SCC0202 – Algoritmos e Estrutura de Dados I

Listas Lineares Dinâmicas Duplamente Encadeadas

Prof.: Dr. Rudinei Goularte

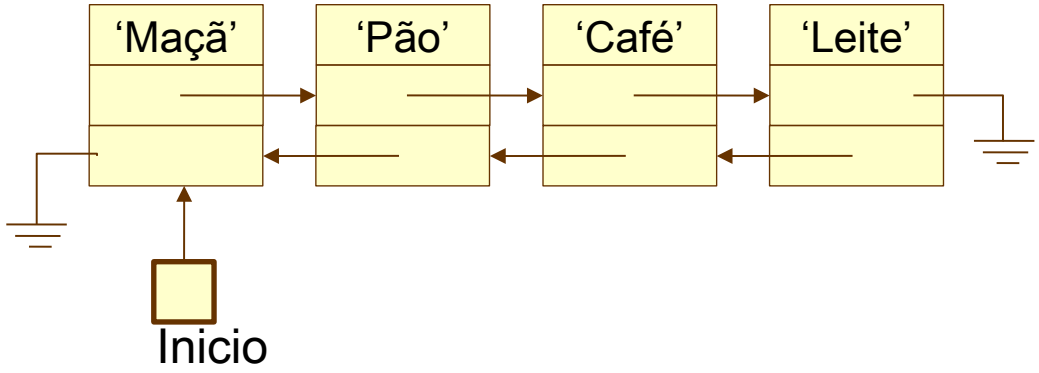
(rudinei@icmc.usp.br)

Instituto de Ciências Matemáticas e de Computação - ICMC

Sala 4-229

Listas Duplamente Encadeadas

- Nesta aula vamos implementar as operações do TAD Listas utilizando Listas Duplamente Encadeadas



Listas Duplamente Encadeadas

- Nas listas duplamente encadeadas, cada nó mantém um ponteiro para o nó anterior e posterior
- A manipulação da lista é mais complexa, porém algumas operações são diretamente beneficiadas
- Por exemplo, as operações de inserção e remoção em uma dada posição

Listas Duplamente Encadeadas

- Aplicações
  - Em geral, qualquer aplicação que necessite “navegação” em dois sentidos.
  - Exemplo: “Playlist” (com respectiva reprodução) de músicas.
    - Skip e Back.

TAD Listas Duplamente Encadeada

- Principais operações
  - Criar lista
  - Apagar lista
  - Verificar se a lista está vazia
  - Imprimir lista
  - Inserir item
  - Remover item (dado uma chave)
  - Recuperar item (dado uma chave)

  - **Interface (.h) da lista é a mesma!!!**

Listas Duplamente Encadeadas

```
1 /*listaDupla.c*/
2  typedef strcut no_ NO;
3  struct no_{
4    ITEM *item;
5    NO *anterior;
6    NO *proximo;
7  };
8
9  struct lista_{
10    NO *inicio;
11    NO *fim;
12    int tamanho;  /*tamanho da lista*/
13 };

...
```

Operações Básicas

- As operações de criar e apagar a lista são simples

```
1 LISTA *lista_criar(void){
2 /*pré-condição: existir espaço na memória.*/
3
4    LISTA *lista = (LISTA *) malloc(sizeof(LISTA));
5    if(lista != NULL) {
6        lista->inicio = NULL;
7        lista->fim = NULL;
8        lista->tamanho = 0;
9    }
10
11  return (lista);
12}
```

  - Nada muda em relação à implementação da lista simplesmente encadeada... (reuso!)

Operações Básicas

- As operações de criar e apagar a lista são simples

```
/*recebe o inicio da lista como argumento e esvazia a mesma*/
8 void lista_esvazia (NO *ptr){
9     if (ptr != NULL){
10       if(ptr->proximo != NULL)
11          lista_esvazia(ptr->proximo);
12
13       item_apagar(&ptr->item);
14       ptr->anterior = NULL;
15       free(ptr); /* apaga o nó*/
16       ptr = NULL;
17    }
18 }

1 void lista_apagar(LISTA **ptr){
2     if (*ptr == NULL)
3        return;
4     lista_esvazia((*ptr)->inicio);
5     free(*ptr);
6     *ptr = NULL;
7 }
```

Operações Básicas

- Inserção
  - Em listas não ordenadas
    - No início ou no fim da lista.
  - Em listas ordenadas
    - Inserir ordenadamente.
- Remoção
  - Em qualquer posição da lista

- Essas implementação são ligeiramente diferentes das implementações para listas simplesmente encadeadas.

Inserir Item (Primeira Posição)

```
/*Insere um novo nó no início da lista. PARA LISTAS NÃO ORDENADAS*/
1 bool lista_inserir_inicio(LISTA *lista, ITEM *i){
2    if ((lista != NULL) && (!lista_cheia(lista)) ) {
3        NO *pnovo = (NO *) malloc(sizeof (NO));
4        pnovo->item = i;
5       if (lista->inicio == NULL){
6          //lista->inicio = pnovo;
7          lista->fim = pnovo;
8          pnovo->proximo = NULL;
9        }
10       else {
11         lista->inicio->anterior = pnovo;
12         pnovo->proximo = lista->inicio;
13       }
14       pnovo->anterior = NULL;
15       lista->inicio = pnovo;
16       lista->tamanho++;
17      return (true);
18   } else
19      return (false);
20 }
```

Inserir Item (Última Posição)

```
/*Insere um novo nó no fim da lista. PARA LISTAS NÃO ORDENADAS*/
1 bool lista_inserir_fim(LISTA *lista, ITEM *item){
2    if ((lista != NULL) && (!lista_cheia(lista)) ) {
3           NO *pnovo = (NO *) malloc(sizeof (NO));
4        pnovo->item = item;
5        if (lista->inicio == NULL){
6           lista->inicio = pnovo;
7           pnovo->anterior = NULL;
8        }
9        else {
10          lista->fim->proximo = pnovo;
11             pnovo->anterior = lista->fim;
12       }
13       pnovo->proximo = NULL;
14       lista->fim = pnovo;
15       lista->tamanho++;
16       return (true);
17   } else
18      return (false);
19 }
```

Remover Item (dado uma chave)

```
1 ITEM *lista_remover(LISTA *lista, int chave){
2    NO *p=NULL;
3    if ( (lista != NULL) && (!lista_vazia(lista)) ){
4       p = lista->inicio;
5       while(p != NULL && (item_chave(p->item) != chave) ) /*Percorre a lista em
6           p = p->proximo;                                  busca da chave*/
7       if(p != NULL){ /*Se a lista não acabou significa que encontrou a chave*/
8          if(p == lista->inicio) /*Se é o 1º da lista basta acertar o ptr inicio*/
9             lista->inicio = p->proximo;
10         else  /*Se não é o 1º da lista, há alguém antes dele para acertar o ptr*/
11            p->anterior->proximo = p->proximo;
12         if(p == lista->fim) /* Ideia do if/else anterior para o fim da lista */
13            lista->fim = p->anterior;
14         else
15            p->proximo->anterior = p→anterior;
16         ITEM *it = p->item;
17         p->proximo = NULL;  p->anterior = NULL;
18         free(p); lista->tamanho--;
19         return(it);
20      }
21   }
22   return(NULL); /*elemento (chave) não está na lista ou lista vazia*/
23 }
```

Exercício (já caiu em provas passadas...)

- Implementar a função para inserir **ordenadamente** itens no TAD lista dinâmica duplamente encadeada

- Fim