

Lista Encadeada

Prof. Marcelo Rosa

Algoritmos e Estrutura de Dados 2 (AE43CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

- 1 Introdução
- 2 Alocação Dinâmica de *Structs*
- 3 Lista Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
 - Exercícios
- 4 Lista Duplamente Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
 - Exercícios
- 5 Lista Circular

1 Introdução

2 Alocação Dinâmica de *Structs*

3 Lista Encadeada

- Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção

- Exercícios

4 Lista Duplamente Encadeada

- Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção

- Exercícios

5 Lista Circular

- Recapitulação sobre listas estáticas
 - Os elementos são armazenados em vetores e são acessíveis individualmente através de índices
 - A alocação é feita de forma estática (bloco contíguo de memória)
 - Implementação: a estrutura contém uma variável para indicar a posição final da lista



- Operações básicas: criar uma lista vazia, inserir, remover, buscar e acessar um item elemento.

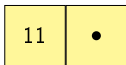
- Vantagens das listas estáticas
 - Simples implementação
- Desvantagens das listas estáticas
 - Custo para retirar itens
 - Caso a lista atinja o limite de armazenamento, não é possível realocar memória
 - Pode ocorrer desperdício de memória se for alocado muito espaço e for utilizado pouco do mesmo
 - Na realidade, não há a operação de remoção (os elementos são geralmente apenas deslocados, mas a cópia do último elemento é mantido)

Introdução

Alternativa: Lista Encadeada

Introdução

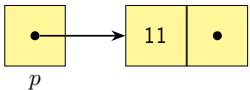
Alternativa: Lista Encadeada



- alocamos memória conforme o necessário

Introdução

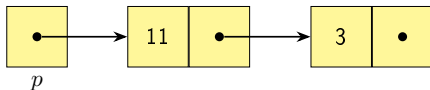
Alternativa: Lista Encadeada



- alocamos memória conforme o necessário
- guardamos um ponteiro para a estrutura em uma variável

Introdução

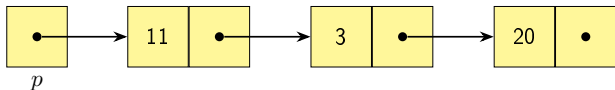
Alternativa: Lista Encadeada



- alocamos memória conforme o necessário
- guardamos um ponteiro para a estrutura em uma variável
- o primeiro **nó** aponta para o segundo

Introdução

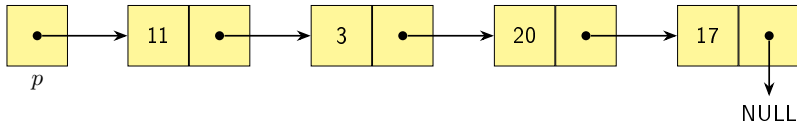
Alternativa: Lista Encadeada



- alocamos memória conforme o necessário
- guardamos um ponteiro para a estrutura em uma variável
- o primeiro **nó** aponta para o segundo
- o segundo nó aponta para o terceiro, e assim por diante

Introdução

Alternativa: Lista Encadeada

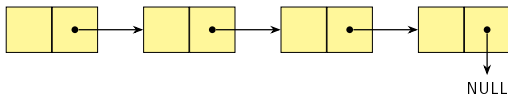


- alocamos memória conforme o necessário
- guardamos um ponteiro para a estrutura em uma variável
- o primeiro **nó** aponta para o segundo
- o segundo nó aponta para o terceiro, e assim por diante
- o último nó aponta para **NULL**

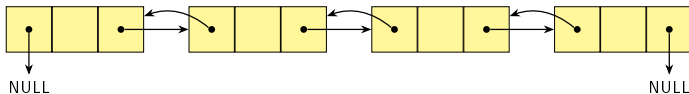
Introdução

Tipos de Listas encadeadas

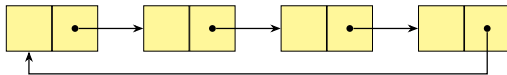
- Simples



- Duplamente



- Circular



- 1 Introdução
- 2 Alocação Dinâmica de *Structs*
- 3 Lista Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
- Exercícios
- 4 Lista Duplamente Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
 - Exercícios
- 5 Lista Circular

- A partir do uso de ponteiros é possível alocar dinamicamente vetores e matrizes de *struct*
- Os comandos seguem o mesmo padrão das variáveis comuns
- A alocação é também feita por meio de funções da *stdlib*:
 - *Malloc()*
 - *free()*

- Exemplo

```
#include <stdlib.h>

typedef struct{
    char nome[120];
    char RG[10];
    char CPF[14];
    Data nasc;
    Endereco end;
}Pessoa;

Pessoa* alocar_vetor(int n){
    Pessoa *p;

    p = (Pessoa*) malloc(n * sizeof(Pessoa));

    return p;
}
```

- 1 Introdução
- 2 Alocação Dinâmica de *Structs*
- 3 Lista Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
- Exercícios
- 4 Lista Duplamente Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
 - Exercícios
- 5 Lista Circular

Nó (ou célula ou node)

Elemento alocado dinamicamente que contém:

- um conjunto de dados
- um ponteiro para outro nó



- **Observação:** um ponteiro estar vazio (aponta para **NULL** em C)

Nó (ou célula ou node)

Elemento alocado dinamicamente que contém:

- um conjunto de dados
- um ponteiro para outro nó



- **Observação:** um ponteiro estar vazio (aponta para **NULL** em C)

Nó apontando para NULL



Nó (ou célula ou node)

Elemento alocado dinamicamente que contém:

- um conjunto de dados
- um ponteiro para outro nó

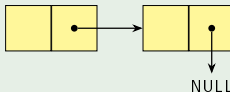
Dados  Ponteiro

- **Observação:** um ponteiro estar vazio (aponta para **NULL** em C)

Nó apontando para NULL



Nó apontando para outro nó



Lista Encadeada

Nó: definição em C

```
1  typedef struct No No;  
2  
3  struct No {  
4      int item;  
5      struct No *next;  
6  };
```

Lista Encadeada

Nó: definição em C

```
1  typedef struct No No;  
2  
3  struct No {  
4      int item;  
5      struct No *next;  
6  };
```

Criação de um nó



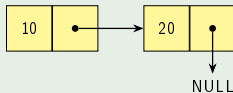
```
7  No *p = (No*) malloc(sizeof(No));  
8  p1->item = 10;  
9  p1->next = NULL;
```

Lista Encadeada

Nó: definição em C

```
1  typedef struct No No;  
2  
3  struct No {  
4      int item;  
5      struct No *next;  
6  };
```

Conectando nós

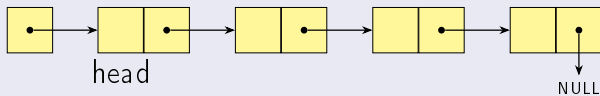


```
10  No *p1 = (No*) malloc(sizeof(No));  
11  No *p2 = (No*) malloc(sizeof(No));  
12  p1->item = 10;  
13  p2->item = 20;  
14  
15  p1->next = p2;  
16  p2->next = NULL;
```

Lista encadeada

Lista encadeada (ou ligada)

- Conjunto de nós ligados entre si de maneira sequencial

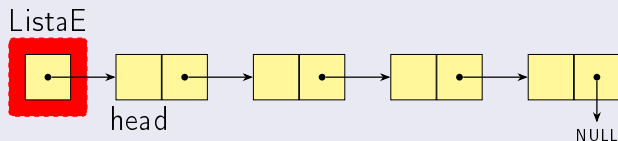


- Em geral, o **primeiro nó** de uma lista encadeada é denominado **cabeça** (*head*)

Lista encadeada

Lista encadeada (ou ligada)

- Conjunto de nós ligados entre si de maneira sequencial



- Em geral, o **primeiro nó** de uma lista encadeada é denominado **cabeça** (*head*)

Criando uma lista

```
27 typedef struct ListaE ListaE;  
28 struct ListaE{  
29     No *head; // Ponteiro para o primeiro elemento da lista  
30 };  
31  
32 ListaE* criar_listaE(){  
33     ListaE* l = (ListaE*) malloc(sizeof(ListaE));  
34     l->head = NULL;  
35     return l;  
36 }
```


Ideia geral

Para encontrar o primeiro elemento com uma chave `key` em uma lista encadeada `l`, em geral, começamos pelo primeiro elemento da lista (`head`) e continuamos percorrendo a lista comparando as chaves de cada elemento com `key` até encontrar uma correspondência ou chegar ao final da lista.

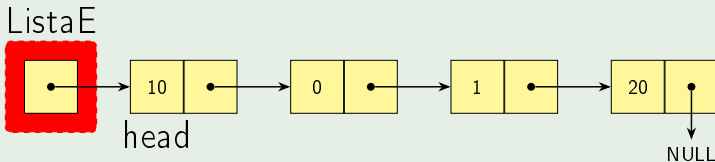
Operação de busca em lista encadeada

Ideia geral

Para encontrar o primeiro elemento com uma chave *key* em uma lista encadeada *l*, em geral, começamos pelo primeiro elemento da lista (*head*) e continuamos percorrendo a lista comparando as chaves de cada elemento com *key* até encontrar uma correspondência ou chegar ao final da lista.

Exemplo

Considere *key* = 20



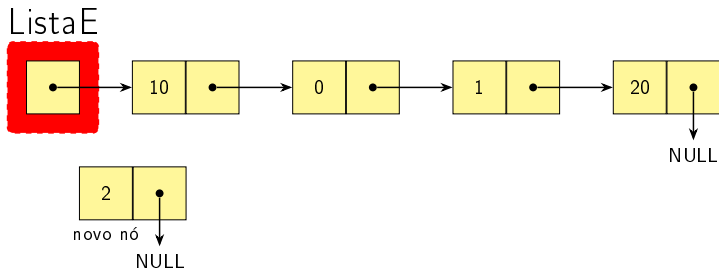
Operação de busca em lista encadeada

Implementação em C

```
37 No* buscar(ListaE *l, int key){
38     No *aux; // Para percorrer a lista, deve ser utilizada
39     // uma variável auxiliar para não perder a
40     // cabeça da lista
41
42     if (l != NULL){
43         aux = l->head;
44
45         // Percorrer a lista encadeada: enquanto a chave não
46         // for encontrada e o valor nulo (NULL) não for
47         // alcançado, percorrer cada nó
48         while (aux != NULL && aux->item != key){
49             aux = aux->next;
50         }
51     }
52     return aux;
53 }
```

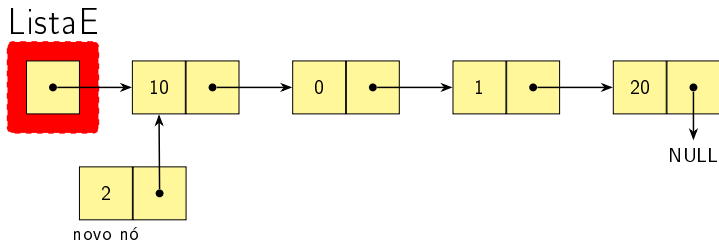
Operação de inserção em lista encadeada

Novo nó no início da lista



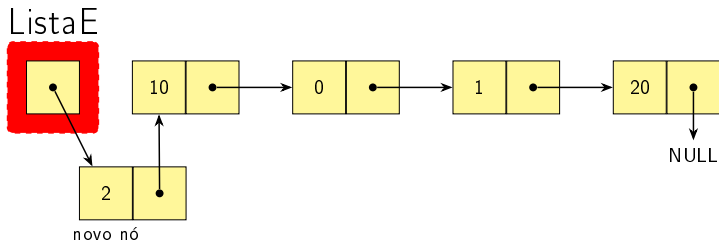
Operação de inserção em lista encadeada

Novo nó no início da lista



Operação de inserção em lista encadeada

Novo nó no início da lista



Operação de inserção em lista encadeada

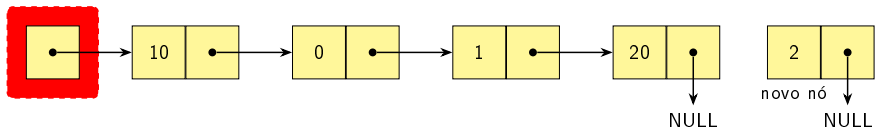
Novo nó no início da lista: implementando em C

```
1 void inserir_primeiro(ListaE *l, int key){
2     No *novo; // Novo nó
3
4     // Caso a lista encadeada seja nula, alocar um espaço para essa estrutura
5     if (l == NULL)
6         l = criar_listaE();
7
8     novo = criar_celula(key); // Criar novo nó
9
10    // Apontar o próximo do novo nó para a cabeça da lista
11    novo->next = l->head;
12
13    l->head = novo; // Atualizar a cabeça da lista para o novo nó
14 }
15
16 No* criar_no(int key){
17     No *novo = (No*) malloc(sizeof(No));
18     novo->item = key;
19     novo->next = NULL;
20     return novo;
21 }
```

Operação de inserção em lista encadeada

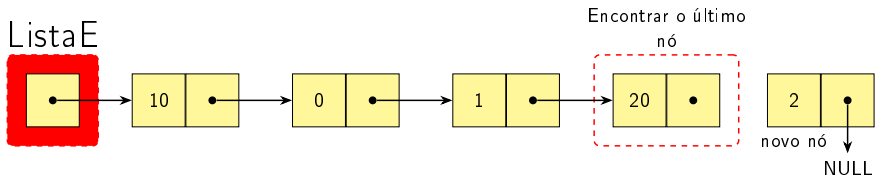
Novo nó no final da lista

ListaE



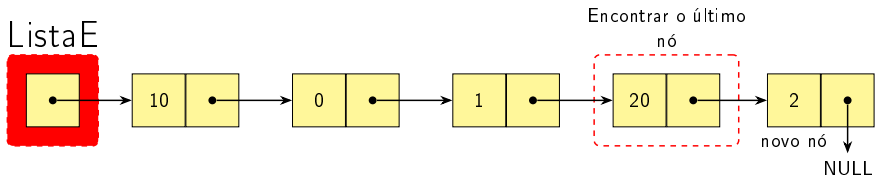
Operação de inserção em lista encadeada

Novo nó no final da lista



Operação de inserção em lista encadeada

Novo nó no final da lista



Operação de inserção em lista encadeada

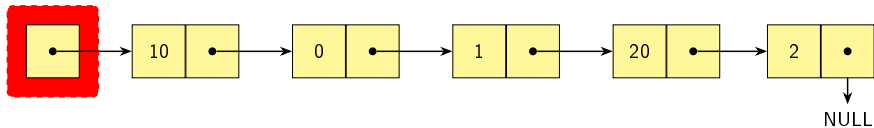
Novo nó no final da lista: implementando em C

```
1 void inserir_ultimo(ListaE *l, int key){
2     No *aux, *novo; // Nó auxiliar e novo, respectivamente
3
4     // Se a lista estiver vazia, não faz sentido percorrê-la
5     if (listaE_vazia(l))
6         inserir_primeiro(l, key);
7     else{
8         aux = l->head;
9         // Percorrer a lista até encontrar o último nó
10        while(aux->next != NULL){
11            aux = aux->next;
12        }
13        novo = criar_celula(key); // Criar novo nó
14        aux->next = novo; // O último elemento da lista aponta para o novo nó
15    }
16 }
17
18 // Retorna 1 se a lista está vazia ou 0, caso contrário
19 int listaE_vazia(ListaE *l){
20     return (l == NULL) || (l->head == NULL);
21 }
```

Operação de remoção em lista encadeada

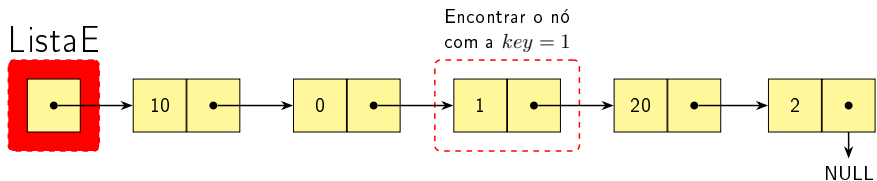
Considere $key = 1$

ListaE



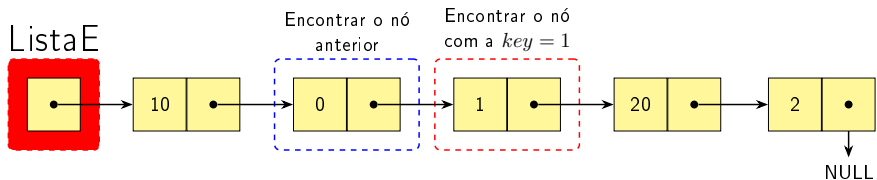
Operação de remoção em lista encadeada

Considere $key = 1$



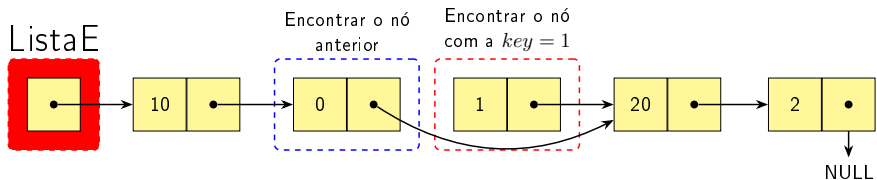
Operação de remoção em lista encadeada

Considere $key = 1$



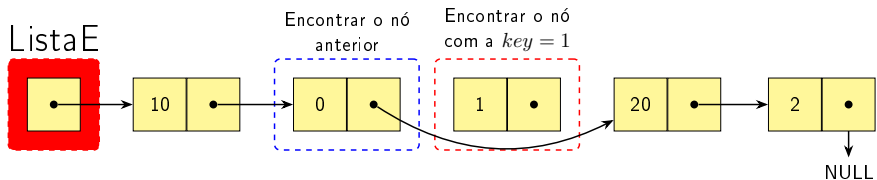
Operação de remoção em lista encadeada

Considere $key = 1$



Operação de remoção em lista encadeada

Considere $key = 1$



Operação de remoção em lista encadeada

```
1  No* remover(ListaE *l, int key){
2      novo *auxA, *auxP = NULL; // nós auxiliares
3      if (!listaE_vazia(l)){
4          auxA = l->head; // apontar o auxA para a cabeça da lista
5          if(auxA->item == key){ // Verificar se o elemento está na cabeça da lista
6              l->head = l->head->next; // Atualizar a cabeça
7          }else{
8              auxP = auxA; // apontar auxP para auxA. Ambos apontam para a cabeça
9              // Procurar a célula que deve ser removida
10             while((auxA != NULL) && (auxA->item != key)){
11                 auxP = auxA; // Guardar o endereço auxA (anterior ao nó procurado)
12                 auxA = auxA->next; // Atualizar auxA
13             }
14         }
15         if (auxA != NULL){
16             // Caso a chave seja encontrada, ou seja, auxA diferente de nulo, fazer o nó
17             //predecessor (auxP caso exista) apontar o ponteiro "next" para o próximo de auxA
18             if(auxP != NULL)
19                 auxP->next = auxA->next;
20             return auxA; // Operação bem-sucedida
21         }
22     }
23     return auxA;
24 }
```

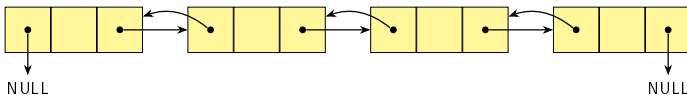
TAD Lista Encadeada Simples

- Exercício 1: para o TAD de lista encadeada apresentado na aula, implemente as seguintes funcionalidades:
 - a) – *void imprimir(ListaE *l)*: imprime a lista
 - b) – *int liberar_LE(ListaE *l)*: Libera a memória alocada para a lista, bem como para nós
 - c) – *void inserir_ordenado(ListaE *l, int k)*: Inserção ordenada de elementos de forma crescente
 - d) – *No* remover_primeiro(ListaE *l)*: Remoção no início
 - e) – *No* remover_ultimo(ListaE *l)*: Remoção no fim
 - f) – *void split(List *l1, List *l2, List *l3)*: a lista *l1* é dividida em duas listas (*l2* e *l3*)
 - g) – *void concatenate(List *l1, List *l2)*: a lista *l2* deve ser concatenada em *l1*
 - h) – *List* merge(List *l1, List *l2)*: as listas *l1* e *l2* devem ser intercaladas em uma nova

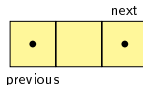
- 1 Introdução
- 2 Alocação Dinâmica de *Structs*
- 3 Lista Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
- Exercícios
- 4 Lista Duplamente Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
 - Exercícios
- 5 Lista Circular

Listas Encadeadas

Lista duplamente encadeada



- Cada nó (célula) possui dois ponteiros
 - Um que aponta para o nó antecessor (*previous*) e outro que aponta para o nó sucessor (*next*)



- Se não for uma lista circular
 - O ponteiro *previous* do primeiro nó é apontado para NULL
 - O ponteiro *next* do último nó é apontado para NULL
- Uma lista duplamente encadeada pode ser circular

Listas Encadeadas

Lista duplamente encadeada

- A principal vantagem da lista encadeada em relação à simples é a facilidade de navegação, que pode ser feita em ambas direções
 - As operações de inserção e remoção são facilitadas (diminui a quantidade de variáveis auxiliares necessárias)
- Se não existir a necessidade de percorrer a lista de trás para frente, a lista encadeada simples é mais vantajosa (devido à simplicidade e também economiza espaços na memória)

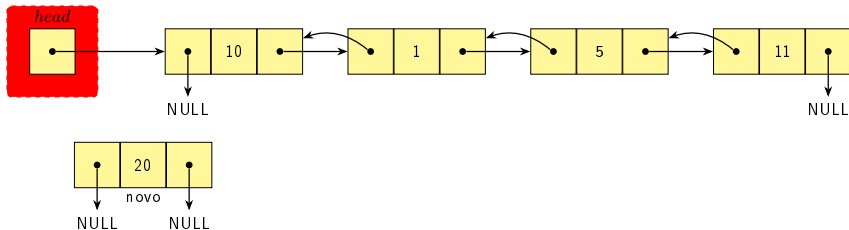
Operação de busca em lista duplamente encadeada

- A operação de busca de um elemento em uma lista duplamente encadeada segue da mesma forma que em lista encadeada simples.

Operação de inserção em lista duplamente encadeada

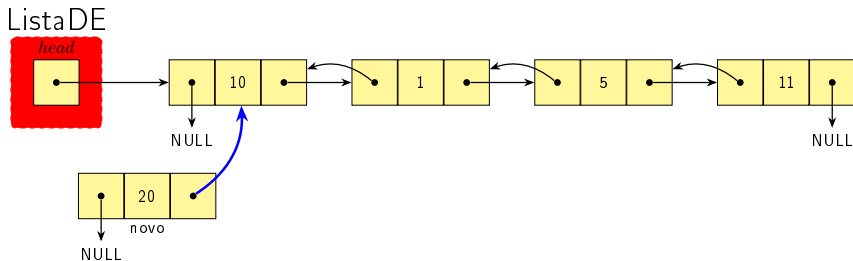
Novo nó no início da lista

ListaDE



Operação de inserção em lista duplamente encadeada

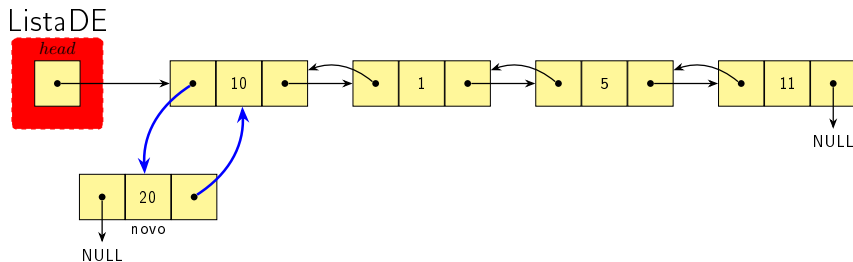
Novo nó no início da lista



- 1 Fazemos o próximo do novo nó (`novo->next`) apontar para a cabeça da lista (`listaDE->head`)

Operação de inserção em lista duplamente encadeada

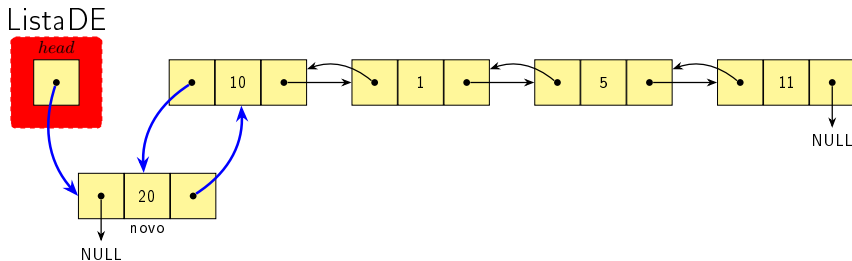
Novo nó no início da lista



- 1 Fazemos o próximo do novo nó (`novo->next`) apontar para a cabeça da lista (`listaDE->head`)
- 2 (Caso a cabeça seja não vazia,) fazemos o antecessor da cabeça (`listaDE->head->previous`) apontar para o novo nó.

Operação de inserção em lista duplamente encadeada

Novo nó no início da lista

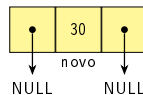
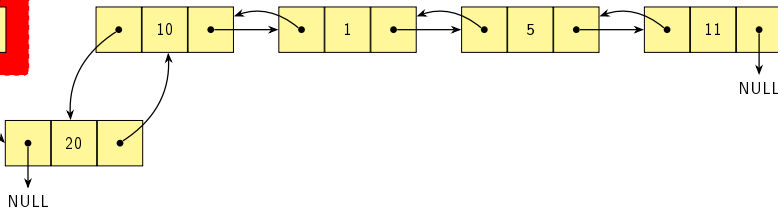


- 1 Fazemos o próximo do novo nó (`novo->next`) apontar para a cabeça da lista (`listaDE->head`)
- 2 (Caso a cabeça seja não vazia,) fazemos o antecessor da cabeça (`listaDE->head->previous`) apontar para o novo nó.
- 3 Por fim, a cabeça da lista (`listaDE->head`) deve apontar para o novo

Operação de inserção em lista duplamente encadeada

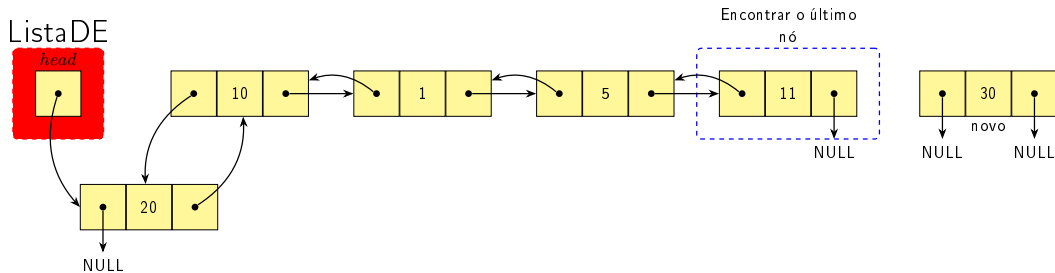
Novo nó no final da lista

ListaDE



Operação de inserção em lista duplamente encadeada

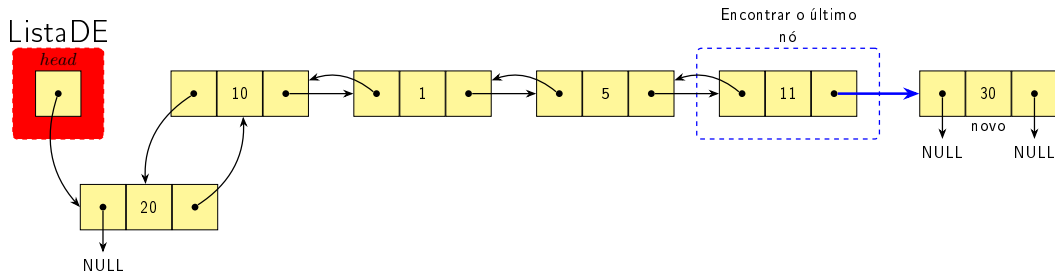
Novo nó no final da lista



- 1 Encontramos o final da lista (fim)

Operação de inserção em lista duplamente encadeada

Novo nó no final da lista

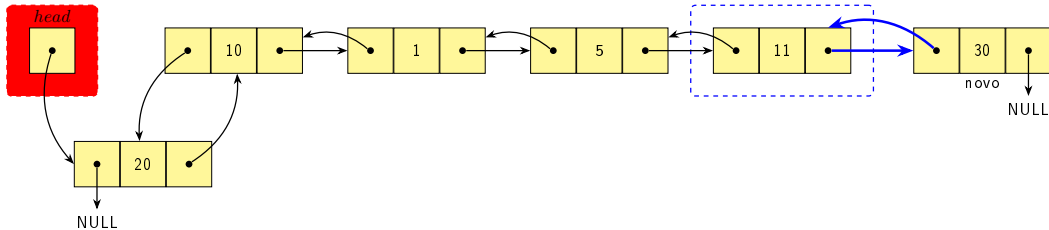


- 1 Encontramos o final da lista (fim)
- 2 fazemos o próximo do fim (fim->next) apontar para o novo nó.

Operação de inserção em lista duplamente encadeada

Novo nó no final da lista

ListaDE

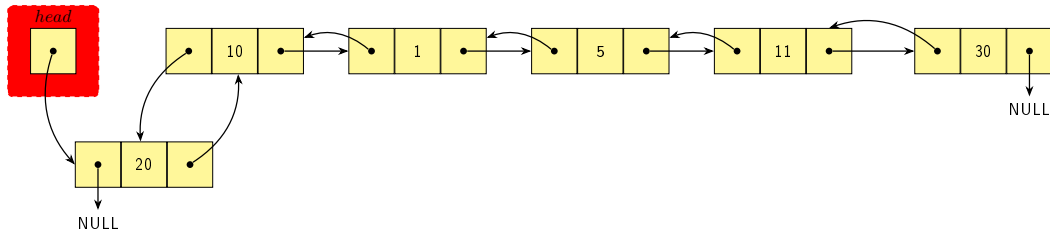


- 1 Encontramos o final da lista (fim)
- 2 fazemos o próximo do fim (fim->next) apontar para o novo nó.
- 3 Por fim, tornamos o nó (fim) o antecessor do novo nó

Operação de remoção em lista duplamente encadeada

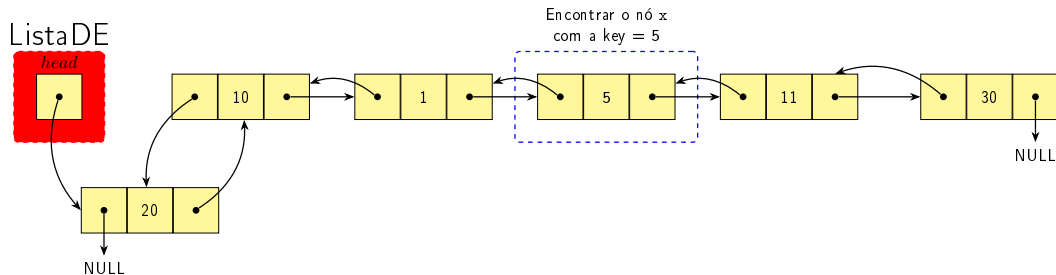
Considere a $key = 5$

ListaDE



Operação de remoção em lista duplamente encadeada

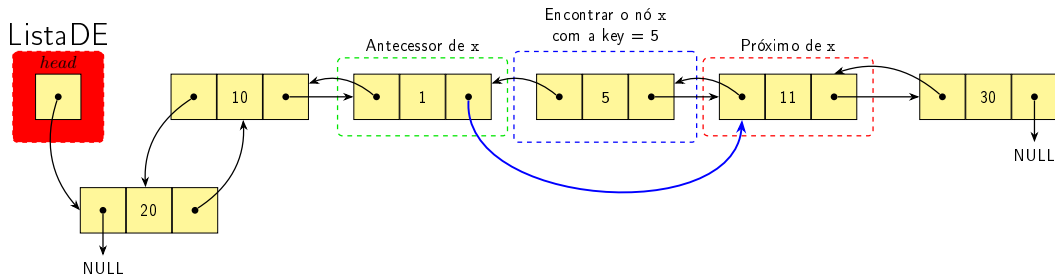
Considere a $key = 5$



- 1 Encontramos o nó (x) com a chave correspondente

Operação de remoção em lista duplamente encadeada

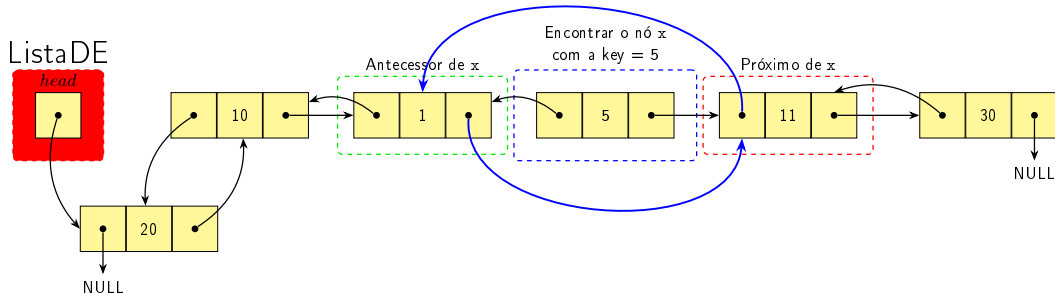
Considere a $key = 5$



- 1 Encontramos o nó (x) com a chave correspondente
- 2 Fazemos o próximo do antecessor de x ($x \rightarrow \text{previous} \rightarrow \text{next}$) ser igual ao próximo de x ($x \rightarrow \text{next}$)

Operação de remoção em lista duplamente encadeada

Considere a $key = 5$



- 1 Encontramos o nó (x) com a chave correspondente
- 2 Fazemos o próximo do antecessor de x ($x \rightarrow \text{previous} \rightarrow \text{next}$) ser igual ao próximo de x ($x \rightarrow \text{next}$)
- 3 Por fim, fazemos o antecessor do próximo de x ($x \rightarrow \text{next} \rightarrow \text{previous}$) ser igual ao antecessor de x ($x \rightarrow \text{previous}$)

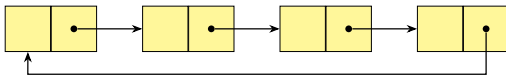
Para o TAD de lista duplamente encadeada apresentada em aula implemente as seguintes funcionalidades

- ❶ NoDE* criar_no(int key): criar um nó
- ❷ ListaDE* criar_ListaDE(): criar uma lista vazia
- ❸ int ListaDE_vazia(ListaDE *l): retornar 1 se a lista está vazia ou 0, caso contrário
- ❹ NoDE* procurar(ListaDE *l, int key): Retornar o ponteiro do nó caso exista, caso contrário retornar NULL
- ❺ void inserir_primeiro(ListaDE *l, int key): inserir um novo nó no início da lista
- ❻ void inserir_ultimo(ListaDE *l, int key): inserir novo nó no fim da lista
- ❼ NoDE* remover(ListaDE *l, int key): remove e retorna o primeiro nó da lista cujo o item é igual a key; caso contrário retorna NULL
- ❽ NoDE* remover_inicio(ListaDE *l): remove e retorna o primeiro nó (ou cabeça) da lista; caso contrário retorna NULL
- ❾ NoDE* remover_ultimo(ListaDE *l): remove e retorna o último nó (ou calda)

- 1 Introdução
- 2 Alocação Dinâmica de *Structs*
- 3 Lista Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
- Exercícios
- 4 Lista Duplamente Encadeada
 - Operações básicas
 - Operação de busca
 - Operação de inserção
 - Operação de remoção
 - Exercícios
- 5 Lista Circular





Listas Encadeadas

Lista circular



- Na lista é formado um ciclo
 - O último elemento aponta para o primeiro em uma lista encadeada circular simples
 - Em uma lista duplamente encadeada circular, além do caso anterior, o primeiro elemento também aponta para o último
- Apesar de não existir, na realidade, primeiro ou último elemento, ainda é necessária a existência de um ponteiro para algum elemento
- A lista circular pode ser útil quando:
 - A busca pode ser feita a partir de qualquer elemento
 - Não há ordenação na lista

- Vantagens?
 - Não é necessária a definição do tamanho máximo da lista
 - Em operações de inclusão e remoção não é necessário o deslocamento de outras células
- Desvantagens?
 - Pode ser necessário percorrer a lista inteira (mesmo se tiver ordenada) para encontrar um item
 - Cada elemento utiliza maior quantidade de memória, pois em cada célula deve ter pelo menos um ponteiro (se for encadeada simples) para apontar para outra célula

-  Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Clifford, S.
Algoritmos: teoria e prática.
Elsevier, 2012.
-  Pereira, S. L.
Estrutura de itens e em C: uma abordagem didática.
Saraiva, 2016.
-  Szwarcfiter, J.; Markenzon, L.
Estruturas de itens e Seus Algoritmos.
LTC, 2010.
-  Tenenbaum, A.; Langsam, Y.
Estruturas de itens usando C.
Pearson, 1995.



Ziviani, M.

Projetos de Algoritmos: com implementações em Pascal e C.
Thomson, 2004.