

Melhores momentos

Lista de nomes em ordem alfabética

AULA 7

0	Carlos	n = 8
1	Eduardo	
2	Helio	
3	Joao	
4	Luiz	
5	Maria	
6	Rui	
7	Sergio	
8		
9		
10		

Remover Joao

0	Carlos	n = 8
1	Eduardo	
2	Helio	
3	Joao	
4	Luiz	
5	Maria	
6	Rui	
7	Sergio	
8		
9		
10		

Remover Joao

0	Carlos	n = 7
1	Eduardo	
2	Helio	
3		
4	Luiz	
5	Maria	
6	Rui	
7	Sergio	
8		
9		
10		

Remover Joao

0	Carlos	n = 7
1	Eduardo	
2	Helio	
3	Luiz	
4		
5	Maria	
6	Rui	
7	Sergio	
8		
9		
10		

Remover Joao

0	Carlos	n = 7
1	Eduardo	
2	Helio	
3	Luiz	
4	Maria	
5		
6	Rui	
7	Sergio	
8		
9		
10		

Inserir Ana

0	Carlos
1	Eduardo
2	Helio
3	Luiz
4	Maria
5	Rui
6	Sergio
7	
8	Walter
9	
10	

n = 9



Inserir Ana

0	Carlos
1	Eduardo
2	Helio
3	Luiz
4	Maria
5	Rui
6	
7	Sergio
8	Walter
9	
10	

n = 9



Inserir Ana

0	Carlos
1	Eduardo
2	Helio
3	Luiz
4	Maria
5	
6	Rui
7	Sergio
8	Walter
9	
10	

n = 9



Inserir Ana

0	Carlos
1	Eduardo
2	Helio
3	Luiz
4	
5	Maria
6	Rui
7	Sergio
8	Walter
9	
10	

n = 9



Inserir Ana

0	Carlos
1	Eduardo
2	Helio
3	
4	Luiz
5	Maria
6	Rui
7	Sergio
8	Walter
9	
10	

n = 9



Inserir Ana

0	Carlos
1	Eduardo
2	
3	Helio
4	Luiz
5	Maria
6	Rui
7	Sergio
8	Walter
9	
10	

n = 9



Inserir Ana

0	Carlos
1	
2	Eduardo
3	Helio
4	Luiz
5	Maria
6	Rui
7	Sergio
8	Walter
9	
10	

$n = 9$

◀ ▶ ↺ ↻ 🔍

Inserir Ana

0	
1	Carlos
2	Eduardo
3	Helio
4	Luiz
5	Maria
6	Rui
7	Sergio
8	Walter
9	
10	

$n = 9$

◀ ▶ ↺ ↻ 🔍

Inserir Ana

0	Ana
1	Carlos
2	Eduardo
3	Helio
4	Luiz
5	Maria
6	Rui
7	Sergio
8	Walter
9	
10	

$n = 9$

◀ ▶ ↺ ↻ 🔍

Inserção em um vetor

Esta função `insere x` entre `v[k-1]` e `v[k]` no vetor `v[0 . . n-1]`. Ela supõe apenas que $0 \leq k \leq n$. A função devolve o novo valor de `n`.

```
int insere (int k, int x, int n, int v[])
{
    int j;
    for (j = n; j > k; j--)
        v[j] = v[j-1];
    v[k] = x;
    return n+1;
}
```

◀ ▶ ↺ ↻ 🔍

Conclusões

No **pior caso** o consumo de tempo da função `insere` é proporcional a `n`.

O **consumo de tempo** da função `insere` é $O(n)$.

$O(n)$ = “é da ordem de `n`”

◀ ▶ ↺ ↻ 🔍

Mais conclusões

Manter uma **lista** em um vetor sujeita a **remoções** e **inserções** pode dar muito trabalho com **movimentações**.

Veremos uma maneira alternativa que pode dar **menos trabalho** com **movimentações**, se estivermos disposto a gastar um pouco **mais de espaço**.

◀ ▶ ↺ ↻ 🔍

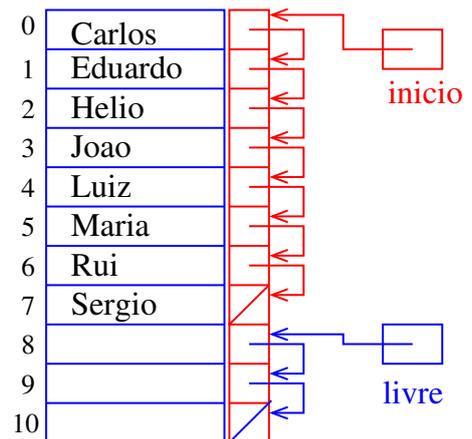
Listas encadeadas em vetores

AULA 8

Lista encadeada



Lista encadeada



Estrutura de uma lista encadeada em vetor

Uma **lista encadeada** (= *linked list* = *lista ligada*) é uma sequência de **células**; cada **célula** contém um **objeto** de algum tipo e o **endereço** da célula seguinte.

```
struct celula {
    int conteudo;
    int prox;
};
typedef struct celula Celula;
Celula v[MAX];
int inicio;
int livre;
```

Imprime conteúdo de uma lista

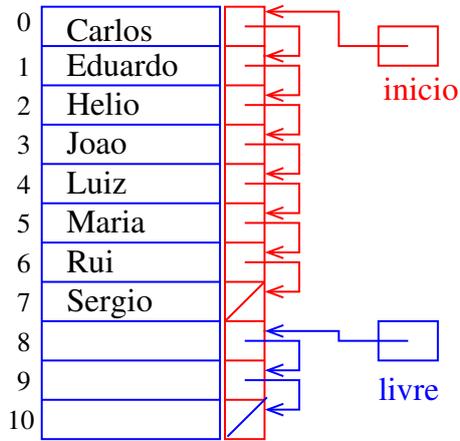
Esta função **recebe** o **índice início** de uma lista encadeada **em um vetor** e imprime os elementos da lista.

```
#define NULO -1
void imprime(int inicio, Celula v[])
{
    int p;

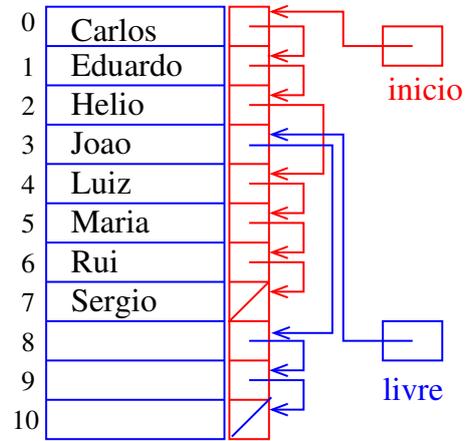
    for (p=inicio; p != NULO; p=v[p].prox)
        printf(" %d ", v[p].conteudo);

    printf("\n");
}
```

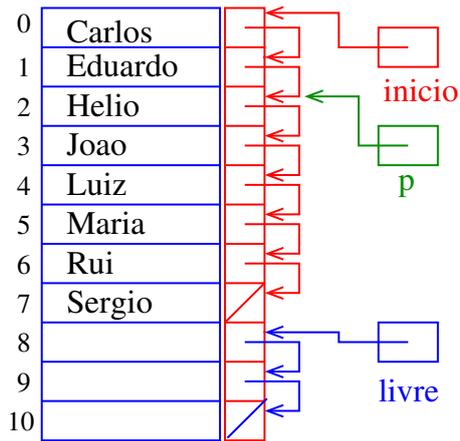
Remover Joao



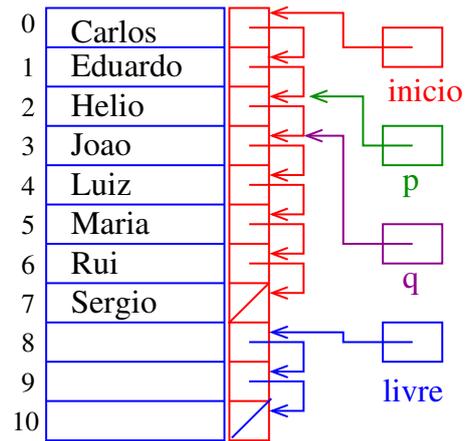
Remover Joao



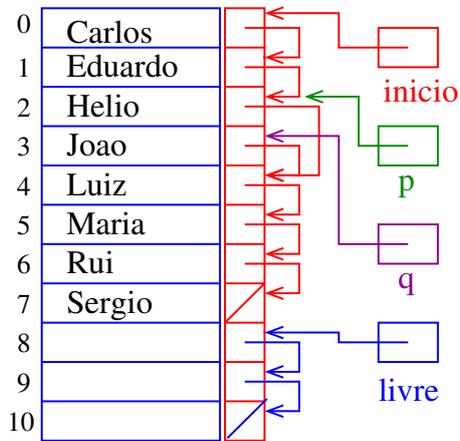
Remover Joao



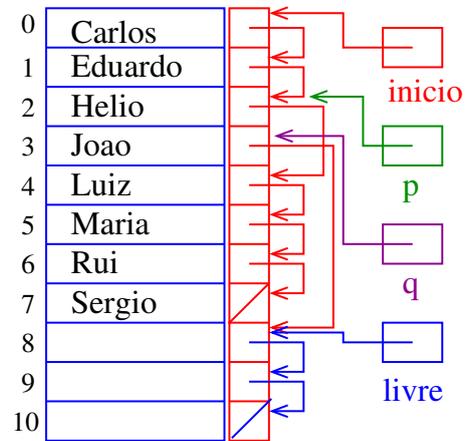
Remover Joao



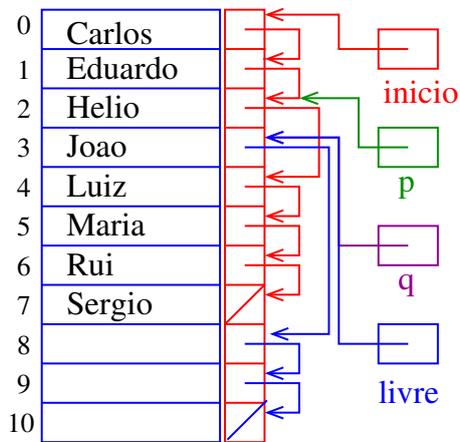
Remover Joao



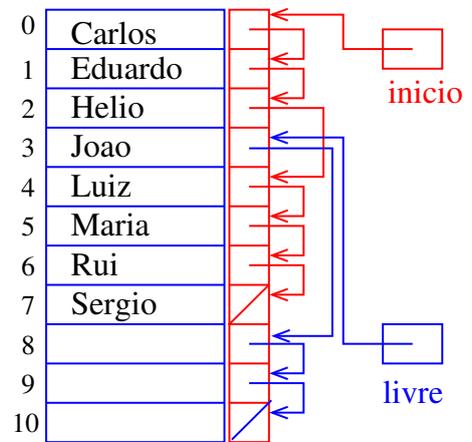
Remover Joao



Remover Joao



Remover Joao



Remoção

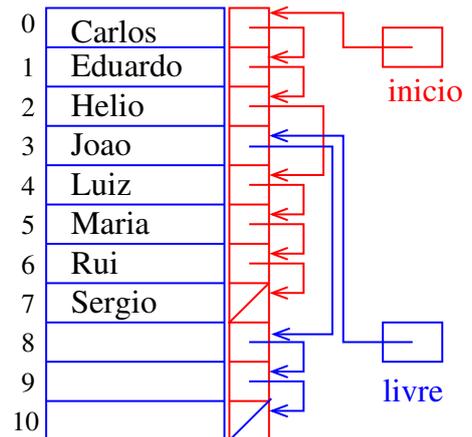
Dado um índice p de uma célula o trecho de código que remove a célula de índice $v[p].prox$ é “essencialmente” o seguinte

```

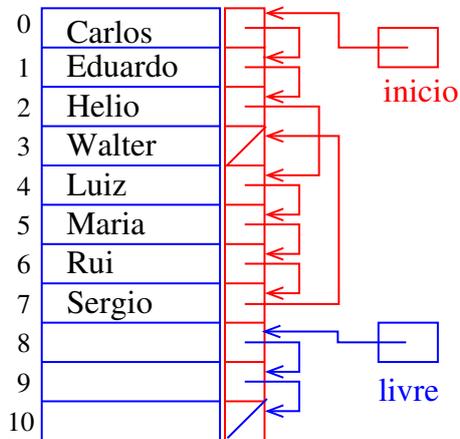
q = v[p].prox;
v[p].prox = v[q].prox;
v[q].prox = livre;
livre = q;
    
```

Dizemos que o consumo de tempo do trecho de código acima é constante, pois não depende do tamanho da lista.

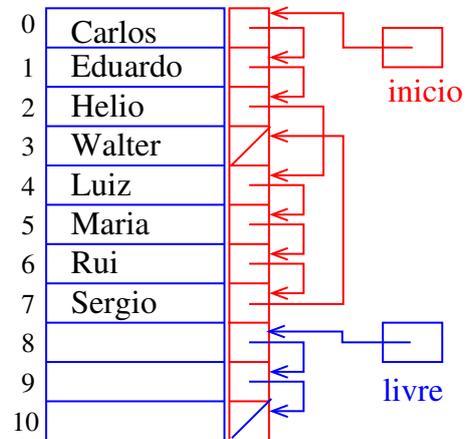
Inserir Walter



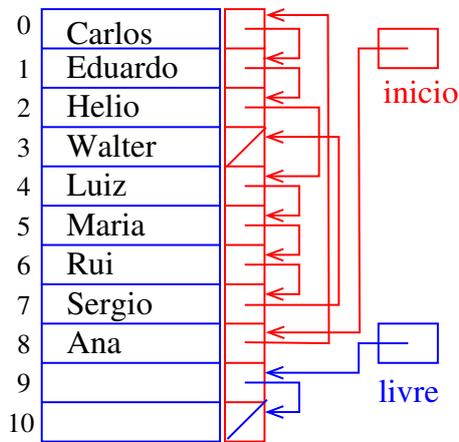
Inserir Walter



Inserir Ana



Inserir Ana



Inserção

O trecho de código para inserir um elemento x no entre as células de índice p e $v[p].prox$ é “especialmente” o seguinte

```
nova = livre;
livre = v[livre].prox;
v[nova].conteudo = x;
v[nova].prox = v[p].prox;
v[p].prox = nova;
```

Dizemos que o **consumo de tempo** do trecho de código acima é **constante**, pois **não depende** do tamanho da lista.

Listas encadeadas em C

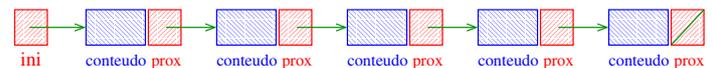
PF 4, S 3.3

<http://www.ime.usp.br/~pf/algoritmos/aulas/lista.html>

Listas encadeadas

Uma **lista encadeada** (= *linked list* = lista ligada) é uma sequência de **células**; cada **célula** contém um **objeto** de algum tipo e o **endereço** da célula seguinte.

Ilustração de uma **lista encadeada** (“sem cabeça”)



Estrutura de uma lista encadeada em C

```
struct celula {
    int conteudo;
    struct celula *prox;
};
typedef struct celula Celula;

Celula *ini;
/* inicialmente a lista esta vazia */
ini = NULL;
```



Endereço de uma lista encadeada

O **endereço** de uma lista encadeada é o endereço de sua **primeira célula**.

Se p é o endereço de uma lista às vezes dizemos que “ p é uma lista”.

Se p é uma lista então

- ▶ $p == \text{NULL}$ ou
- ▶ $p \rightarrow \text{prox}$ é uma lista.

