unipac.br
Barbacena

Bacharelado em Ciência da Computação

Estruturas de Dados

Material de Apoio

Parte I – *TAD e Lista por Vetor*

Prof. Nairon Neri Silva
naironsilva@unipac.br

2º sem / 2021

Tipos abstratos de Dados

- **Dados**: são abstrações da realidade, para a qual se conhece um modelo conceitual (físico-matemático), que permite transformá-los numa informação útil.
- **Processamento**: sequência finita e organizada de operações sobre os dados.
- **Informação**: Resultado do processamento dos dados, gerada como saída.

Tipos abstratos de Dados

- Um tipo de dado refere-se ao conjunto de valores a que uma constante pertence, ou que podem ser assumidos por uma variável ou expressão, ou que podem ser retornados por uma função (Wirth, 1976).
- Um **Tipo Abstrato de Dados (TAD)** pode ser visto como um modelo matemático acompanhado das operações definidas sobre o modelo, independente de qualquer linguagem de programação (Ziviani, 2005).
- **Exemplo**: o conjunto dos números inteiros acompanhado das operações aritméticas fundamentais forma um exemplo de um tipo abstrato de dados.

Tipos abstratos de Dados

- TAD podem ser considerados generalizações de tipos primitivos de dados.

- Da mesma forma que uma função é usada para encapsular partes de um algoritmo, o tipo abstrato de dados pode ser usado para encapsular tipos de dados.

Tipos abstratos de Dados

- Tipos abstratos de dados são extensivamente utilizados como base para o projeto de algoritmos.

- A definição do tipo e todas as operações definidas sobre o tipo podem ser localizadas numa única seção do programa.

TAD = Estrutura de Dados + Operações

Exemplos

- TADs mais comuns e utilizados:
  - Listas lineares
  - Filas
  - Pilhas
  - Árvores

Estrutura de Dados – Listas Lineares

- Uma lista linear é uma sequência de *n* elementos de um tipo de dado:

$$\boldsymbol{x1}, \boldsymbol{x2}, \boldsymbol{x3}, ..., \boldsymbol{xn}$$

cuja propriedade estrutural envolve as posições relativas de seus elementos. Se $n > 0$, então:

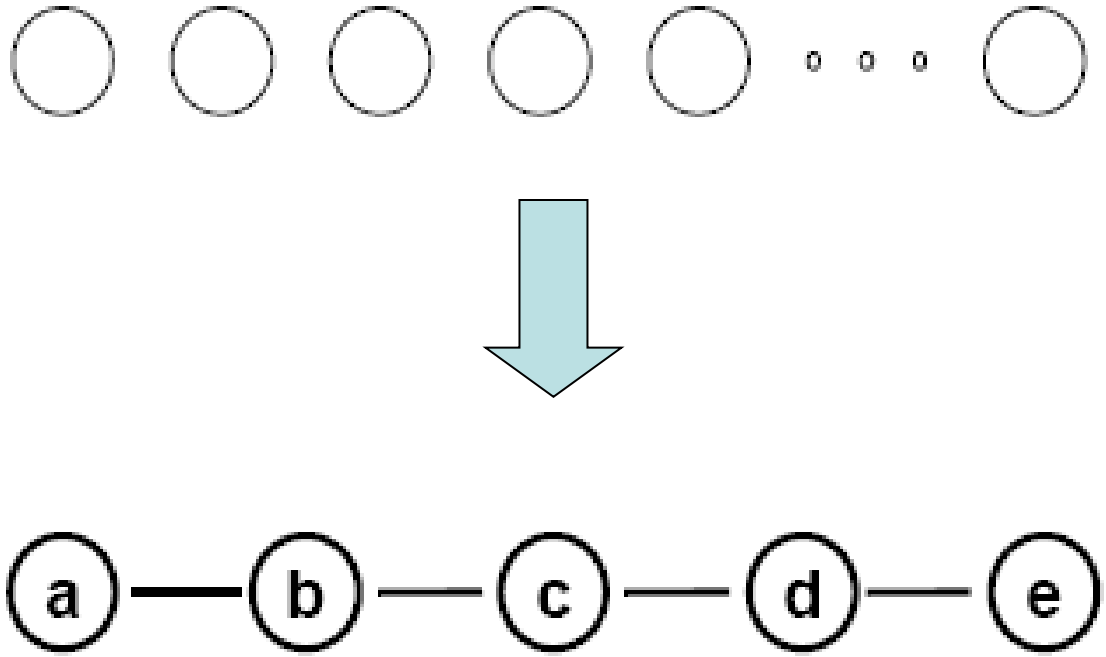
o $x1$ é o primeiro elemento.

o xn é o último elemento

o xi é sucessor de xi-1, para $i = 2, 3, ..., n$

o xi é antecessor de xi+1, para $i = 1, 2, ..., n$-1

Listas Lineares



Listas Lineares

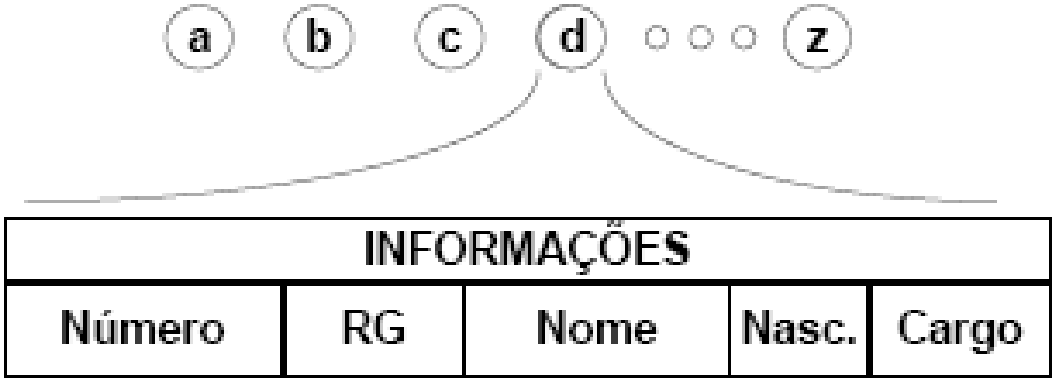
- Listas lineares são muito úteis em aplicações onde número de elementos não é conhecido, a priori.

Exemplos:

- ✓ Lista de arquivos de uma pasta;
- ✓ Lista de programas instalados;
- ✓ Lista de fontes *true type* instaladas;
- ✓ Lista de impressoras compartilhadas em uma rede;
- ✓ Cadastro de funcionários de uma empresa;
- ✓ Itens de estoque de uma empresa.

Listas Lineares

- Estrutura interna é abstraída.



Listas Lineares

- Para criar um TAD Lista, é necessário definir um conjunto de operações sobre os objetos do tipo Lista.

- O conjunto de operações a ser definido depende de cada aplicação, não existindo um conjunto de operações que seja adequado a todas as aplicações.

Listas Lineares

- Operações Comuns (muitas vezes necessárias):
  1. Criar uma lista linear inicialmente vazia.
  2. Inserir um elemento na *i*-ésima posição,
  3. Retirar o *i*-ésimo elemento,
  4. Acessar/Alterar o *i*-ésimo elemento,
  5. Retornar o número de elementos
  6. Localizar determinado elemento.
  7. “Limpar” a lista, isto é, retirar todos os seus elementos.

Listas Lineares

- Outras Operações (Dependem da aplicação específica):
  - · Fundir duas listas em uma.
  - · Dividir uma lista em duas.
  - · Copiar uma lista.
  - · Ordenar os elementos segundo algum critério.
  - · Verificar se uma lista está ordenada.
  - · Inverter a ordem dos elementos.
  - · Trocar a ordem de dois elementos

Implementação de Listas

As implementações mais utilizadas são:

- **Implementação sequencial:** através vetores, alocados estática (durante a compilação) ou dinamicamente (durante a execução).

- **Implementação encadeada:** através de uma lista encadeada por meio de ponteiros (alocação dinâmica).

Implementação Sequencial

Vantagens:

- Acesso direto a qualquer elemento.
- Implementação mais simples.

Desvantagens:

- A inserção ou exclusão de um elemento no meio da lista implica na movimentação de todos os elementos que o sucedem.
- A lista não pode crescer acima da capacidade do tamanho do vetor estático.
- A lista pode crescer acima do tamanho do vetor dinâmico, mas esse redimensionamento do vetor, pode requerer a movimentação de todos os itens.

Implementação Encadeada

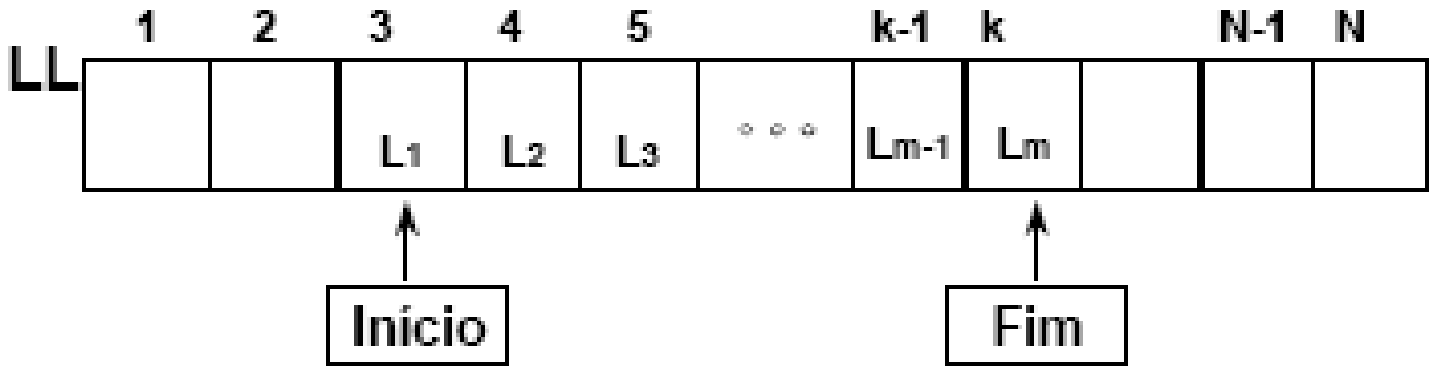
Vantagens

- A lista pode crescer à vontade, até o limite da memória.
- Facilidade de inserção/remoção de elementos nos extremos da lista.

Desvantagens

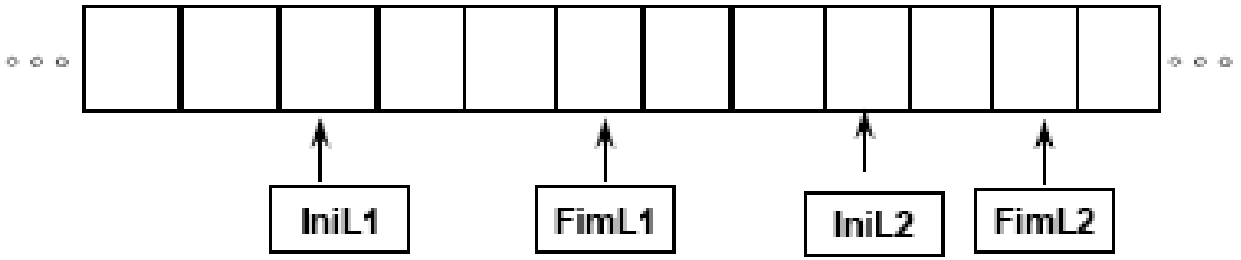
- Dificuldade de acesso. Para acessar um elemento deve-se percorrer a lista do início até chegar ao elemento.
- Maior consumo de memória, pois cada elemento contém um ponteiro para o próximo elemento. Todavia, esse consumo extra pesa relativamente menos, à medida que o tamanho do registro aumenta.
- Implementação um pouco mais difícil.

Listas – Implementação Sequencial



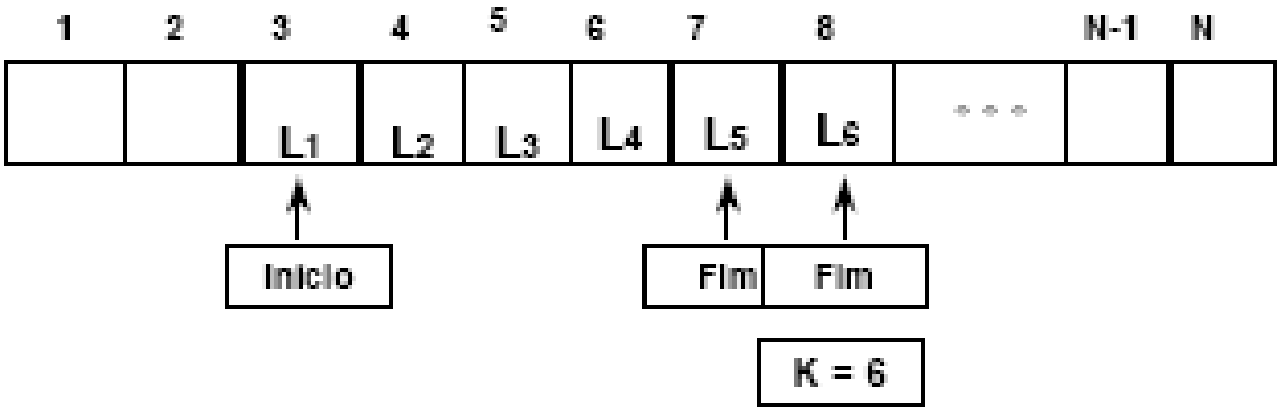
Listas – Implementação Sequencial

Um mesmo arranjo pode ser utilizado para mais de uma lista linear



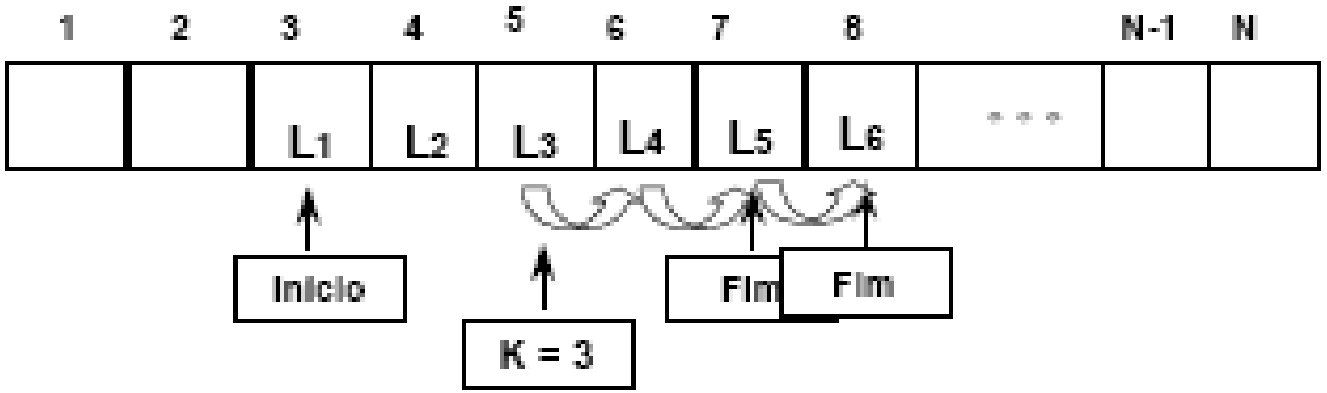
Listas – Implementação Sequencial

- **Inserir como último nodo**



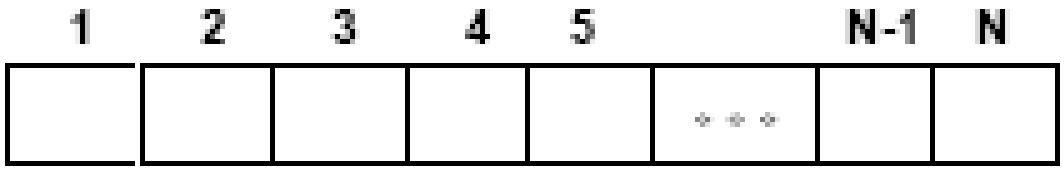
Listas – Implementação Sequencial

- **Inserir no meio da lista linear**



Listas – Implementação Sequencial

- Criar uma lista linear inicialmente vazia.



- Variáveis de controle: início e fim iguais a zero.

Listas – Implementação Sequencial

Vamos implementar uma lista sequencial no Codeblocks começando do zero

Exercícios

Implemente funções para:

1) Verificar se determinado item está presente na lista. (utilize o ID como referência para a busca)
2) Trocar a ordem de dois elementos
3) Alterar o conteúdo de um item da lista
4) Imprimir um item específico (busca pelo código)
5) Zerar a lista
6) Fundir (unir) duas listas em uma.
7) Copiar o conteúdo de uma lista para outra lista vazia.