



SIN211 Algoritmos e Estruturas de Dados

Prof. João Batista Ribeiro

joao42ibatista@gmail.com



Universidade Federal de Viçosa

Slides baseados no material da Prof.^a Rachel Reis



Aula de Hoje

- Lista auto-organizada
- Lista encadeada ordenada
- Ponteiro para ponteiro



Situação-problema

- Imagine que você precise desenvolver uma função para armazenar os contatos de um celular. O desenvolvimento deve ser projetado de forma a facilitar a pesquisa por um contato.
 - 1) Como você sugere o armazenamento dos contatos? Por que?
 - 2) E a parte de pesquisa? Como você sugere que ela seja feita?
 - 3) Alguma sugestão para melhorar a eficiência da busca?



Lista Encadeada - Problema

- 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 para 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

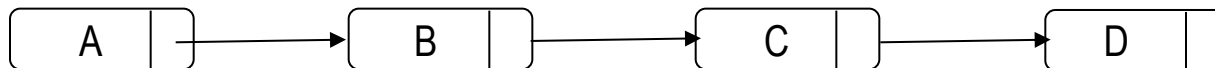
- Os três primeiros métodos (mover para frente, transposição e contagem) permitem colocar os elementos mais prováveis de serem acessados no início da lista.
- O método da ordenação tem a vantagem na busca de informação, sobretudo se a informação não está na lista, pois a busca pode terminar sem que seja necessário pesquisar toda a lista.



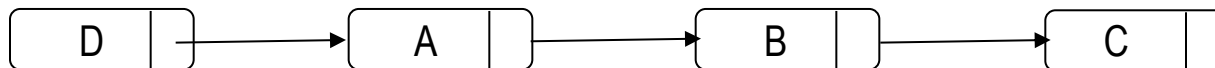
Lista auto-organizada - Métodos

1) Método de mover-para-frente

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



Elemento procurado → D



→ Desvantagem:

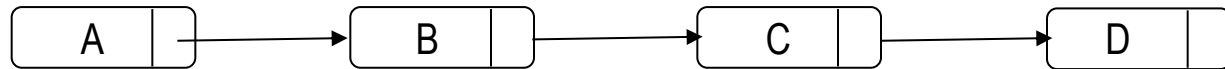
- Uma única pesquisa pelo elemento não implica que o registro será frequentemente pesquisado
- O método é mais “caro” em uma lista linear estática do que em uma lista linear dinâmica. **Por quê?**



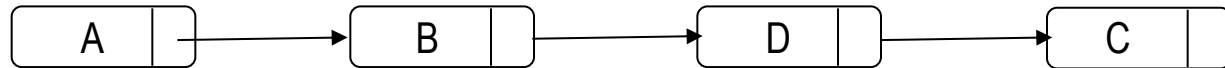
Lista Auto-organizada - Métodos

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:**

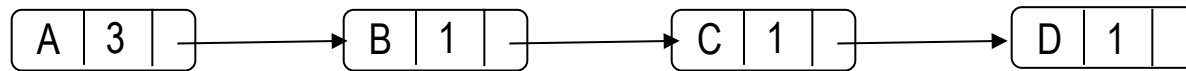
- Se avançarmos o elemento uma posição na lista sempre que ele for pesquisado, garantiremos que ele avançará para o início da lista apenas se for pesquisado com frequência.



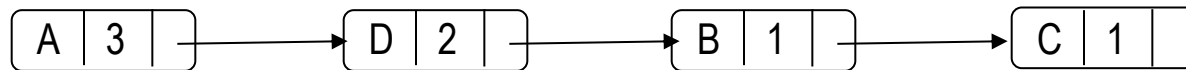
Lista Auto-organizada - Métodos

3) Método da contagem

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



Elemento procurado → D



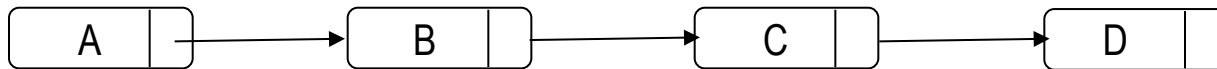
→ O método de contagem pode ser agrupado na categoria dos métodos de ordenação se a frequência é parte da informação.



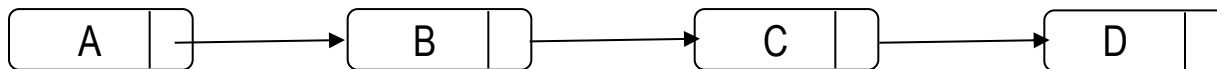
Lista Auto-organizada - Métodos

4) Método da ordenação

- A lista é ordenada de acordo com sua informação



Elemento procurado → D





Atividade

1) Implementar os seguintes métodos de busca para lista auto-organizada:

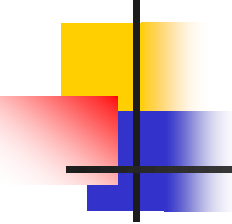
- a) Mover para frente
- b) Transposição
- c) Contagem

2) Pense em uma situação-problema que descreva uma aplicação para os algoritmos acima diferente daqueles trabalhados em sala de aula.



Lista Encadeada Ordenada

- Inserção Ordenada
 - Insere um nó mantendo a lista ordenada
- Algoritmo
 - Se a lista estiver vazia
 - Faz o ponteiro externo apontar para o novo nó
 - Caso contrário, encontra o primeiro nó maior ou igual ao novo nó
 - Faz o novo nó apontar para o nó atual
 - Faz o nó anterior apontar para o novo nó



Lista Ordenada - Implementação

```
CELULA* inserirListaOrdem(CELULA* lista, int x){
    CELULA* atual = lista;
    CELULA* anterior = NULL;
    CELULA* q;

    q = getnode();
    if(q != NULL){
        q->info = x;
        q->next = NULL;

        /*procura posicao para insercao*/
        while(atual != NULL && atual->info < x){
            anterior = atual;
            atual = atual->next;
        }
    }
    ...
}
```

Inserir nó mantendo
a lista ordenada



Lista Ordenada - Implementação

```
/*insere elemento*/
if(atual == lista){ /*insere no começo*/
    q->next = lista;
    lista = q;
} else{ /*insere no meio/fim*/
    anterior->next = q;
    q->next = atual;
}
return lista;
} else { // Fim do if(q != NULL)
    printf ("\nERRO na alocação do nó.\n");
    return NULL;
}
}
```



Lista ordenada - Implementação

```
int main() {
    CELULA *ptrlista;
    ptrlista = init(ptrlista);
    ptrlista = inserirListaOrdem(ptrlista, 7);
    exhibe_lista(ptrlista);
    ptrlista = inserirListaOrdem(ptrlista, 3);
    exhibe_lista(ptrlista);
    ptrlista = inserirListaOrdem(ptrlista, 5);
    exhibe_lista(ptrlista);

    getch();
    return 0;
}
```



Ponteiro para ponteiro

- Um ponteiro para um ponteiro é como se você deixasse dentro do seu armário um papel com o endereço da casa de um amigo.

- Notação:

```
tipo_da_variável **nome_da_variável;
```

onde

****nome_da_variável** é o conteúdo final da variável apontada

***nome_da_variável** é o conteúdo do ponteiro intermediário.



Ponteiro para ponteiro

- Exemplo:

```
int main() {  
    int **ptrPtr;  
    int *Ptr;  
    int a = 10;  
    Ptr = &a;  
    ptrPtr = &Ptr;  
    printf("Valor de a: %d", **ptrPtr);  
    getchar();  
    return 0;  
}
```



Lista Encadeada - Implementação

- Por que o tipo de retorno da função init tem que ser CELULA*?

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

```
int main(){  
    CELULA *ptrLista;  
    ptrLista = init(ptrLista);  
}
```



Lista Encadeada - Implementação

- Por que o tipo de retorno da função init tem que ser CELULA*?

```
int main(){  
    CELULA *ptrLista;  
  
}
```

Memória

Função main →





Lista Encadeada - Implementação

- Por que o tipo de retorno da função init tem que ser CELULA*?

```
int main(){  
    CELULA *ptrLista;  
    ptrLista = init(ptrLista);  
}
```

```
CELULA* init (CELULA *lista){  
  
}
```

Memória

Função main →

LIXO
ptrLista 001

Função init →

LIXO
lista 0020



Lista Encadeada - Implementação

- Por que o tipo de retorno da função init tem que ser CELULA*?

```
int main(){  
    CELULA *ptrLista;  
    ptrLista = init(ptrLista);  
}
```

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

Memória

Função main →

LIXO
ptrLista 001

Função init →

NULL
lista 0020



Lista Encadeada - Implementação

- Por que o tipo de retorno da função init tem que ser CELULA*?

```
int main(){  
    CELULA *ptrLista;  
    ptrLista = init(ptrLista);  
}
```

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

Memória

Função main →

NULL
ptrLista 001

Função init →

NULL
lista 0020



Lista Encadeada - Implementação

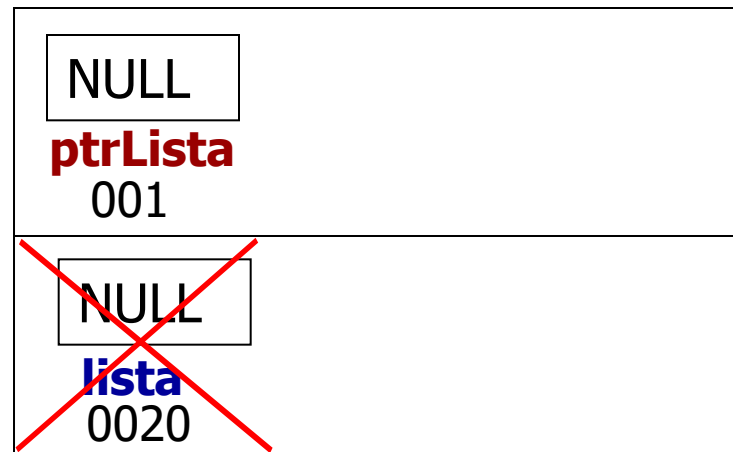
- Por que o tipo de retorno da função init tem que ser CELULA*?

```
int main() {  
    CELULA *ptrLista;  
    ptrLista = init(ptrLista);  
}
```

Memória

Função main →

Função init →





Lista Encadeada - Implementação

- E se usarmos ponteiro para ponteiro?

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




Lista Encadeada - Implementação

- E se usarmos ponteiro para ponteiro?

```
int main(){  
    CELULA *ptrLista;  
}
```

Memória

Função main →





Lista Encadeada - Implementação

- E se usarmos ponteiro para ponteiro?

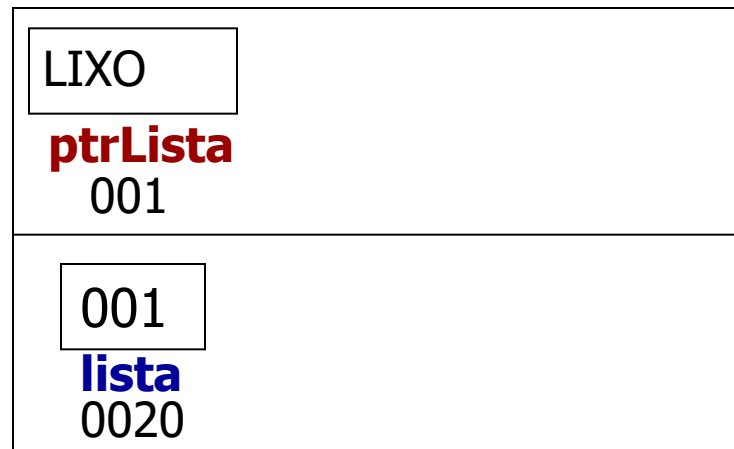
```
int main(){  
    CELULA *ptrLista;  
    init(&ptrLista);  
}
```

```
void init (CELULA **lista){  
  
}
```

Memória

Função main →

Função init →



Lista Encadeada - Implementação

- E se usarmos ponteiro para ponteiro?

```
int main() {  
    CELULA *ptrLista;  
    init(&ptrLista);  
}
```

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

Memória

Função main →

NULL
ptrLista
001

Função init →

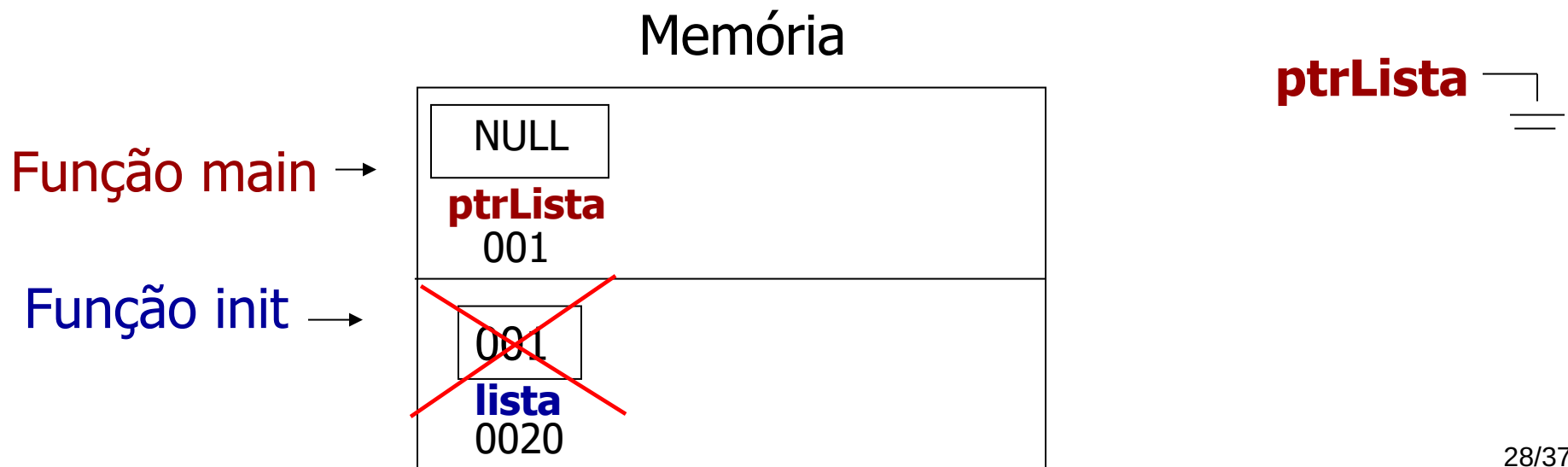
001
lista
0020

ptrLista →
↑
lista

Lista Encadeada - Implementação

- E se usarmos ponteiro para ponteiro?

```
int main() {  
    CELULA *ptrLista;  
    init(&ptrLista);  
}
```





Lista Encadeada - Implementação

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

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

→ Inicia o ponteiro externo para uma lista encadeada



Lista encadeada - Implementação

- Inserir novo nó no início da lista encadeada

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

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



Lista encadeada - Implementação

- Inserir novo nó no final da lista encadeada

```
void 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 - Implementação

■ Inserir novo nó no final da lista encadeada (cont.)

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

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




Lista Encadeada - Implementação

- Remover nó início da lista encadeada

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



Lista Encadeada - Implementação

- Remover nó específico da lista encadeada

```
int 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 - Implementação

- Remover nó da lista encadeada (cont.)

```
    else {  
        while (aux->next != q)  
            aux = aux->next;  
        aux->next = q->next;  
        freenode (q) ;  
    }  
    return 1; // removeu  
}  
return 0; //não removeu  
}
```



Lista Encadeada - Implementação

```
int main() {  
    CELULA *ptrlista;  
    int del;  
    init(&ptrlista);  
    insere_inicio(&ptrlista, 7);  
    exhibe_lista(ptrlista);  
    insere_fim(&ptrlista, 3);  
    exhibe_lista(ptrlista);  
    del = remove_valor (&ptrlista, 3);  
    if (del == 0)  
        printf("\nErro!");  
    exhibe_lista(ptrlista);  
    getchar();  
}
```



Leitura Recomendada

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