BC1424
Algoritmos e Estruturas de Dados I

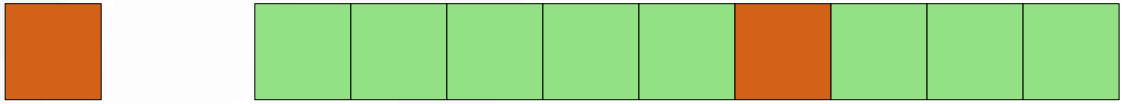
Aula 07:
Listas encadeadas

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

1Q-2015

Operações em vetores

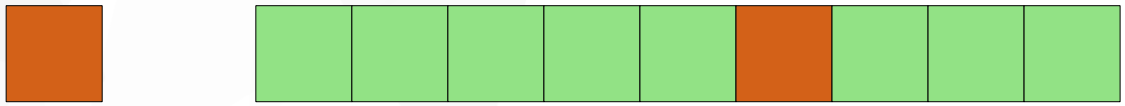
Busca →
(dado um elemento)



O(n) ou O(lg n)

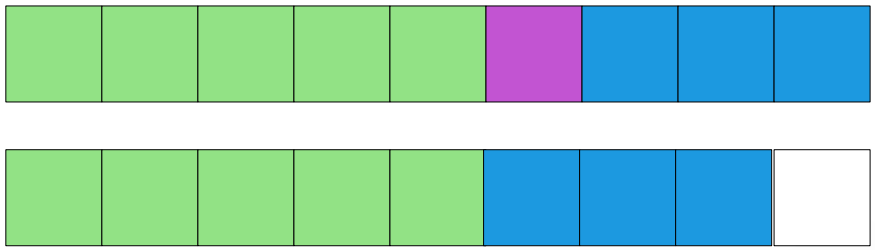
Operações em vetores

Busca →
(dado um elemento)



O(n) ou O(lg n)

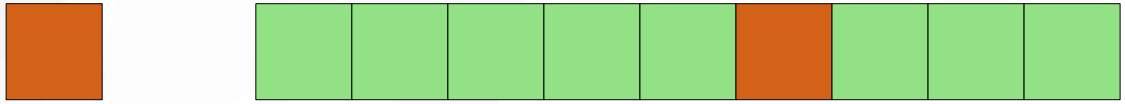
Remoção →
(dado um índice)



O(n)

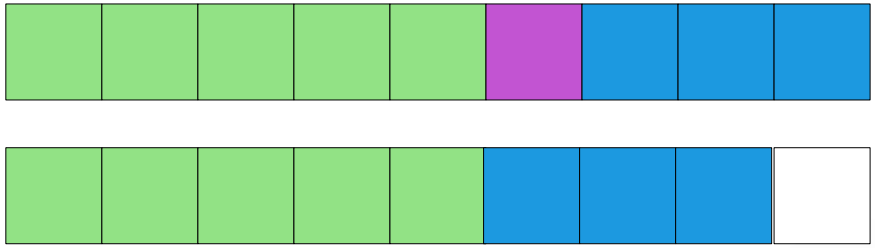
Operações em vetores

Busca →
(dado um elemento)



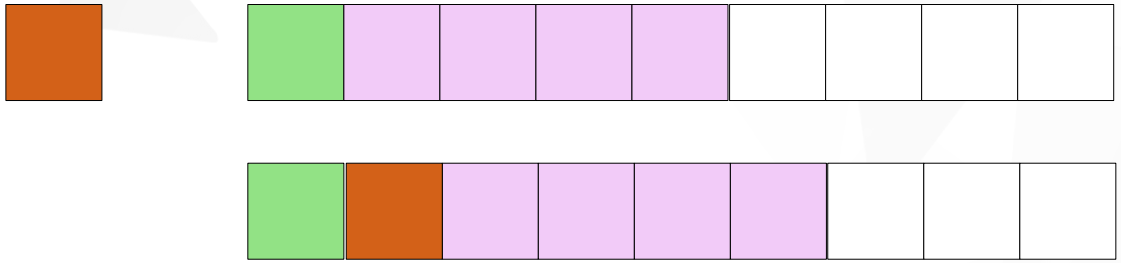
O(n) ou O(lg n)

Remoção →
(dado um índice)



O(n)

Inserção →
(dado um índice
e um elemento)



O(n)

Listas encadeadas

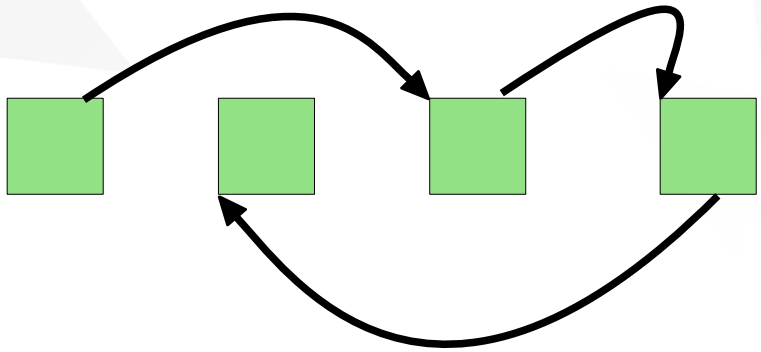
- Uma lista encadeada é uma representação de **uma sequência de objetos** na memória do computador.

- As células que armazenam elementos consecutivos da sequência **não ficam necessariamente em posições consecutivas** da memória.

Listas encadeadas

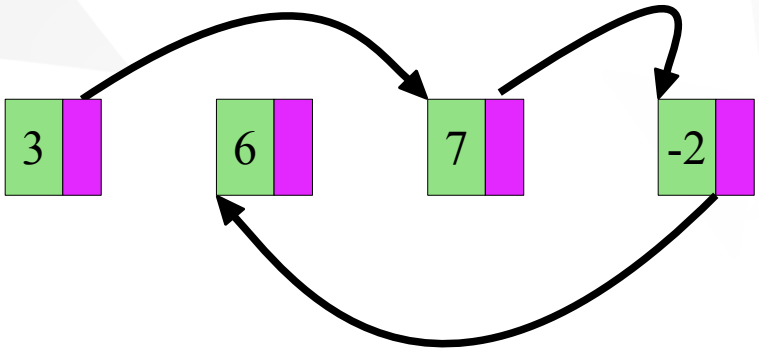
- Uma lista encadeada é uma representação de **uma sequência de objetos** na memória do computador.

- As células que armazenam elementos consecutivos da sequência **não ficam necessariamente em posições consecutivas** da memória.



Definição

- Uma lista encadeada é uma sequência de registros que armazenam células.
  - → Cada célula contém um objeto de determinado tipo.
  - → Cada célula contém o endereço para a célula seguinte.



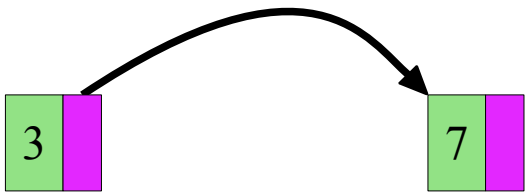
No caso da última célula, o endereço é NULL

Definição

- Suporemos que os objetos armazenados nas células são to tipo int.

- A estrutura de células pode ser definida como:

```
struct cel {
    int          conteudo;
    struct cel   *seg;
};
```



Definição

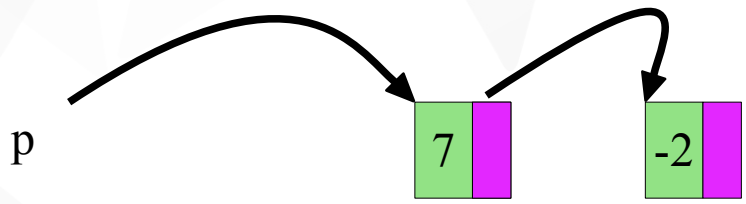
- É conveniente tratar as células como um novo tipo de dados, que chamaremos **celula**:

```
struct cel {
    int          conteudo;
    struct cel   *seg;
};

typedef struct cel celula;
```

```c
#include <stdio.h>

struct cel {
    int          conteudo;
    struct cel  *seg;
};

typedef struct cel celula;

int main()
{
    celula c;     // c eh uma celula
    celula *p;    // p eh um ponteiro para uma celula

    c.conteudo = 3;
    c.seg = NULL;

    printf("Conteudo: %d, End. seguinte: %p", c.conteudo, c.seg);
}
```

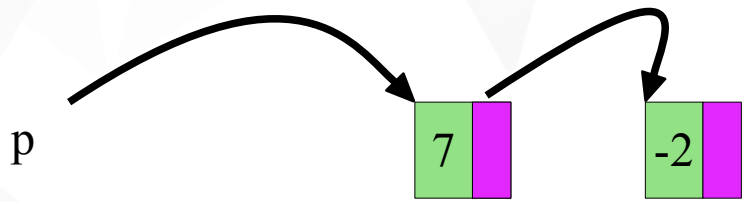
```
onteudo: 3, End. seguinte: (nil)
```

- Se p é um endereço para uma célula:
  - → como acessar ao conteúdo dessa celula?
  - → como obter o endereço da célula seguinte?




```
celula c;
celula *p;

p = &c;
```

- Se p é um endereço para uma célula:
  - → como acessar ao conteúdo dessa celula?
  - → como obter o endereço da célula seguinte?




```
celula c;
celula *p;

p = &c;
```

```
printf("Conteudo: %d, End. seguinte: %p\n", (*p).conteudo, (*p).seg );

printf("Conteudo: %d, End. seguinte: %p\n", p->conteudo,    p->seg );
```

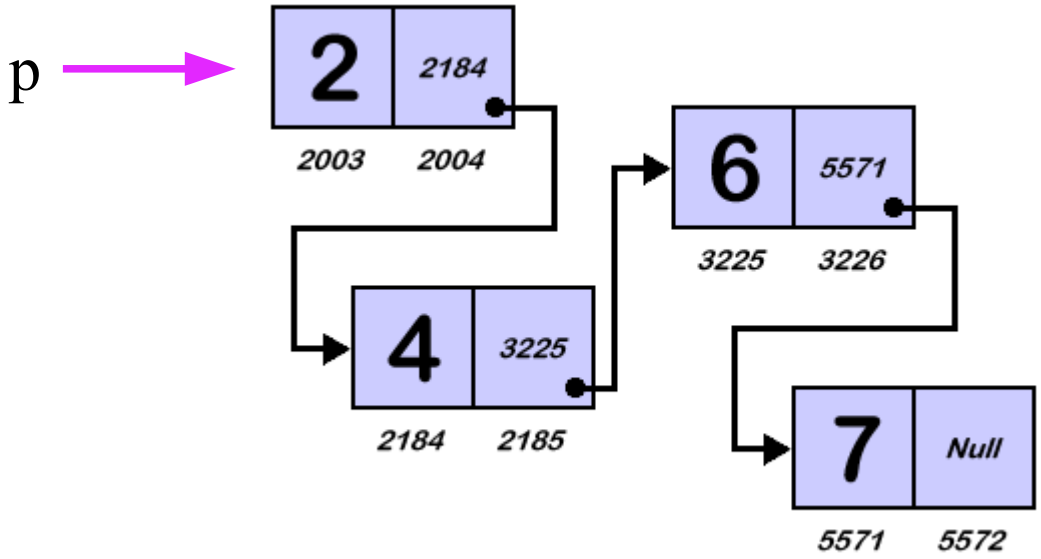
```c
#include <stdio.h>

struct cel {
    int          conteudo;
    struct cel  *seg;
};

typedef struct cel celula;

int main()
{
    celula c;    // c eh uma celula
    celula *p;   // p eh um ponteiro para uma celula

    c.conteudo = 3;
    c.seg = NULL;
    p = &c;

    printf("Conteudo: %d, End. seguinte: %p\n", (*p).conteudo, (*p).seg );
    printf("Conteudo: %d, End. seguinte: %p\n", p->conteudo,    p->seg );
}
```

Listas

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

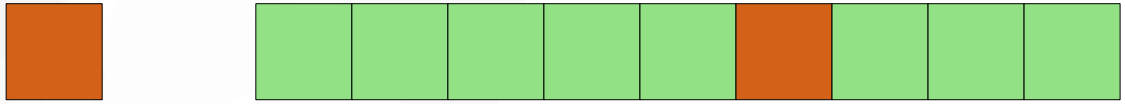
- Se **p** é o endereço de uma lista, podemos dizer, "**p é uma lista**".

```
celula c;
celula *p;

p = &c;
```



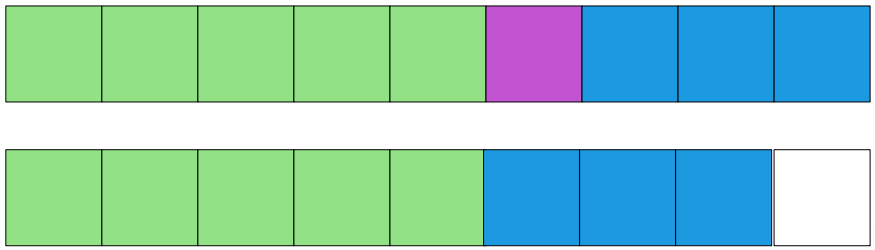
Operações em vetores

Busca →
(dado um elemento)



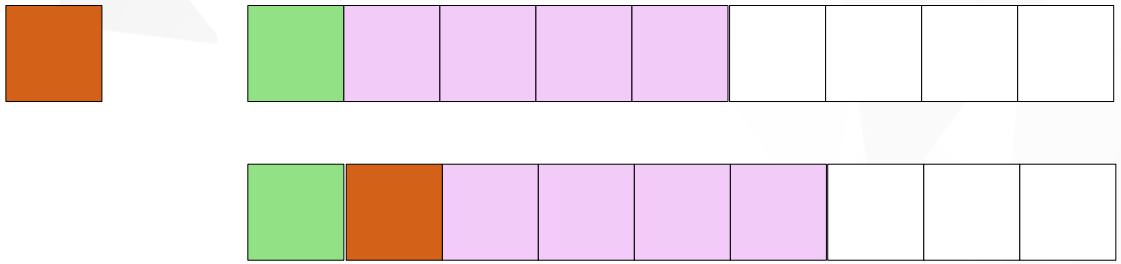
O(n) ou O(lg n)

Remoção →
(dado um índice)



O(n)

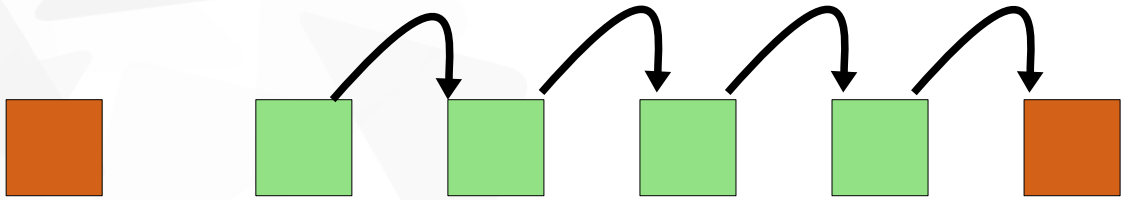
Inserção →
(dado um índice
e um elemento)



O(n)

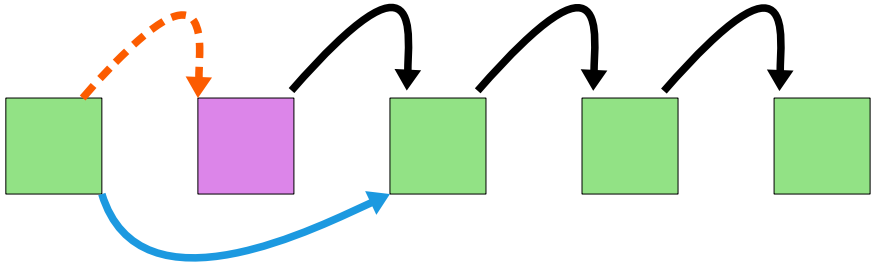
Operações considerando listas encadeadas

Busca →
(dado um elemento)



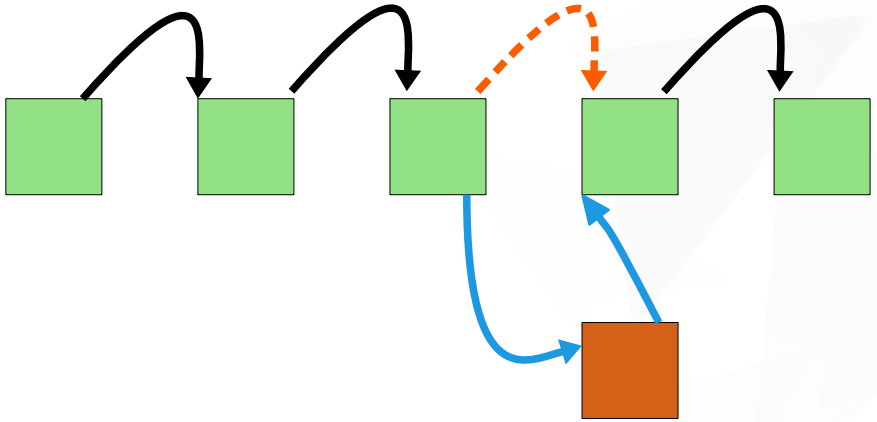
O(n)

Remoção →
(dado um ponteiro)



O(1)

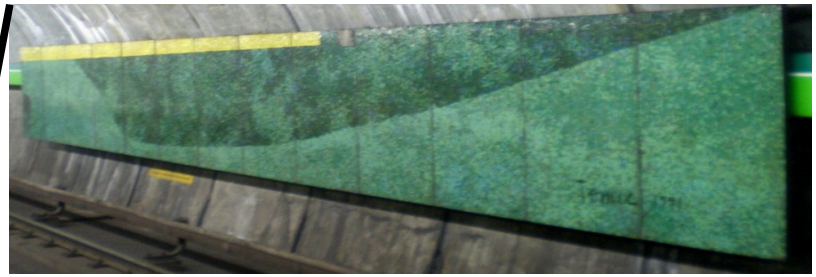
Inserção →
(dado um ponteiro
e um elemento)

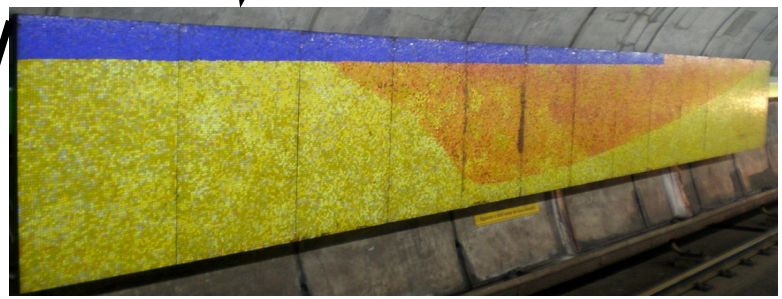


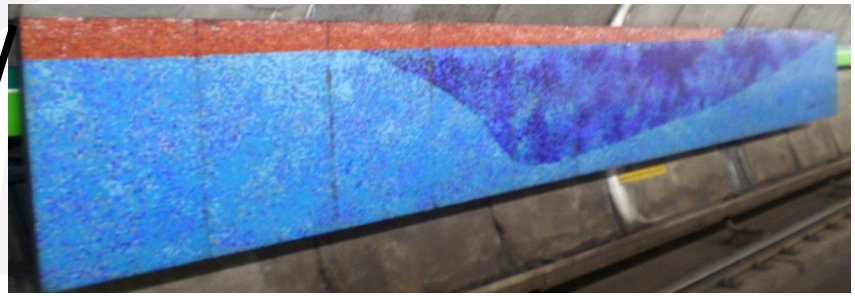
O(1)

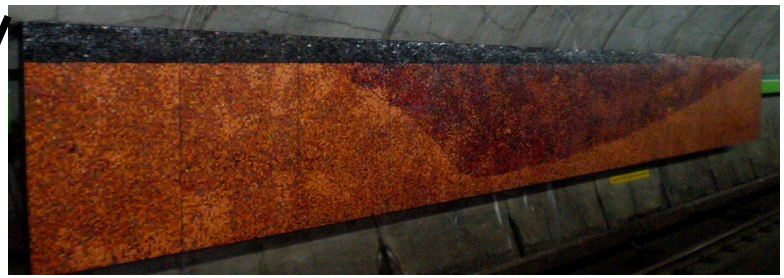
Estação Consolação: “Quatro estações” (1991) Mosaico abstrato feito de pastilhas vitrificadas por Tomie Ohtake

Estação Consolação: “Quatro estações” (1991) Mosaico abstrato feito de pastilhas vitrificadas por Tomie Ohtake









null

Listas com cabeça e sem cabeça

- Uma lista encadeada pode ser vista de 2 maneiras diferentes, dependendo dopapel que sua primeira célula desempenha.

- **Lista com cabeça:**

  A primeira célula serva, apenas, para marcar o início da lista (seu conteúdo é irrelevante)

- **Lista sem cabeça:**

  O conteúdo da primeira célula é tão relevante quanto o das demais.

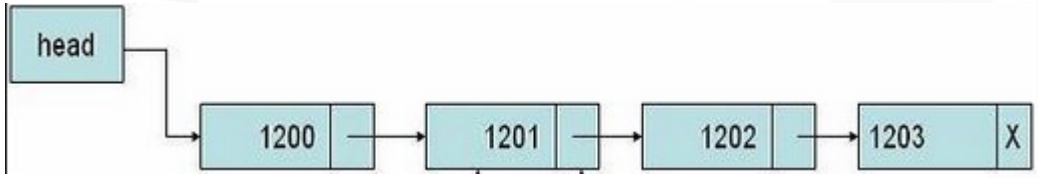
Para criar uma lista vazia

- **Lista com cabeça:**

```
celula *lst;
lst = (celula *) malloc(sizeof(celula));
lst->seg = NULL;
```

- **Lista sem cabeça:**

```
celula *lst;
lst = NULL;
```



Para criar uma lista vazia

- **Lista com cabeça:**

"Mais fáceis de manipular"

```
celula *lst;
lst = (celula *) malloc(sizeof(celula));
lst->seg = NULL;
```

- **Lista sem cabeça:**

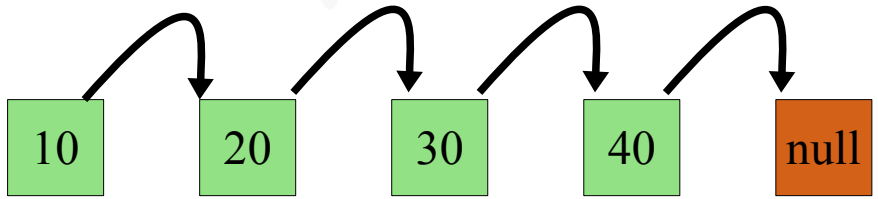
"Mais puras"

```
celula *lst;
lst = NULL;
```

Exercício 01

Elabore um programa que permita:

- Criar uma lista ligada, com cabeça, contendo os seguintes elementos
- Apresente, na tela, o conteúdo de cada celula.

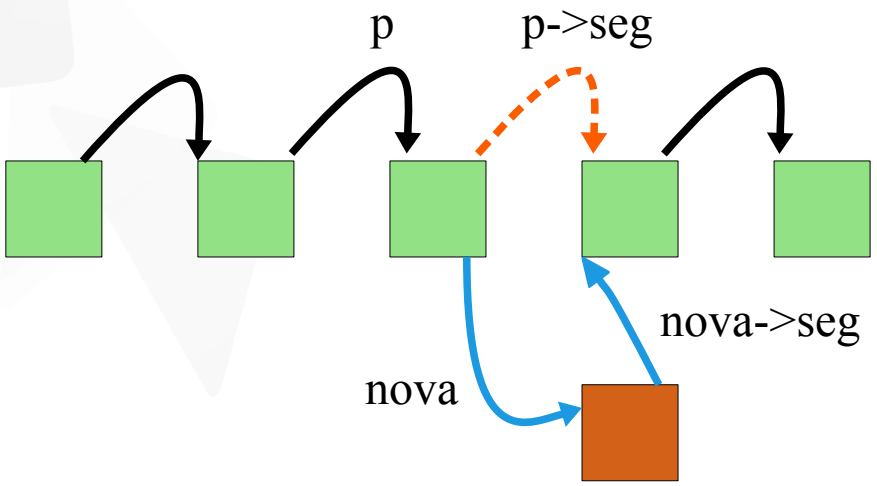



```
struct cel {
    int          conteudo;
    struct cel  *seg;
};

typedef struct cel celula;
```

```
celula *lst;
lst = (celula *) malloc(sizeof(celula));
lst->seg = NULL;
```

Inserção de nova célula

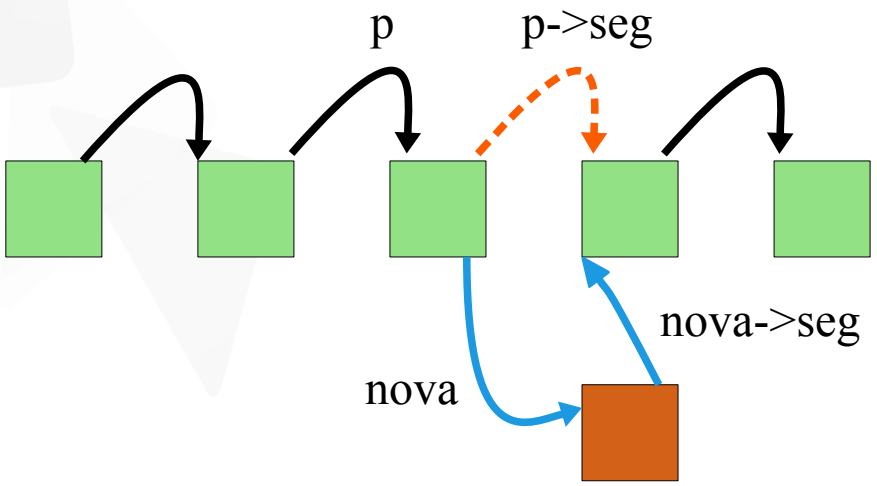
Inserção →
(dado um ponteiro
e um elemento)



O(1)

Inserção de nova célula

Inserção →
(dado um ponteiro
e um elemento)

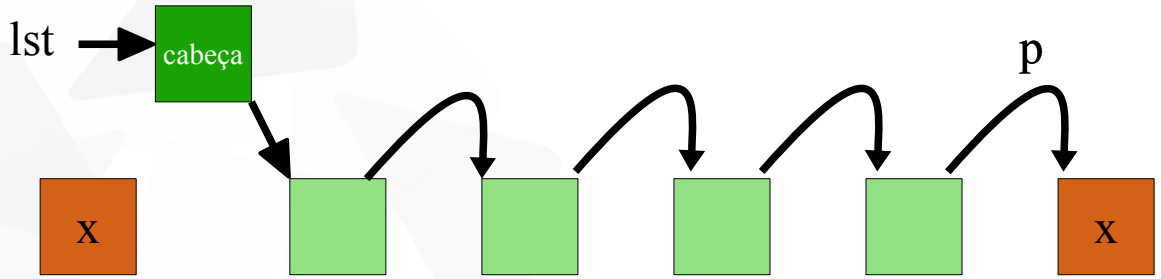


O(1)

```
void Insere (int y, celula *p) {
     celula *nova;
     nova = (celula *) malloc(sizeof(celula));
     nova->conteudo = y;

     nova->seg = p->seg;
     p->seg = nova;
}
```

```c
int main()
{
    celula *lst, *temp;
    lst = (celula *) malloc(sizeof(celula));
    lst->seg = NULL;

    int v[] = {10,20,30,40};
    int i;

    // criacao da lista de celulas
    temp = lst;
    for (i=0; i<sizeof(v)/sizeof(v[0]); i++) {
        Insere (v[i], temp);
        temp = temp->seg;
    }

    // visualizacao da lista
    for (temp=lst->seg; temp!=NULL; temp=temp->seg) {
        printf("%d \n", temp->conteudo);
    }
}
```

Busca em uma lista encadeada



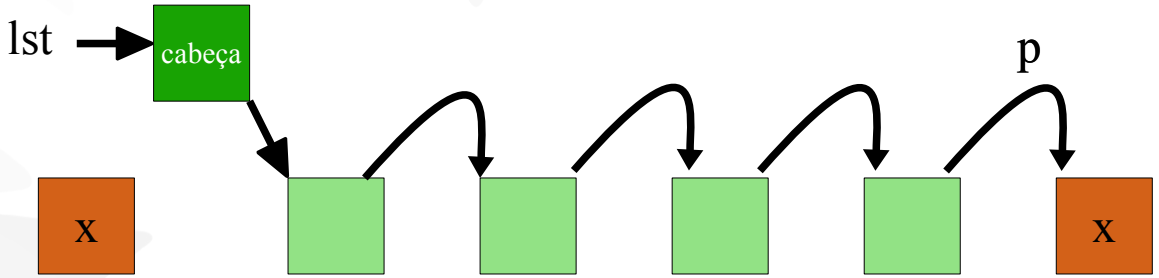
Busca →
(dado um elemento)

O(n)

Exercício 02

Elabore uma função que permita fazer a busca de uma celula que contenha um número inteiro x em uma lista com cabeça.

- Se a célula não existir na lista, devolva NULL.

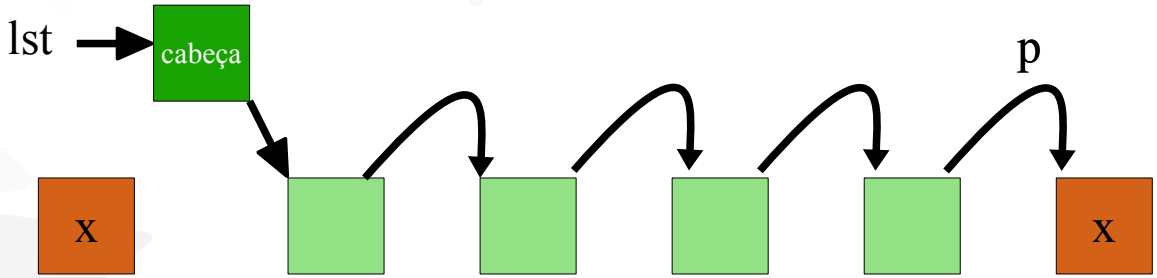



```
celula *Busca(int x, celula *lst)
```

Exercício 02

Elabore uma função que permita fazer a busca de uma celula que contenha um número inteiro x em uma lista com cabeça.

- Se a célula não existir na lista, devolva NULL.

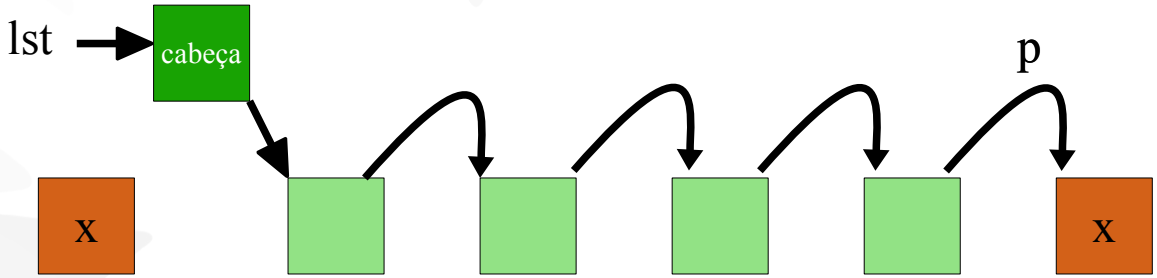



```
celula *Busca(int x, celula *lst) {
    celula *p;
    p = lst->seg;

    while(p!=NULL && p->conteudo!=x) {
        p = p->seg;
    }
    return p;
}
```

Exercício 02

Elabore uma função que permita fazer a busca de uma celula que contenha um número inteiro x em uma lista com cabeça.

- Se a célula não existir na lista, devolva NULL.



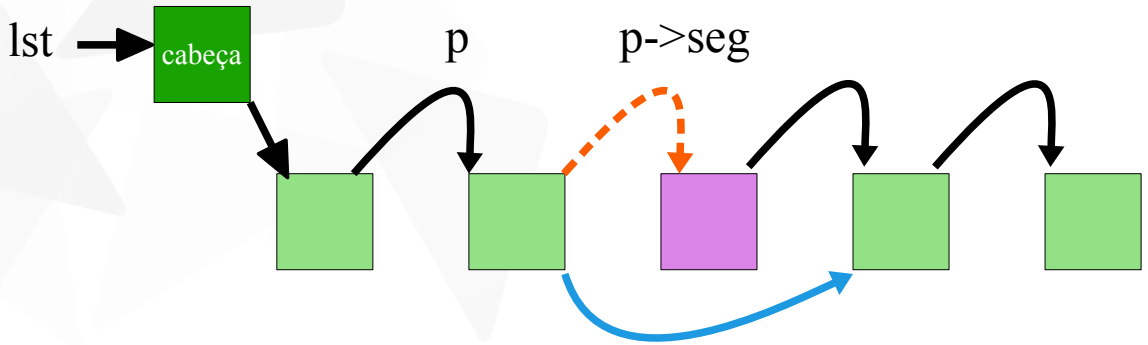

```
celula *Busca(int x, celula *lst) {
    celula *p;
    p = lst->seg;

    while(p!=NULL && p->conteudo!=x) {
        p = p->seg;
    }
    return p;
}
```

Veja a versão recursiva No livro do Prof. Feofiloff

```c
int main()
{
    celula *lst, *temp;
    lst = (celula *) malloc(sizeof(celula));
    lst->seg = NULL;

    int v[] = {10,20,30,40};
    int i;

    // criacao da lista de celulas
    temp = lst;
    for (i=0; i<sizeof(v)/sizeof(v[0]); i++) {
        Insere (v[i], temp);
        temp = temp->seg;
    }

    // visualizacao da lista
    for (temp=lst->seg; temp!=NULL; temp=temp->seg) {
        printf("%d \n", temp->conteudo);
    }

    // teste de busca
    temp = Busca(30, lst);
    if (temp!=NULL)
        printf("%d \n", temp->conteudo);

    temp = Busca(300, lst);
    if (temp!=NULL)
        printf("%d \n", temp->conteudo);
}
```

Remoção em uma lista encadeada

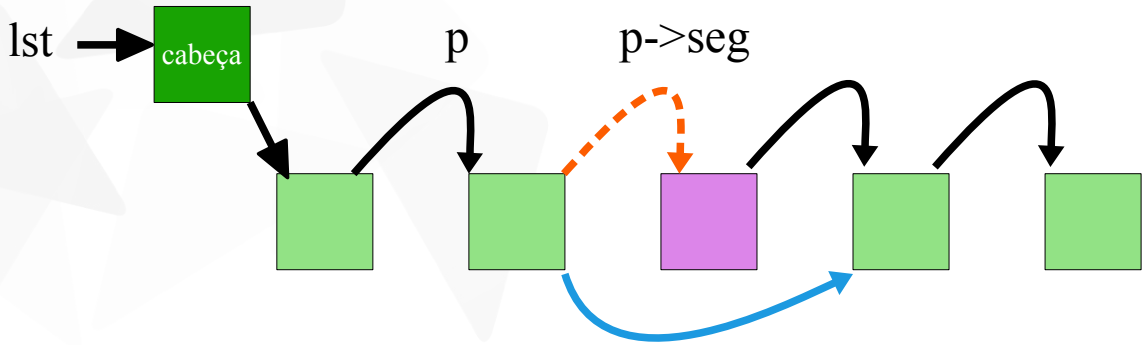
Remoção →
(dado um ponteiro)



O(1)

Remoção em uma lista encadeada

Remoção →
(dado um ponteiro)

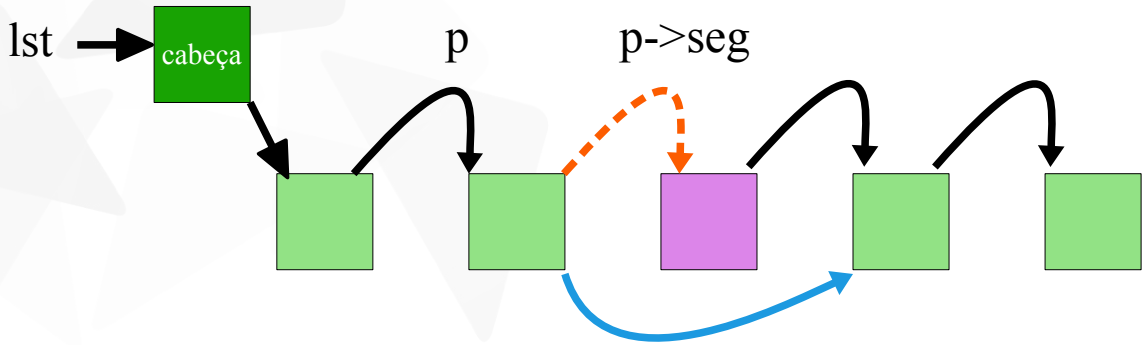


O(1)

```
void *Remove(celula *p) {
    celula *lixo;

    lixo = p->seg;
    p->seg = lixo->seg;

    free(lixo);
}
```

Remoção em uma lista encadeada

Remoção →
(dado um ponteiro)



O(1)

```
void *Remove(celula *p) {
    celula *lixo;

    lixo = p->seg;
    p->seg = lixo->seg;

    free(lixo);
}
```

**Veja a versão BuscaERemove
No livro do Prof. Feofiloff**

Lista 04: https://www.hackerrank.com

- **Cavity Map**
- **Cut the sticks**
- **Sherlock and Planes**

Será utilizado um programa de deteção de plágio em todas as submissões!
Plágio → reprovação

Data: 18/Março (domingo) até às 23h50.

Envio: Através do Tidia.

Arquivos:

Para cada exercício-problema deverá submeter:

- O código fonte: nome do arquivo → RA_nomeDoProblema.c
- O comprovante de aceitação (screenshot) → RA_nomeDoProblema.pdf

Exemplo: 10123456_solveMeFirst**.c**
10123456_solveMeFirst**.pdf**