



Estruturas de Dados

Aula 9: Listas (parte 1)

24/06/2013

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>



Listas Lineares

- Forma simples de interligar os elementos de um conjunto.
- Agrupa informações referentes a um conjunto de elementos que se relacionam entre si de alguma forma.
- São úteis em aplicações tais como manipulação simbólica, gerência de memória, simulação e compiladores.
- Inúmeros tipos de dados podem ser representados por listas. Alguns exemplos de sistemas de informação são: informações sobre os funcionários de uma empresa, notas de alunos, itens de estoque, etc.

Listas Lineares (2)



- Estrutura em que as operações **inserir, retirar e localizar** são definidas.
- Itens da lista podem ser **acessados, inseridos ou retirados**.
- Podem **crescer ou diminuir** de tamanho durante a execução de um programa, de acordo com a demanda.
- Duas listas podem ser **concatenadas** para formar uma lista única, ou uma pode ser **partida** em duas ou mais listas.
- Podem ser adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.

Definição Lista Lineares



- Seqüência de zero ou mais itens $x_1; x_2; \dots; x_n$, na qual x_i é de um determinado tipo e n representa o tamanho da lista linear.
- Sua principal propriedade estrutural envolve as posições relativas dos itens em uma dimensão.
 - Assumindo $n \geq 1$, x_1 é o primeiro item da lista e x_n é o último item da lista.
 - x_i precede x_{i+1} para $i = 1; 2; \dots; n - 1$
 - x_i sucede x_{i-1} para $i = 2; 3; \dots; n$
 - o elemento x_i é dito estar na i -ésima posição da lista.

TAD Lista: Exemplos



- Exemplos de operações possíveis:
 - Criar uma lista linear vazia.
 - Inserir um novo item imediatamente após o i -ésimo item.
 - Retirar o i -ésimo item.
 - Localizar o i -ésimo item para examinar e/ou alterar o conteúdo de seus componentes.
 - Combinar duas ou mais listas lineares em uma lista única.
 - Partir uma lista linear em duas ou mais listas.
 - Fazer uma cópia da lista linear.
 - Ordenar os itens da lista em ordem ascendente ou descendente, de acordo com alguns de seus componentes.
 - Pesquisar a ocorrência de um item com um valor particular em algum componente.

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

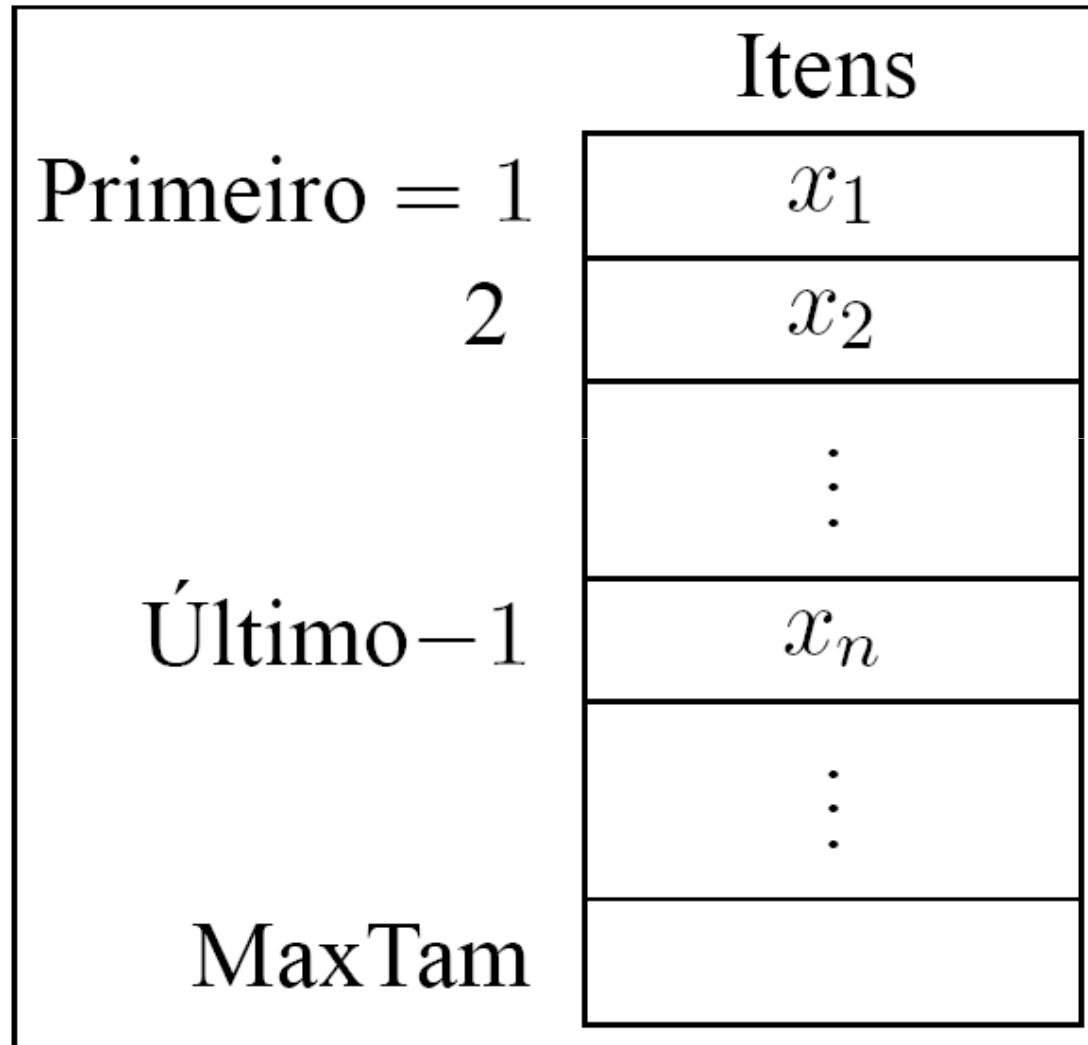


- Há varias maneiras de implementar listas lineares.
- Cada implementação apresenta vantagens e desvantagens particulares.
- Vamos estudar duas maneiras distintas
 - Usando alocação **sequencial e estática** (com **vetores**).
 - Usando alocação **não sequencial e dinâmica** (com **ponteiros**): *Estruturas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



- Armazena itens em **posições contíguas de memória.**
- A lista pode ser percorrida em qualquer direção.
- A inserção de um novo item pode ser realizada após o último item com custo constante.
- A inserção de um novo item no meio da lista requer um deslocamento de todos os itens localizados após o ponto de inserção.
- Retirar um item do início da lista requer um deslocamento de itens para preencher o espaço deixado vazio.

Listas Lineares em Alocação Seqüencial e Estática (2)



Estrutura de Listas com Alocação Seqüencial e Estática



- Os itens são armazenados em um **vetor** de tamanho suficiente para armazenar a lista.
- O campo **Último** contém a posição após o último elemento da lista.
- O i-ésimo item da lista está armazenado na i-ésima posição do vetor, $0 \leq i \leq \text{Último}$.
- A constante **MaxTam** define o tamanho máximo permitido para a lista.

Estrutura de Listas com Alocação Seqüencial e Estática (2) – arquivo.h



```
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);  
TipoItem* Retira (Posicao p, TipoLista* Lista);  
void Imprime (TipoLista* Lista);  
TipoItem* InicializaTipoItem();  
void ModificaValorItem (TipoItem* x, int valor);  
void ImprimeTipoItem(TipoItem* x);
```

Estrutura de Listas com Alocação Seqüencial e Estática (3) - arquivo.c



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

#define InicioVetor 0
#define MaxTam 1000

struct tipoitem {
    int valor;
    /* outros componentes */
};

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

Implementação TAD Lista com Vetores



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


Implementação TAD Lista com Vetores



```
/* Faz a lista ficar vazia */
```

```
void FLVazia (TipoLista* Lista)  
{  
    Lista->Primeiro = InicioVetor;  
    Lista->Ultimo = Lista->Primeiro;  
}
```

```
/*Verifica se a lista está vazia*/
```

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

Implementação TAD Lista com Vetores (2)



```
/* Insere x após o último elemento da
   lista */
void Insere (TipoItem* x, TipoLista
            *Lista)
{
    if (Lista ->Ultimo >= MaxTam)
        printf ("Lista está cheia\n");
    else
    {
        Lista ->Item[Lista->Ultimo] = *x;
        Lista->Ultimo++;
    }
}
```

Implementação TAD Lista com Vetores (3)



*/*Opção que não modifica o lista.h */*

```
TipoItem* Retira (Posicao p, TipoLista* Lista)
{
    int Aux; TipoItem* item;
    item = (TipoItem*) malloc(sizeof(TipoItem));
    if (Vazia(Lista) || p >= Lista->Ultimo)
    {
        printf ("A posição não existe!\n");
        return NULL;
    }
    *item = Lista->Item[p]; Lista->Ultimo--;
    for (Aux = p; Aux < Lista->Ultimo; Aux++)
        Lista->Item[Aux] = Lista->Item[Aux+1];
    return item;
}
```

Implementação TAD Lista com Vetores (3)



*/*Opção que modifica o lista.h */*

```
void Retira (Posicao p, TipoLista *Lista, TipoItem* item)
{
    int Aux;
    if (Vazia(Lista) || p >= Lista->Ultimo)
    {
        printf ("A posição não existe!\n");
        return;
    }
    *item = Lista->Item[p]; Lista->Ultimo--;
    for (Aux = p; Aux < Lista->Ultimo; Aux++)
        Lista->Item[Aux] = Lista->Item[Aux+1];
    return;
}
```

Implementação TAD Lista com Vetores(4)



```
/*Imprime os itens da lista na ordem de ocorrência */  
void Imprime (TipoLista* Lista)  
{  
    int Aux;  
    printf ("Imprime Lista Estatica: ");  
  
    for (Aux = Lista->Primeiro; Aux < Lista->Ultimo;  
        Aux++)  
    {  
        printf ("%d\n", Lista->Item[Aux].valor);  
    }  
}
```

TipoItem



- Como o TipoItem é opaco, precisamos de operações no TAD que manipulam este tipo:
 - InicializaTipoItem: cria um TipoItem
 - ModificaValorTipoItem: modifica o campo valor de um TipoItem
 - ImprimeTipoItem: Imprime o campo valor de um TipoItem

TipoItem (cont.)



```
TipoItem* InicializaTipoItem() {  
    TipoItem* item = (TipoItem*)malloc(sizeof(TipoItem));  
    return item;  
}
```

```
void ModificaValorItem (TipoItem* item, int valor) {  
    item->valor = valor;  
}
```

```
void ImprimeTipoItem (TipoItem* item){  
    printf ("Campo valor: %d ", item->valor);  
}
```

Lista com alocação sequencial e estática: vantagens e desvantagens



- Vantagem: economia de memória (os ponteiros são implícitos nesta estrutura).
- Desvantagens:
 - custo para inserir ou retirar itens da lista, que pode causar um deslocamento de todos os itens, no pior caso;
 - em aplicações em que não existe previsão sobre o crescimento da lista, a utilização de arranjos em linguagens como o Pascal e o C pode ser problemática pois, neste caso, o tamanho máximo da lista tem de ser definido em tempo de compilação.

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

