Estruturas de Dados

Listas Lineares

Prof. MSc. Raphael Gomes
raphael@ifg.edu.br

O que veremos hoje...

- *Listas Lineares*
- *Formas de Implementação*
- *Listas Desordenadas e Ordenadas*
- *Exemplos*

Conceito

- *Uma Lista Linear é um **agregado linear** de elementos, ou seja, é uma estrutura que permite representar um conjunto de dados de forma a preservar a **relação de ordem linear** entre os dados.*

Lista Linear

- ***Definição**: sequência de **zero ou mais** elementos $a_1, a_2, ..., a_n$ sendo*
  - ✓ a_i elementos de um mesmo tipo
  - ✓ n o tamanho da lista linear

- ***Propriedade fundamental**: os elementos têm relações de ordem na lista*
  - ✓ a_i precede a_{i+1} (e a_i sucede a_{i-1});
  - ✓ a_1 é o primeiro elemento da lista
  - ✓ a_n é o último elemento da lista

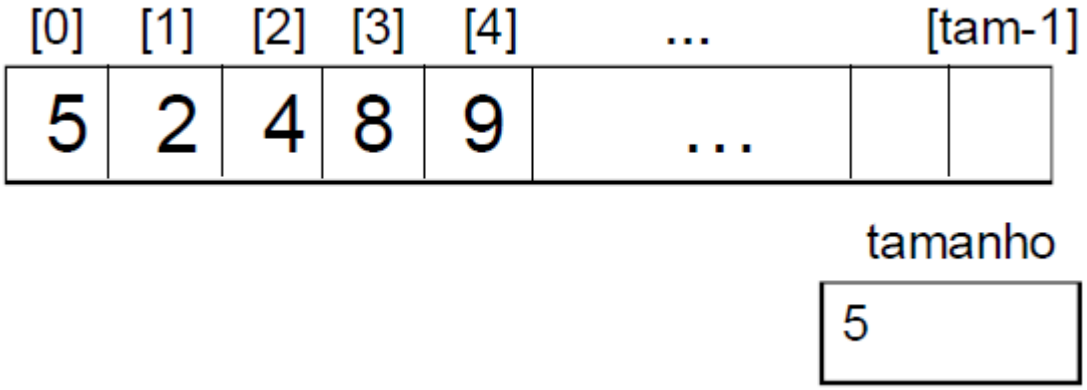
Métodos para a Representação dos Dados

- *Representação baseada em vetor*
- *Representação baseada em ligações*
- *Representação simulando ponteiros*

Representação Baseada em Vetor

- *Um vetor (array) é utilizado para armazenar tanto a lista de elementos quanto as referências para os mesmos (índices do vetor).*

- *Graficamente:*



Problema

- *Exige existência de um conjunto de bytes consecutivos na memória capaz de comportar todo o conjunto de elementos do vetor.*

- *Desperdício de memória versus Falta de memória*

Representação Baseada em Ligações

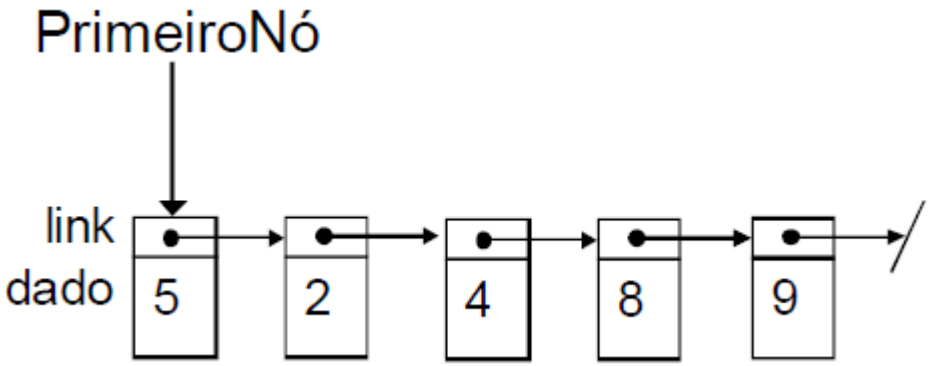
- *Os elementos podem ser armazenados em um conjunto de posições **não sequencial** de localizações de memória.*

- *Cada elemento, neste caso, deve manter explicitamente, um campo indicando a localização do próximo elemento.*

- *Tal campo, em geral, é chamado de **ponteiro** ou **campo de ligação** (link).*

Representação Baseada em Ligações

- *Graficamente:*



- *Neste tipo de representação a estrutura cresce/diminui dinamicamente à medida que novos elementos vão sendo inseridos/removidos,* ***evitando****, desse modo, o* ***desperdício de memória****.*

Representação Simulando Ponteiros

- *Similar à lista ligada, mas os "ponteiros" são substituídos por números inteiros.*

- *Vantagem*
  - ✓ É possível armazenar os elementos em qualquer posição do vetor, simulando as ligações utilizando os próprios índices para os elementos.

Representação Simulando Ponteiros

- *É necessário ter uma variável para controlar o início da lista e outra responsável por gerenciar a lista de elementos não ocupados.*

- *Graficamente:*

PrimeiroNó | 2 |

nó	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
próx		6	7			1	-1	5		
elemento		8	5			4	9	2		

Operações

- *Vazia ( )*
- *Tamanho ()*
- *Pegue (Índice)*
- *ÍndiceDe (x)*
- *Remove (Índice)*
- *Adicione (Índice, x)*
- *Imprime ( )*

Operações Básicas de uma Lista

- *Inicia*
  - pós-condição: Lista vazia;
  - funcionalidade: cria uma lista vazia
  - resultado: retorno de uma lista vazia criada

- *Vazia*
  - funcionalidade: Testa se a lista está vazia ou não
  - resultado: TRUE se a lista está vazia; senão FALSE

Operações Básicas de uma Lista

- *Remove*
  - pré-condição: Lista tem n>1 elementos e pos <= posição do último;
  - pós-condição: Lista tem n – 1 elementos;
  - funcionalidade: Retorna o item x que está na posição p da lista mantendo a ordenação dos demais
  - resultado: item removido

Operações Básicas de uma Lista

- *Adicione*
  - pré-condição: Lista tem n>= 0 elementos e n <TamMax;
  - pós-condição: Lista tem n+1 elementos;
  - funcionalidade: Insere um elemento na lista

Características das Listas Lineares Baseadas em Vetor

- *Armazenados fisicamente em posições consecutivas;*
- *Elementos na lista podem estar ordenados ou não;*
- *Inserção de um elemento em uma lista:*
  - ✓ Desordenada: insere o elemento na posição a[i], o que causa o deslocamento a direita do elemento de a[i] ao último;
  - ✓ Ordenada: insere o elemento mantendo a ordem.
- *Eliminação do elemento a[i] requer o deslocamento à esquerda do a[i+1] ao último*

Vantagens e Desvantagens

- *Vantagens*
  - ✓ Acesso direto indexado a qualquer elemento (facilita modificações dos conteúdos)
  - ✓ Tempo constante para acessar os elementos (depende só do índice)

- *Desvantagens*
  - ✓ Movimentação para a inserção e remoção de elementos
  - ✓ Conhecimento a priori do tamanho do número máximo da lista (aplicações em que não existe previsão sobre o crescimento da lista não é indicado)

Implementação (lista_linear_vetor.h)

```
#ifndef LISTA_LINEAR_VETOR_H_
#define LISTA_LINEAR_VETOR_H_

    // Capacidade da lista, ou seja, quantos elementos cabem nela
    #define CAPACIDADE 100

    // Valor para indicar um elemento nulo
    #define NULO -1

    // Tipo dos dados armazenados na lista
    typedef int T;

    typedef struct {
            // Vetor para armazenar o conjunto de elementos da
            // lista
            T elemento[CAPACIDADE];
            // Contador do número de elementos ocupados
            int tamanho;
    } Lista;
```

Implementação (lista_linear_vetor.h)

```
    // Definição dos métodos
    void cria(Lista *l);
    int vazia(Lista *l);
    int tamanho(Lista *l);
    T pegue(int indice, Lista *l);
    int indiceDe(T oElemento, Lista *l);
    void vejaIndice(int indice, Lista *l);
    T removeLista(int indice, Lista *l);
    void adicioneLista(int indice, T oElemento, Lista *l);
    void imprime(Lista *l);

#endif /* LISTA_LINEAR_VETOR_H_ */
```

Implementação (lista_linear_vetor.c)

```c
#include "lista_linear_vetor.h"
#include <stdio.h>
#include <stdlib.h>

/* Cria uma lista vazia inicializando propriamente as variáveis */
void cria(Lista *l) {
     int i;

     l->tamanho = 0;

     for (i = 0; i < CAPACIDADE; i++) {
          l->elemento[i] = NULO;
     }
}

/* Verifica se a lista está vazia.
 * Retorna 1 se e somente se a lista estiver vazia, 0 caso contrário. */
int vazia(Lista *l) {
     return (l->tamanho == 0);
}

/* Tamanho da lista.
 * Retorna o número corrente de elementos na lista. */
int tamanho(Lista *l) {
     return l->tamanho;
}
```

Implementação (lista_linear_vetor.c)

```c
/* Busca elemento em determinada posição da lista.
 * Retorna o elemento localizado na posição indice.
 * Termina o programa quando o índice fornecido não estiver entre 0 e tamanho-1. */
T pegue(int indice, Lista *l) {
	vejaIndice(indice, l);
	return l->elemento[indice];
}

/* Busca o índice de determinado elemento.
 * Retorna o índice da 1a ocorrência do elemento na lista ou -1 se não encontrar. */
int indiceDe(T oElemento, Lista *l) {
	int i;
	for (i = 0; i < l->tamanho; i++) {
		if (l->elemento[i] == oElemento) {
			return i; // elemento encontrado
		}
	}
	return -1;
}

/* Verifica a validade de determinado índice. Um índice válido está entre 0 e tamanho-1.
 * Termina o programa quando indice não estiver entre 0 e tamanho – 1. */
void vejaIndice(int indice, Lista *l) {
	if (indice < 0 || indice >= l->tamanho) {
		printf("Índice = %d tamanho = %d \n", indice, l->tamanho);
		exit(1);
	}
}
```

Implementação (lista_linear_vetor.c)

```
/**
 * Remove o elemento localizado no indice especificado. Todos os elementos
 * com índice superior ao índice fornecido terão seus valores decrementados
 * em uma unidade.
 * Retorna o elemento removido.
 * Termina o programa quando indice não estiver entre 0 e tamanho - 1.
 */
T removeLista(int indice, Lista *l) {
      int i;

      vejaIndice(indice, l); // índice válido
      T elementoRemovido = l->elemento[indice];

      // deslocar elementos com índice superior
      for (i = indice + 1; i < l->tamanho; i++) {
              l->elemento[i - 1] = l->elemento[i];
      }

      l->tamanho--;
      l->elemento[l->tamanho] = NULO;
      return elementoRemovido;
}
```

Implementação (lista_linear_ligada.c)

```
/* Adiciona um novo elemento a lista em uma dada posição. Todos os elementos
 * com índice igual ou superior ao índice fornecido terão seus índices
 * incrementados
 * Termina o programa quando a lista estiver cheia ou quando indice não
 * estiver entre 0 e tamanho - 1.
 */
void adicioneLista(int indice, T oElemento, Lista *l) {
     int i;
     if (l->tamanho == CAPACIDADE) { // sem espaço
             printf("Lista cheia \n");
             exit(1);
     }
     if (indice < 0 || indice > l->tamanho) {
             printf("Índice = %d tamanho = %d \n", indice, l->tamanho);
             exit(1);
     }

     // desloca elementos uma posição à direita
     for (i = l->tamanho - 1; i >= indice; i--) {
             l->elemento[i + 1] = l->elemento[i];
     }
     l->elemento[indice] = oElemento;
     l->tamanho++;
}
```

Implementação (lista_linear_ligada.c)

```
/**
 * Imprime o conteúdo de uma lista usando a saída padrão
 */
void imprime(Lista *l) {
     int i;

     if (l->tamanho == 0) {
             printf("Lista vazia");
             return;
     }

     printf("[");
     for (i = 0; i < l->tamanho; i++) {
             if (i == l->tamanho - 1) {
                     printf("%d]\n", l->elemento[i]);
             } else {
                     printf("%d\t", l->elemento[i]);
             }
     }
}
```

Teste (main_vetor.c)

```c
#include "lista_linear_vetor.h"
#include <stdio.h>
#include <stdlib.h>

int main() {
     Lista lista;
     T item;
     int vetor[CAPACIDADE];
     int i, j, k, n, max = 10;

     cria(&lista);

     /*Gera uma permutacao aleatoria de chaves entre 1 e max*/
     ...

     /*Insere cada chave na lista */
     for (i = 0; i < max; i++) {
             item = vetor[i];
             adicioneLista(i, item, &lista);
             printf("Inseriu: %d \n", item);
     }
     imprime(&lista);
```

Teste (main_vetor.c)

```
        /*Retira cada chave da lista */
        for (i = 0; i < max; i++) { /*escolhe uma chave aleatoriamente */
                j = (int) ((lista.tamanho - 1) * rand() / (RAND_MAX + 1.0));
                /*retira chave apontada */
                item = removeLista(j, &lista);
                printf("Retirou: %d\n", item);
        }
        imprime(&lista);

        return 0;
}
```

Exercícios

- *Modificar a implementação de forma que a capacidade da lista seja fornecido pelo usuário*
  - ✓ Dica: Modifique o método de criação da lista

Exercícios

- *Dadas duas listas L1 e L2, que possam estar desordenadas e contendo elementos repetidos:*
  - ✓ Verifique se L1 está ordenada ou não (a ordem deve ser crescente)
  - ✓ Faça uma cópia da lista L1 em uma outra lista L2;
  - ✓ Faça uma cópia da Lista L1 em L2, eliminando elementos repetidos;
  - ✓ Inverta L1 colocando o resultado em L2;
  - ✓ Inverta L1 colocando o resultado na própria L1;
  - ✓ Intercale L1 com a lista L2, gerando a lista L3. Considere que L1, L2 são ordenadas.
  - ✓ Assumindo que os elementos da lista L1 são inteiros positivos, forneça os elementos que aparecem o maior e o menor número de vezes (forneça os elementos e o número de vezes correspondente)

Bibliografia

- *M. T. GOODRICH, R. TAMASSIA, Estrutura de Dados e Algoritmos em Java, 2a Ed. Bookman, 2002.*