Listas Circulares, com nó cabeça e ordenadas

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

Baseado nos slides do Prof. Rudinei Goularte

Conteúdo

- ➔ Listas Ligadas com Nó Cabeça
- ➔ Listas Ligadas Circulares
- ➔ Listas Ligadas Ordenadas

- ◆ Obs: ligada, aqui, é sinônimo de encadeada.

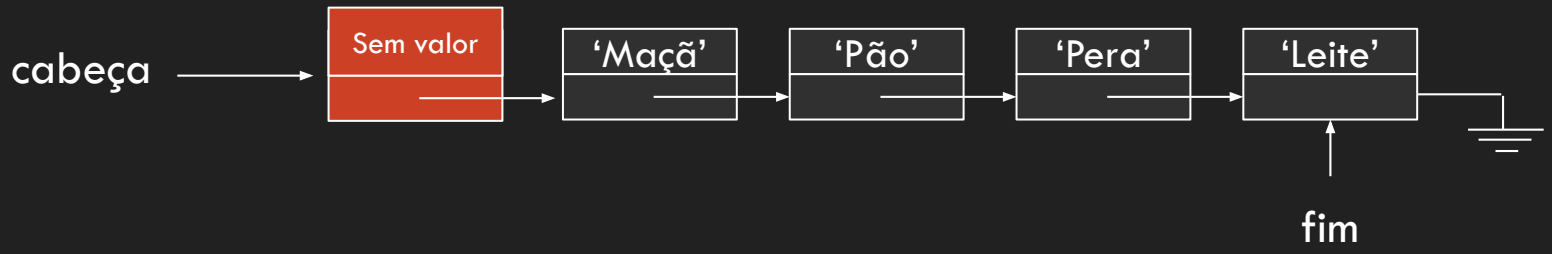
Listas Ligadas com Nó Cabeça

- A operação **mais complexa** é a **remoção** de um elemento dado uma chave
- Isso porque o algoritmo precisa apontar para o item anterior ao que será removido, o que, no caso da **remoção do primeiro elemento**, configura uma **exceção** que precisa ser tratada à parte

Listas Ligadas com Nó Cabeça

- Uma **solução** que **simplifica a implementação** é substituir o ponteiro para início por um **nó cabeça**
- Um nó cabeça é um nó normal da lista, mas esse é sempre o **primeiro nó** e a **informação** armazenada **não tem valor**

Listas Ligadas com Nó Cabeça



Nó Cabeça e Lista Vazia

- A lista com nó cabeça será vazia quando o próximo do nó cabeça apontar para NULL

```
struct lista_{
    NO *cabeca;
    NO *fim;
    int tamanho;
};
boolean lista_vazia(LISTA *lista) {
    if (lista != NULL && (lista->cabeca->proximo == NULL))
        return (TRUE);
    return (FALSE);
}
```

Criar Lista

```
LISTA *lista_criar(void) {
    LISTA *lista = (LISTA *) malloc(sizeof(LISTA));
    if(lista != NULL) {
        lista->cabeca = (NO *) malloc(sizeof(NO));
        if (lista->cabeca == NULL)
            return(NULL);
        lista->cabeca->proximo = NULL;
        lista->fim = NULL;
        lista->tamanho = 0;
    }
    return(lista);
}
```

Implementação das Demais Operações

- A implementação das demais operações é similar a lista ligada padrão (sem nó cabeça), a única alteração é substituir as referências ao ponteiro início pelo próximo do nó cabeça
- O grande ganho é na remoção dado uma chave, já que não é necessário tratar separadamente quando o item a se remover é o primeiro

Remover nó (dado uma chave)

```
boolean lista_remover(LISTA *lista, int chave) {
    if (lista != NULL) {
        NO *p = lista->cabeca;

        while (p->proximo != NULL && (item_get_chave(p->proximo->item)) != chave){
            p = p->proximo;
        }

        if (p->proximo != NULL) {
            NO *aux = p->proximo;
            p->proximo = aux->proximo;
            aux->proximo = NULL;
            if (aux == lista->fim)
                lista->fim = p;
            lista->tamanho --;
            free(aux); aux = NULL;
            return (TRUE);
        }
    }
    return (FALSE);
}
```

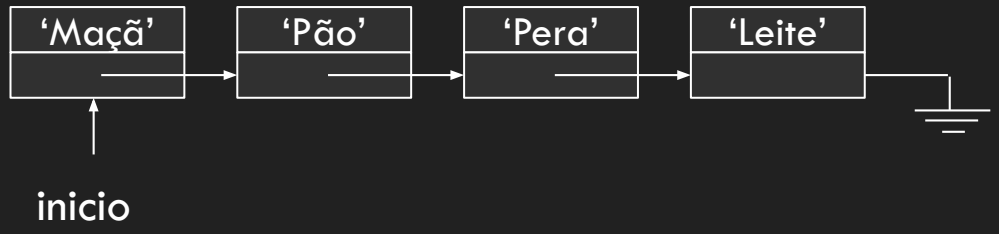
Exercício

- ➔ Implementar as demais operações do TAD listas usando o conceito de lista ligada com nó cabeça

Listas Ligadas Circulares

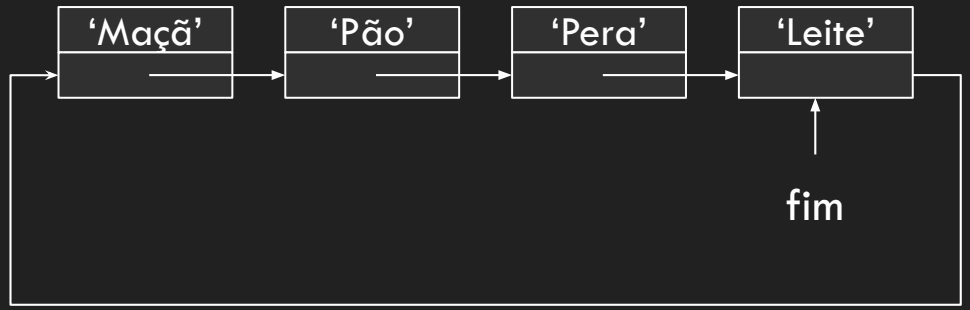
Listas Ligadas Circulares

- Um diferente tipo de implementação de listas ligadas substitui a definição de que *o próximo do último é NULL* por ***o próximo do último é o primeiro***



Listas Ligadas Circulares

➔ Um diferente tipo de implementação de listas ligadas substitui a definição de que o próximo do último é NULL por o próximo do último é o primeiro



Listas Ligadas Circulares

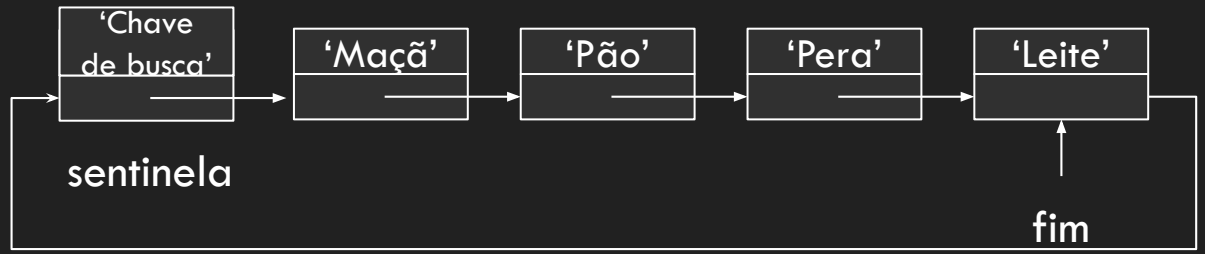
- A partir de um nó da lista pode-se chegar a qualquer outro nó!!!!
- Nessa implementação somente um ponteiro para o fim da lista é necessário, não sendo necessário um ponteiro para o início. (Por quê?)

Listas Ligadas Circulares (Sentinela)

- ➔ No caso especial da **busca** em listas circulares, o emprego de um nó cabeça pode reduzir a quantidade de testes necessários
- ➔ A ideia é colocar a chave de busca no nó cabeça e começar a busca no próximo nó

Listas Ligadas Circulares (Sentinela)

- Se o item encontrado for a cabeça, a busca não teve sucesso. Assim um teste é “economizado” já que não é preciso testar se a lista acabou
- Nesse caso, o nó cabeça é chamado de **sentinela**



Listas Ligadas Circulares (Sentinela)

```
struct lista_{
    NO *sentinela;
    NO *fim;
    int tamanho;
};

ITEM *busca(LISTA *lista, int chave) {
    item_set_chave(&(lista->sentinela->item), chave);
    NO *p = lista->sentinela;
    do {
        p = p->proximo;
    } while (item_get_chave(p->item) != chave);
    return ((p != lista->sentinela) ? p->item : NULL);
}
```

Listas Ligadas Ordenas

Listas Ligadas Ordenadas

- Suponha uma lista ligada ordenado contendo os itens 6, 8 e 10. Inserir, ordenadamente, a sequência: 9, e 3.

  - Na lousa...

Listas Ligadas Ordenadas

- Comentários sobre a implementação
  - Não precisa do ponteiro **fim** porque a inserção será em qualquer posição de lista
  - Novamente o emprego do nó cabeça facilita a implementação uma vez que vamos buscar a posição anterior à de inserção, e no caso de ser o menor item da lista isso não representará exceção

Listas Ligadas Ordenadas

```
boolean lista_inserir_ordenado(LISTA *lista, ITEM *i) {
    NO *p = NULL, *n = NULL;
    if(lista != NULL && (!lista_cheia(lista)) ) {
    p = lista->cabeca;
    while(p->proximo != NULL && (item_get_chave(p->proximo->item) < item_get_chave(i))
        p = p->proximo;
        n = (NO *) malloc (sizeof(NO)); n->item = i;
        n->proximo = p->proximo;
        p->proximo = n;
        if (n->proximo == NULL)
            lista->fim = n;
        lista->tamanho++;
        return(TRUE);
    }
    return(FALSE);
}
```

Listas Ligadas Ordenadas

- Inserir ordenado x Ordenar a cada inserção
  - Custo?

Listas Ligadas Ordenadas

- Buscas em listas ordenadas
  - Pode-se tirar vantagem da ordenação

Listas Ligadas Ordenadas

```
ITEM *busca(LISTA *lista, int chave) {
    if (lista != NULL) {
        NO *aux = lista->cabeca->proximo;
        while (aux != NULL && (item_get_chave(aux->item) < chave)) {
            aux = aux->proximo;
            if (item_get_chave(aux->item) == chave)
                return(aux->item);
            else
                return (NULL);
        }
    }
}
```

Listas Ligadas Ordenadas

- Outras operações
  - Não deixam a lista desordenada!

Referências

- ZIVIANI, N. Projeto de Algoritmos, Thomson, 2a. Edição, 2004.