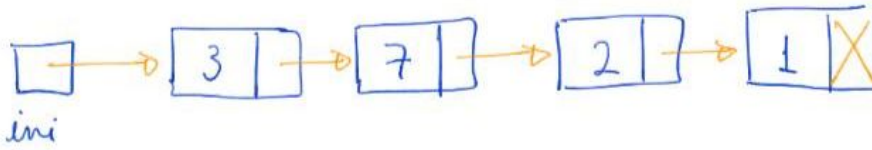


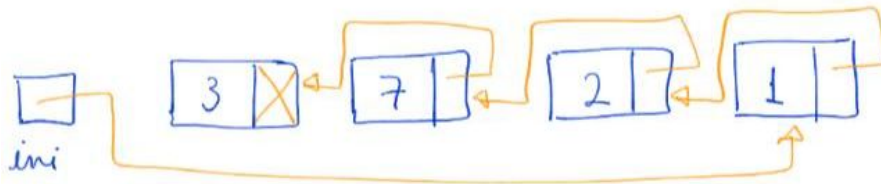
AED1 - Aula 11

Inversão de uma lista, listas encadeadas em vetores

Inversão de uma lista



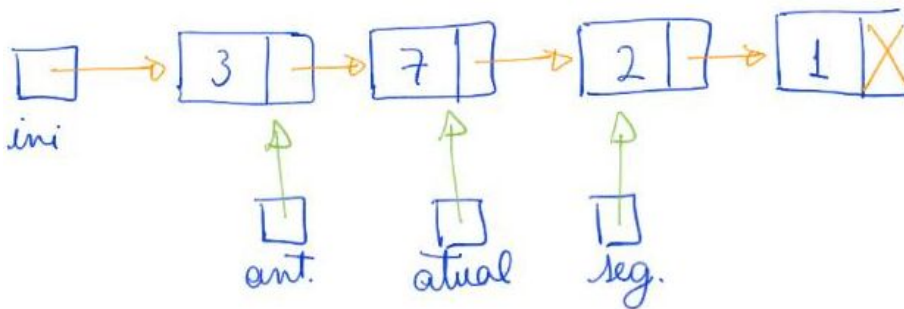
antes da inversão



depois da inversão

Usamos três apontadores:

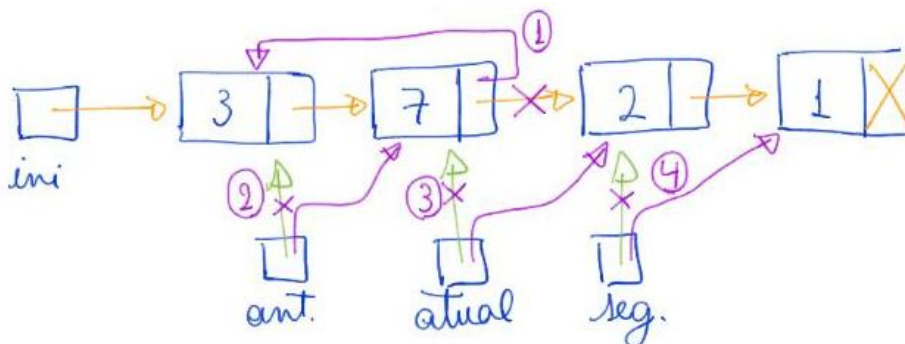
- ant, que aponta para a célula anterior,
- atual, que aponta para a célula corrente,
- seg, que aponta para a célula seguinte.



ideia do algoritmo p/ inversão

Em cada iteração:

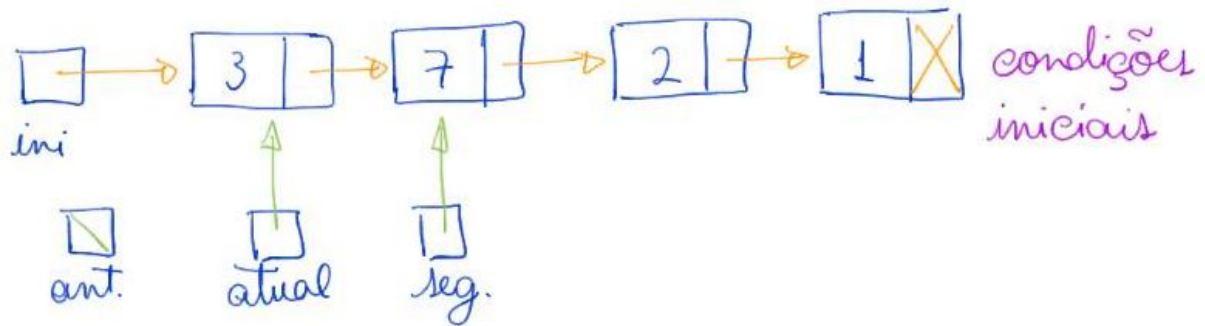
- atual->prox passa a apontar para anterior.



inverte e avança os apontadores

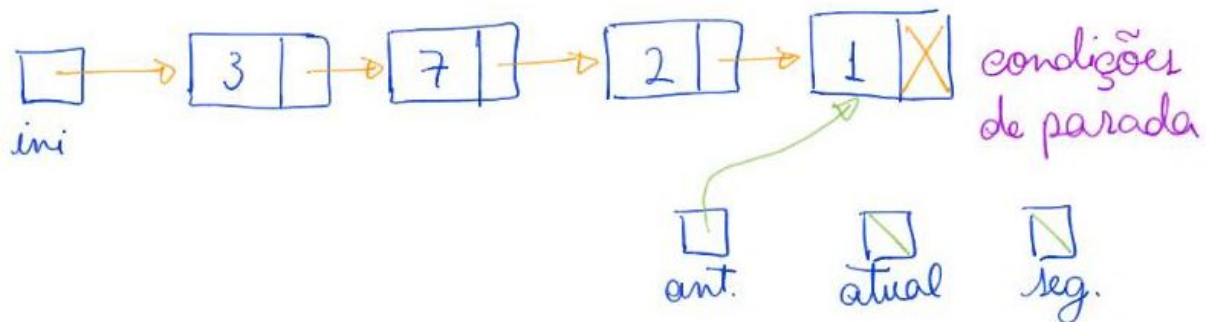
No início, atual aponta para o início da lista

- e anterior aponta para NULL.



Termina o laço quando:

- atual aponta para NULL.



- Caso em que anterior aponta para a última célula,
 - que agora é a primeira.

Sem nó cabeça:

- Versão que devolve o novo apontador para o início da lista.

Celula ***inverta1**(Celula *lst)

```

{
    // ant = anterior, atual = atual, seg = seguinte
    Celula *ant, *atual, *seg;
    ant = NULL; // coloca NULL no novo final da lista
    atual = lst;
    while (atual != NULL)
    {
        seg = atual->prox;
        atual->prox = ant; // momento da inversão
        ant = atual;
        atual = seg;
    }
    return ant;
}

```

- Exemplos de uso:

```

ini = inverta1(ini); // inverte toda a lista
ini->prox = inverta1(ini->prox); // inverte após primeiro
elemento

```

- Versão que usa apontador de apontador.

```

void inverta2(Celula **plst)
{
    // ant = anterior, atual = atual, seg = seguinte
    Celula *ant, *atual, *seg;
    ant = NULL; // coloca NULL no novo final da lista
    atual = *plst;
    while (atual != NULL)
    {
        seg = atual->prox;
        atual->prox = ant; // momento da inversão
        ant = atual;
        atual = seg;
    }
    *plst = ant;
}

```

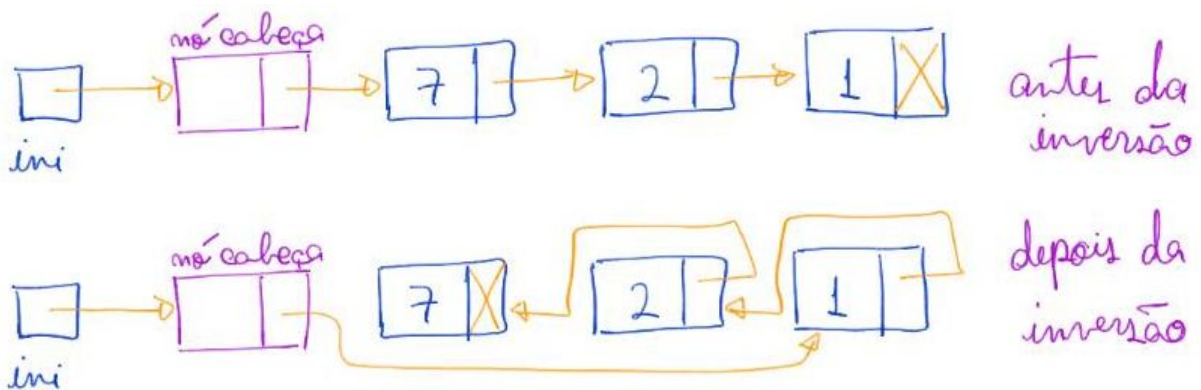
- Exemplos de uso:

```

inverta2(&ini); // inverte toda a lista
inverta2(&ini->prox); // inverte após primeiro elemento

```

Com nó cabeça:



```

void inverta(Celula *lst)
{
    // ant = anterior, atual = atual, seg = seguinte
    Celula *ant, *atual, *seg;

```

```

ant = NULL; // coloca NULL no novo final da lista
atual = lst->prox; // começa depois do nó cabeça
while (atual != NULL)
{
    seg = atual->prox;
    atual->prox = ant; // momento da inversão
    ant = atual;
    atual = seg;
}
lst->prox = ant; // conecta lista ao nó cabeça
}

```

- Exemplos de uso:

```
inverta(ini);
```

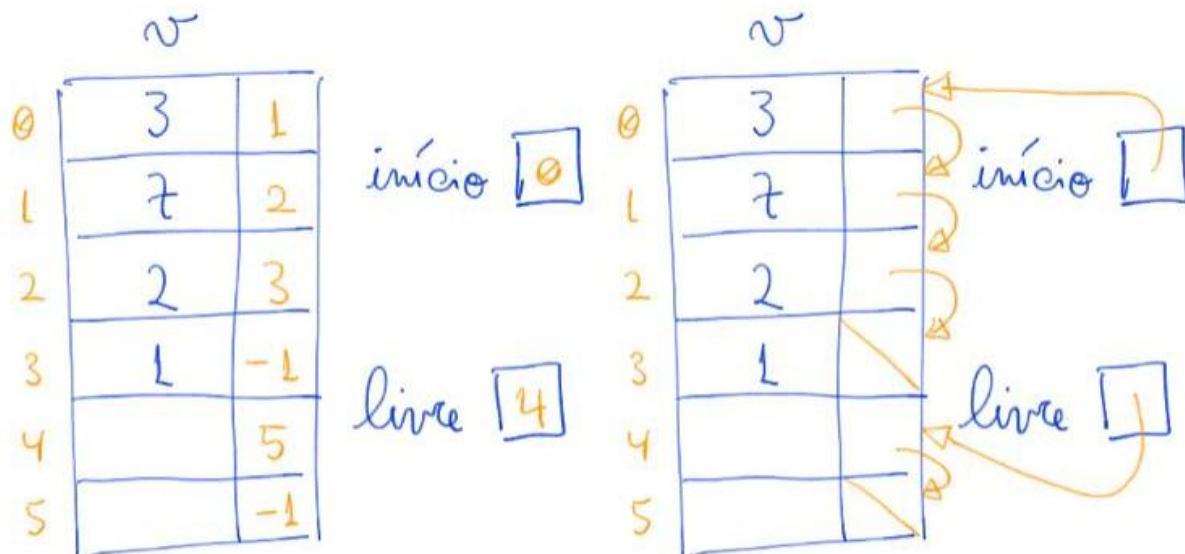
Qual a eficiência de tempo destes algoritmos?

- $O(n)$, sendo n o número de elementos da lista.

Qual a eficiência de espaço destes algoritmos?

- $O(1)$, pois só usa auxiliares que não dependem do tamanho da lista.

Listas encadeadas em vetores



Definições e estruturas:

```

#define MAX 1000
#define NULO -1

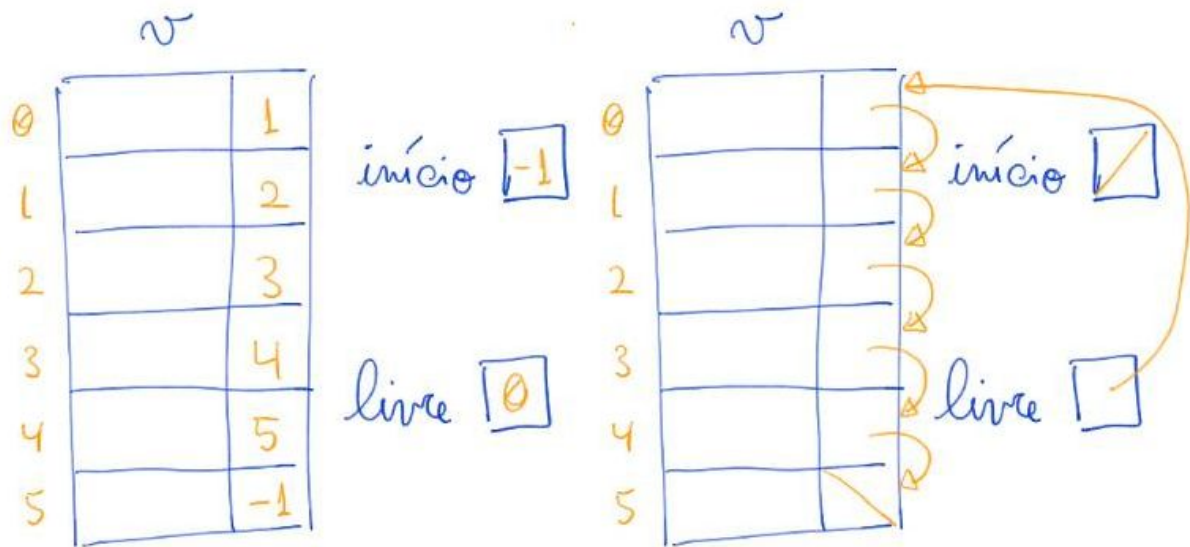
```

```

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

```

Inicialização:



```

Celula v[MAX];
int inicio, livre;

// Inicializa a lista livres com todas as posições
livre = 0;
for (i = 0; i < MAX - 1; i++)
    v[i].prox = i + 1;
v[MAX - 1].prox = NULO;
// e declara a lista de fato vazia
inicio = NULO;

```

- Observe que é necessário manter o controle das posições disponíveis.
 - Para tanto, essas são mantidas na lista "livre".

Impressão:

```

void imprime(Celula v[], int inicio)
{
    int p;

```

```

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

```

- Exemplo de uso:

```

imprime(v, inicio);

```

Busca:

```

int busca(Celula v[], int inicio, int x)
{
    int p;
    for (p = inicio; p != NULO && v[p].conteudo != x; p = v[p].prox)
        ;
    return p;
}

```

- Exemplo de uso:

```

int p = busca(v, inicio, 10);
if (p != NULO)
    printf("%d\n", v[p].conteudo);
else
    printf("nao encontrou\n");

```

Seleção:

```

int selecao(Celula v[], int inicio, int k)
{
    int p, pos;
    for (p = inicio, pos = 0; p != NULO && pos < k; p = v[p].prox,
pos++)
        ;
    return p;
}

```

- Exemplo de uso:

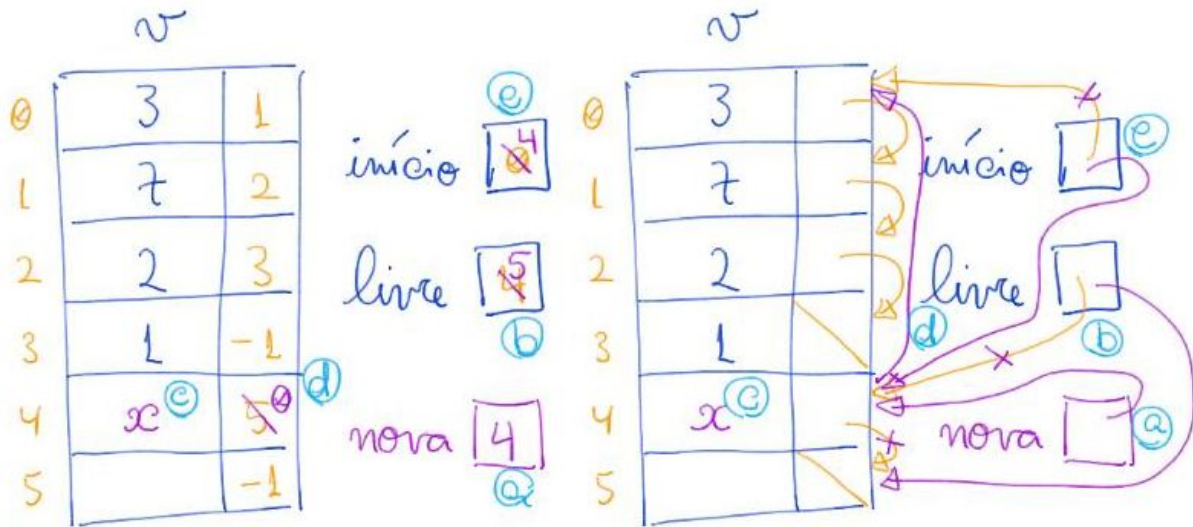
```

int q = selecao(v, inicio, 10);
if (q != NULO)
    printf("%d\n", v[q].conteudo);
else

```

```
printf("nao existe\n");
```

Inserir no início da lista:



```
// insere o elemento x no início da lista
```

```
void insereInicio(Celula v[], int *pinicio, int x, int *plivre)
{
    int nova;
    nova = *plivre;
    *plivre = v[*plivre].prox;
    v[nova].conteudo = x;
    v[nova].prox = *pinicio;
    *pinicio = nova;
}
```

○ Note que, a inserção remove um elemento do início da lista "livre".

● Exemplo de uso:

```
printf("Insere n elementos na lista\n");
for (i = 0; i < n; i++)
    insereInicio(v, &inicio, i, &livre);
imprime(v, inicio);
```

Inserir depois de p:

```
// insere o elemento x entre v[p] e v[p].prox
```

```
void insereDepois(Celula v[], int p, int x, int *plivre)
{
    int nova;
```

```

nova = *plivre;
*plivre = v[*plivre].prox;
v[nova].conteudo = x;
v[nova].prox = v[p].prox;
v[p].prox = nova;
}

```

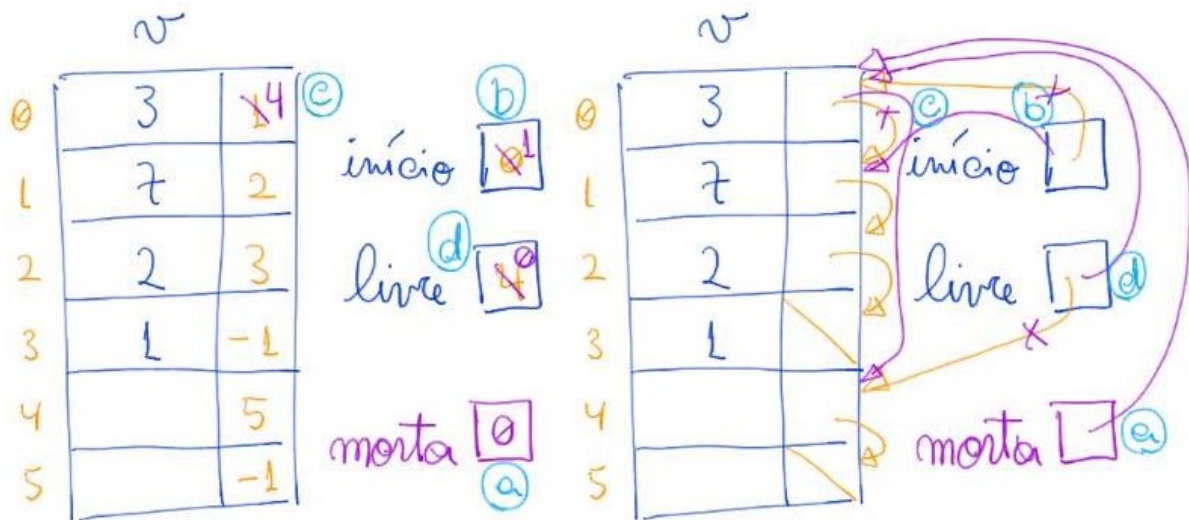
○ Note que, a inserção remove um elemento do início da lista “livre”.

- Exemplos de uso:

```
insereDepois(v, inicio, -77, &livre);
```

```
insereDepois(v, v[inicio].prox, -44, &livre);
```

Remove do início:



```

// remove a celula do inicio
void removeInicio(Celula v[], int *pinicio, int *plivre)
{
    int morta = *pinicio;
    *pinicio = v[morta].prox;
    v[morta].prox = *plivre;
    *plivre = morta;
}

```

○ Note que, a remoção insere um elemento no início da lista “livre”.

- Exemplo de uso:

```
removeInicio(v, &inicio, &livre);
```

Remove o seguinte a p:

```
// remove a celula de índice v[p].prox
```



```

void removeProximo(Celula v[], int p, int *plivre)
{
    int morta = v[p].prox;
    v[p].prox = v[morta].prox;
    v[morta].prox = *plivre;
    *plivre = morta;
}

```

- Note que, a remoção insere um elemento no início da lista “livre”.
- Exemplos de uso:

```

removeProximo(v, inicio, &livre);
removeProximo(v, v[inicio].prox, &livre);

```

Qual a eficiência de tempo destes algoritmos?

- Imprime, busca e seleção são $O(n)$, sendo n o número de elementos da lista.
- As inserções e remoções são $O(1)$.

Qual a eficiência de espaço destes algoritmos?

- $O(1)$, pois só usa auxiliares cujo tamanho total
 - não é proporcional ao tamanho da lista.