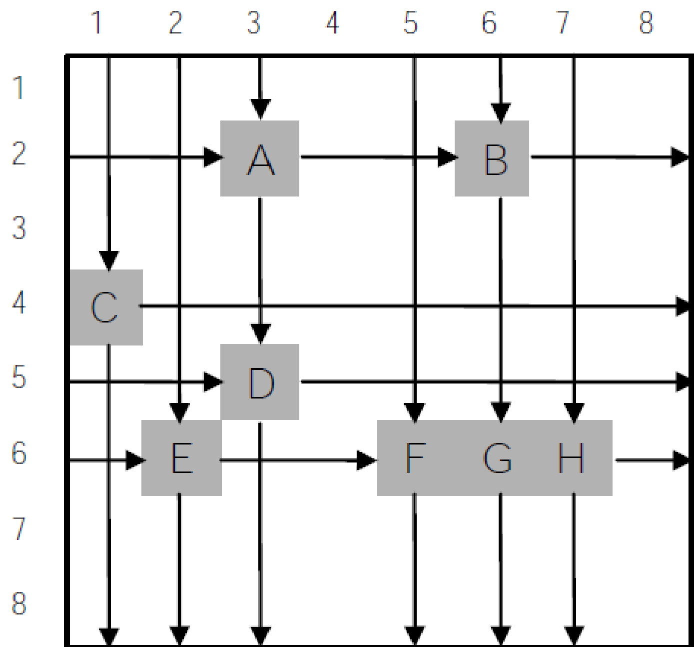
Implementação de matrizes esparsas por linhas (lista ligada organizada por linha)

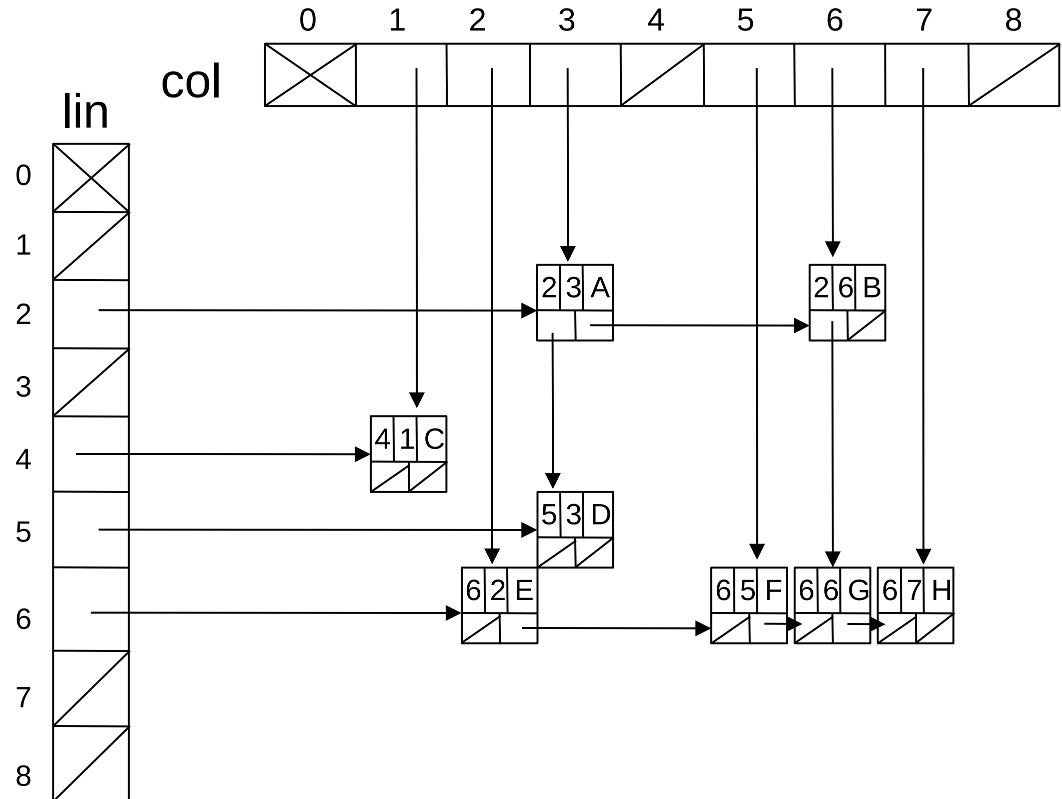


Ok só para armazenar a matriz, mas ruim para buscar elementos específicos, inserção e remoção:

- Busca pela linha i: tem que passar primeiro por todos os nós das i-1 linha anteriores
- Busca pela coluna j: tem olhar tudo!

Solução: Implementação de matrizes esparsas por listas cruzadas



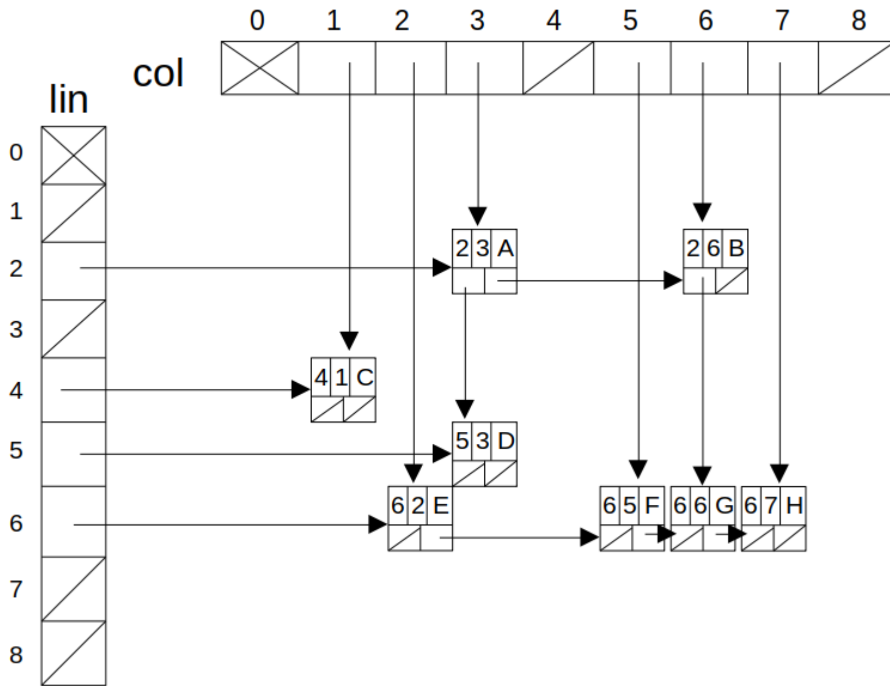


Vantajoso quando:

```
(MAXCOL+1)*(MAXLIN+1)*sizeof(TIPOINFO) > N * sizeof(NO) + sizeof(NO*)*(MAXLIN+MAXCOL+2)
```

N = número de elementos não nulos na matriz

Solução: Implementação de matrizes esparsas por listas cruzadas




```
typedef struct estrutura
{
        int lin;
        int col;
        TIPOINFO info;
        struct estrutura *proxL;
        struct estrutura *proxC;
} NO;
```

permite o cruzamento das listas

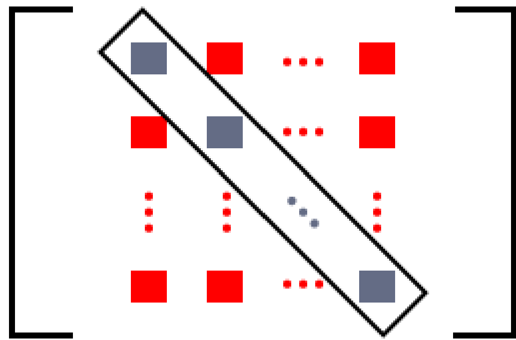
```
typedef struct
{
      NO* lin[MAXLIN+1]; // para indexar até MAXLIN
      NO* col[MAXCOL+1]; // para indexar até MAXCOL
} MATRIZ;
```

permite o acesso instantâneo à lista de uma linha ou coluna específica

Tipos particulares de matrizes esparsas

Matriz diagonal

Só a diagonal **principal** PODE possuir elementos diferentes de zero



$$m_{ij} = \begin{cases} 0, \text{ se } i \neq j \\ ?, \text{ se } i = j \end{cases}$$

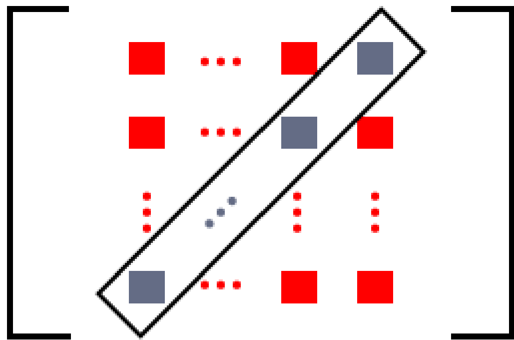
Ex: $A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}$

Qual seria uma representação eficiente de uma matriz diagonal M_{nxn}?

Por um vetor v de tamanho n: v[i] = m[i][i]

Matriz diagonal

O mesmo se fosse a diagnonal **secundária** (M_{nxn})



$$m_{ij} = \begin{cases} 0, \text{ se } j \neq (n-i+1) \\ ?, \text{ se } j = (n-i+1) \end{cases}$$

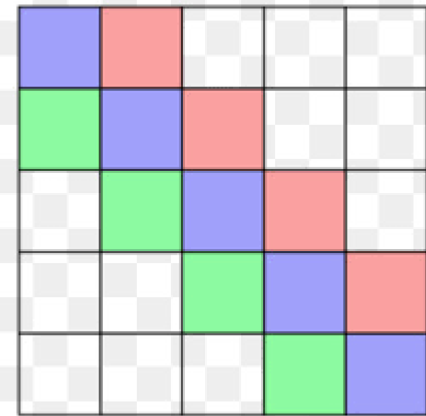
Ex: $B = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 6 & 0 \\ 5 & 0 & 0 \end{bmatrix}$

Qual seria uma representação eficiente dessa uma matriz diagonal M_{nxn}?

Por um vetor v de tamanho n: v[i] = m[i][n-i+1]

Matriz tridiagonal

Elementos diferentes de zero na diagnonal e nas duas diagonais imediatamente acima e abaixo dessa



$$m_{ij} = \begin{cases} 0, \text{ se } |i - j| > 1 \\ ?, \text{ se } |i - j| \leq 1 \end{cases}$$

i	j	2i+j-2
1	1	1
1	2	2
2	1	3
2	2	4
2	3	5
3	2	6
3	3	7
3	4	8
4	3	9
4	4	10
4	5	11
5	4	12
5	5	13

Qual seria uma representação eficiente?

Por um vetor v de tamanho 3*n -2 : m[i][j] = v[2*i + j - 2]

Matrizes Triangulares

Elementos não nulos na diagonal principal e:

- acima : matriz triangular superior (só interessa m_{ij} se $i \leq j$)
- abaixo : matriz triangular inferior (só interessa m_{ij} se $i \geq j$)

Qual seria uma representação eficiente?

Nr de elementos não nulos: $(n^2 + n) / 2$

i \ j	1	2	3	4	5
1	1	2	4	7	11
2	2	3	5	8	12
3	3	4	6	9	13
4	4	5	7	10	14
5	5	6	8	11	15

Superior (ordenada por colunas):

$m[i][j] = v[(i^2 - i)/2 + j]$

i \ j	1	2	3	4	5
1	1	2	3	4	5
2	2	3	4	5	6
3	4	5	6	7	8
4	7	8	9	10	11
5	11	12	13	14	15

Inferior (ordenada por linhas):

$m[i][j] = v[(j^2 - j)/2 + i]$

Tenho alguns vídeos no eaulas...

https://eaulas.usp.br/portal/video?idItem=30799