SIN 211 - Algoritmos e Estruturas de Dados

(Lista Duplamente Encadeada)

Profº: Joelson Antônio dos Santos

Universidade Federal de Viçosa
Instituto de Ciências Exatas e Tecnológicas
Campus de Rio Paranaíba - MG

joelsonn.santos@gmail.com
Sala: BBT 233

17 de abril de 2018

Aula de Hoje

1. Lista Duplamente Encadeada

Créditos

O material desta aula é composto por adaptações e extensões dos originais gentilmente cedidos pelos professores **Moacir Pereira Ponti** e **Rachel Reis**.

Lista Simplesmente Encadeada

- Cada célula possui basicamente dois campos:
  - Informação (**info**).
  - Ponteiro para a próxima célula (**prox**).

Lista Duplamente Encadeada

- Cada célula possui três campos:
  - Informação (**info**).
  - Ponteiro para a próxima célula (**prox**).
  - Ponteiro para a célula anterior (**ant**).
- Um ponteiro externo é usado para o primeiro elemento da lista[1].

[1] Podem haver implementações que também usam um ponteiro adicional para o último elemento da lista.

Lista Duplamente Encadeada

- Quais as vantagens de se trabalhar com listas encadeadas? E as desvantagens?

- Essas vantagens e desvantagens permanecem para as listas duplamente encadeadas!

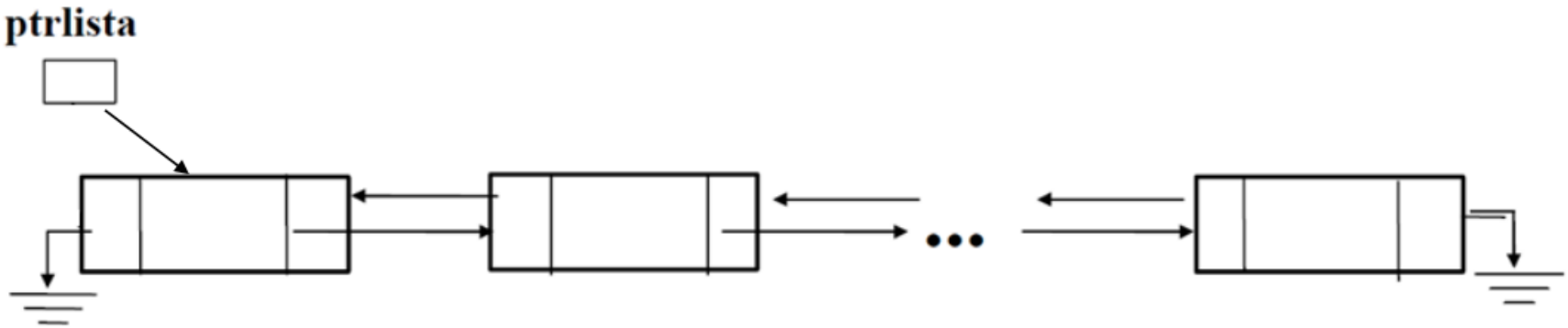
Lista Duplamente Encadeada

- A lista duplamente encadeada pode ser utilizada quando existe uma necessidade de se trabalhar com os elementos anteriores e posteriores de uma informação.

- Já fazíamos isso usando um ponteiro *anteriorAuxiliar* em algumas operações.

Lista Duplamente Encadeada

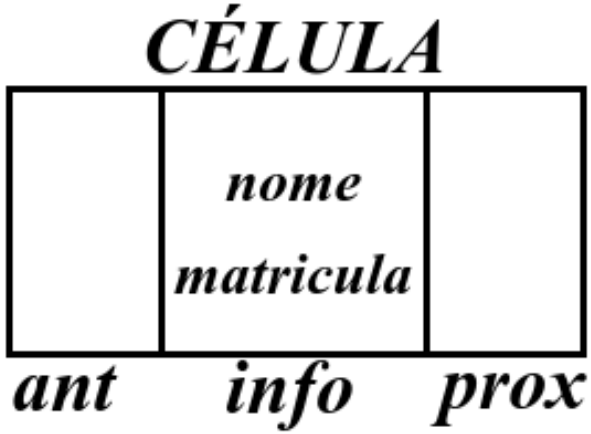
- Vantagens:
  - A partir de uma célula, podemos acessar as células adjacentes (anterior e posterior).
  - Permite que a lista seja percorrida em ambas as direções.
- Desvantagem: requer mais memória.



Lista Duplamente Encadeada

```c
typedef struct sPessoa{
    char nome[50];
    int matricula;
}PESSOA;

typedef struct sCell{
    PESSOA info;
    struct sCell* prox;
    struct sCell* ant;
}CELULA;
```



Lista Duplamente Encadeada

- Operações comuns para esta lista:
  - Criação/Declaração
  - Inicialização
  - Inserção
  - Percurso (mostrar lista)
  - Substituição de valor
  - Remoção
  - Busca ou pesquisa

Lista Duplamente Encadeada - Criação/Declaração

- Exemplo de declaração de um ponteiro externo para a lista na função principal:

```
int main(){

    CELULA* ptrLista;
```

Lista Duplamente Encadeada - Inicialização

```
void inicializarLista(CELULA** lista){
    (*lista) = NULL;
}
```

Lista Duplamente Encadeada - Inserir Início

```
int inserirInicio(CELULA** lista, PESSOA elemento){
    CELULA* p = criarCelula();

    if(p == NULL){
        printf("\n Erro de alocacao de memoria");
        return 0;
    }
    p->info = elemento;
    p->prox = NULL;
    p->ant = NULL;
```

Lista Duplamente Encadeada - Inserir Início (continuação)

```
    if(listaVazia(lista)){
        (*lista) = p;
        return 1;
    }
    p->prox = (*lista);
    (*lista)->ant = p;
    (*lista) = p;
  return 1;
}
```

Lista Duplamente Encadeada - Inserir Fim

```
int inserirFim(CELULA** lista, PESSOA elemento){
    CELULA* p = criarCelula();
    CELULA* aux;
    if(p == NULL){
        printf("\n Erro de alocacao de memoria");
        return 0;
    }
    p->info = elemento;
    p->prox = NULL;
    p->ant = NULL;
```

Lista Duplamente Encadeada - Inserir Fim (continuação)

```
        if(listaVazia(lista)){
            (*lista) = p;
            return 1;
        }
        aux = (*lista);
        while(aux->prox != NULL){
            aux = aux->prox;
        }
        p->ant = aux;
        aux->prox = p;
    return 1;
}
```

Lista Duplamente Encadeada - Remover Início

```
PESSOA removerInicio(CELULA** lista){
    CELULA* removida;
    PESSOA elementoRemovido;
    if(listaVazia(lista)){
      printf("\n Lista Vazia!");
      strcpy(elementoRemovido.nome, "   ");
      elementoRemovido.matricula = -1;
      return elementoRemovido;
    }
```

Lista Duplamente Encadeada - Remover Início (continuação)

```
    removida = (*lista);
    elementoRemovido = removida->info;
    (*lista) = (*lista)->prox;

    if((*lista) != NULL){
      (*lista)->ant = NULL;
    }
    free(removida);
  return elementoRemovido;
}
```

Lista Duplamente Encadeada - Remover Fim

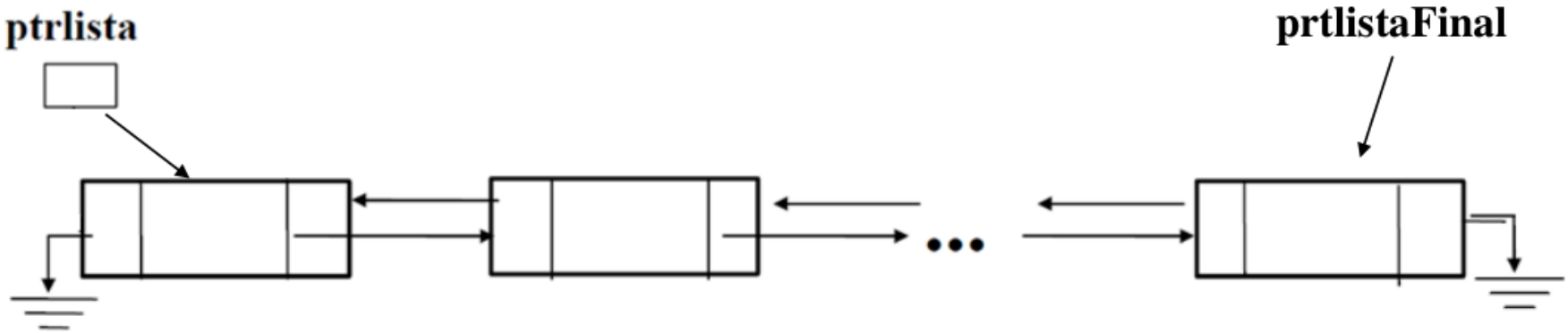
```
PESSOA removerFim(CELULA** lista){
    CELULA* removida;
     PESSOA elementoRemovido;
     if(listaVazia(lista)){
       printf("\n Lista Vazia!");
       strcpy(elementoRemovido.nome, "   ");
       elementoRemovido.matricula = -1;
       return elementoRemovido;
     }
```

Lista Duplamente Encadeada - Remover Fim (continuação)

```
        removida = (*lista);
        while(removida->prox != NULL){
            removida = removida->prox;
        }
        elementoRemovido = removida->info;
        (removida->ant)->prox = NULL;
        free(removida);
    return elementoRemovido;
}
```

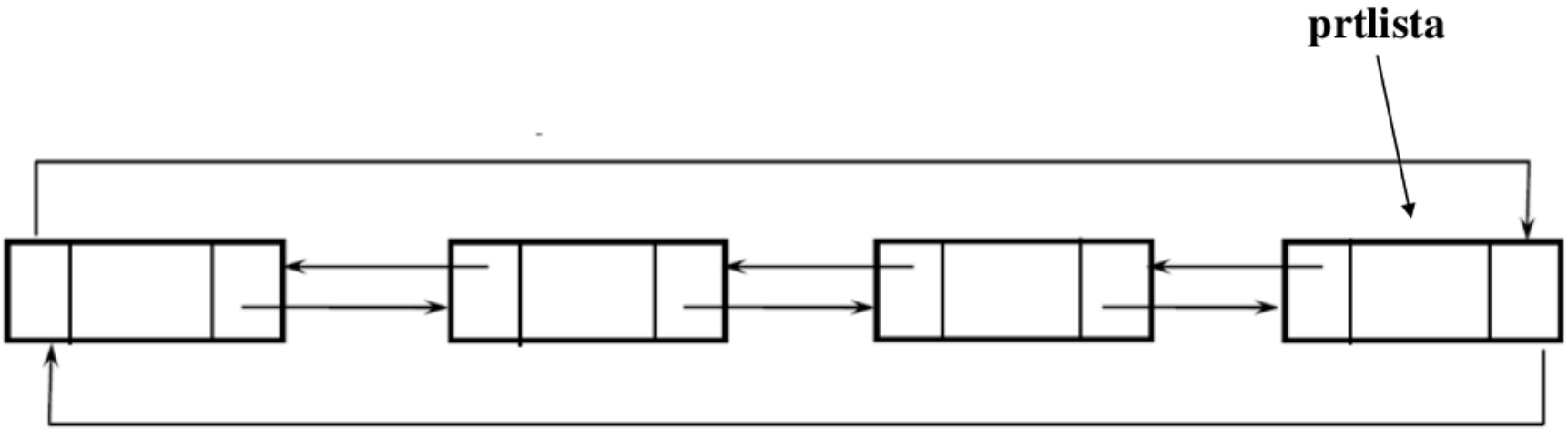
Lista Duplamente Encadeada

- Uma outra forma de implementação para a lista duplamente encadeada seria utilizando um ponteiro para apontar para o seu final.



Lista Duplamente Encadeada

- Neste caso, poderemos também implementar nossa lista duplamente encadeada de maneira circular.



Bibliografia Básica

- DROZDEK, Adam. Estrutura de Dados e Algoritmos em C++. Editora Pioneira Thomson Learning, 2005.
  - Pág 85, seção 3.3 (Listas Circulares) - até pág. 96
  - Pág 80, seção 3.2 (Lista Duplamente Ligada) – até pág. 84

- TENENBAUM A., LANGSAM Y. e AUGENSTEIN M. J. Estrutura de Dados usando C. Editora Makron, 1995.
  - Pág 279, seção 4.5 (Lista Circular) – até pág 280
  - Pág 294 (Lista Duplamente Ligada) – até pág. 300
  - Pág 287 (Nós de Cabeçalho) – até pág. 291