

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Presidente Epitácio

Estruturas de Dados – ED1C3

Listas Dinâmicas

PROF. MARCELO ROBERTO ZORZAN

DISCIPLINA: ESTRUTURAS DE DADOS I

AULA 08

Aula de Hoje

Lista Linear Dinâmica

Lista Auto-Organizada

Melhor solução: Lista Estática ou Dinâmica?

- Jorge que acabou de se formar no IFSP conseguiu emprego em uma empresa de telefonia de grande porte na cidade de São Paulo. Como era do seu interesse ele foi contratado para trabalhar na área de desenvolvimento para celular. O módulo que Jorge terá que implementar corresponde ao módulo de “contatos”.

Por quê???

Alocação: Sequencial x Encadeada

- Alocação de Memória Sequencial
 - itens agrupados em células consecutivas de memória
- Alocação de Memória Encadeada
 - itens ocupam células espalhadas por toda memória
 - itens são armazenados em blocos de memória denominados nós

Lista Encadeada

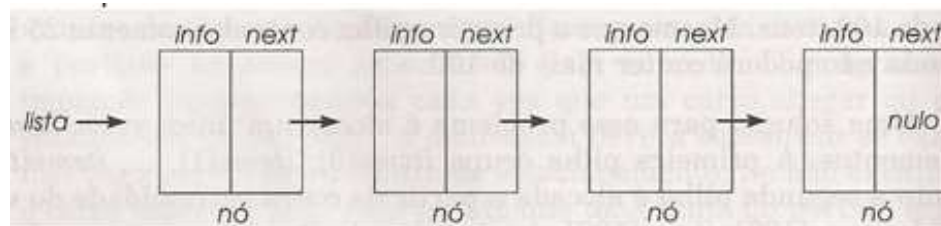
- Algumas vezes não é possível resolver o problema de forma sequencial
- Com *arrays* temos o problema de custo de inserção e/ou remoção no meio da lista

Solução???

- Uso de lista encadeada, também conhecida como lista ligada
- Uma lista encadeada é uma estrutura de dados linear e dinâmica

Lista Encadeada

- Definição: estrutura de dados que mantém uma coleção de itens em ordem linear sem exigir que eles ocupem posições consecutivas de memória.
- Lista Simplesmente Encadeada
 - itens são armazenados em blocos de memória denominados nós
 - cada nó possui dois campos:
 - 1) campo de informação
 - 2) campo do endereço do nó seguinte da lista



Como você sugere a implementação dessa estrutura de dados?

Lista Encadeada

- Vantagens:
 - Facilidade para inserção e remoção de itens em posições arbitrárias;
 - Pouca movimentação dos dados em memória;
 - Útil em aplicações em que o tamanho máximo da lista não precisa ser definido *a priori*.

Lista Encadeada

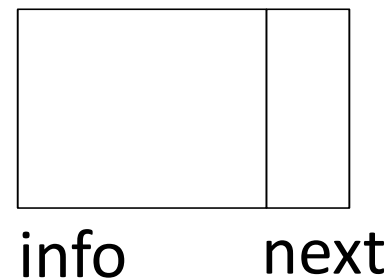
- Desvantagens:
 - Pode consumir mais tempo para preparação do sistema para alocar e liberar armazenamento;
 - Em média, o acesso a um item é mais oneroso que o acesso direto oferecido pelos *arrays*.

Lista Encadeada

(definição)

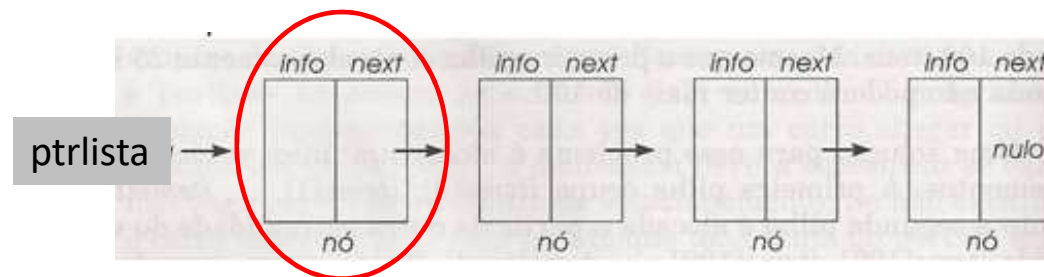
- Definir um nó para uma lista encadeada linear cujos nós são alocados dinamicamente
 - Considere que o campo informação seja um valor do tipo primitivo **int**

```
typedef struct cell
{
    int info;
    struct cell *next;
} CELULA;
```



Lista Encadeada

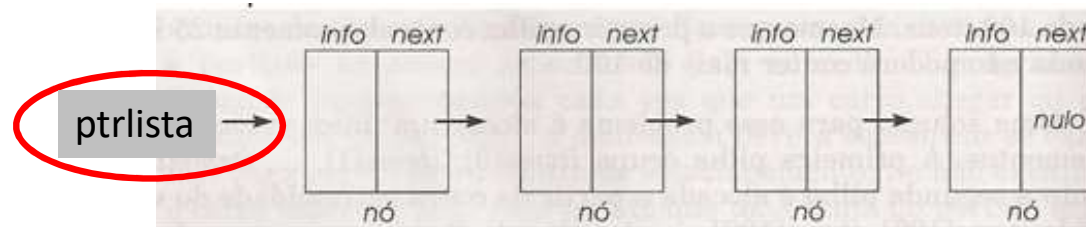
- Uma célula *c* pode ser declarados como:
CELULA *c*;
- Se *c* é uma célula então
 - *c.info* é o conteúdo da célula e
 - *c.next* é o endereço da próxima célula



Lista Encadeada

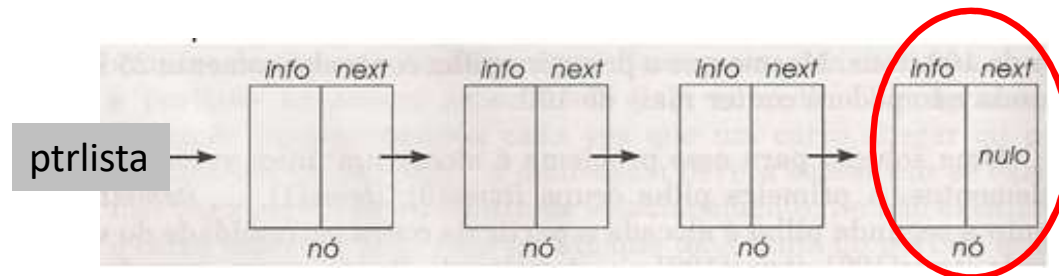
```
int main()
{
    CELULA * ptrlista; /* ponteiro externo ptrlista que
                        aponta para o primeiro nó da lista */
}
```

- Sendo ptrlista o endereço da primeira célula então
 - **ptrlista->info** é o conteúdo da célula e
 - **ptrlista->next** é o endereço da próxima célula



Lista Encadeada

- O campo do próximo endereço do último nó da lista contém um valor especial, conhecido como NULL, que não é um endereço válido.
- Esse ponteiro nulo (ou NULL) é usado para indicar o final de uma lista.



Lista Encadeada

(implementação)

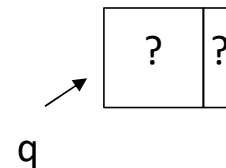
- Criar um nó dinamicamente:

```
CELULA* getnode()  
{  
    return (CELULA *) malloc (sizeof (CELULA)) ;  
}
```

→ A função getnode() aloca/cria um nó para uma lista encadeada.

CELULA q = getnode() ;

- obtém um nó vazio;
- q irá conter o endereço desse nó.



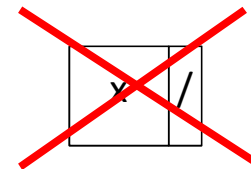
Lista Encadeada

(implementação)

- Liberar o espaço de memória ocupado por um nó.

```
void freenode (CELULA *q)
{
    free (q) ;
}
```

→ **q** é o endereço do nó a ser liberado



Lista Encadeada

(implementação)

- Inicializar o ponteiro externo à lista encadeada linear com o valor NULL

```
CELULA* init (CELULA *lista)
{
    lista = NULL;
    return lista;
}
```

→ Inicia o ponteiro para uma lista encadeada

lista



→ lista é o endereço do ponteiro externo para uma lista encadeada

Lista Encadeada

(implementação)

- Verificar se a lista encadeada está vazia

```
int empty (CELULA *lista)
{
    if (lista == NULL)
        return 1;
    return 0;
}
```

→ lista é o endereço do primeiro nó da lista encadeada

Lista Encadeada

(implementação)

- Imprimir os elementos da lista encadeada

```
void exibe_lista (CELULA *lista) {  
    CELULA *aux;  
  
    aux = lista;  
    while (aux != NULL) {  
        printf ("%d\t", aux->info);  
        aux = aux->next;  
    }  
    printf ("\n");  
}
```

→ lista é o endereço do primeiro nó da lista encadeada

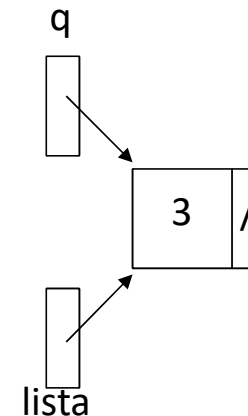
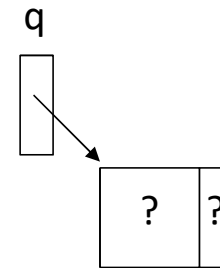
Lista Encadeada

(Criando um primeiro nó)

```
q = getnode();  
if (q != NULL) {  
    q->info = x; /* 3 */  
    q->next = NULL;  
}
```

```
lista = q;
```

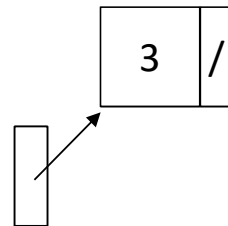
→ Lembrando que **q** armazena o endereço do novo nó e **lista** armazena o endereço do primeiro nó da lista encadeada.



Lista Encadeada

(Inserindo um nó no início da lista)

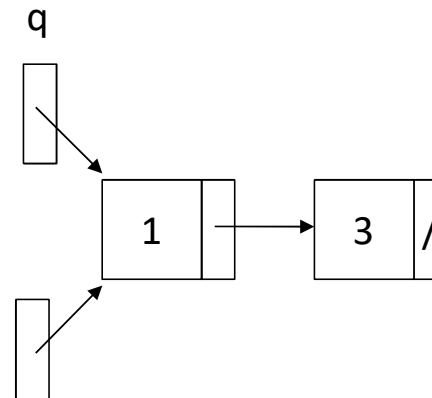
Antes



lista

```
q = getnode();  
if (q != NULL)  
{  
    q->info = x; /* 1 */  
    q->next = lista;  
    lista = q;  
}
```

Depois



lista

Lista Encadeada

(Inserindo um nó no início da lista)

```
CELULA* insere_inicio (CELULA *lista, int x)
{
    CELULA *q;

    q = getnode();
    if (q != NULL) {
        q->info = x;
        q->next = lista;
        lista = q;
        return lista;
    }
    else {
        printf("\nERRO na alocao do nó.\n");
        return NULL;
    }
}
```

→ lista : ponteiro externo para uma lista encadeada

→ x : inteiro que indica o valor a ser inserido

Lista Encadeada

(Inserindo um nó no final da lista)

```
CELULA* insere_fim (CELULA *lista, int x) {  
    CELULA *q;  
    CELULA *aux;  
  
    q = getnode ();  
    if (q != NULL) {  
        q->info = x;  
        q->next = NULL;  
  
        if (empty(lista))  
            lista = q;  
    }  
}
```

Lista Encadeada

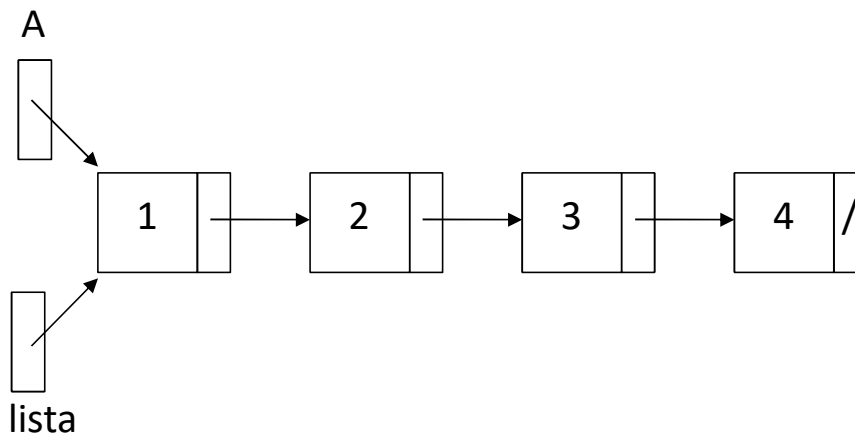
(Inserindo um nó no final da lista)

```
else{ //percorre lista até chegar ao ultimo nó
    aux = lista;
    while (aux->next != NULL)
        aux = aux->next;

    aux->next = q;
}
return lista;
} // Fim do if(q != NULL)
else {
    printf ("\nERRO na alocação do nó.\n");
    return NULL;
}
}
```

Como remover um nó no início da lista?

Antes



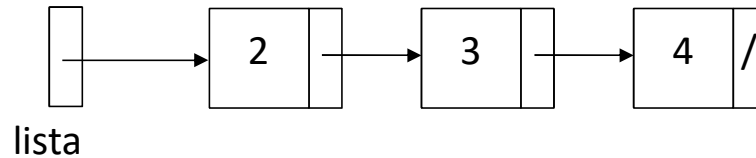
//Remoção do nó A

`x = A->info;`

`lista = lista->next;`

`freenode (A);`

Depois



Lista Encadeada

(Remover um nó no início da lista)

```
CELULA* remove_inicio (CELULA *lista) {  
    CELULA *q;  
  
    q = lista;  
    if (!empty(lista)) { //há itens na lista  
        lista = q->next;  
        freenode (q);  
        return lista;  
    }  
    else {  
        printf ("\nERRO: lista vazia.\n");  
        return NULL;  
    }  
}
```


Lista Encadeada

(Pesquisar um valor na lista)

```
CELULA* pesquisa (CELULA *lista, int x) {  
    CELULA *q;  
  
    if (!empty(lista)) {  
        q = lista;  
        while (q != NULL) {  
            if (q->info == x)  
                return q; //encontrou o nó  
            q = q->next;  
        }  
    }  
    return NULL; //não encontrou o nó  
}
```

→ Caso exista item com valor indicado em x, retorna o endereço do primeiro nó que contém tal valor

Lista Encadeada

(Remover um nó da lista)

```
CELULA* remove_valor (CELULA *lista, int x) {  
    CELULA *q;  
    CELULA *aux;  
  
    if ((q = pesquisa (lista, x)) != NULL)  
    {  
        aux = lista;  
        if (aux == q) //nó está no inicio da lista  
            remove_inicio (lista);  
  
        ...  
    }
```

Lista Encadeada

(Remover um nó da lista)

```
...  
    else {  
        while (aux->next != q)  
            aux = aux->next;  
        aux->next = q->next;  
        freeNode (q) ;  
    }  
    return lista; //removeu  
}  
return NULL; //não removeu  
}
```

Lista Encadeada

(Função main)

```
int main(){
    CELULA *list;
    list = init(list);
    list = insere_inicio (list, 7) ;
    list = insere_fim (list, 3);
    list = insere_fim (list, 9);
    exhibe_lista(list);
    list = remove_valor (list, 3);
    exhibe_lista(list);

    getch();

    return 0;
}
```

Lista Encadeada

(Busca Sequencial)

As listas encadeadas exigem a busca sequencial para localizar um elemento ou descobrir que ele não está na lista.

A busca inicia no começo da lista e pára nos seguintes casos:


- Quando encontra o elemento procurado
- Quando o fim da lista é atingido sem encontrar o elemento desejado

Lista Auto-Organizada

(Motivação)

Pode-se melhorar a eficiência da busca organizando a lista dinamicamente.

Existem formas diferentes para organizar as listas:

1. Método de mover para frente: O elemento localizado deve ser colocado no início da lista.
 2. Método da transposição: O elemento localizado deve ser trocado com seu predecessor, exceto se ele estiver na primeira posição da lista.
 3. Método da contagem: A lista deve ser ordenada pelo número de vezes que os elementos são acessados.
 4. Método da ordenação: A lista é ordenada de acordo com sua informação.
- 

Lista Auto-Organizada

(Motivação)

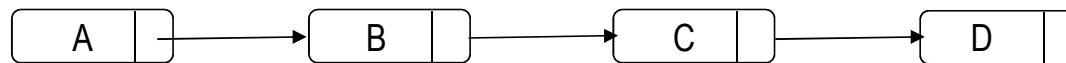
Os três primeiros métodos (mover para frente, transposição e contagem) permitem colocar os elementos mais prováveis de serem acessados no início da lista.

O método da ordenação tem a vantagem na busca de informação, sobretudo se a informação não está na lista pois a busca pode terminar sem que seja necessário pesquisar toda a lista.

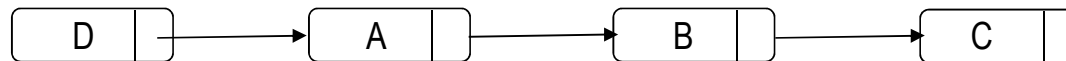
Lista Auto-Organizada (Métodos)

1) Método de mover-para-frente

- O elemento localizado deve ser colocado no início da lista.



Elemento procurado → D



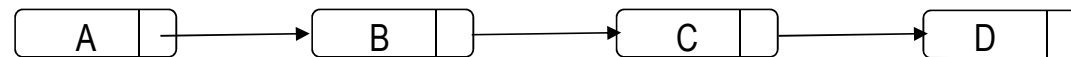
→ Desvantagem:

- Uma única pesquisa pelo elemento não implica que o registro será freqüentemente pesquisado
- O método é mais “caro” em uma lista linear estática do que em uma lista linear dinâmica. **Por quê?**

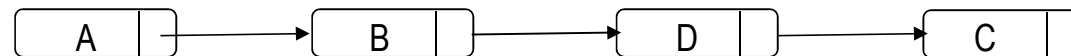
Lista Auto-Organizada (Métodos)

2) Método da transposição

- Depois de localizar o elemento desejado, troque-o com seu predecessor, a menos que ele esteja no início da lista



Elemento procurado → D



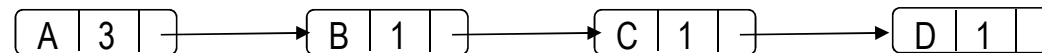
→ Vantagem:

- Se avançarmos o elemento uma posição na lista sempre que ele for pesquisado, garantiremos que ele avançará para o início da lista apenas se for pesquisado com frequência.

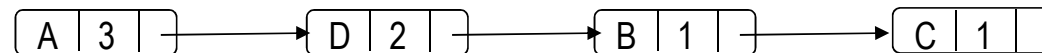
Lista Auto-Organizada (Métodos)

3) Método da contagem

- Ordene a lista pelo número de vezes que os elementos estão sendo acessados



Elemento procurado → D

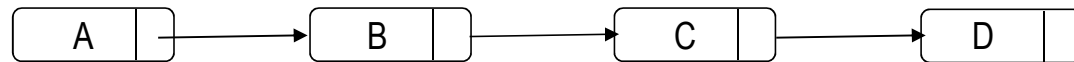


→ O método de contagem pode ser agrupado na categoria dos métodos de ordenação se a frequência é parte da informação.

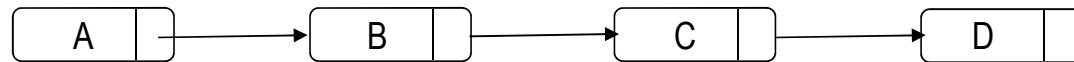
Lista Auto-Organizada (Métodos)

4) Método da ordenação

- A lista é ordenada de acordo com sua informação



Elemento procurado → D



Leitura Recomendada

- TENENBAUM A., LANGSAM Y. e AUGENSTEIN M. J.
Estrutura de Dados usando C. Editora Makron, 1995.
 - Pág 223, seção 4.2 - até pág. 229
 - Pág 231 (“Operações getnode e freenode”) – até pág. 233
 - Pág 256 (“Listas ligadas usando variáveis dinâmicas”) – até pág. 258
 - Pág 260 (“Exemplos de operações de listas em C”) – até pág. 262
- DROZDEK, Adam. Estrutura de Dados e Algoritmos em C++. Editora Pioneira Thomson Learning, 2005.
 - Pág 91, seção 3.5 (Listas Auto-Organizadas)- até pág. 94

Exercício

1) Crie um arquivo ListaLinearDinamica.c e implementa as seguintes informação de uma lista linear dinâmica:

- ✓ Definição
- ✓ Operações
 - init
 - getnode
 - freenode
 - empty
 - exhibe_lista
 - insere_inicio, insere_fim
 - remove_inicio, remove_valor
 - pesquisa

Exercício

- 2) Crie um arquivo `AplicacaoListaLinearDinamica.c` que manipule as informações de uma lista linear dinâmica a partir das funções criadas no item 1.

Atividades

- 1) Implementar lista linear dinâmica (arquivos `ListaLinearDinamica.c` e `AplicacaoListaLinearDinamica.c`)
- 2) Implementar os seguintes métodos de busca para listas auto-organizadas:
 - a) Mover para frente
 - b) Transposição
 - c) Contagem