



Estruturas de Dados

Aula 02: TAD e Lista Linear

Prof. Marcelo Zorzan



Assuntos da Aula

- Tipo Abstrato de Dados
- Estrutura de Dados
- Lista Linear Estática



Tipo Abstrato de Dado

- Um Tipo Abstrato de Dado (TAD) é um modelo matemático acompanhado das operações definidas sobre o modelo.
- Exemplo de Tipo Abstrato de Dado:
 - 1- Conjunto dos números inteiros:

Possíveis valores: ..., -200, -199, ..., 0, ..., 199, 200, ...

Operações: somar, subtrair, multiplicar, dividir, resto da divisão, igualdade, desigualdade...

***Na definição de um TAD, não há preocupação com a eficiência de tempo e espaço. Esta é uma preocupação de sua implementação.**



Estruturas de Dados

- A eficiência de um programa está fortemente associada à forma como seus dados são organizados
- Dados quando estão organizados de uma forma coerente representam uma estrutura de dados
- Estrutura de dados é o ramo da computação que estuda os diversos mecanismos de organização de dados para atender aos diferentes requisitos de processamento
- De acordo com o modo como um conjunto de dados é organizado e como as operações são efetuadas sobre estes dados, pode-se solucionar de forma simples problemas extremamente complexos.



Estruturas de Dados

- Estruturas de dados são usadas para representar o modelo matemático de um Tipo Abstrato de Dado, ou seja, são usadas para trabalhar a implementação de um TAD.



Situação-Problema

- Imagine que você precise criar um programa para armazenar os seguintes dados de uma pessoa:
 - nome completo,
 - estado civil,
 - data de nascimento (dia, mês e ano),
 - endereço (nome da rua, numero, cidade, cep, estado, país),
 - conta bancária (número, agência, saldo, nome do banco).



Situação-Problema – Solução 1

```
int main(){
    char nomeCompleto[20], estadoCivil[10];
    int diaDataNasc, mesDataNasc, anoDataNasc;
    char nomeRua[30], cidade[15], cep[10], estado[2], pais[10];
    int numeroResidencia, numeroAgenciaBanco;
    char numeroConta[10], nomeBanco[10];
    float saldo;
    ...
    return 0;
}
```

O que você acha da solução acima?



Situação-Problema – Solução 2

E a solução abaixo?

```
typedef struct sPessoa{  
    char nomeCompleto[20], estadoCivil[10];  
    int diaDataNasc, mesDataNasc, anoDataNasc;  
    char nomeRua[30], cidade[15], cep[10], estado[2], pais[10];  
    int numeroResidencia, numeroAgenciaBanco;  
    char numeroConta[10], nomeBanco[10];  
    float saldo;  
} Pessoa;
```

```
int main(){  
    Pessoa pes;  
    return 0;  
}
```




Situação-Problema – Solução 3

```
typedef struct sData{  
    int dia,  
    int mes,  
    int ano;  
} Data;
```

```
typedef struct sEndereco{  
    char nomeRua[30];  
    int numResidencia;  
    char cidade[15];  
    char cep[10];  
    char estado[2];  
    char pais[10];  
} Endereco;
```

```
typedef struct sConta{  
    int numeroAgencia;  
    char numeroConta[10];  
    char nomeBanco[10];  
    float saldo;  
} Conta;
```

```
typedef struct sPessoa{  
    char nomeCompleto[20];  
    char estadoCivil[10];  
    Data dataNasc;  
    Endereco end;  
    Conta contaCorrente;  
} Pessoa;
```

```
int main(){  
    Pessoa pes;  
    return 0;  
}
```



Estruturas de Dados

- E se quiséssemos cadastrar 10 pessoas?

```
int main(){  
    Pessoa pes [10];  
    return 0;  
}
```



Estruturas de Dados

- E se quiséssemos que o usuário do sistema definisse o número de pessoas a serem cadastradas?

```
int main(){  
    Pessoa *pes;  
    int num;  
    printf("Digite o número de pessoas: ")  
    scanf("%d", &num);  
    pes = (Pessoa*) malloc(num*sizeof(Pessoa));  
  
    return 0;  
}
```



Estruturas de Dados

- Podemos considerar “*arrays*” e “registros” como Estruturas de Dados?

"De acordo com o modo como um conjunto de dados é organizado e como as operações são efetuadas sobre estes dados, pode-se solucionar de forma simples problemas complexos. "



Estruturas de Dados

- Estrutura de dados :
 - Células de memória com espaço para uma unidade de informação
 - Descrevem um tipo relacionado a uma quantidade de bytes necessários para armazenar a informação
- Estrutura de dados simples:
 - **Arranjos**: são estruturas de dados homogêneas de n -dimensões
 - **Registros** (ou *structs*): são estruturas de dados heterogêneas que agrupam uma série de células de memória ou ainda outras estruturas de dados.



Estruturas de Dados

- Estruturas de dados complexas:
 - Lista dinâmicas,
 - Filas,
 - Pilhas,
 - Árvores,
 - Grafos.



Estruturas de Dados - Lista

- Permite resolver problemas como:
 - Filas de prioridades,
 - Percurso de um transporte,
 - Etc.

- Exemplo

→ Lista sequencial:

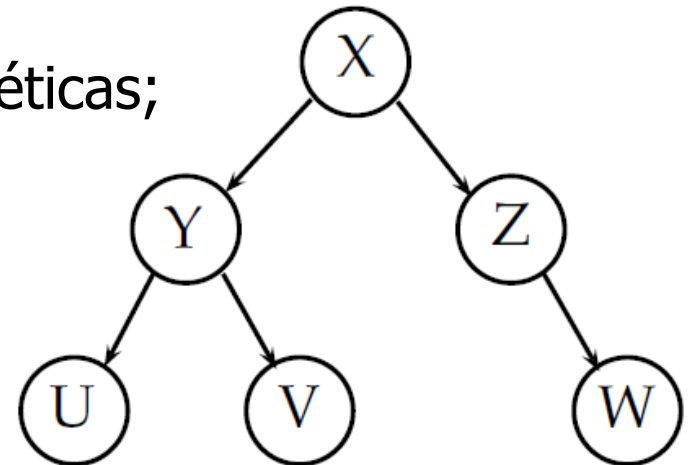
1	2	3	4
---	---	---	---

→ Lista encadeada:

1	→	2	→	3	→	4	
---	---	---	---	---	---	---	--

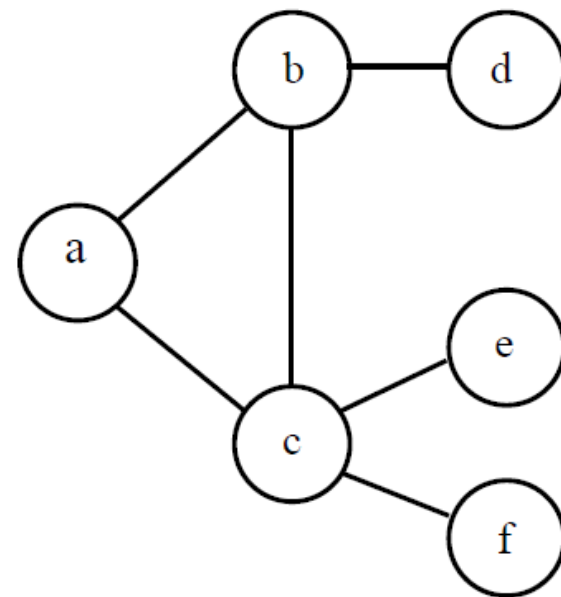
Estruturas de Dados - Árvore

- Estrutura de dados em árvore permitem modelar uma série de problemas computacionais como:
 - Problemas que envolvem hierarquia
 - Estruturas de diretórios, subdiretórios e arquivos;
 - Interfaces gráficas com o usuário (organização dos menus, por exemplo);
 - Organização de expressões aritméticas;
 - Etc.



Estruturas de Dados - Grafos

- Estrutura de dados em grafos permitem modelar uma série de problemas computacionais como:
 - Redes,
 - Percurso de um transporte (ônibus, carro, trem, etc)
 - Controle de uma malha viária,
 - Etc.





Lista Linear

- São estruturas flexíveis, que podem crescer ou diminuir durante a execução do programa, de acordo com a demanda
- Itens podem ser acessados, inseridos ou retirados de uma lista.
- Listas são adequadas para aplicações nas quais não é possível prever a demanda por memória. (listas dinâmicas)



Lista Linear

- Definição: seqüência de zero ou mais elementos a_1, a_2, \dots, 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



Implementação do TAD

Lista

- Tipos de Implementação:
 1. Através de arranjos (*arrays*) – listas estáticas
 2. Através de apontadores ou ponteiros – listas dinâmicas

- Em qualquer uma das implementações, deve-se:
 1. Definir a Estrutura de Dados
 2. Definir as Operações

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



Lista - Operações usuais

- Inicializar a lista
- Acessar um elemento
- Buscar um valor
- Remover um elemento
- Inverter uma lista
- Ordenar uma lista
- Concatenar duas listas
- Intercalar duas lista
- Inserir um novo elemento
- Verificar se a lista está cheia
- Verificar se a lista está vazia
- Modificar um elemento existente
- Determinar o tamanho da lista



Lista usando *array* - implementação

- Em um *array* os itens da lista são armazenados em posições contíguas de memória.
- A lista pode ser percorrida em qualquer direção.
- Inserção de um elemento em uma lista:
 - Desordenada: insere o elemento na última posição;
 - Ordenada: insere o elemento na posição a_i , o que causa o deslocamento de todos itens localizados após o ponto de inserção.
- Da mesma forma, retirar um item do início da lista requer um deslocamento de itens para preencher o espaço deixado vazio.



Lista usando *array* - implementação

```
#define TAMF 10
int main(){
    int i, elem;
    Tipo lista_a[TAMF];
    elem = -1;
    i = 0;
    while (i < TAMF){
        if(busca == lista_a[i]){
            elem = i;
            break;
        }
        i++;
    }
    if(elem > -1)
        printf("Elemento encontrado = %d", elem);
    else
        printf("Elemento não encontrado");
}
```

- Tamanho constante
- Busca sequencial, exemplo:



Situação-Problema

- Jorge, que acabou de se formar no IFSP, conseguiu emprego em uma empresa de telefonia de grande porte na cidade de São Paulo. Como era do seu interesse ele foi contratado para trabalhar na área de desenvolvimento para celular. O módulo que Jorge terá que implementar corresponde ao módulo de “contatos”.

Que estrutura de dados você sugere que ele utilize e quais operações seriam interessante implementar para essa estrutura?



Situação-Problema

- “contatos” para celular

Definir Lista Linear

Estrutura
de Dados



Lista Linear

Definição

Operação 1

Operação 2

Operação 3

Operação n

Inicializar

Inserir

Remover

Pesquisar

Estática ou
Dinâmica?



Lista usando *array* - implementação

- Definição da lista linear estática

```
#define TAMF 10

typedef struct lista
{
    char valor[TAMF];
    int n;
}LISTA;
```

- A variável **n** representa o índice do último elemento inserido na lista

- Exemplo:

'a'	'b'	'c'	'd'						
0	1	2	3	4	5	6	7	8	9

n = 3



Lista usando *array* - implementação

- Inicializar a lista

```
void lista_inicializar(LISTA *L)
{
    L->n = -1;
}
```

- A variável **n** é inicializada com valor -1 para indicar que a lista está vazia.



Lista usando *array* - implementação

```
int main()
{
    LISTA *L;
    L = (LISTA*) malloc(sizeof(LISTA));
    lista_inicializar(L);
}
```

ou

```
int main()
{
    LISTA L;
    lista_inicializar(&L);
}
```

Formas de criar
uma variável do
tipo LISTA



Lista usando *array* - implementação

- Acesso a um elemento

```
char lista_acessar(LISTA *L, int k) {  
    if ( (k < 0) || (k > (L->n)) )  
        printf("Erro - elemento fora dos limites");  
    else  
        return L->valor[k];  
}
```

- A função `lista_acessar` retorna o elemento localizado na posição de índice `k` da lista, no caso, um caracter.



Lista usando *array* - implementação

- Buscar um valor

```
int lista_buscar(LISTA *L, char val_b{
    int elem = -1;
    int i = 0;
    while (i <= (L -> n)) {
        if(val_b == L -> valor[i]) {
            elem = i;
            i++;
            break;
        }
    }
    return elem;
}
```

- A função lista_buscar retorna a localização (índice) de um determinado elemento na lista.



Lista usando *array* - implementação

- Verificar se a lista está cheia

```
int lista_cheia(LISTA *L) {  
    if (L -> n+1 == TAMF)  
        return 1;  
    else  
        return 0;  
}
```



Lista usando *array* - implementação

- Verificar se a lista está vazia

```
int lista_vazia(LISTA *L) {  
    if ( L->n == -1 )  
        return 1;  
    else  
        return 0;  
}
```




Lista usando *array* - implementação

- Insere um novo elemento no fim (topo) da lista

```
int lista_inserirfim(LISTA *L, char elem)
{
    if(lista_cheia(L))
    {
        printf("Erro - lista cheia");
        return 1;
    }
    L -> valor[L->n+1] = elem;
    L -> n = L -> n+1;

    return 0;
}
```



Lista usando *array* - implementação

- Insere um novo elemento na posição k

```
int lista_inserir(LIST *L, int k, char elem){  
    if(lista_cheia(L)){  
        printf("Erro - lista cheia");  
        return 1;  
    }  
    if(k < 0 || k > L->n+1){  
        printf("Erro - elemento fora dos limites");  
        return 1;  
    }  
    ...  
}
```



Lista usando *array* - implementação

```
...  
int i;  
  
for (i=(L->n+1); i>k; i--) {  
    L->valor[i] = L->valor[i-1];  
}  
  
L->valor[k] = elem;  
L->n = L->n+1;  
return 0;  
}
```



Lista usando *array* - implementação

- Modificar um elemento existente

```
int lista_modificar(LISTA *L, int k, char elem) {  
    if (k < 0 || k > L->n) {  
        printf("Erro - elemento fora dos limites");  
        return 1;  
    }  
  
    L->valor[k] = elem;  
    return 0;  
}
```



Lista usando *array* - implementação

- Remover um elemento

```
int lista_remove(LISTA *L, int k) {
    if (k < 0 || k > L->n) {
        printf("Erro - elemento fora dos limites");
        return 1;
    }

    for (i = k; i < (L->n); i++) {
        L->valor[i] = L->valor[i+1];
    }

    L->n = L->n-1;
    return 0;
}
```



Lista usando *array* - implementação

- Determinar o tamanho da lista

```
int lista_tamanho(LISTA *L) {  
    return L->n+1;  
}
```



Lista usando *array* - implementação

- Outras operações:
 - Concatenar duas listas
 - Intercalar duas listas
 - Inverter uma lista
 - Ordenar uma lista



Lista usando *array* - implementação

```
int main()
{
    LISTA L;
    int suc;
    lista_inicializar(&L);
    suc = lista_inserirfim(&L, 'a');
    if(suc == 0)
        printf("\n\nSucesso na insercao!\n");
    suc = lista_inserirfim(&L, 'b');
    if(suc == 0)
        printf("\n\nSucesso na insercao!\n");
    suc = lista_remove(&L, L.n);
    if(suc == 0)
        printf("\n\nSucesso na remocao! \n");
}
```

- Crie uma função `lista_exibir` para imprimir os elementos da lista.



Lista usando *array*

- Pontos Fortes

- Fácil endereçamento
- Aritmética simples (endereços)
- Fácil inserção e remoção de elementos no final da lista

- Pontos fracos

- Difícil inserção e remoção dos elementos no meio da lista
- Difícil movimentação de elementos na lista

Solução?



Leitura Recomendada

- ZIVIANI N. Projeto de Algoritmos: com implementações em Pascal e C, Ed. Pioneira Thomson Learning, 2002.

→ Capítulo 1, seções 1.1 e 1.2

→ Capítulo 3, seções 3.1, subseção 3.1.1



Exercício

1) Desenvolva um programa em Linguagem C que permita fazer as seguintes operações sobre uma lista linear estática de números inteiros positivos:

- (a) inserir um elemento em uma posição específica;
- (b) remover um elemento de uma posição específica;
- (c) acessar um elemento de uma posição específica;
- (d) procurar um determinado elemento.
- (e) exibir os elementos do vetor

Quantos *bytes* seu programa **principal** ocupa para armazenar dados?

OBS: Defina funções para cada operação.