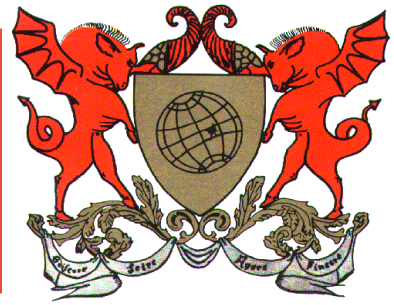


Universidade Federal de Viçosa
Campus de Florestal

Algoritmos e Estruturas de Dados I (CCF 211)

Listas Lineares (Cap03 – Seção 3.1 - Ziviani)

Profa.Thais R. M. Braga Silva
<thais.braga@ufv.br>

Listas Lineares

- Uma das formas mais simples de interligar os elementos de um conjunto (ordem sequêncial)
- Estrutura em que as operações inserir, retirar e localizar são definidas
- Podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda
- Itens podem ser acessados, inseridos ou retirados de uma lista

Listas Lineares

- Duas listas podem ser concatenadas para formar uma lista única, ou uma pode ser partida em duas ou mais listas
- Adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível
- São úteis em aplicações tais como manipulação simbólica, gerência de memória, simulações e compiladores.

Definição de Listas Lineares

- **Sequência de zero ou mais itens $x_1, x_2, \cdots, x_n$**
  - 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 $i=1,2,\cdots,n-1$ (x_n não possui sucessor)
  - x_i sucede x_{i-1} para $i = 2,3,\cdots,n$ (ninguém precede x_1)
  - o elemento x_i está na i-ésima posição da lista

TAD Lista Linear

- O que deveria conter?
  - Estrutura de dados para representar a lista linear
  - Conjunto de operações que atuam sobre a lista

- Quais operações deveriam fazer parte deste conjunto?

O conjunto de operações a ser definido depende de cada aplicação.

TAD Lista Linear

- **Um conjunto de operações necessário a uma maioria de aplicações é:**

1) Criar uma lista linear vazia
2) Inserir um novo item imediatamente após o i-ésimo item
3) Retirar o i-ésimo item
4) Localizar o i-ésimo item para examinar e/ou alterar o conteúdo de seus componentes (percorrer ou acessar)
5) Combinar duas ou mais listas lineares em uma lista única

TAD Lista Linear

- **Um conjunto de operações necessário a uma maioria de aplicações é (continua):**

6) Dividir uma lista linear em duas ou mais listas
7) Fazer uma cópia da lista linear
8) Ordenar os itens da lista em ordem ascendente ou descendente, de acordo com alguns de seus componentes
9) Pesquisar a ocorrência de um item com um valor particular em algum componente

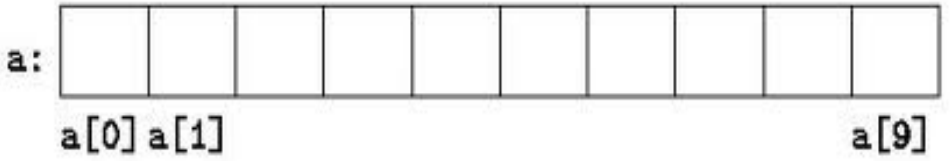
Exemplo de Protótipo para Operações

- **Exemplo de Conjunto de Operações:**
  1) **FLVazia(Lista)**. Faz a lista ficar vazia
  2) **LInsere(Lista, x)**. Insere x após o último item da lista
  3) **LRetira(Lista, p, x)**. 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
  4) **LEhVazia(Lista)**. Esta função retorna **true** se lista vazia; senão retorna **false**
  5) **LImprime(Lista)**. Imprime os itens da lista na ordem de ocorrência

Implementações de Listas Lineares

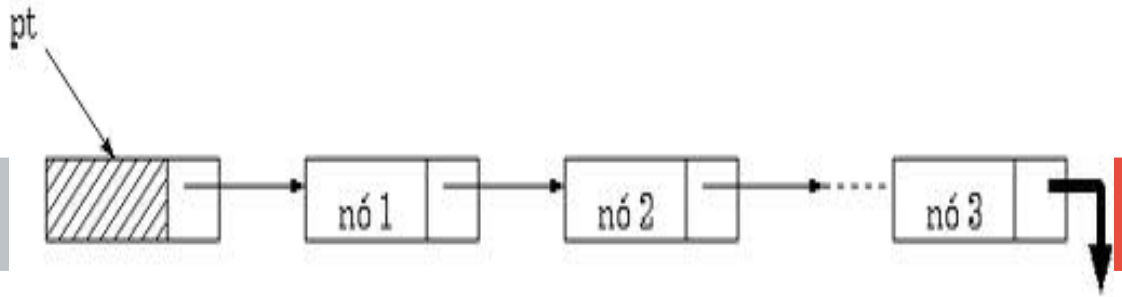
- Várias estruturas de dados podem ser usadas para representar listas lineares, cada uma com vantagens e desvantagens particulares

- As duas representações mais utilizadas são as implementações por meio de arranjos e de apontadores (estruturas auto-referenciadas – lista encadeada)

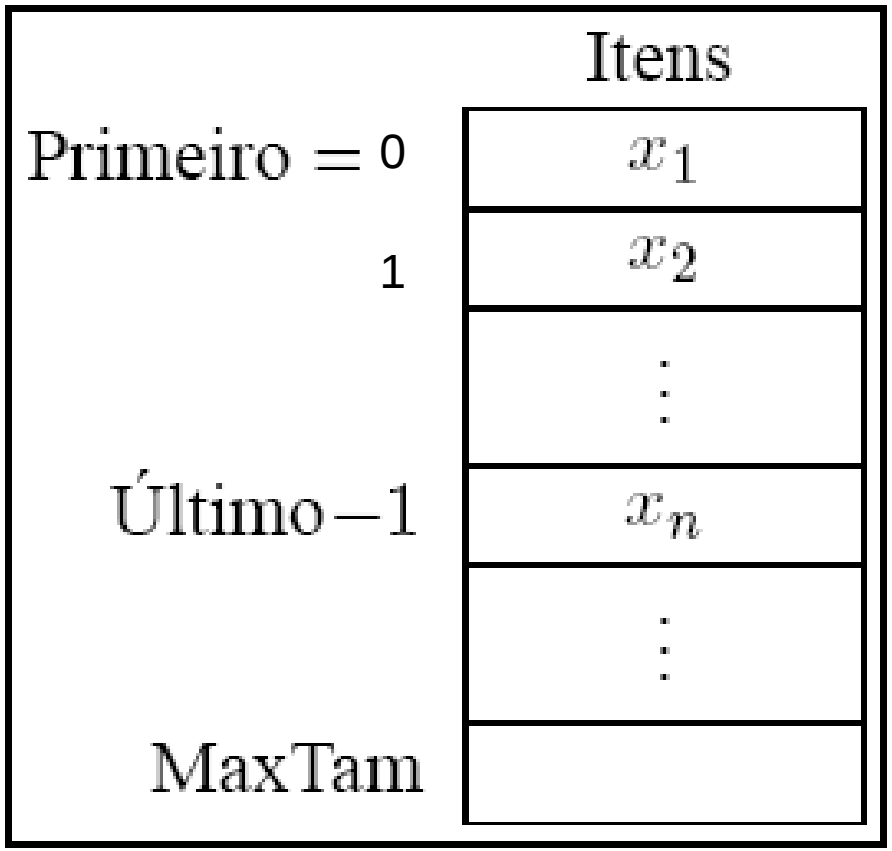


Arranjo / Vetor

Lista Encadeada



Implementação de Lista por meio de Arranjo



- Os itens da lista são armazenados 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 deslocamento de itens para preencher o espaço deixado vazio

Estrutura da Lista Usando Arranjo

- Os itens são armazenados em um arranjo de tamanho suficiente para armazenar a lista
- O campo Último aponta para a posição seguinte a do último elemento da lista, ou seja, para primeira posição disponível (facilita semântica para lista vazia)
- O *i*-ésimo item da lista está armazenado na (*i* - 1)-ésima posição do array, $0 \leq i < \text{Último}$
- A constante MaxTam define o tamanho máximo permitido para a lista

Arranjos em Linguagem C

- Conjunto de variáveis do mesmo tipo
- Declaração *<tipo> <nome> [<tamanho>];*
- Compilador aloca espaço de memória igual ao número de bytes do tipo vezes tamanho do vetor
- Bytes alocados de maneira contígua na memória
- Índice iniciando de 0 para acessar posições do arranjo
- Limites do arranjo não são verificados pelo compilador na indexação

Arranjos como Ponteiros - Linguagem C

- Ao declarar um arranjo, o compilador reserva o espaço de memória, cria um ponteiro do tipo do vetor e o aponta para a primeira posição alocada

- nome_vetor[0] é equivalente à *(nome_vetor)
- nome_vetor[índice] é equivalente à *(nome_vetor+índice)

Arranjos como Ponteiros - Linguagem C

- Ponteiro que aponta para um vetor pode ser indexado

```
#include <stdio.h>

int main ()

{

        int matrx [10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        int *p;

        p=matrx;

        printf ("O terceiro elemento do vetor e: %d",p[2]);

        return(0);

}
```

Estrutura da Lista Usando Arranjo

arquivo.h

```
#include <stdio.h>
#include <stdlib.h>
#define InicioArranjo   0
#define MaxTam          1000

typedef int TChave;

typedef int Apontador;

typedef struct {
  TChave Chave;
  /* outros componentes */
} TItem;

typedef struct {
  TItem Item[MaxTam];
  Apontador Primeiro, Ultimo;
} TLista;
```

Estrutura da Lista Usando Arranjo

arquivo.h

```
void FLVazia(TLista* pLista);

int LEhVazia(TLista* pLista);

int LInsere(TLista* pLista, TItem x);

int LRetira(TLista* pLista, Apontador p, TItem *pX);

void LImprime(TLista* pLista);
```

Operações sobre Lista Usando Arranjo

arquivo.c

```
void FLVazia(TLista* pLista)
{
  pLista->Primeiro = InicioArranjo;
  pLista->Ultimo = pLista->Primeiro;
}  /* FLVazia */

int LEhVazia(TLista* pLista)
{
  return (pLista->Ultimo == pLista->Primeiro);
}  /* LEhVazia */

int LInsere(TLista* pLista, TItem x)
{
  if (pLista->Ultimo == MaxTam)
    return 0; /* lista cheia */
  pLista->Item[pLista->Ultimo++] = x;
  return 1;
}  /* LInsere */
```

Operações sobre Lista Usando Arranjo

arquivo.c

```
int LRetira(TLista* pLista, Apontador p, TItem *pX)
{
  int cont;

  if (LEhVazia(pLista) || p >= pLista->Ultimo || p < 0)
    return 0;

  *pX = pLista->Item[p];
  pLista->Ultimo--;
  for (cont = p+1; cont <= pLista->Ultimo; cont++)
    pLista->Item[cont - 1] = pLista->Item[cont];
  return 1;
}  /* LRetira */
```

Operações sobre Lista Usando Arranjo

arquivo.c

```
void LImprime(TLista* pLista)
{
  int i;
  for (i = pLista->Primeiro; i < pLista->Ultimo; i++)
    printf("%d\n", pLista->Item[i].Chave);
}  /* LImprime */
```

Lista Usando Arranjo
Vantagens e Desvantagens

- **Vantagem:**
  - economia de memória (os apontadores 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 C pode ser problemática porque o tamanho máximo da lista tem de ser definido em tempo de compilação

Exercícios

- Implementar um TAD que represente algum item (ex: carro, livro, pessoa). Implementar o TAD Lista Linear para esse tipo de item usando arranjo como ED.
- Implementar as seguintes funções extras:
  - Concatenar
  - Intercalar (*Merge*)
  - Copiar uma lista