Estrutura de Dados

Aula 6 : Lista Dinâmica Duplamente Encadeada

Prof. MSc. Fausto Sampaio

https://github.com/Fausto14/estrutura_de_dados

Centro Universitário UniFanor - Wyden

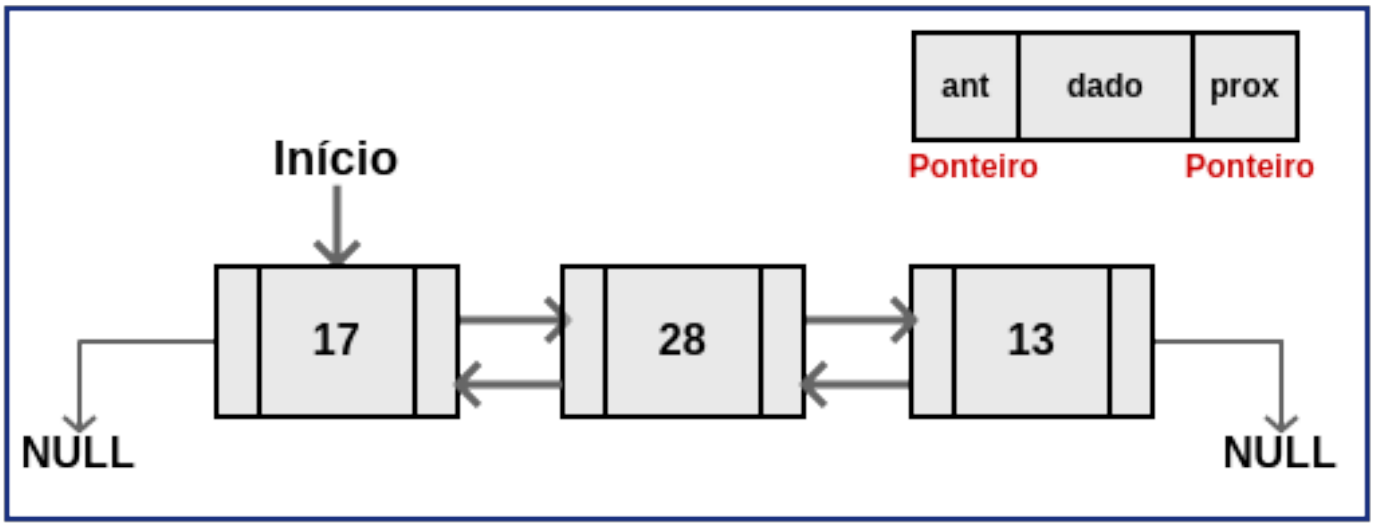
20 de novembro de 2019

Sumário

Lista Dinâmica Duplamente Encadeada

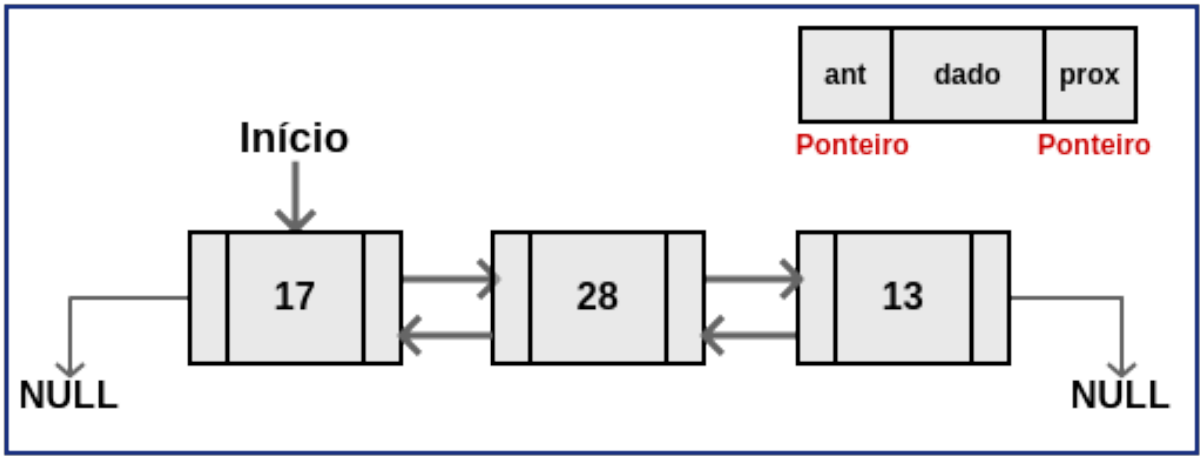
Definição

- **Lista Dinâmica Duplamente Encadeada:** Tipo de lista onde cada elemento aponta para o seu sucessor (prox) e antecessor (ant) na lista;
- Usa um ponteiro especial para o primeiro elemento da lista e uma indicação de final de lista nos dois sentidos;



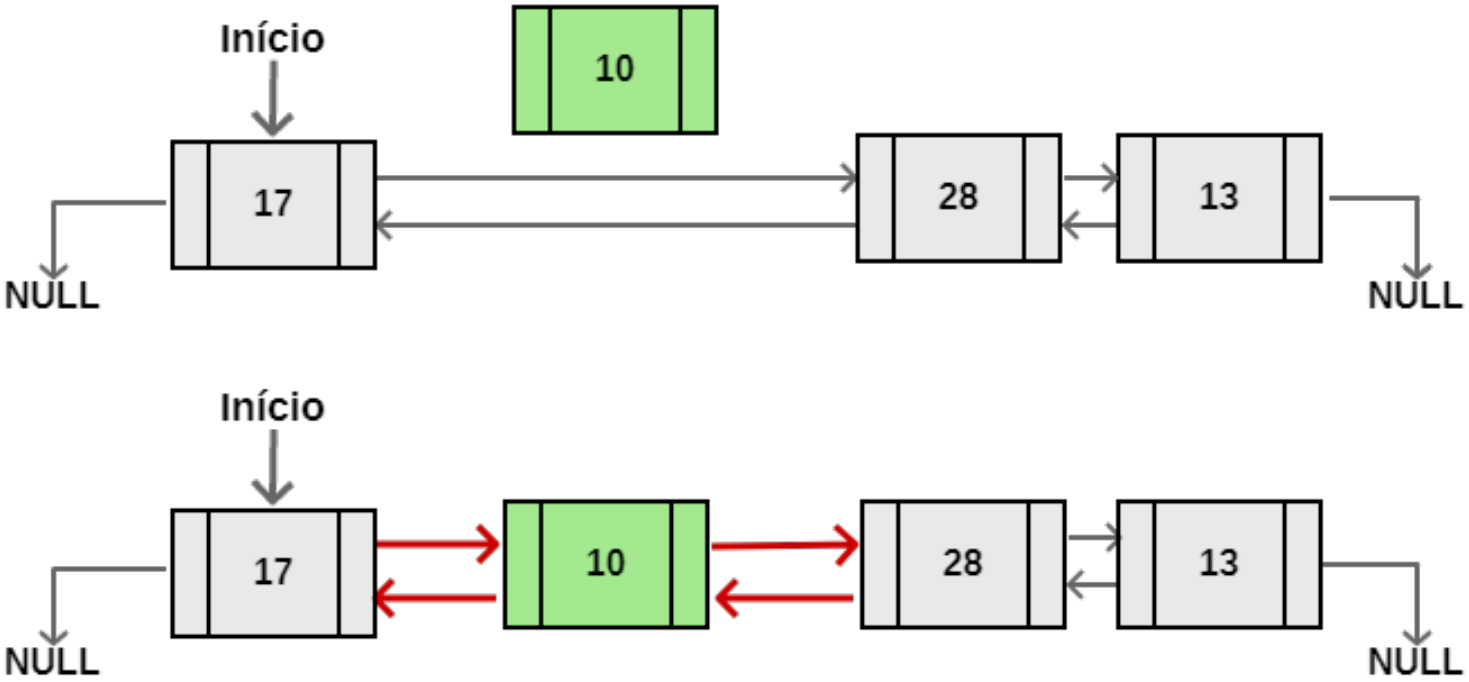
Definição

- Cada elemento é tratado como um ponteiro que é alocado dinamicamente, a medida que os dados são inseridos;
- Para guardar o primeiro elemento, utilizamos um **ponteiro para ponteiro**;
- Um **ponteiro para ponteiro** pode guardar o endereço de um **ponteiro**;
- Assim, fica fácil mudar quem está no início da lista mudando o **conteudo** do **ponteiro para ponteiro**.



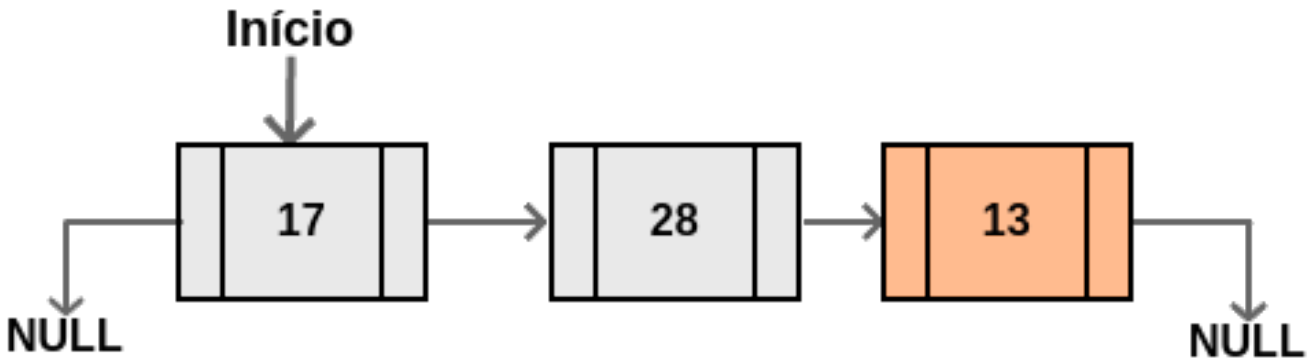
Vantagens

- Melhor utilização dos recursos de memória;
- Não precisa movimentar os elementos nas operações de inserção e remoção;



Desvantagens

- Acesso indireto aos elementos;
- Necessidade para percorrer a lista para acessar um elemento.



Quando utilizar

- Não há ncessidade de garantir um espaço mínimo para a execução do aplicativo;
- Inserção/Remoção em lista ordenada são as operações mais frequentes;
- Necessidade de acessar informação de um elemento antecessor.

Implementação em C

ListaDinEncadDupla.h

- os protótipos das funções;
- o tipo de dado armazenado na lista;
- o ponteiro "lista".

ListaDinEncadDupla.c

- o tipo de dado "lista";
- implementar as suas funções.

Protótipos

ListaDinEncadDupla.h

```
1  //Arquivo ListaDinEncadDupla.h
2  struct aluno{
3      int matricula;
4      char nome[30];
5      float n1,n2,n3;
6  };
7
8  typedef struct elemento* Lista;
9
10 Lista* cria_lista();
11 void libera_lista(Lista* li);
12 int tamanho_lista(Lista* li);
13 int lista_vazia(Lista* li);
14 int insere_lista_final(Lista* li, struct aluno al);
15 int insere_lista_inicio(Lista* li, struct aluno al);
16 int insere_lista_ordenada(Lista* li, struct aluno al);
17 int remove_lista(Lista* li, int mat);
18 int remove_lista_inicio(Lista* li);
19 int remove_lista_final(Lista* li);
20 int consulta_lista_pos(Lista* li, int pos, struct aluno *al);
21 int consulta_lista_mat(Lista* li, int mat, struct aluno *al);
22 void imprime_lista(Lista* li);
```

Definição

ListaDinEncadDupla.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "ListaDinEncadDupla.h" //inclui os Protótipos

//Definição do tipo lista
struct elemento{
    struct elemento *ant;
    struct aluno dados;
    struct elemento *prox;
};
//facilitar as manipulações dentro das implementações
typedef struct elemento Elem;
```

Criar Lista

Procedimentos

- Alocar memória para um ponteiro tipo Lista;
- Fazer o ponteiro que indica o início da lista, apontar para NULL.

ListaDinEncadDupla.c

```
14 Lista* cria_lista(){
15     Lista* li = (Lista*) malloc(sizeof(Lista));
16     if(li != NULL)
17         *li = NULL;
18     return li;
19 }
```

Liberar Lista

Procedimentos

- Percorrer toda a lista;
- Liberar memória referente a cada elemento da lista;
- Liberar memória referente à estrutura geral da lista;

ListaDinEncadDupla.c

```
21 void libera_lista(Lista* li){
22     if(li != NULL){
23         Elem* no;
24         while((*li) != NULL){
25             no = *li;
26             *li = (*li)->prox;
27             free(no);
28         }
29         free(li);
30     }
31 }
```

Tamanho da Lista

Procedimentos

- Percorrer toda a lista;
- Para cada elemento acessado incrementar o contador em 1;
- retornar o valor do contador;

ListaDinEncadDupla.c

```
199 int tamanho_lista(Lista* li){
200     if(li == NULL)
201         return 0;
202     int cont = 0;
203     Elem* no = *li;
204     while(no != NULL){
205         cont++;
206         no = no->prox;
207     }
208     return cont;
209 }
```

Lista Cheia

Procedimentos

- Retornar false (0);

Observação

- Na implementação dinâmica, a lista nunca será cheia, pois o limite da lista é a quantidade de memória do computador;

ListaDinEncadDupla.c

```
211 int lista_cheia(Lista* li){
212     return 0;
213 }
```

Lista Vazia

Procedimentos

- verdadeiro: se a estrutura da lista for NULL ou se o ponteiro de início apontar para NULL;
- caso contrário: falso;

ListaDinEncadDupla.c

```
215 int lista_vazia(Lista* li){
216     if(li == NULL)
217         return 1;
218     if(*li == NULL)
219         return 1;
220     return 0;
221 }
```

Inserir - Início

Procedimentos

- Fazer o ponteiro **prox** do novo elemento apontar para o ínicio da lista;
- Fazer o ponteiro **ant** do novo elemento apontar para NULL;
- Se a lista não for vazia, fazer o ponteiro **ant** do elemento do ínicio da lista apontar para o novo elemento;
- Fazer o ponteiro que indica o início da lista, apontar para o novo elemento.

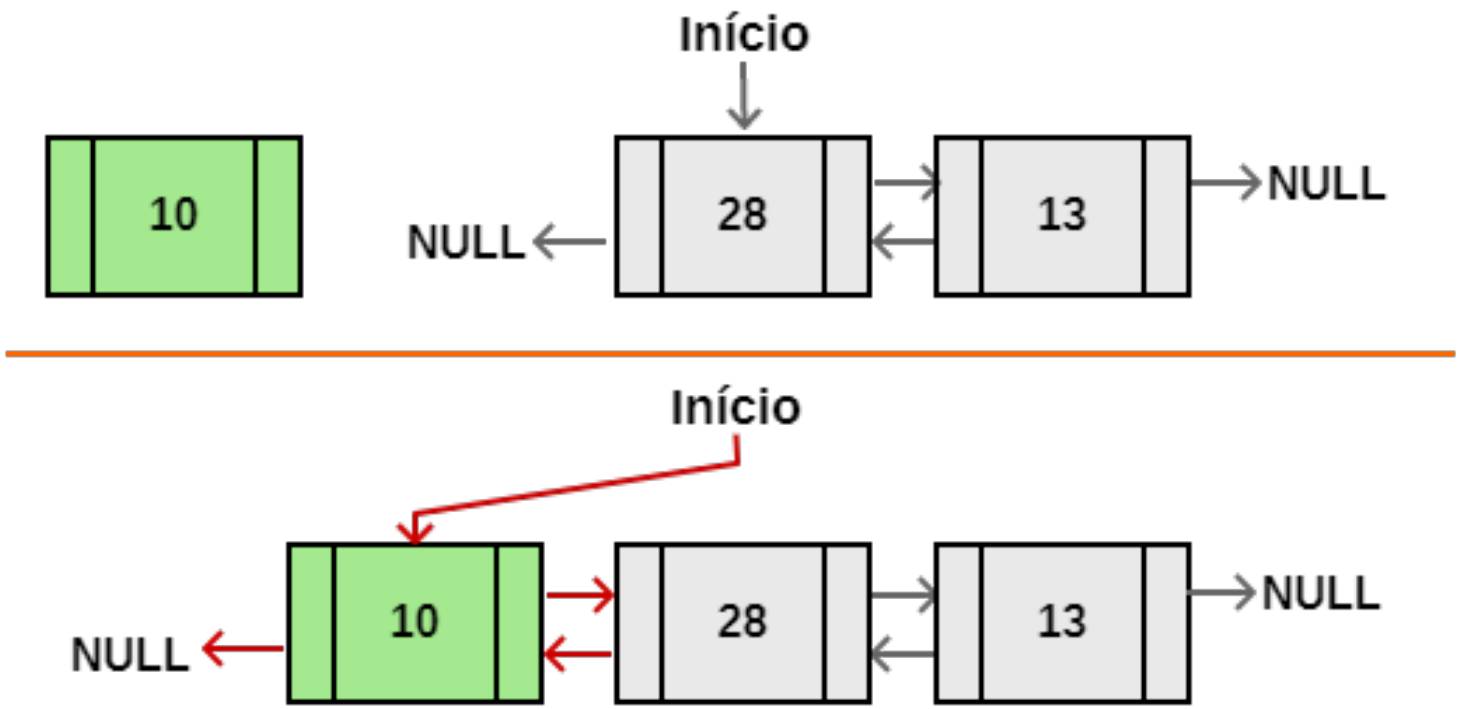
ListaDinEncadDupla.c

```
89  int insere_lista_inicio(Lista* li, struct aluno al){
90      if(li == NULL)
91          return 0;
92      Elem* no;
93      no = (Elem*) malloc(sizeof(Elem));
94      if(no == NULL)
95          return 0;
96      no->dados = al;
97      no->prox = (*li);
98      no->ant = NULL;
99      if(*li != NULL)//lista não vazia: apontar para o anterior!
100         (*li)->ant = no;
101     *li = no;
102     return 1;
103 }
```

Inserir - Início da Lista

Atenção

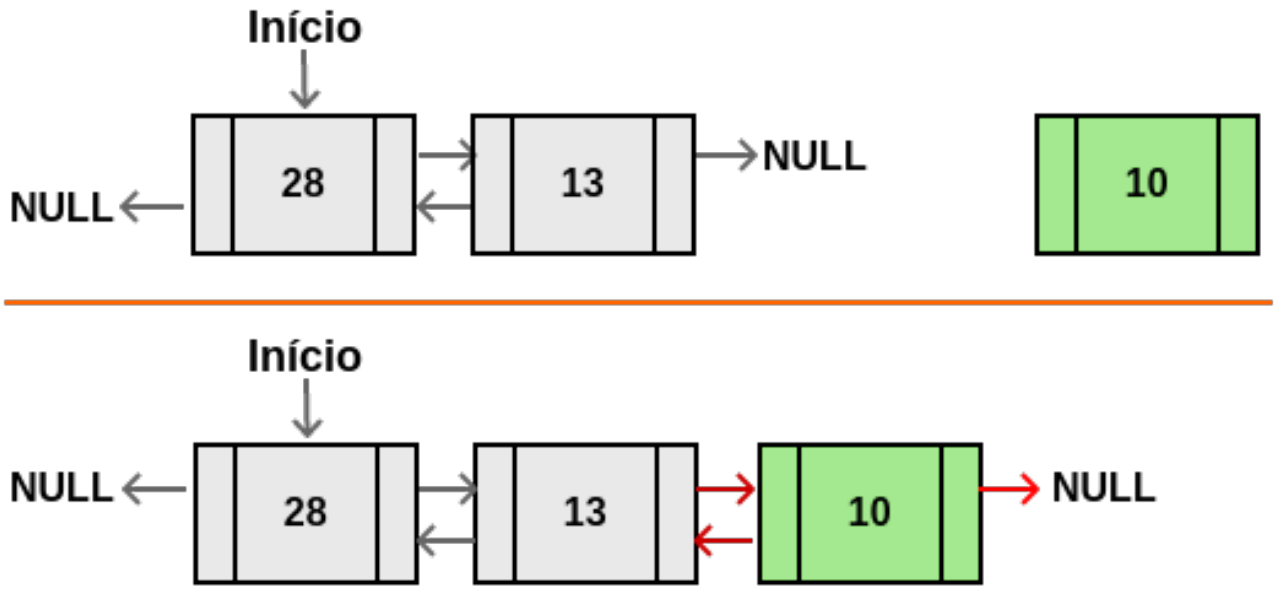
- A inserção é parecida com a da **Lista Dinâmica Encadeada**;
- Deve-se apenas considerar que agora temos dois ponteiros para atualizar: anterior e próximo;



Inserir - Final da Lista

Procedimentos

- Buscar onde inserir: acessar o último elemento da lista;
- Fazer o ponteiro **prox** do último elemento apontar para o novo elemento;
- Fazer o ponteiro **prox** do novo elemento apontar para NULL;
- Fazer o ponteiro **ant** do novo elemento apontar o último elemento;



Inserir - Final da Lista

ListaDinEncadDupla.c

```
65 int insere_lista_final(Lista* li, struct aluno al){
66     if(li == NULL)
67         return 0;
68     Elem *no;
69     no = (Elem*) malloc(sizeof(Elem));
70     if(no == NULL)
71         return 0;
72     no->dados = al;
73     no->prox = NULL;
74     if((*li) == NULL){//lista vazia: insere início
75         no->ant = NULL;
76         *li = no;
77     }else{
78         Elem *aux;
79         aux = *li;
80         while(aux->prox != NULL){
81             aux = aux->prox;
82         }
83         aux->prox = no;
84         no->ant = aux;
85     }
86     return 1;
87 }
```

Remover - Inicio, Meio ou Final da Lista

Procedimentos

- Buscar o elemento a ser removido: usar algum critério de busca;
- Verificar:
  - Se o elemento a ser removido for o primeiro: fazer o ponteiro de início da lista apontar para o **prox** do elemento que está sendo revovido;
  - Se o elemento a ser removido NÃO for o primeiro: no elemento que está sendo revovido, fazer o ponteiro **prox** do **ant** apontar para **prox**;
- Se o elemento a ser removido NÃO for o último: no elemento que está sendo revovido, fazer o ponteiro **ant** do **prox** apontar para **ant**;
- Liberar memória do elemento que está sendo removido: free();

Cuidado

- Não se pode remover de uma lista vazia;
- Removendo o último nó, a lista fica vazia;

Remover - Inicio, Meio ou Final da Lista

ListaDinEncadDupla.c

```
140 int remove_lista(Lista* li, int mat){//TERMINAR
141     if(li == NULL)
142         return 0;
143     if((*li) == NULL)//lista vazia
144         return 0;
145     Elem *no = *li;
146     while(no != NULL && no->dados.matricula != mat){
147         no = no->prox;
148     }
149     if(no == NULL)//não encontrado
150         return 0;
151 
152     if(no->ant == NULL)//remover o primeiro?
153         *li = no->prox;
154     else
155         no->ant->prox = no->prox;
156 
157     if(no->prox != NULL)//não é o último?
158         no->prox->ant = no->ant;
159 
160     free(no);
161     return 1;
162 }
```

Consultar na Lista

- Tipos:
  - Consulta por posição;
  - Consulta por conteúdo;
- Ambos dependem de busca (percorrer os elementos) até encontrar o desejado;
- Segue os mesmos passos da consulta em Listas Dinâmica Encadeada;

Referências

Referências

- André Ricardo Backes, CAPÍTULO 5 - Listas, Editor(s): André Ricardo Backes, **Estrutura de Dados Descomplicada em Linguagem C**, Elsevier Editora Ltda., 2016, Pages 77-191, ISBN 9788535285239.