

Cap. VIII – NOÇÕES DE ESTRUTURAS DE DADOS

8.1 - Introdução

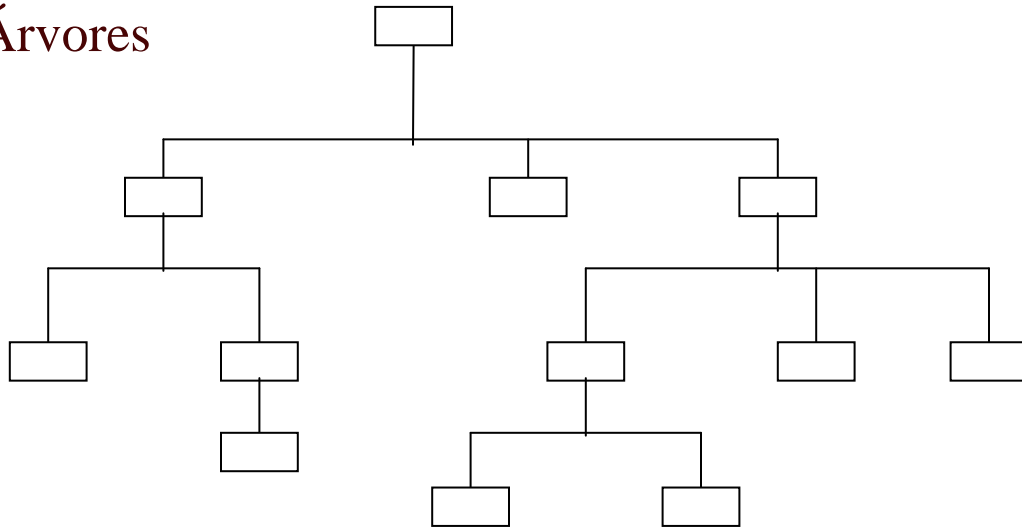
- O modo de armazenar informações na memória tem grande influência sobre o tempo de execução de um programa.
- A natureza das informações sugere a utilização de certos modelos de armazenamento:

– Listas Lineares:



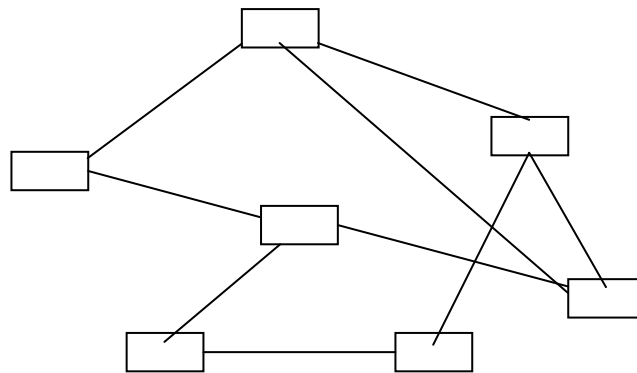
- ✍ Tabelas em geral,
- ✍ Empregados de uma empresa,
- ✍ Clientes de um banco,
- ✍ Livros de um biblioteca,
- ✍ Tabelas em banco de dados relacionais

– Árvores



- ✍ Organização de livros e cursos,
- ✍ Organogramas de empresas,
- ✍ Expressões aritméticas,
- ✍ Estrutura de um programa,
- ✍ Jogos de um campeonato.

– Grafos:



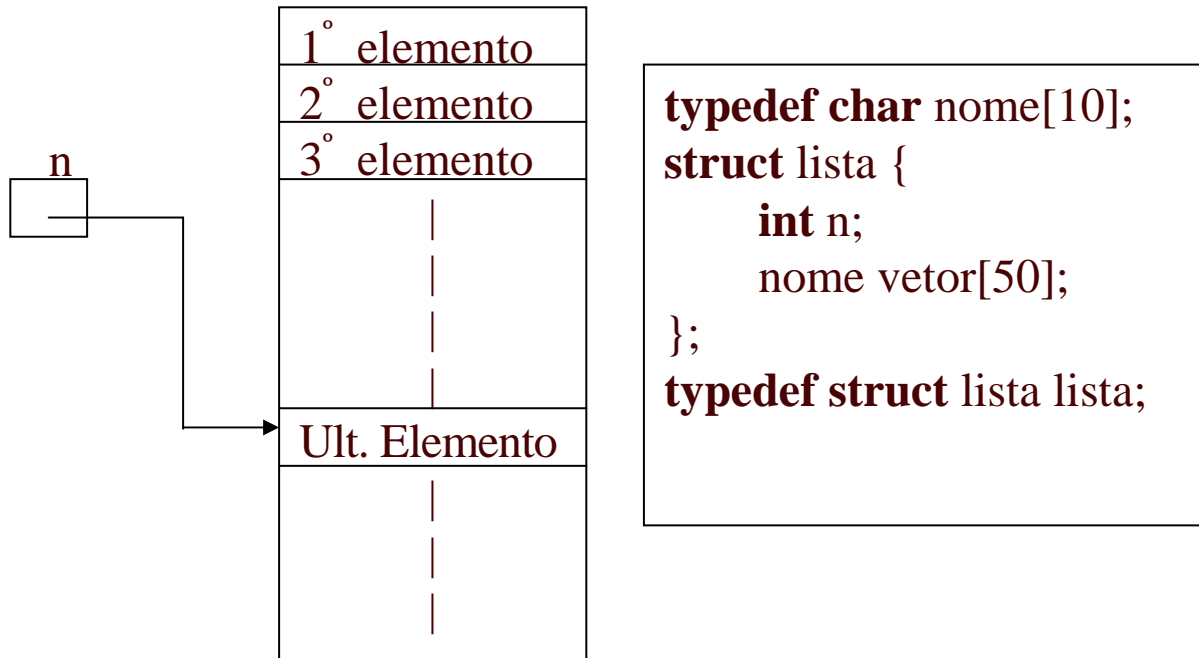
- ✍ Tarefas de um projeto,
 - ✍ Sistema rodoviário de uma região,
 - ✍ Rede de computadores,
 - ✍ Fornecimento de produtos entre fábricas,
 - ✍ Dependências entre comandos de um programa.
- Cada um desses modelos pode ser implementado através

de diversas estruturas de dados.

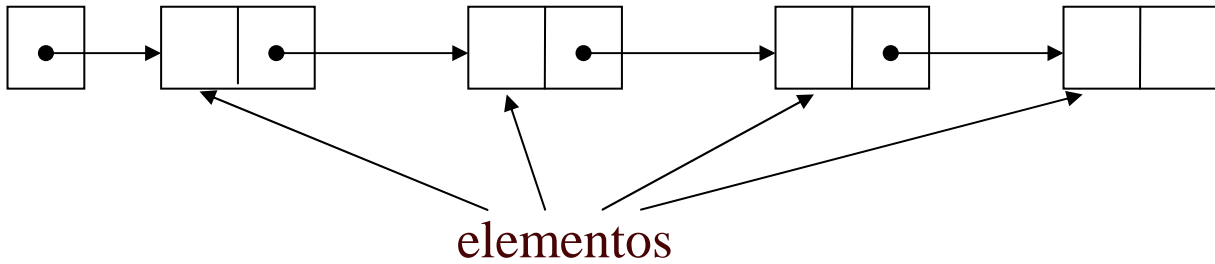
- A escolha de uma estrutura para um modelo numa determinada aplicação afeta a eficiência do programa.

8.2 - Listas Lineares Encadeadas

- Uma Lista Linear pode ser implementada por uma variável indexada e um inteiro:



- É a chamada **estrutura contígua** para listas.
- Operações de inserir e deletar elementos são ineficientes: muita movimentação de elementos.
- Uma alternativa: **estrutura encadeada**:



- Os elementos são guardados em nós
- Cada nó guarda:
 - um elemento
 - um ponteiro para o próximo nó
- Declarações:

```
typedef char nome [10];
typedef struct noh noh;
typedef noh * lista;

struct noh {
    nome elem;
    noh * prox;
};
lista l1,l2,l3;
```
- **Exemplo 8.1:** programa para formar uma lista encadeada de elementos do tipo inteiro:

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

typedef struct noh noh;
typedef noh *lista;
typedef noh *posicao;
struct noh {
    int elem;
    noh *prox;
};

void formalista (lista *l) {
    posicao p;
    int i, n;

    printf ("Numero de elementos da lista: ");
    scanf ("%d", &n);
    printf ("\nElementos:\n\t");
    for (p = *l = (noh *) malloc (sizeof (noh)),
        i = 1; i <= n; i++) {
        p->prox = (noh *) malloc (sizeof (noh));
        p = p->prox;
        scanf ("%d", &p->elem);
    }
    p->prox = NULL;
}

```

```

void escreverlista (lista l) {
    posicao p;
    printf ("\n\t");
    for (p = l; p->prox != NULL; p = p->prox)
        printf ("%4d", p->prox->elem);
    printf ("\n");
}

```

```

void main ( ) {
    lista l;

    formalista (&l);
    printf ("\nConfirmacao:");
    escreverlista (l);
}

```

- Para inserir e deletar elementos da lista, não é necessário mover muitas informações.
- Encontrada a posição de inserção, é só criar um novo nó e encadeá-lo na lista.
- Encontrada a posição de deleção, é só retirar o nó do encadeamento.

- **Exemplo 8.2:** Funções para inserir e deletar elementos de uma lista linear encadeada e ordenada:

– Inserir:

```
void inserir (int num, lista l) {  
    posicao p, q;  
    for (p = l; p->prox != NULL && p->prox->elem < num;  
        p = p->prox);  
    q = p->prox;  
    p->prox = malloc (sizeof (noh));  
    p->prox->elem = num;  
    p->prox->prox = q;  
}
```

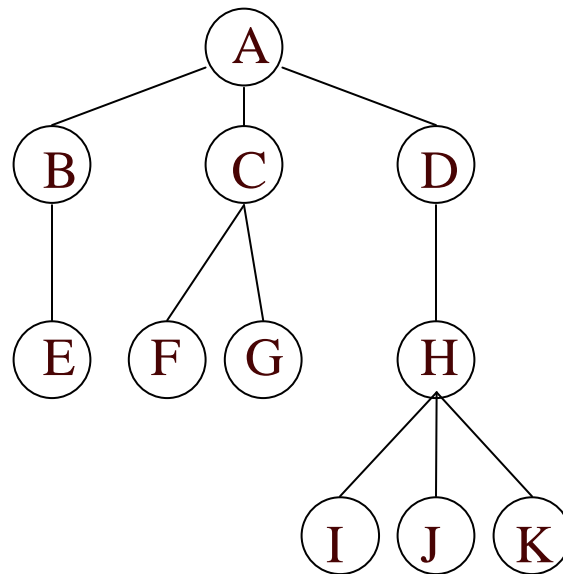
– Deletar:

```
void deletar (int num, lista l) {  
    posicao p, q;  
    for (p = l; p->prox != NULL && p->prox->elem < num;  
        p = p->prox);  
    if (p->prox != NULL && p->prox->elem == num) {  
        q = p->prox;  
        p->prox = q->prox;  
        free (q);  
    }  
}
```

8.3 – Árvores

8.3.1 – Definições

- Esquema:



- **Árvore:** coleção de nós sobre os quais existe uma hierarquia **paterna** (relação *pai de*).
- **Raiz:** nó de máxima hierarquia de uma árvore.
- **Definição recursiva:**

1 – Um único nó é, por si mesmo, uma árvore. Ele é a raiz dessa árvore.

2 – Seja \underline{n} um nó e A_1, A_2, \dots, A_k , árvores de raízes n_1, n_2, \dots, n_k , respectivamente. Pode-se construir uma nova árvore, fazendo \underline{n} pai de n_1, n_2, \dots, n_k . Nessa árvore, \underline{n} é a raiz e A_1, A_2, \dots, A_k são as subárvores da raiz.

- **Folha:** nó sem filhos.
- **Irmãos:** filhos de um mesmo nó.
- **Floresta:** conjunto de zero ou mais árvores disjuntas.
- **Forma parentesada:** uma das formas de representar uma árvore:

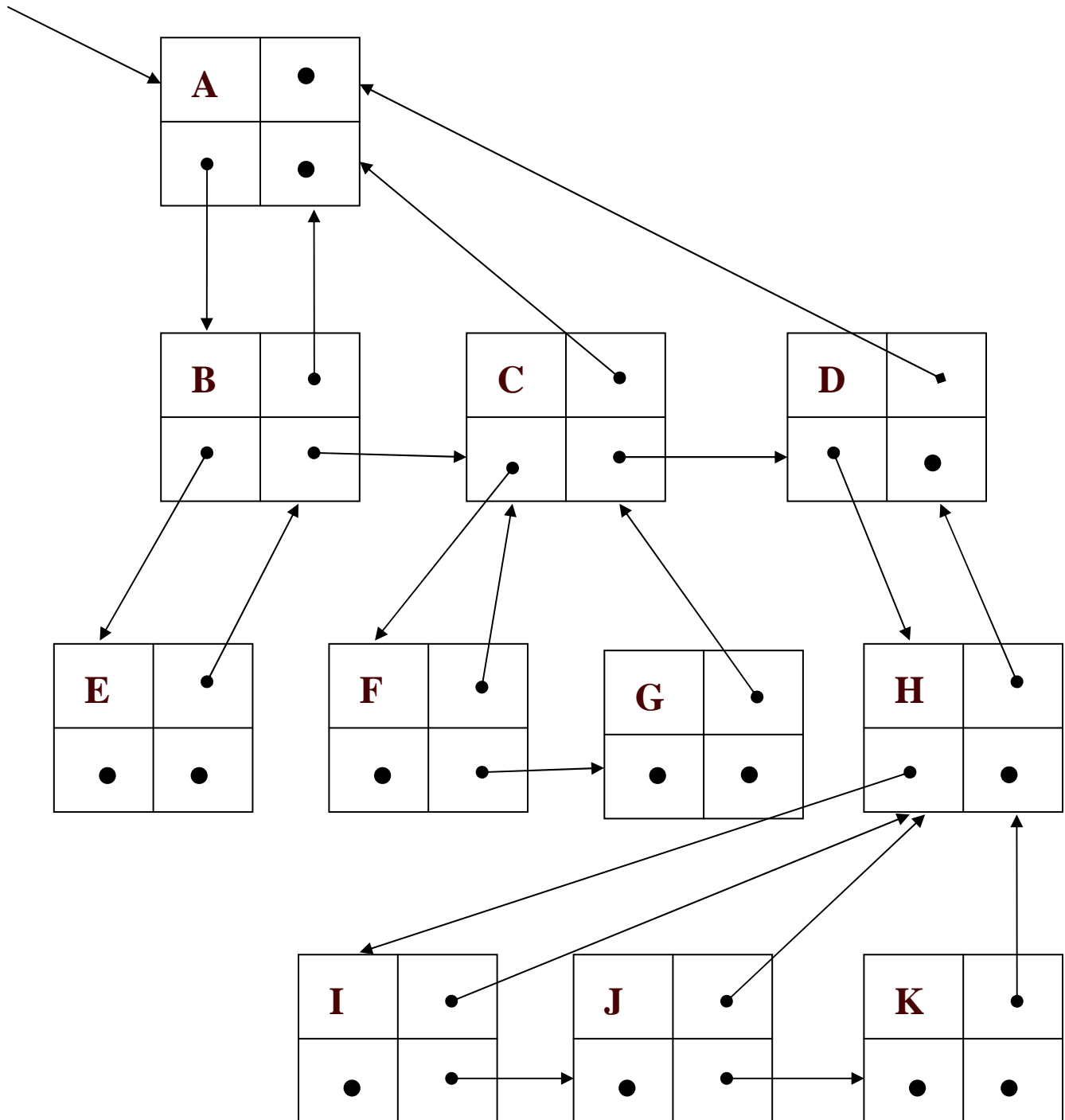
(A (B (E)) (C (F) (G)) (D (H (I) (J) (K))))

8.3.2 – Estruturas de dados

- Árvores também podem ser armazenadas em variáveis indexadas. É a chamada “estrutura contígua”.

	1	2	3	4	5	6	7	8	9	10	11
Info	A	B	E	C	F	G	D	H	I	J	K
Nº filhos	3	1	0	2	0	0	1	3	0	0	0
Pai	0	1	2	1	4	4	1	7	8	8	8

- Usando encadeamentos é mais fácil a manipulação:
“Encadeamento de pais e irmãos”:



- **Exemplo 8.3:** Programa para:

- Ler a forma parentesada de um árvore,
- Montar sua estrutura de dados e
- Escrever a árvore por ordem de nível.

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0

/*      Declaracoes de tipos      */

typedef struct celfila celfila;
typedef struct fila fila;
typedef struct celula celula;
typedef celula *noh;
struct celfila {
    noh elem;   celfila *prox;
};
struct fila {
    celfila *fr, *tr;
};
typedef celula *arvore;
struct celula {
    char info;   celula *pai, *fesq, *idir;
};
```

```
/*          Prototipos de funcoes          */
```

```
void analise (void);  
void escreverarvore (void);  
void armazena (noh *, noh);  
void erro (char []);  
void initfila (fila *);  
void entrarfila (noh, fila *);  
noh deletar (fila *);  
char vazia (fila);
```

```
/*          Variaveis globais          */
```

```
char expr[100];  
arvore arv;  
int i;  
char err;
```

```
/*          Programa principal: le a forma parentesada,  
          analisa e armazena a arvore, imprimindo-a. */
```

```
void main () {  
    err = FALSE;  
    printf ("Forma parentesada: ");  
    scanf ("%s", expr);  
    analise ();  
    if (err == TRUE) arv = NULL;  
    escreverarvore ();  
}
```

```

/*      Funcao analise: analisa sintaticamente a
        expressao e armazena-a na estrutura de
        encadeamento de pais e irmaos
    */

void analise () {
    i = 0;
    if (expr[i] == '$') arv = NULL;
    else if (expr[i] != '(') erro ("$ ou ( esperado");
    else {
        i++;
        armazena (&arv, NULL);
    }
    if (expr[i] != '$') erro("$ esperado");
}

/*      Funcao erro: emite mensagem de erro e da
        sinal para anular a arvore errada
    */

void erro (char s[ ]) {
    err = TRUE;
    printf ("\n\t%s", s);
}

```

```
/*      Funcao armazena (n, pai): armazena o no n na
        arvore, ligando-o a seu pai; chama a si propria
        para armazenar os filhos de n      */
```

```
void armazena (noh *n, noh pai) {
    noh ult;
    if (expr[i] < 'A' || expr[i] > 'Z') erro ("Letra esperada");
    else {
        *n = (celula *) malloc (sizeof (celula));
        (*n)->info = expr[i];
        (*n)->pai = pai;
        (*n)->fesq = NULL;
        (*n)->idir = NULL;
        for (i++; expr[i] == '('; ) {
            i++;
            if ((*n)->fesq == NULL) {
                armazena (&(*n)->fesq, *n);
                ult = (*n)->fesq;
            }
            else {
                armazena (&ult->idir, *n);
                ult = ult->idir;
            }
        }
        if (expr[i] != ')') erro (") esperado");
        else i++;
    }
}
```

```

/*      Funcao escrevearvore: escreve os nos da
        arvore em ordem de nivel      */

void escreverarvore () {
    fila fp, fs;
    noh x, y;
    printf ("\n");
    if (arv == NULL) printf ("\n\tArvore vazia");
    else {
        initfila (&fs);
        entrarfila (arv, &fs);
        do {
            printf ("\n");
            fp = fs;
            initfila (&fs);
            while (vazia(fp) == FALSE) {
                x = deletar(&fp);
                printf ("  %c(", x->info);
                if (x->pai == NULL) printf ("#");
                else printf ("%c)", x->pai->info);
                for (y = x->fesq; y != NULL; y = y->idir)
                    entrarfila (y, &fs);
            }
        } while (vazia (fs) == FALSE);
    }
}

```

```
/*          Funcao initfila: inicializa uma fila de nos      */
```

```
void initfila (fila *f) {  
    (*f).fr = (celfila *) malloc (sizeof (celfila));  
    (*f).tr = (*f).fr;  
    (*f).fr->prox = NULL;  
}
```

```
/*          Funcao entrarfila: entra com um no em uma      */  
          fila
```

```
void entrarfila (noh x, fila *f) {  
    (*f).tr->prox = (celfila *) malloc (sizeof (celfila));  
    (*f).tr = (*f).tr->prox;  
    (*f).tr->elem = x;  
    (*f).tr->prox = NULL;  
}
```



```
/*      Funcao deletar: deleta o elemento da frente  
de uma fila, retornando seu valor      */
```

```
noh deletar (fila *f) {  
    noh x;  
    celfila *p;  
    p = (*f).fr->prox;  
    x = p->elem;  
    if ((*f).tr == p) (*f).tr = (*f).fr;  
    (*f).fr->prox = p->prox;  
    free (p);  
    return x;  
}
```

```
/*      Funcao vazia: verifica se uma fila estah  
vazia  
      */
```

```
char vazia (fila f) {  
    char c;  
    if (f.fr == f.tr) c = TRUE;  
    else c = FALSE;  
    return c;  
}
```