INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
CATARINENSE
Campus Videira

Listas
Alocação Encadeada Simples

Professor: Manassés Ribeiro

Conceito

- Podem ser implementadas utilizando vetores ou, mais comumente, utilizando alocação dinâmica de memória;
- Na alocação encadeada, a ordem lógica das informações pode ser (e geralmente é) diferente da ordem física.

Conceito

- Exemplo de alocação da palavra BANANA

Primeiro →

	Info	Elo
0	A	4
1		
2	B	10
3		
4	N	8
5		
6	N	0
7		
8	A	-1
9		
10	A	6

- Na alocação encadeada, as informações devem ser SEMPRE acessadas conforme a ordem lógica, nunca pela ordem física.

Conceito

- Na alocação encadeada, o Nodo deve conter pelo menos duas informações: **INFO** e **ELO**.
- **INFO:** campo que contém as informações que são armazenadas na lista e;
- **ELO:** campo que faz o encadeamento das informações, responsável pela definição da ordem lógica.

Conceito

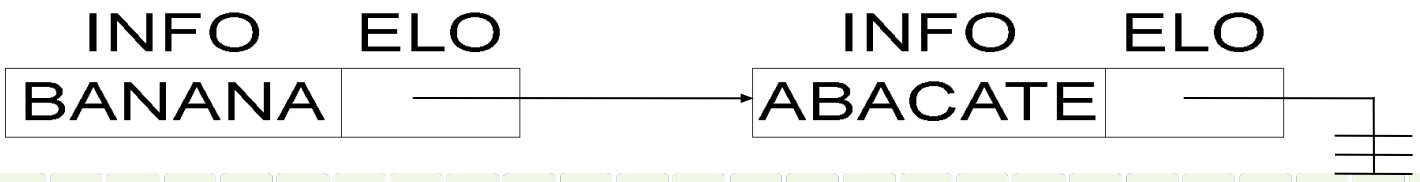
- O campo INFO, por sua vez, pode ser dividido em n outros campos. O campo ELO sempre conterá o índice do próximo nodo na ordem lógica.

Alocação Dinâmica de Memória (ADM)

- É o processo que aloca (solicita/reserva) memória em tempo de execução;
- É utilizado quando não se sabe ao certo quanto de memória será necessário para realizar determinada operação;
- Essencial para evitar o desperdício de memória.

Lista Encadeada com ADM

- As Listas Encadeadas, utilizando alocação dinâmica de memória (ADM), são compostas por elementos individuais cada um ligado por um único ponteiro. Cada elemento consiste em duas partes: um membro de dados (INFO), e um ponteiro próximo (ELO).

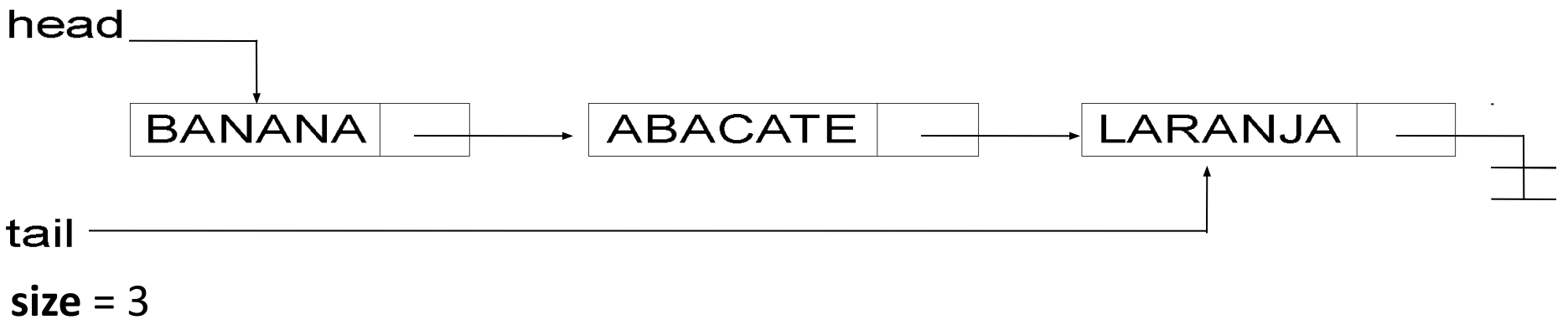


Lista Encadeada com ADM

- Usando esta estrutura de dois membros, uma lista encadeada é formada definindo-se o ponteiro **next (ELO)** de cada elemento para que ele aponte para o elemento que segue (o próximo).
- O elemento **next** do último elemento é definido para **NULL**.
- O elemento no inicio da lista é a cabeça (**head**) e o elemento no final da lista é a cauda (**tail**).

Lista Encadeada com ADM

- Exemplo:



size = 3

Lista Encadeada com ADM

- Para acessar um elemento em uma lista encadeada inicia-se pela cabeça da lista (*head*) e percorre a lista por meio dos ponteiros próximo (*next*) do elementos sucessivos movendo-se de elemento a elemento até que o elemento desejado seja encontrado ou até chegar na cauda (*tail*).

Lista Encadeada com ADM

As principais funções em listas são:

- Cria lista;
- Cria elemento/nodo;
- Insere elemento na lista;
- Remove elemento da lista;
- Percorre a lista (encontra elementos).

Estrutura da Lista

A estrutura principal da lista encadeada simples, utilizando ADM, é composta por:

- Elemento *head*;
- Elemento *tail*;
- Tamanho da lista (um número inteiro que indica quantos elementos, nodos, compõem a lista).

Algoritmo da função insere

1.Cria novo elemento

2.Aloca memória para novo elemento

3.Atribui valor ao novo elemento

4.Verifica se elemento pivô é NULL, caso verdadeiro

4.1.Verifica se a lista esta vazia, caso verdadeiro

4.1.1.Insere o novo elemento como primeiro elemento da lista e vai para o item “6”.

4.2.Caso a lista não esteja vazia

4.2.1.Insere o novo elemento no início da lista e vai para o item “6”.

5.Caso o elemento pivô não seja NULL

5.1.Verifica se o elemento pivô é o último elemento da lista, caso verdadeiro

5.1.1.Insere o novo elemento no final da fila e vai para o item “6”.

5.2.Caso o elemento pivô não seja o último elemento da lista

5.2.1.Insere o novo elemento no meio da lista, logo após o elemento pivô e vai para o item “6”.

6.Atualiza o tamanho da lista

Algoritmo da função remove

1. Cria um elemento para ser usado como elemento antigo
2. Verifica se a lista esta vazia, caso verdadeiro
    2.1 Retorna um sinal de erro indicando que a lista está vazia e encerra o algoritmo
3. Verifica se o elemento pivô é NULL, caso verdadeiro
    3.1 É atribuído ao elemento antigo o elemento cabeça da lista
    3.2 É definido como novo elemento cabeça da lista o segundo elemento da lista
    3.3 Verifica se o novo elemento cabeça for NULL, caso verdadeiro executa o passo 3.3.1, caso falso, vai para o passo "5".
        3.3.1 Define a lista como lista vazia e vai para o passo "5"
4. Caso o elemento pivô não for NULL
    4.1 Verifica se o elemento pivô é o último elemento da lista, caso verdadeiro
        4.1.1 Retorna um sinal indicando que o elemento pivô já é o último elemento da lista e encerra a execução do algoritmo
    4.2 Caso o elemento pivô não seja o ultimo elemento da lista
        4.2.1 É atribuído ao elemento antigo o próximo elemento do pivô
        4.2.2 É atribuído ao *next* do elemento pivô o *next* do elemento antigo
        4.2.3 Verifica se o *next* do elemento pivô é NULL, caso verdadeiro executa o passo 4.2.3.1, caso falso executa o passo "5"
            4.2.3.1 Define o elemento pivô como *tail* da lista e vai para o passo "5"
5. Libera a memória do elemento antigo
6. Atualiza o tamanho da lista

Algoritmo da função insere

```
insercao(lista, elementoPivo, dado)
inicio
        Cria (novo_elemento)
        Aloca_memória (novo_elemento)
        novo_elemento->dado = dado

        if (elementoPivo == NULL)
                if (lista vazia)
                        lista->tail = novo_elemento

                novo_elemento->next = lista->head
                lista->head = novo_elemento
        senao
                se (elementoPivo->next == NULL)
                        lista->tail = novo_elemento

                novo_elemento->next = elementoPivo->next
                elementoPivo->next = novo_elemento

        fimse
        atualiza_tamanho_lista
fim
```

Algoritmo da função remove

```
remocao(lista, elementoPivo)
inicio
        Cria (elemento_antigo)
        se (lista vazia)
                retorna lista vazia

        se (elementoPivo == NULL)
                elemento_antigo = lista->head
                lista->head = lista->head->next

                se (lista->head == NULL)
                        lista->tail = NULL
        else
                se (elementoPivo->next == NULL)
                        retorna fim da lista

                elemento_antigo = elementoPivo->next
                elementoPivo->next = elementoPivo->next->next
                        if (elementoPivo->next == NULL)
                                lista->tail = elementoPivo
                }


                libera_memória(elemento_antigo)

                atualiza_tamanho_lista
fim
```

Atividades de fixação

Dada a lista de elementos ao lado (que inicialmente está vazia) realizar as operações em lista encadeada simples. Ao final apresentar a representação gráfica do resultado final da lista indicando a quantidade de elementos restantes.

Seq.	Elemento	Pivô	Operações
1º		tail	remoção
2º	João	tail	inserção
3º	Maria	head	inserção
4º	Lucas	“Maria”	inserção
5º		“Lucas”	remoção
6º	Sandro	Null	inserção
7º	Pedro	Null	inserção
8º		tail	remoção
9º		head	remoção
10º		Null	remoção

Implementação de lista encadeada simples em C

- Implementar uma lista encadeada simples em C utilizando os conceitos e algoritmos vistos em aula.
- O tipo de dados dos elementos da lista podem ser valores inteiros, alfanuméricos (strings), ou ponteiros para outros tipos abstratos de dados.
  - O ideal é que a lista já seja construída para armazenar tipos abstratos de dados.

Principais funções

- **Cria lista:**
  - função que cria a lista e aloca a memória necessária para a lista;
- **Cria elemento:**
  - função que cria um elemento (nodo) novo, aloca memória necessária e atribui um valor para este novo elemento;
- **Insere elemento na lista:**
  - função que insere este novo elemento em uma lista seguindo o algoritmo apresentado nos slides 13 e 15;
- **Remove elemento da lista:**
  - função que insere este novo elemento em uma lista seguindo o algoritmo apresentado nos slides 14 e 16;
- **Percorre a lista para encontrar elementos (pesquisa):**
  - função que percorre a lista, a partir da cabeça (head) até encontrar o elemento que está sendo buscado ou atingir o fim da lista;

Representação em C

Representação de Elementos em Lista encadeada Simples usando C

```
typedef struct sElemento{
    struct sElemento *next;
    int dado; //Depende do tipo de dados que se deseja trabalhar
} Elemento;
```

Representação de Lista encadeada Simples em C

```
typedef struct sLista{
    struct sElemento *head;
    struct sElemento *tail;
    int size;
} Lista;
```