

SCC0202 – Algoritmos e Estrutura de Dados I

Listas Dinâmicas Encadeadas
circulares, com nó-cabeça, ordenadas

Prof.: Dr. Rudinei Goularte

(rudinei@icmc.usp.br)

Instituto de Ciências Matemáticas e de Computação - ICMC

Sala 4-229

Conteúdo



Listas Encadeadas com Nó Cabeça

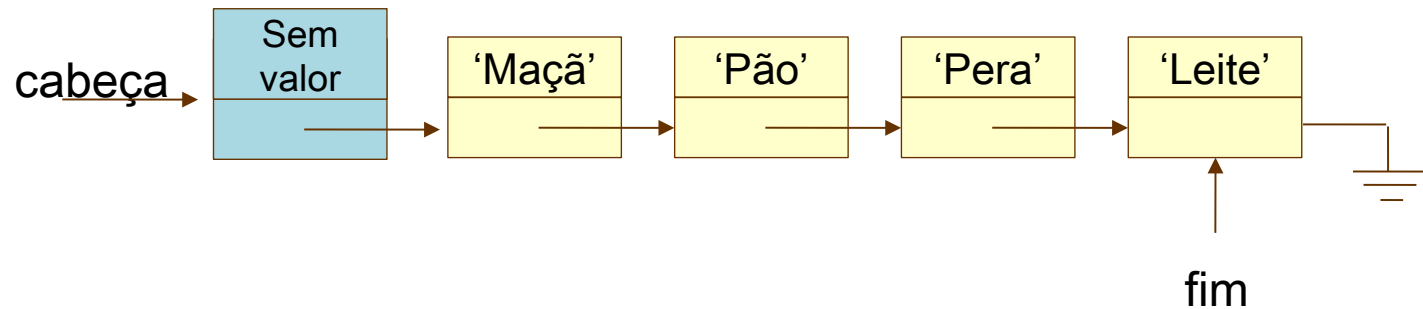
Listas Encadeadas Circulares

Listas Encadeadas Ordenadas

Listas Encadeadas 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
- 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 Encadeadas com Nó Cabeça



Listas Encadeadas com Nó Cabeça

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

```
1  struct lista_{
2      NO *cabeca;
3      NO *fim;
4      int tamanho;
5  };
6
7  bool lista_vazia(LISTA *lista) {
8      if (lista != NULL && (lista->cabeca->proximo == NULL))
9          return (true);
10     return (false);
11 }
```

Criar Lista

```
1 LISTA *lista_criar(void) {
2     LISTA *lista = (LISTA *) malloc(sizeof(LISTA));
3
4     if(lista != NULL) {
5         lista->cabeca = (NO *) malloc(sizeof(NO));
6         if (lista->cabeca == NULL)
7             return(NULL);
8         lista->cabeca->proximo = NULL;
9         lista->fim = NULL;
10        lista->tamanho = 0;
11    }
12    return(lista);
13 }
```

Implementação das Demais Operações

- A implementação das demais operações é similar a lista encadeada 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)

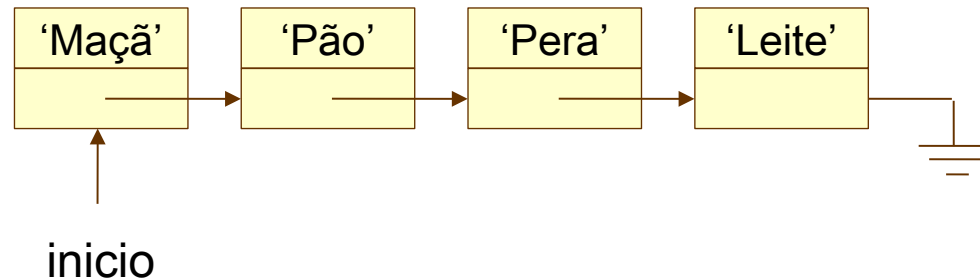
```
1 bool lista_remove(LISTA *lista, int chave) {
2     if (lista != NULL) {
3         NO *p = lista->cabeca;
4
5
6         while (p->proximo != NULL && (item_get_chave(p->proximo->item)) != chave) {
7
8             p = p->proximo;
9         }
10
11         if (p->proximo != NULL) {
12             NO *aux = p->proximo;
13             p->proximo = aux->proximo;
14             aux->proximo = NULL;
15
16             if (aux == lista->fim)
17                 lista->fim = p;
18             lista->tamanho --;
19
20             free(aux); aux = NULL;
21             return (true);
22         }
23     }
24     return (false);
25 }
```


Exercício

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

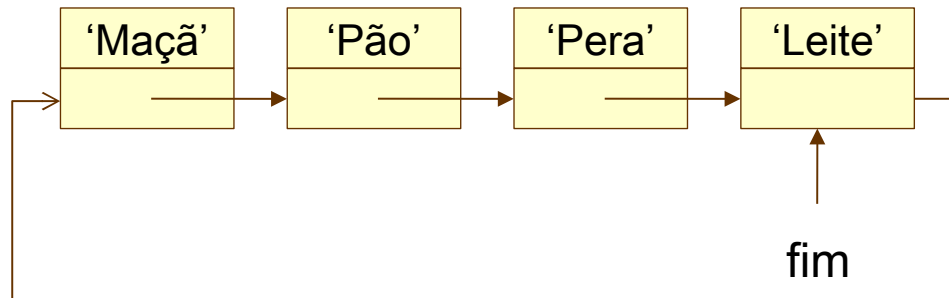
Listas Encadeadas Circulares

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



Listas Encadeadas Circulares

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

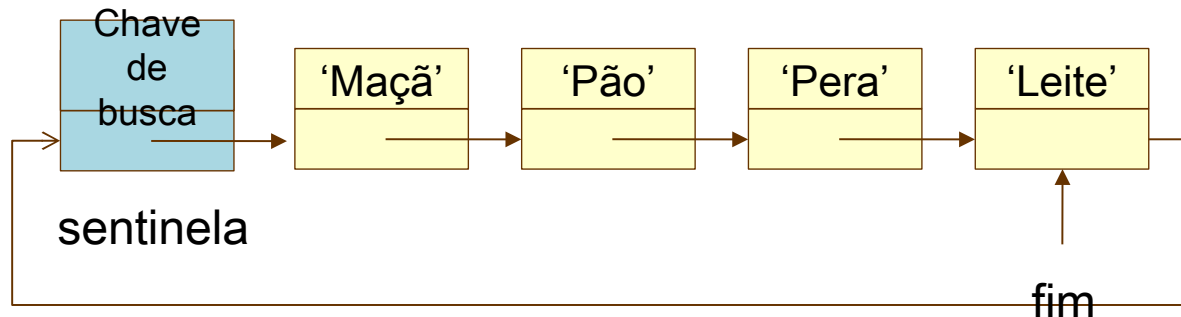


Listas Encadeadas Circulares

- 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 Encadeadas 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
- Se o item encontrado for o 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 Encadeadas Circulares (Sentinela)

```
1
2
3 struct lista_{
4     NO *sentinela;
5     NO *fim;
6     int tamanho;
7 };
8
9
10 ITEM *busca(LISTA *lista, int chave) {
11     item_set_chave(&(lista->sentinela->item), chave);
12     NO *p = lista->sentinela;
13
14     do {
15         p = p->proximo;
16     } while (item_get_chave(p->item) != chave);
17
18     return ((p != lista->sentinela) ? p->item : NULL);
19 }
```

Listas Encadeadas Ordenadas

- Suponha uma lista encadeada ordenada contendo os itens 6, 8 e 10. Inserir, ordenadamente, a sequência: 9, e 3.
- ▣ Na lousa...

Listas Encadeadas 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 Encadeadas Ordenadas

```
1 boolean lista_inserir_ordenado(LISTA *lista, ITEM *i) {
2     NO *p = NULL, *n = NULL;
3
4     if(lista != NULL && (!lista_cheia(lista)) ) {
5         p = lista->cabeca;
6
7         while(p->proximo != NULL && (item_get_chave(p->proximo->item) <
                                                    item_get_chave(i))
8             p = p->proximo;
9         n = (NO *) malloc (sizeof(NO)); n->item = i;
10        n->proximo = p->proximo;
11        p->proximo = n;
12        if (n->proximo == NULL)
13            lista->fim = n;
14        lista->tamanho++;
15        return(TRUE);
16    }
17    return(FALSE);
18}
```

Listas Encadeadas Ordenadas

- Inserir ordenado x Ordenar a cada inserção
 - ▣ Custo?

Listas Encadeadas Ordenadas


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

```
1  ITEM *busca(LISTA *lista, int chave) {  
2      if (lista != NULL) {  
3          NO *aux = lista->cabeca->proximo;  
4  
5          while (aux != NULL && (item_get_chave(aux->item) < chave)) {  
6              aux = aux->proximo;  
7              if (item_get_chave(aux->item) == chave)  
8                  return(aux->item);  
9          else  
10             return (NULL);  
11     }  
12 }
```

Listas Encadeadas Ordenadas



- Outras operações
 - ▣ Não deixam a lista desordenada!

- 
- Material baseado nos originais produzidos pelos professores:
 - ▣ Gustavo E. de A. P. A. Batista
 - ▣ Fernando V. Paulovich