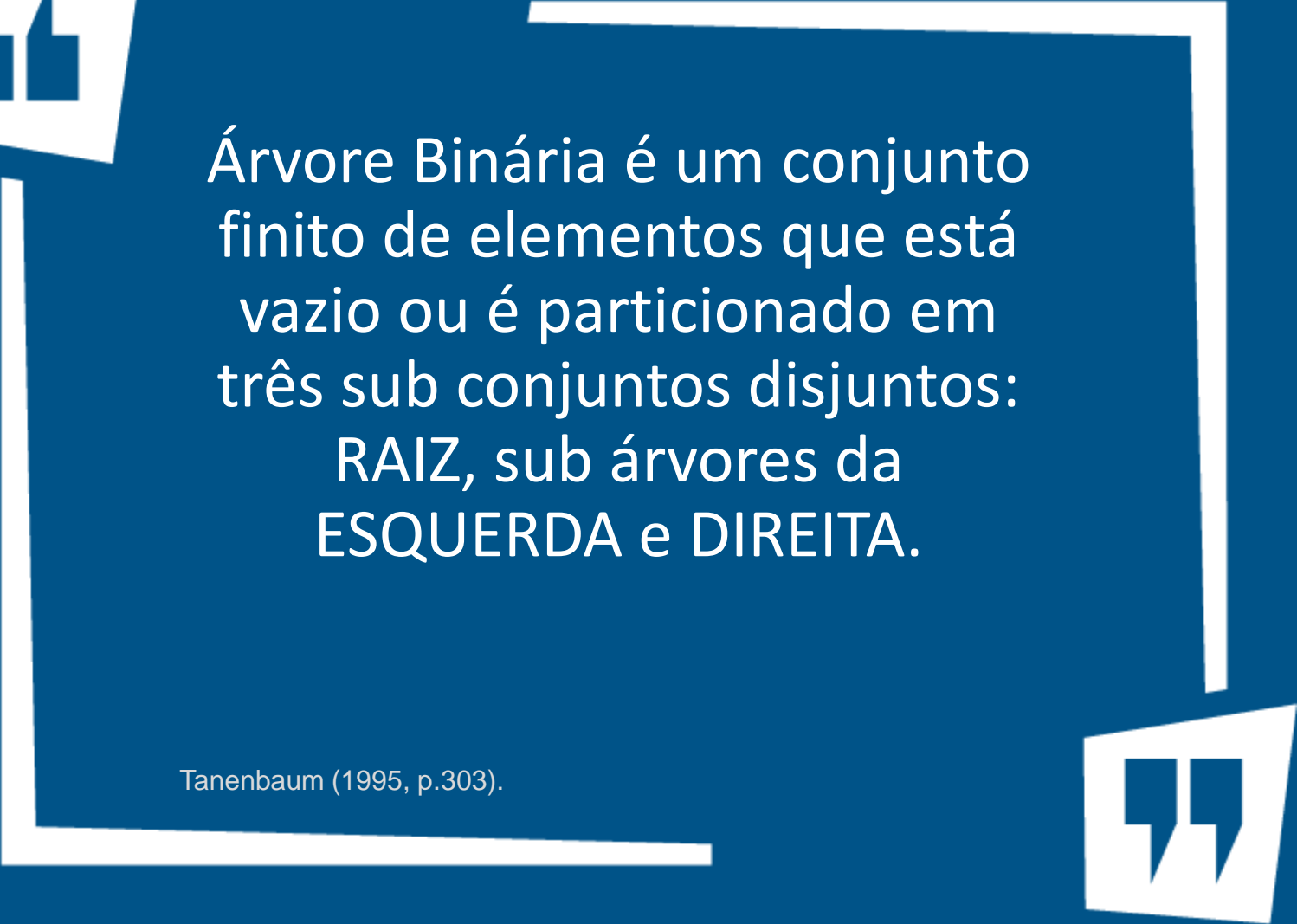



Estrutura de Dados II



Unidade 3 – Árvore Binária em C

Prof. Sandro T. Pinto



Árvore Binária é um conjunto finito de elementos que está vazio ou é particionado em três sub conjuntos disjuntos: RAIZ, sub árvores da ESQUERDA e DIREITA.

Tanenbaum (1995, p.303).

Unidade 3

Árvores Binárias

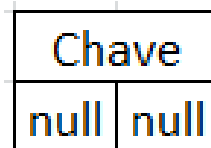
Como representar uma árvore binária em código?

Unindo os nós

Como representamos os nós?

Criando uma estrutura que guarde a informação necessária para a montagem da árvore, exemplo:

Criar 2 ponteiros: um para sub árvore da esquerda e outra para a sub árvore da direita, e também um campo para chave e os dados.



```
struct noArvore {  
    char info;  
    struct noArvore* esq;  
    struct noArvore* dir;  
};
```

Unidade 3

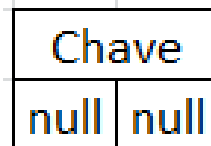
Árvores Binárias

Estrutura básica:

Guardamos aqui somente a chave? **não**. Podemos guardar qualquer informação para sua árvore.

Se estamos buscando alguma coisa precisamos de: uma **chave de busca**. Para verificar cada nó.

```
struct noArvore {  
    char info;  
    struct noArvore* esq;  
    struct noArvore* dir;  
};
```



Ponteiros para subir a árvore da esquerda e da direita.

Unidade 3

Implementando árvore binária em C

Existem várias formas de implementar uma árvore binária. A mais simples delas é usar um vetor de nós. Cada nó possui pelo menos três valores: **pai**, **esquerda** e **direita**. O Atributo **pai** vai apontar para a posição no vetor do **pai do nó**. O atributo **esquerda** vai armazenar a posição da raiz da **sub árvore esquerda**, e o atributo **direita** guarda a posição da raiz da **sub árvore direita**. Vamos adicionar mais um atributo, **dado**, que irá armazenar o **valor do nó**.

```
//Estrutura
Struct str_no {
    char dado;
    int esquerda;
    int direita;
    int pai;
}
```

Unidade 3

Implementando Árvore Binária em C

Com a estrutura já definida, precisamos criar uma variável do tipo **vetor** de **str_no**. Esse vetor terá o tamanho definido por uma constante chamado **tamanho**. Precisamos também de uma variável do tipo **inteiro** que servirá de índice para o nosso vetor.

```
//Constantes
#define tamanho 100

//Variáveis
Struct str_no arvore[tamanho];
Int indice=0;
}
```

Unidade 3

Implementando Árvore Binária em C

Para inserir um nó na árvore, é preciso saber o seu valor, quem é o **pai**, e se ele é um filho a **esquerda** ou **direita**.

Mesmo sabendo quem é o **pai**, antes de fazer a inserção no vetor, é preciso encontrar sua localização. Para isso é preciso criar uma função chamada de **arvore_procura**, que retorna um valor inteiro (posição no vetor) e têm como parâmetro o nome do **pai**.

```
//Procura nó
Int arvore_procura(char dado) {
    if (indice != 0) {
        for (int i = 0; i<indice; i++) {
            if (arvore[i].dado == dado) {
                return (i);
            }
        }
    }
    else {
        return (0);
    }
}
```


Unidade 3

Implementando Árvore Binária em C

Note que o programa faz uma verificação no valor da variável **índice**. Se o valor for **0** significa que a árvore está vazia e o valor a ser inserido será a raiz da árvore. A função leva em conta que o **pai** está presente na árvore. Já posto que não foi previsto nenhum tipo de retorno de erro para caso o **pai** não ser encontrado.

```
//Procura nó
Int arvore_procura(char dado) {
    if (indice != 0) {
        for (int i = 0; i<indice; i++) {
            if (arvore[i].dado == dado) {
                return (i);
            }
        }
    }
    else {
        return (0);
    }
}
```

Unidade 3

Implementando Árvore Binária em C

A função **arvore_insere** terá como parâmetro um valor inteiro que representa a posição do pai no vetor, o valor dado digitado pelo usuário e a sua posição de **descendência (se é filho do lado esquerdo ou direito)**.

```
//Inserir nó
void arvore_insere(int pai, char dado, int lado) {
    switch (lado) {
        case E:
            arvore[pai].esquerda = indice;
            arvore[indice].dado=dado;
            arvore[indice].pai= pai;
            arvore[indice].esquerda = -1;
            arvore[indice].direita = -1;
            indice ++;
        break;
        case D:
            arvore[pai].direita = indice;
            arvore[indice].dado=dado;
            arvore[indice].pai= pai;
            arvore[indice].esquerda = -1;
            arvore[indice].direita = -1;
            indice ++;
        break;
    }
}
```

Unidade 3

Implementando Árvore Binária em C

Inicialmente criamos uma variável chamada **indice**, que guarda a primeira posição livre da árvore. O **parâmetro pai** recebido na **função** indica qual a posição do nó pai.

Se novo nó for **filho a esquerda**, atribuiremos o valor do indice ao atributo esquerdo da árvore.

Case E `arvore[pai].esquerda = indice;`

No caso de **filho a direita**, colocamos o valor de indice no atributo direita.

Case D `arvore[pai].direita = indice;`

Unidade 3

Implementando Árvore Binária em C

O próximo passo é guardar o nome do nó no atributo dado e a referência do **pai**. Marcamos **-1** os valores de esquerda e direita para identificar que ambos os ponteiros não apontam para um **sub árvore**. Veja o código:

```
arvore[indice].dado=dado;  
arvore[indice].pai= pai;  
arvore[indice].esquerda = -1;  
arvore[indice].direita = -1;  
indice ++;
```

Unidade 3

Implementando Árvore Binária em C

Foi apresentado uma forma de armazenar uma **árvore binária** em um vetor. Essa implementação é bem simples e rápida, mas **longe do ideal**. Isso porque voltamos ao **mesmo problema de estruturas anteriores** em que **muita memória é alocada** na execução do programa.

Esse programa procura sempre a primeira posição livre no vetor para posicionar um novo nó na árvore. Dessa forma, a ordenação da estrutura estará diretamente ligada à **ordem em que os nós foram adicionados**.

Unidade 3

Uma Árvore Binária Diferente

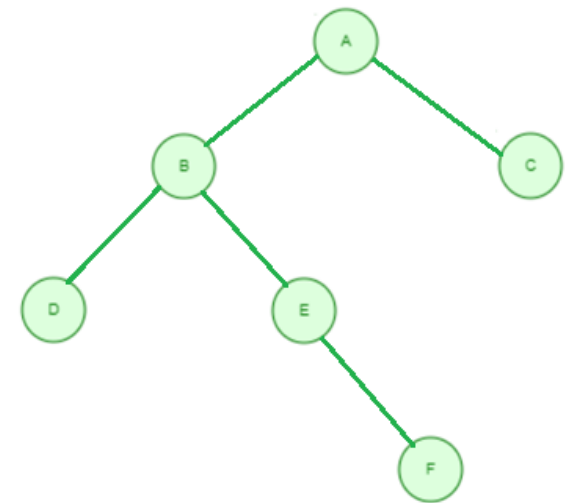
Outra forma de armazenar uma árvore binária em um vetor é **reservar as posições de acordo com o nível** e descendência de cada nó.

O primeiro nó a ser armazenado é a raiz da árvore e ele ficará na primeira posição do vetor. Lembrando que a primeira posição do vetor em C é o 0(zero).

Unidade 3

Uma Árvore Binária Diferente

Vamos simular a inserção da árvore da Figura em um vetor de 16 posições. O primeiro nó a ser inserido será a raiz da árvore A e ocupará a **posição 0(zero)** do vetor.

[illegible]

Unidade 3

Uma Árvore Binária Diferente

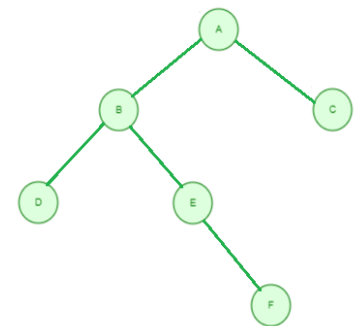
Como o objetivo é manter a árvore indexada dentro do vetor, vamos reservar a primeira posição denominada de **p** logo após o nó para armazenar o filho a esquerda, e a segunda posição para o filho a direita.

Assim, para um nó armazenado numa posição $p(0)$ qualquer, seu filho esquerdo estará na:

posição $p+1 = B$

E se o filho estiver a direita na:

posição $p+2 = C$

[illegible]

Unidade 3

Uma Árvore Binária Diferente

Usando a formula que foi proposta, para encontrar os filhos B com $p+1=2$ da posição do vetor, e C com $p+2=3$ da posição do vetor, **não serve para a ordenação** toda a árvore binária em vetor de dados.

Uma árvore binária cresce de forma geométrica posto que cada nó tem dois filhos, que por sua vez, são duas sub árvores que podem estar vazias ou não. Independentemente de o filho existir, seu espaço precisa ser reservado no vetor.

Unidade 3

Uma Árvore Binária Diferente

Adequando a formula para o vetor atender a uma árvore binária deve ficar assim:

$2 * p + 1$ para o filho a esquerda

$2 * p + 2$ para o filho a direita

Unidade 3

Uma Árvore Binária Diferente

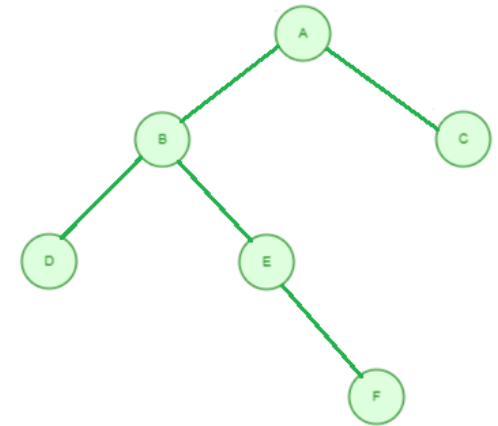
Aplicando as novas formulas para o nó A que está na posição 0(zero), temos que:

2^*p+1 $2^*0+1=1$ posição do nó B da esquerda

2^*p+2 $2^*0+2=2$ posição do nó C da direita

Unidade 3

Uma Árvore Binária Diferente



Para B, que está na posição 1, temos :

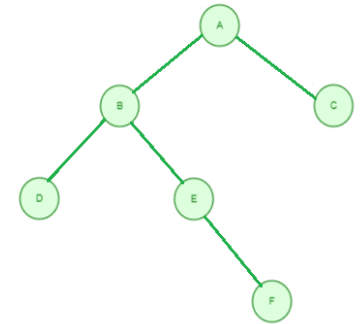
2^{*p+1} $2^{*1+1}=3$ posição do nó D filho esquerdo de B

2^{*p+2} $2^{*1+2}=4$ posição do nó E filha direita de B

[illegible]

Unidade 3

Uma Árvore Binária Diferente



O próximo será D, que também é uma folha. Então será reservado no vetor as posições

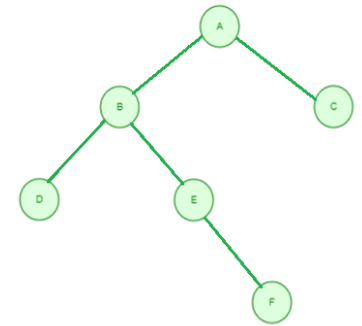
$2 * p + 1$ $2 * 3 + 1 = 7$ posição do nó __ filho esquerdo de D

$2 * p + 2$ $2 * 3 + 2 = 8$ posição do nó __ filha direita de D

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E	-	-	-	-							

Unidade 3

Uma Árvore Binária Diferente



O último nó de nível 2 (E) possui um filho direita (F), que será armazenado na posição:

$2 * p + 1$ $2 * 4 + 1 = 9$ posição do nó __ filho esquerdo de E

$2 * p + 2$ $2 * 4 + 2 = 10$ posição do nó F filho direita de E

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E	-	-	-	-	-	F					

Unidade 3



Referências:

SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. Estruturas de dados e seus algoritmos. 2ed. Rio de Janeiro: LTC, 1994. 320p.

TENENBAUM, Aaron M.; LANGSAM, Yedidiah; AUGENSTEIN, Moshé J.. Estruturas de dados usando C. São Paulo: Makron Books, 1995. 884p.

VELOSO, Paulo et al.. Estruturas de dados. Rio de Janeiro: Campus, 2001. 228p

Atividades - Unidade 3

