Listas Duplamente Ligadas

SCC0202 - Algoritmos e Estruturas de Dados I

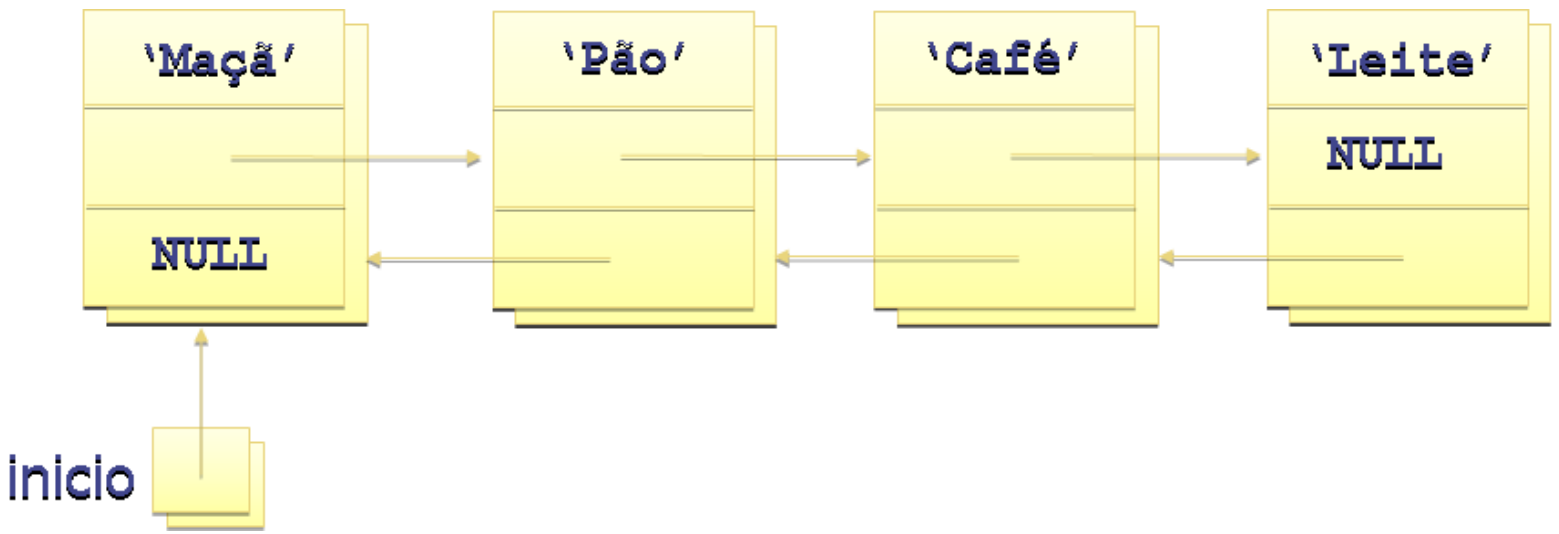
Prof. Fernando V. Paulovich
http://www.icmc.usp.br/~paulovic
paulovic@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

11 de setembro de 2013

Listas Duplamente Ligadas

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



Listas Duplamente Ligadas

- Nas listas duplamente ligadas, cada nó mantém um ponteiro para o nó anterior e posterior

Listas Duplamente Ligadas

- Nas listas duplamente ligadas, cada nó mantém um ponteiro para o nó anterior e posterior
- A manipulação da lista é mais complexa, porém algumas operações são diretamente beneficiadas

Listas Duplamente Ligadas

- Nas listas duplamente ligadas, 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 Ligadas

```
1  typedef struct lista_ligada LISTA_LIGADA;
2
3  typedef struct NO {
4    ITEM *item;
5    struct NO *proximo;
6    struct NO *anterior;
7  } NO;
8
9  struct lista_ligada {
10   NO *inicio;
11   NO *fim;
12   int tamanho;
13 };
```

TAD Listas Duplamente Ligadas

Principais operações

- Criar lista
- Apagar lista
- Verificar se a lista está vazia
- Imprimir lista
- Inserir item (última posição)
- Inserir item (primeira posição)
- Remover item (dado uma chave)
- Recuperar item (dado uma chave)
- Contar número de itens

Operações Básicas

- As operações de criar e apagar a lista são simples

```
1  LISTA_LIGADA *criar_lista() {
2    LISTA_LIGADA *lista = (LISTA_LIGADA *)malloc(sizeof (LISTA_LIGADA));
3
4    if(lista != NULL) {
5      lista->inicio = NULL;
6      lista->fim = NULL;
7      lista->tamanho = 0;
8    }
9
10   return lista;
11 }
```

Operações Básicas

- As operações de criar e apagar a lista são simples

```
1  void apagar_no(NO *no) {
2    apagar_item(&no->item);
3    free(no);
4  }
5
6  void apagar_lista(LISTA_LIGADA **lista) {
7    NO *paux = (*lista)->inicio;
8    NO *prem = NULL;
9
10   while(paux != NULL) {
11     prem = paux;
12     paux = paux->proximo;
13     apagar_no(prem);
14   }
15
16   free(*lista);
17   *lista=NULL;
18 }
```

Inserir Item (Primeira Posição)

```
1  int inserir_item_inicio(LISTA_LIGADA *lista, ITEM *item) {
2    if(!vazia(lista)) {
3      NO *pnovo = (NO *)malloc(sizeof(NO));
4
5      if(pnovo != NULL) {
6        pnovo->item = item;
7        pnovo->anterior = NULL;
8        pnovo->proximo = lista->inicio;
9
10       if(lista->inicio == NULL) {
11         lista->fim = pnovo;
12       } else {
13         lista->inicio->anterior = pnovo;
14       }
15
16       lista->inicio = pnovo;
17       lista->tamanho++;
18     }
19     return 1;
20   }
21   return 0;
22 }
```

Inserir Item (Última Posição)

```
1  int inserir_item_fim(LISTA_LIGADA *lista, ITEM *item) {
2    if(!cheia(lista)) {
3      NO *pnovo = (NO *)malloc(sizeof(NO));
4
5      if(pnovo != NULL) {
6        pnovo->item = item;
7        pnovo->proximo = NULL;
8        pnovo->anterior = lista->fim;
9
10       if(lista->inicio == NULL) {
11         lista->inicio = pnovo;
12       } else {
13         lista->fim->proximo = pnovo;
14       }
15
16       lista->fim = pnovo;
17       lista->tamanho++;
18       return 1;
19     }
20   }
21   return 0;
22 }
```

Remover Item (dado uma chave)

```
1  int remover_item(LISTA_DUPLAMENTE_LIGADA *lista, int chave) {
2    if (!vazia(lista)) {
3      NO *prem = lista->inicio;
4      while(prem != NULL && prem->item->chave != chave) {
5        prem = prem->proximo;
6      }
7
8      if(prem != NULL) {
9        if(prem != lista->inicio) {
10         prem->anterior->proximo = prem->proximo;
11       } else {
12         lista->inicio = prem->proximo;
13       }
14
15       if(prem != lista->fim) {
16         prem->proximo->anterior = prem->anterior;
17       } else {
18         lista->fim = prem->anterior;
19       }
20
21       lista->tamanho--;
22       apagar_no(prem);
23       return 1;
24     }
25   }
26   return 0;
27 }
```

Exercício

- Se a lista for duplamente encadeada circular, as exceções no momento da remoção são evitadas

Exercício

- Se a lista for duplamente encadeada circular, as exceções no momento da remoção são evitadas
- Se a lista apresentar um nó cabeça (sentinela), a busca pode ser melhorada

Exercício

- Se a lista for duplamente encadeada circular, as exceções no momento da remoção são evitadas
- Se a lista apresentar um nó cabeça (sentinela), a busca pode ser melhorada
- Implemente todas as operações do TAD lista (apresentadas anteriormente) usando uma lista circular duplamente encadeada com nó cabeça