



Estruturas de Informação

Aula 10: Listas (parte 2)

23/04/2009

Fontes Bibliográficas



- Livros:
 - Projeto de Algoritmos (Nivio Ziviani): Capítulo 3;
 - Introdução a Estruturas de Dados (Celes, Cerqueira e Rangel): Capítulo 10;
 - Estruturas de Dados e seus Algoritmos (Szwarcfiter, et. al): Capítulo 2;
 - Algorithms in C (Sedgewick): Capítulo 3;
- Slides baseados nas transparências disponíveis em:
<http://www.dcc.ufmg.br/algoritmos/transparencias.php>

TAD Lista (1)



/* Faz a lista ficar vazia */

- FLVazia(Lista).
 - Input: L (Lista)
 - Output: L'
 - Pré-condição: L é definida
 - Pós-condição: L' é definida e vazia

/* Insere x após o último elemento da lista */

- Insere(x, Lista). Insere x após o último
 - Input: x (Item da Lista) e L (Lista)
 - Output: L'
 - Pré-condição: L é definida e x é um Item válido da lista
 - Pós-condição: L' é definida e vazia e o elemento item de L' é igual a x

TAD Lista (2)



/*Retorna o item x que está na posição p da lista, retirando-o da lista e deslocando os itens a partir da posição p+1 para as posições anteriores */

- Retira(p, Lista, x)
 - Input: p (posição válida da lista) e L (Lista)
 - Output: x (item da lista da posição p)
 - Pré-condição: L é definida e p é uma posição válida da lista
 - Pós-condição: L' é a lista L sem o item x, com todos os itens deslocados de uma posição

TAD Lista (3)



/*Verifica se a lista está vazia*/

- Vazia(Lista)
 - Input: L (Lista)
 - Output: B (*true* se lista vazia; senão retorna *false*)
 - Pré-condição: L é definida
 - Pós-condição: L não é modificada
- /*Imprime os itens da lista na ordem de ocorrência */
- Imprime(Lista)
 - Input: L (Lista)
 - Output: nenhum
 - Pré-condição: L é definida e não está vazia
 - Pós-condição: L não é modificada e seus elementos são impressos

Implementação de Listas Lineares



- Há varias maneiras de implementar listas lineares.
- Cada implementação apresenta vantagens e desvantagens particulares.
- Vamos estudar duas maneiras distintas
 - Usando alocação sequencial e estática (com vetores).
 - Usando alocação não sequencial e dinâmica (com ponteiros): *Estruturas Encadeadas*.

Listas Lineares em Alocação Sequencial e Estática (2)



Itens	
Primeiro = 1	x_1
2	x_2
	$\vdots$
Último-1	x_n
	$\vdots$
MaxTam	

Estrutura de Listas com Alocação Sequencial e Estática (2) – lista.h



```
typedef int TipoChave;
typedef int Posicao;
typedef struct tipoitem TipoItem;
typedef struct tipolista TipoLista;

TipoLista* InicializaLista();
void FLVazia (TipoLista* Lista);
int Vazia (TipoLista* Lista);
void Insere (TipoItem* x, TipoLista* Lista);
void Retira (Posicao p, TipoLista* Lista,
             TipoItem* Item);
void Imprime (TipoLista* Lista);
void SetTipoChave (TipoItem* x, TipoChave n);
```

Estrutura de Listas com Alocação Sequencial e Estática (3) - lista.c



```
#include <stdio.h>
#include "lista.h"

#define InicioVetor 0
#define MaxTam 1000

struct tipoitem {
    TipoChave Chave;
    /* outros componentes */
};

struct tipolista{
    TipoItem Item[MaxTam];
    Posicao Primeiro, Ultimo;
};
```

Listas com alocação não sequencial e dinâmica

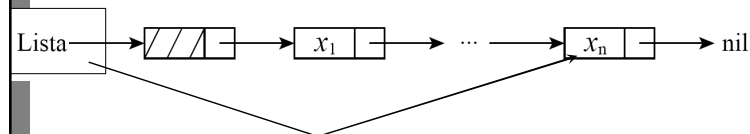


- Cada item é encadeado com o seguinte mediante uma variável do tipo Ponteiro.
- Permite utilizar posições não contíguas de memória.
- É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.
- Há uma célula cabeça para simplificar as operações sobre a lista
- Estrutura Encadeada

Listas com alocação não sequencial e dinâmica



- Cada item é encadeado com o seguinte mediante uma variável do tipo Ponteiro.
- Permite utilizar posições não contíguas de memória.
- É possível inserir e retirar elementos sem necessidade de deslocar os itens seguintes da lista.
- Há uma célula cabeça para simplificar as operações sobre a lista



Estrutura da Lista com Alocação não Sequencial e Dinâmica



- A lista é constituída de células.
- Cada célula contém um item da lista e um ponteiro para a célula seguinte.
- O registro (struct) TipoLista contém um ponteiro para a célula cabeça e um ponteiro para a última célula da lista.

Estrutura da Lista com Alocação não Sequencial e Dinâmica (2) – lista.h



```
typedef int TipoChave;
typedef int Posicao;
typedef struct tipoitem TipoItem;
typedef struct tipolista TipoLista;

TipoLista* InicializaLista();
void FLVazia (TipoLista* Lista);
int Vazia (TipoLista* Lista);
void Insere (TipoItem* x, TipoLista* Lista);
void Retira (Posicao p, TipoLista* Lista,
             TipoItem* Item);
void Imprime (TipoLista* Lista);
void SetTipoChave (TipoItem* x, TipoChave n);
```

Estrutura da Lista com Alocação não Sequencial e Dinâmica (2) – arquivo.c



```
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

struct tipoitem{
    TipoChave Chave;
    /* outros componentes */
};

struct Celula_str {
    TipoItem Item;
    Ponteiro Prox;
};

struct tipolista{
    Ponteiro Primeiro, Ultimo;
};
```

Implementação TAD Lista com Ponteiros



```
TipoLista* InicializaLista()
{
    TipoLista* lista =
        (TipoLista*)malloc(sizeof(TipoLista));
    return lista;
}
```

Implementação TAD Lista com Ponteiros



```
void FLVazia (TipoLista *Lista)
{
    Lista->Primeiro = (Ponteiro) malloc (sizeof
        (Celula));
    Lista->Ultimo = Lista->Primeiro;
    Lista->Primeiro->Prox = NULL;
}

int Vazia (TipoLista* Lista)
{
    return (Lista->Primeiro == Lista->Ultimo);
}
```

Implementação TAD Lista com Ponteiros (2)



```
void Insere (TipoItem* x, TipoLista
*Lista)
{
    Lista->Ultimo->Prox = (Ponteiro)
    malloc(sizeof(Celula));
    Lista->Ultimo = Lista->Ultimo->Prox;
    Lista->Ultimo->Item = *x;
    Lista->Ultimo->Prox = NULL;
}
```

Implementação TAD Lista com Ponteiros (3)



```
// retorna o tamanho da lista, tirando célula
cabeça
int tamanho (TipoLista* Lista)
{
    Ponteiro p;
    int i = 0;
    for (p=Lista->Primeiro->Prox; p!=NULL; p=p-
    >Prox)
        i++;
    return i;
}
```

Implementação TAD Lista com Ponteiros (3)



```
/* O item retirado é o seguinte apontado por p */
void Retira (Posicao p, TipoLista *Lista, TipoItem *Item)
{
    Ponteiro q, q2;
    int i;
    int t = tamanho (Lista);
    if (Vazia(Lista) || p>=t)
    {
        printf ("ERRO: Lista vazia ou posicao nao existe\n");
        return;
    }
    q = Lista->Primeiro;
    //encontra o elemento antes da posicao desejada
    for (i=0; i<p; i++)
        q = q->Prox;
    q2 = q->Prox;
    *Item = q2->Item;
    q->Prox = q2->Prox;
    if (q2->Prox == NULL) Lista->Ultimo = q2;
    free (q2);
}
```

Implementação TAD Lista com Ponteiros(4)



```
void Imprime (TipoLista* Lista)
{
    Ponteiro Aux;
    Aux = Lista->Primeiro->Prox;
    while (Aux != NULL)
    {
        printf ("%d\n", Aux->Item.Chave);
        Aux = Aux->Prox;
    }
}
```

Lista com alocação não sequencial e dinâmica:
vantagens e desvantagens



- Vantagens:
 - Permite inserir ou retirar itens do meio da lista a um custo constante (importante quando a lista tem de ser mantida em ordem).
 - Bom para aplicações em que não existe previsão sobre o crescimento da lista (o tamanho máximo da lista não precisa ser definido *a priori*).
- Desvantagem: utilização de memória extra para armazenar os ponteiros.