

## Aviso

## AULA 7

Amanhã, no horário da monitoria, o Vinicius dará uma aula sobre o `gdb` ([The GNU Debugger Project](http://www.gnu.org/software/gdb/)).

- ▶ **Quando:** das 12h às 13h, terça-feira, 26 de agosto
- ▶ **Onde:** auditório do CCSL

## Hoje

- ▶ listas em vetores
- ▶ listas encadeadas em vetores
- ▶ listas encadeadas em C



Fonte: <http://onlinefungags.com/>

## Listas em vetores

PF 3

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

## Lista de nomes em ordem alfabética

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

$n = 8$

## Remover Joao

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

$n = 8$

### Remover Joao

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

n = 7

< > < > < > < > < > < >

### Remover Joao

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

n = 7

< > < > < > < > < > < >

### Remover Joao

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

n = 7

< > < > < > < > < > < >

### Remover Joao

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

n = 7

< > < > < > < > < > < >

### Remover Joao

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

n = 7

< > < > < > < > < > < >

### Busca em um vetor

A função recebe  $x$ ,  $n \geq 0$  e  $v$  e devolve um índice  $k$  em  $0..n-1$  tal que  $x == v[k]$ .

Se tal  $k$  não existe, devolve -1

```
int busca(int x, int n, int v[])
{
    int k;
    k = n-1;
    while (k >= 0 && v[k] != x)
        k -= 1;
    return k;
}
```

< > < > < > < > < > < >

## Busca recursiva em vetor

A função recebe  $x$ ,  $n \geq 0$  e  $v$  e devolve um índice  $k$  em  $0..n-1$  tal que  $x == v[k]$ .  
Se tal  $k$  não existe, devolve  $-1$

```
int buscaR(int x, int n, int v[])
{
    if (n == 0) return -1;
    if (x == v[n-1]) return n-1;
    return buscaR(x, n-1, v);
}
```

Navigation icons

## Remoção em vetor

Esta função recebe  $0 \leq k < n$  e remove o elemento  $v[k]$  do vetor  $v[0..n-1]$ .  
A função devolve o novo valor de  $n$ .

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

Navigation icons

## Conclusões

No pior caso o consumo de tempo da função `remove` é proporcional a  $n$ .

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

$O(n)$  = "é da ordem de  $n$ "

Navigation icons

## Conclusões

No pior caso o consumo de tempo da função `busca` é proporcional a  $n$ .

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

$O(n)$  = "é da ordem de  $n$ "

Navigation icons

## Remoção recursiva

A função recebe  $0 \leq k < n$  e remove o elemento  $v[k]$  do vetor  $v[0..n-1]$ .  
A função devolve o novo valor de  $n$ .

```
int removeR(int k, int n, int v[])
{
    if (k == n-1) return n-1;
    v[k] = v[k+1];
    return removeR(k+1, n, v);
}
```

Navigation icons

## Inserir Walter

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

$n = 7$

Navigation icons



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;
}
```

Navigation icons

## Conclusões

No **piores 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`"

Navigation icons

## Listas encadeadas em vetores

## Inserção recursiva

Recebe  $0 \leq k \leq n$  e `insere` `x` entre `v[k-1]` e `v[k]` no vetor `v[0..n-1]`. A função devolve o novo valor de `n`. (Hmmm. Essa função devia se `void...`)

```
int insereR(int k, int x, int n, int v[])
{
    if (k == n) v[n] = x;
    else {
        v[n] = v[n-1];
        insereR(k, x, n-1, v);
    }
    return n+1;
}
```

Navigation icons

## 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**.

Navigation icons

## Lista encadeada

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

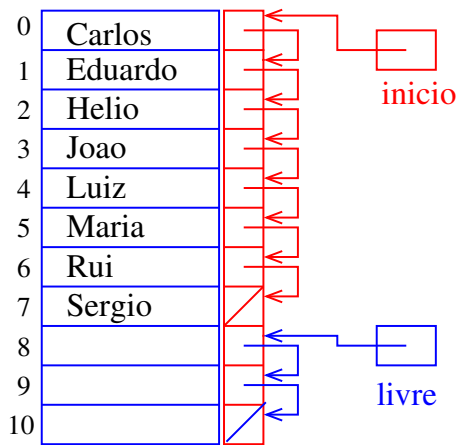
0  
início

8  
livre

Navigation icons

Navigation icons

## 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.

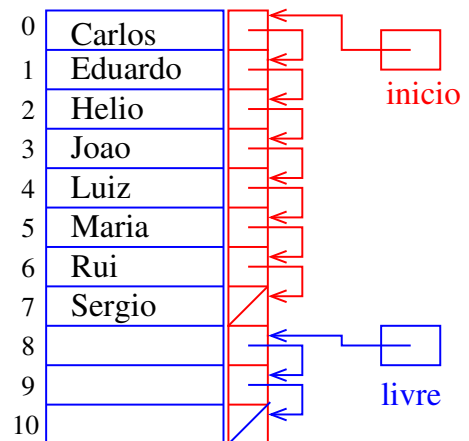
```
struct celula {
    int conteudo;
    int prox;
};
typedef struct celula Celula;
Celula v[MAX];
int inicio;
int livre;
```

## Imprime conteúdo de uma lista

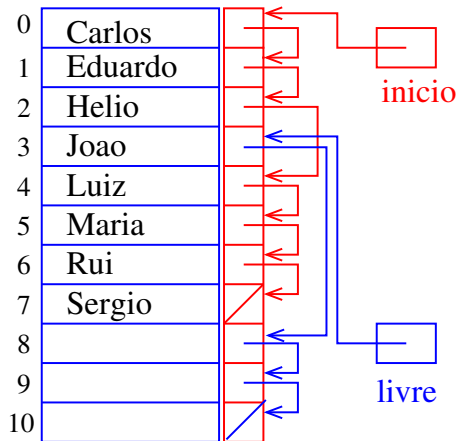
Esta função **recebe** o **índice inicio** de uma lista encadeada **em um vetor** e imprime os elementos da lista.

```
# 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");
}
```

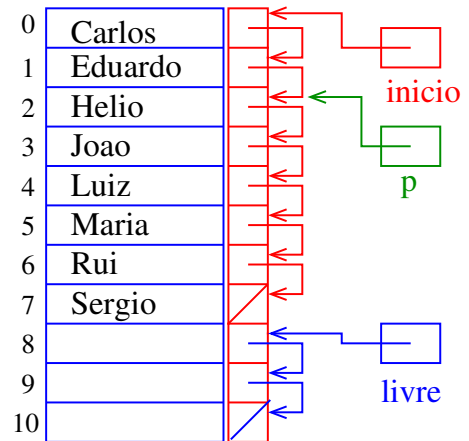
## Remover Joao



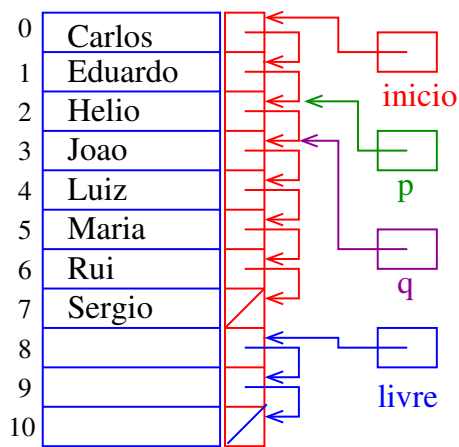
## Remover Joao



## Remover Joao

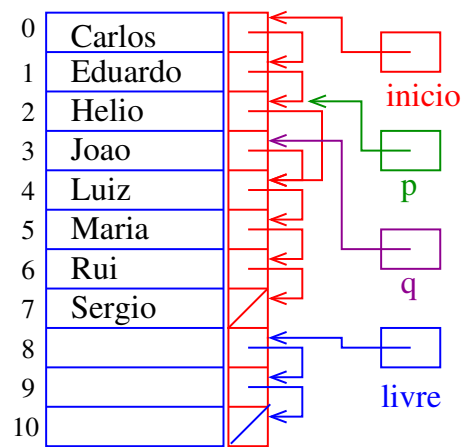


### Remover Joao



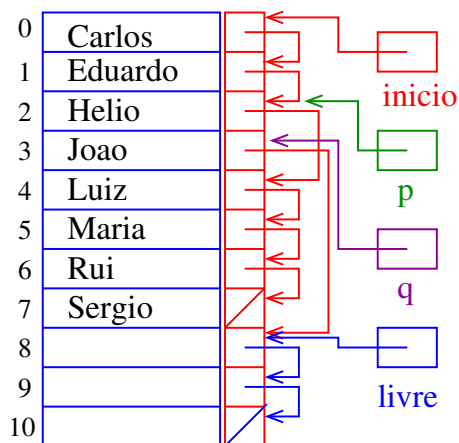
Navigation icons: back, forward, search, etc.

### Remover Joao



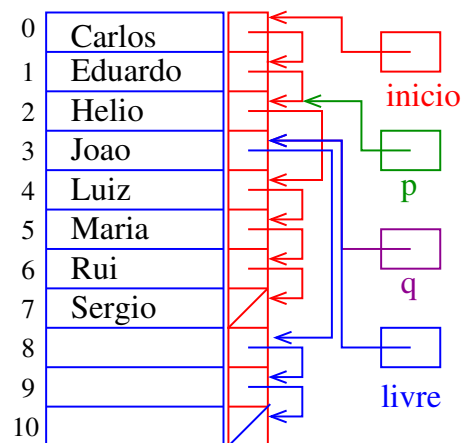
Navigation icons: back, forward, search, etc.

### Remover Joao



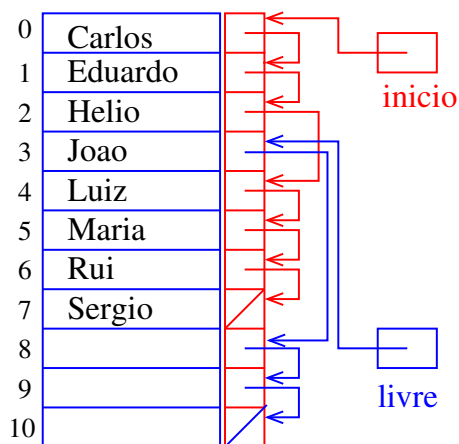
Navigation icons: back, forward, search, etc.

### Remover Joao



Navigation icons: back, forward, search, etc.

### Remover Joao



Navigation icons: back, forward, search, etc.

### 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

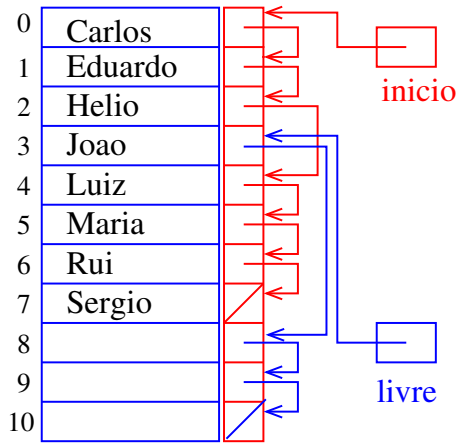
```
q = v[p].prox;  
v[p].prox = v[q].prox;  
v[q].prox = livre;  
livre = q;
```

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

Navigation icons: back, forward, search, etc.

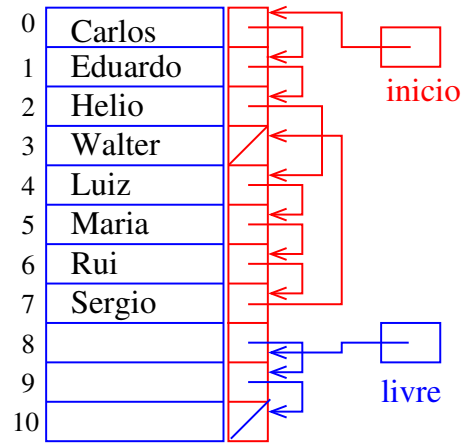


Inserir Walter



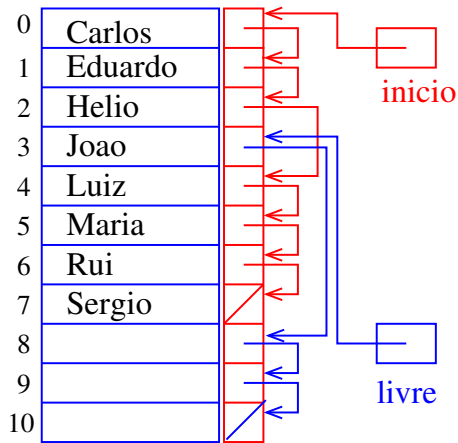
< > < > < > < > < > < >

Inserir Walter



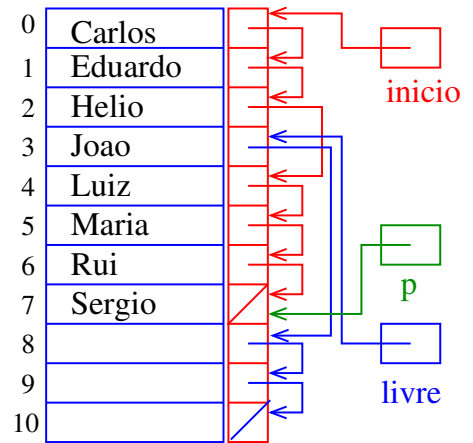
< > < > < > < > < > < >

Inserir Walter



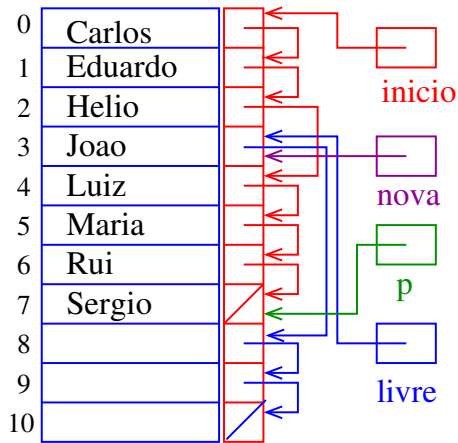
< > < > < > < > < > < >

Inserir Walter



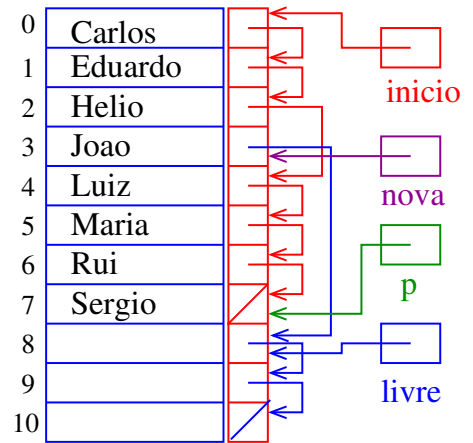
< > < > < > < > < > < >

Inserir Walter



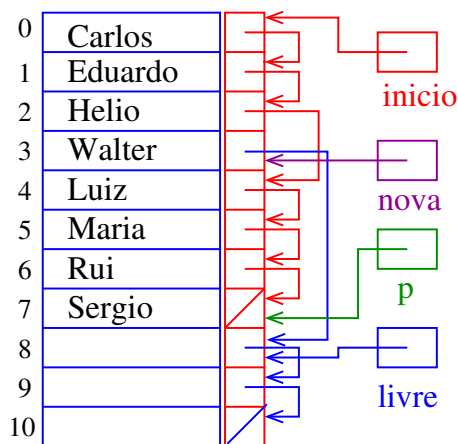
< > < > < > < > < > < >

Inserir Walter



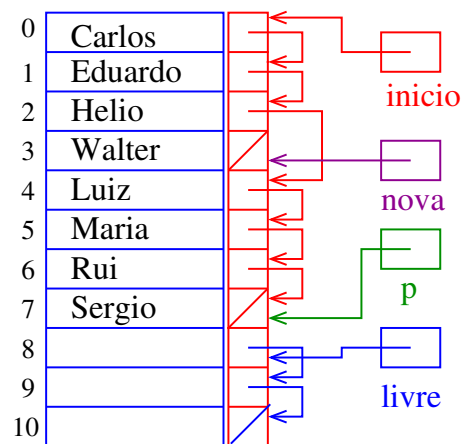
< > < > < > < > < > < >

### Inserir Walter



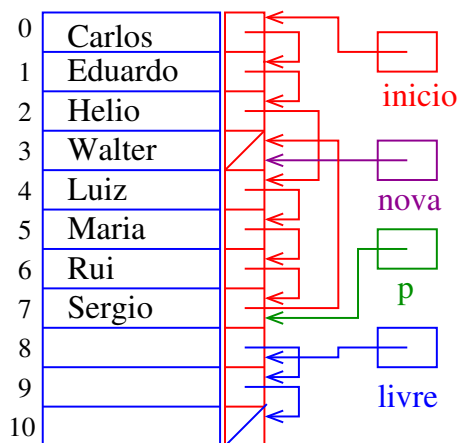
Navigation icons: back, forward, search, etc.

### Inserir Walter



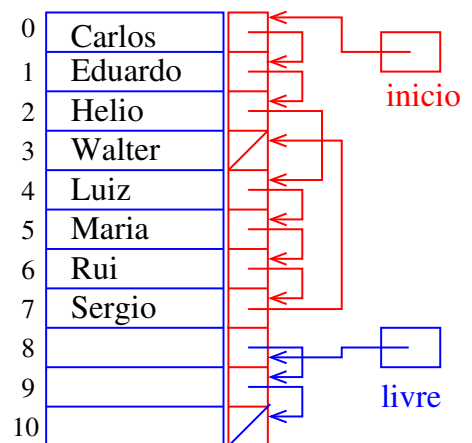
Navigation icons: back, forward, search, etc.

### Inserir Walter



Navigation icons: back, forward, search, etc.

### Inserir Walter



Navigation icons: back, forward, search, etc.

### Inserção

O trecho de código para inserir uma nova célula com o elemento  $x$  entre as células de índice  $p$  e  $v[p].prox$  é “essencialmente” o seguinte

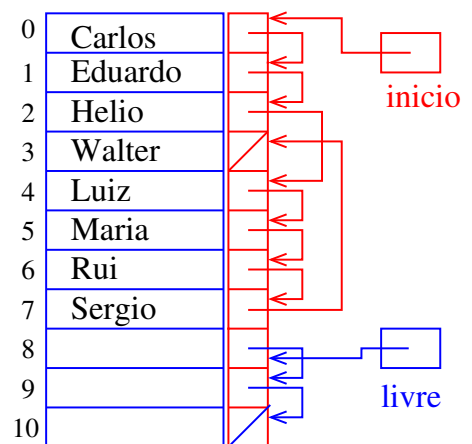
```

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.

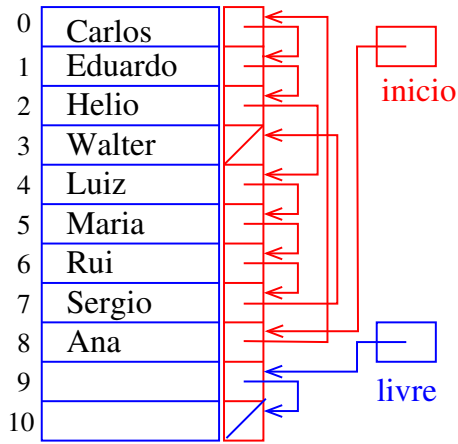
Navigation icons: back, forward, search, etc.

### Inserir Ana



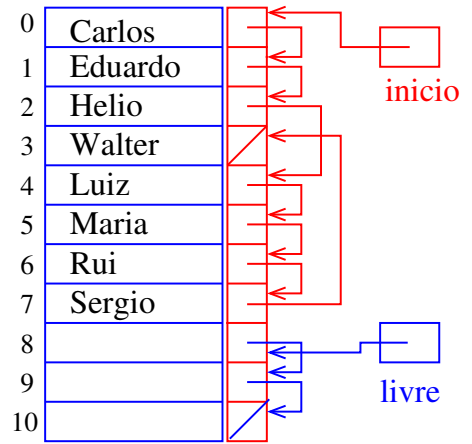
Navigation icons: back, forward, search, etc.

### Inserir Ana



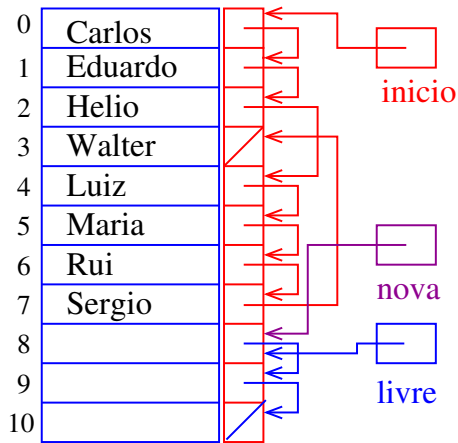
Navigation icons: back, forward, search, etc.

### Inserir Ana



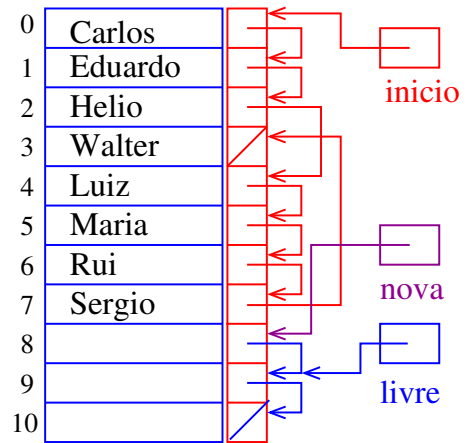
Navigation icons: back, forward, search, etc.

### Inserir Ana



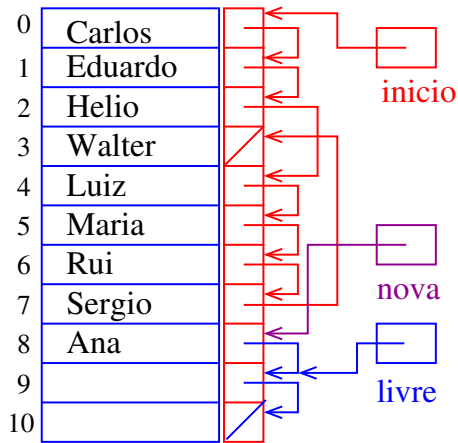
Navigation icons: back, forward, search, etc.

### Inserir Ana



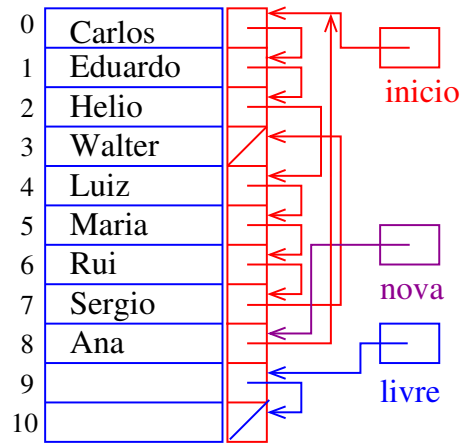
Navigation icons: back, forward, search, etc.

### Inserir Ana



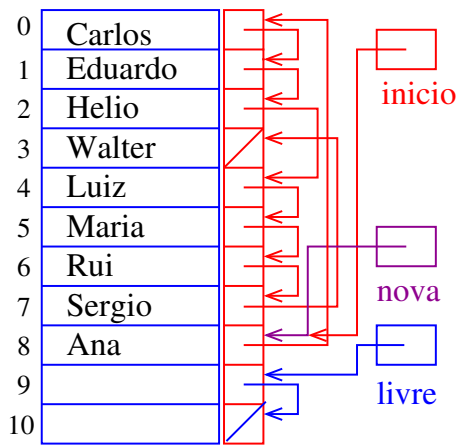
Navigation icons: back, forward, search, etc.

### Inserir Ana

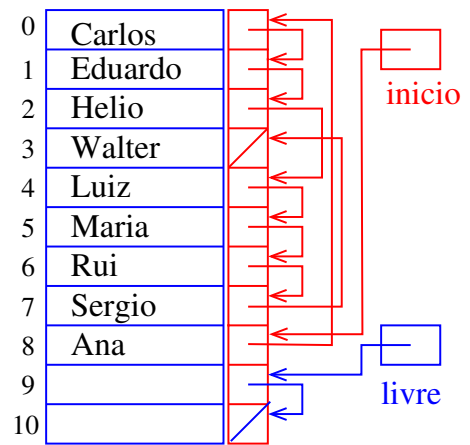


Navigation icons: back, forward, search, etc.

## Inserir Ana



## Inserir Ana



## Inserção no início

O trecho de código para inserir uma nova célula com o elemento  $x$  no **início** da lista;

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

PF 4, S 3.3

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

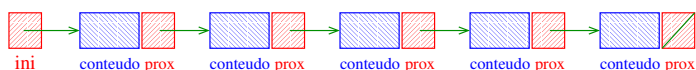
Dizemos que o **consumo de tempo** do trecho de código acima é **constante**, pois **não depende** do tamanho da lista.

## Listas encadeadas em C

### 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 para listas encadeadas em C

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

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



## Endereços listas encadeadas

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 == NULL` ou
- ▶ `p->prox` é uma lista.

< ▶ ◀ ◂ ◃ ≡ ☰ ↻

## Busca em listas encadeadas

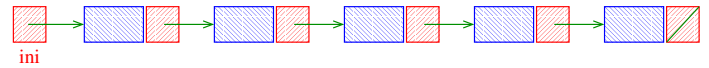
Esta função recebe um inteiro `x` e uma lista `ini`. A função devolve o endereço de uma célula que contém `x`. Se tal célula não existe, a função devolve `NULL`.

```
Celula *busca (int x, Celula *ini)
{
    Celula *p;
    p = ini;
    while (p != NULL && p->conteudo != x)
        p = p->prox;
    return p;
}
```

< ▶ ◀ ◂ ◃ ≡ ☰ ↻

## Imprime conteúdo de uma lista

Esta função imprime o conteúdo de cada célula de uma lista encadeada `ini`.



```
void imprima (Celula *ini)
{
    Celula *p;
    for (p = ini; p != NULL; p=p->prox)
        printf("%d\n", p->conteudo);
}
```

< ▶ ◀ ◂ ◃ ≡ ☰ ↻