Aula 11 - Estrutura de dados 1 – Uniube

Prof. Marcos Lopes

34 9 9878 0925

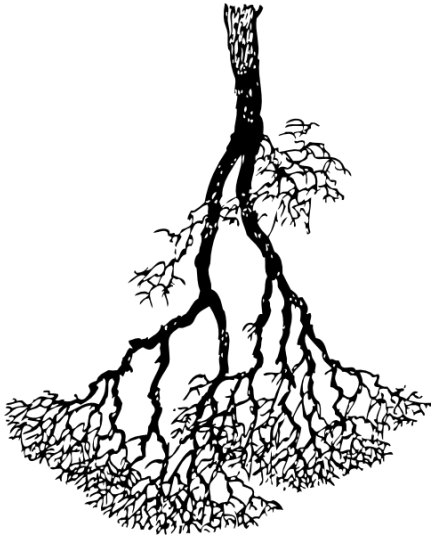
Estrutura de dados do tipo árvore binária:

Uma árvore binária é uma estrutura de dados mais geral que uma lista encadeada.

Vamos introduzir algumas operações básicas sobre árvores binárias.

As árvores da computação têm a curiosa tendência

de crescer para baixo…



Nós e seus filhos:

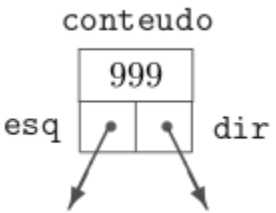
Uma árvore binária (= binary tree) é um conjunto de nós que satisfaz certas condições.

As condições não serão dadas explicitamente, mas elas ficarão implicitamente claras no contexto.

Os registros serão chamados de “nós” (poderiam também ser chamados células).

Cada nó tem um endereço. Suporemos, por enquanto, que cada nó tem apenas três campos: um número inteiro e dois ponteiros para nós. Os nós podem, então, ser definidos assim:

```
typedef struct reg {
      int  conteudo;
      noh *esq;
      noh *dir;
} noh;
```



Nós e seus filhos (continuação):

O campo conteudo é a carga útil do nó; os dois outros campos servem apenas para dar estrutura à árvore. O campo esq de cada nó contém NULL ou o endereço de outro nó.

Uma hipótese análoga vale para o campo dir.

Se o campo esq de um nó P é o endereço de um nó E, diremos que E é o filho esquerdo de P.

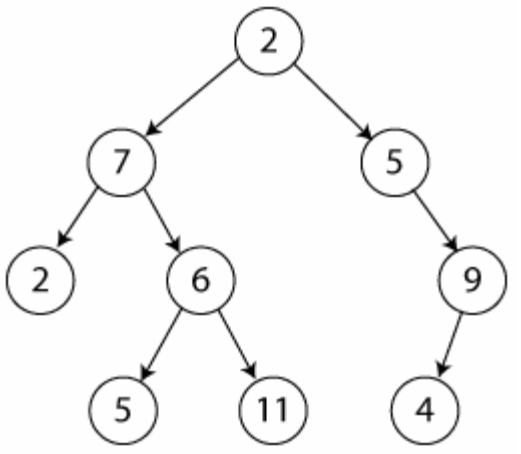
Analogamente, se P.dir é igual a &D, diremos que D é o filho direito de P. Se um nó F é filho (esquerdo ou direito) de P, diremos que P é o pai de F. Uma folha (= leaf) é um nó que não tem filho algum.

É muito conveniente confundir, verbalmente, cada nó com seu endereço.

Assim, se x é um ponteiro para um nó (ou seja, se x é do tipo *noh), dizemos considere o nó x em lugar de considere o nó cujo endereço é x.

Nós e seus filhos (continuação):

A figura abaixo mostra dois exemplos de árvores binárias. Do lado esquerdo, temos uma curiosa árvore binária na natureza. Do lado direito, uma representação esquemática de uma árvore binária cujos nós contêm os números 2, 7, 5, etc.



Exercícios 1)

a) Dê uma lista das condições que um conjunto de nós deve satisfazer para ser considerado uma árvore binária.

Árvores e subárvores:

Suponha que x é um nó.

Um descendente de x é qualquer nó que possa ser alcançado pela iteração das instruções `x = x->esq` e `x = x->dir` em qualquer ordem.

(É claro que essas instruções só podem ser iteradas enquanto x for diferente de NULL. Estamos supondo que NULL é de fato atingido mais cedo ou mais tarde.)

Um nó x juntamente com todos os seus descendentes é uma árvore binária.

Dizemos que x é a raiz (= root) da árvore. Se x tiver um pai, essa árvore é subárvore de alguma árvore maior. Se x é NULL, a árvore é vazia.

Para qualquer nó x, o nó `x->esq` é a raiz da subárvore esquerda de `x` e `x->dir` é a raiz da subárvore direita de x.

Endereço de uma árvore e definição recursiva:

O endereço de uma árvore binária é o endereço de sua raiz.

É conveniente confundir, verbalmente, árvores com seus endereços: dizemos considere a árvore r em lugar de considere a árvore cuja raiz tem endereço r.

Essa convenção sugere a introdução do nome alternativo arvore para o tipo-de-dados ponteiro-para-nó:

```
typedef noh *arvore;        // árvore
```

Dada essa convenção, podemos dar uma definição recursiva de árvore binária: um ponteiro-para-nó r é uma árvore binária se

1) r é NULL ou
2) r->esq e r->dir são árvores binárias.

Muitos algoritmos sobre árvores ficam mais simples quando escritos em estilo recursivo.

Exercícios 2)

a) Sequências de parenteses. Árvores binárias têm uma relação muito íntima com certas sequências bem-formadas de parênteses. Discuta essa relação.

b) Expressões aritméticas. Árvores binárias podem ser usadas, de maneira muito natural, para representar expressões aritméticas (como ((a+b)*c-d)/(e-f)+g, por exemplo). Discuta os detalhes.

Varredura esquerda-raiz-direita:

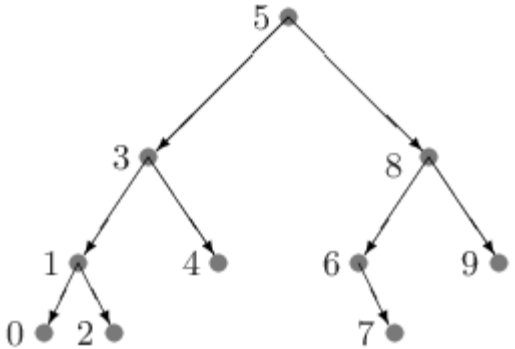
Uma árvore binária pode ser percorrida de muitas maneiras diferentes.

Uma maneira particularmente importante é a esquerda-raiz-direita, ou e-r-d, também conhecida como inorder traversal, ou varredura infixa, ou varredura central.

A varredura e-r-d visita:

1) a subárvore esquerda da raiz, em ordem e-r-d,
2) a raiz,
3) a subárvore direita da raiz, em ordem e-r-d,

nessa ordem.



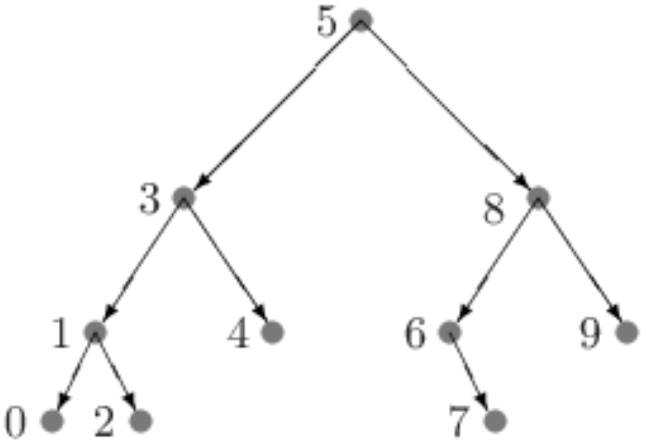
Na figura ao lado, os nós estão numeradas na ordem em que são visitados pela varredura e-r-d.

Varredura esquerda-raiz-direita (continuação):

Eis uma função recursiva que faz a varredura e-r-d de uma árvore binária:

```
// Recebe a raiz r de uma árvore binária e
// imprime os conteúdos dos seus nós
// em ordem e-r-d.

void erdPrint (arvore r) {
   if (r != NULL) {
      erd (r->esq);
      printf ("%d\n", r->conteudo);
      erd (r->dir);
   }
}
```



É um excelente exercício escrever uma versão iterativa dessa função.

Exercícios 3)

a) Considere o modelo de dados de um nó de uma árvore binária e o esqueleto de código a seguir e complete o que está faltando (onde tem //código aqui):

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

// Definição da estrutura do nó da árvore
struct TreeNode {
    int data;
    struct TreeNode *left;
    struct TreeNode *right;
};

// Função para inserir um nó na árvore
struct TreeNode* insertNode(struct TreeNode* root, int value) {
    // Se a árvore estiver vazia, retorna um novo nó
    //código aqui

    // Caso contrário, percorre a árvore até achar a posição do nó
    //código aqui

    // Retorna o ponteiro para o nó raiz (sem alterações se o valor já existir na árvore)
    return root;
}
```

```
// Função para imprimir a árvore em ordem
void printInOrder(struct TreeNode* root) {
    //código aqui
}

// Função principal
int main() {
    struct TreeNode* root = NULL;

    // Inserindo nós na árvore
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);

    // Imprimindo a árvore em ordem
    printf("Árvore em ordem: ");
    printInOrder(root);
    printf("\n");

    return 0;
}
```