

Árvore Binária de Busca

Prof. Flavio B. Gonzaga
flavio.gonzaga@unifal-mg.edu.br
Universidade Federal de Alfenas
UNIFAL-MG

Árvore Binária de Busca

- Árvore Binária de Busca generalizam a ideia de listas encadeadas crescentes;
- Considerando a estrutura de cada nó como sendo:

```
typedef struct noArvore_ noArvore;  
struct noArvore_  
{  
    int valor;  
    noArvore * esq;  
    noArvore * dir;  
};
```

- Uma árvore binária deste tipo é de busca se cada nó **p** tem a seguinte propriedade: o valor de **p** é maior ou igual ao valor de cada nó da subárvore esquerda de **p** e; menor ou igual ao valor de cada nó da subárvore direita de **p**;

Árvore Binária de Busca

- Inserção de nó (recursivo):

```
noArvore * insere (noArvore * raiz, noArvore * n) {  
    if (raiz == NULL) return n;  
    if (raiz->valor > n->valor)  
        raiz->esq = insere(raiz->esq, n);  
    else  
        raiz->dir = insere (raiz->dir, n);  
    return raiz;  
}
```

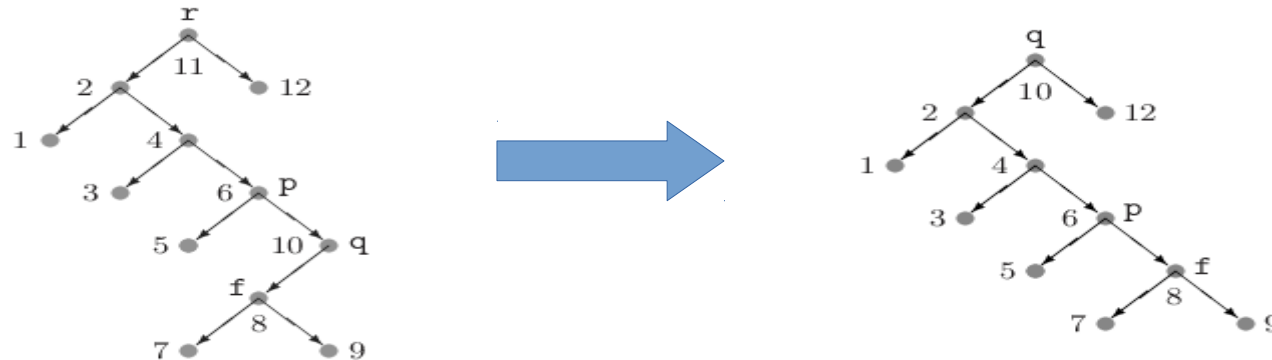
Árvore Binária de Busca

- Busca de nó (recursivo):

```
noArvore * busca (noArvore * raiz, int k) {  
    if (raiz == NULL || raiz->valor == k)  
        return raiz;  
    if (raiz->valor > k)  
        return busca (raiz->esq, k);  
    else  
        return busca (raiz->dir, k);  
}
```

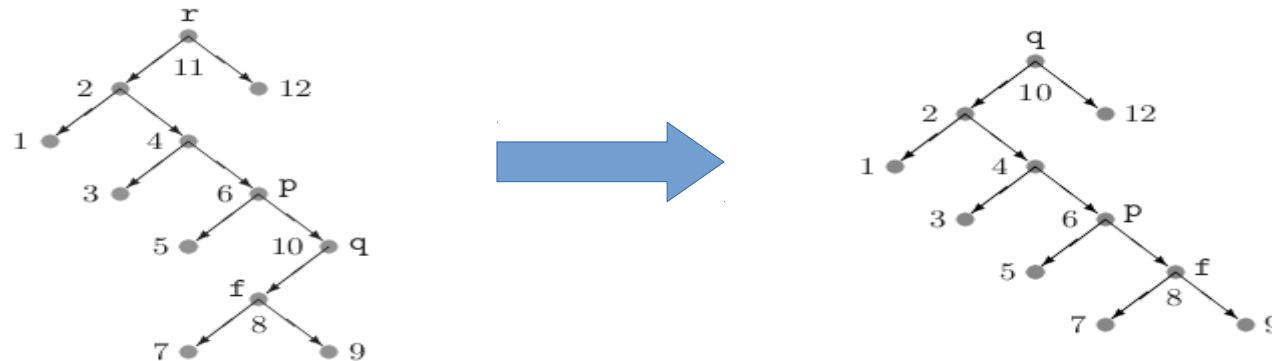
Árvore Binária de Busca

- Remove nó (raiz):
 - Se a raiz não tem um dos filhos, basta que o outro filho assuma o papel de raiz;
 - Senão, faça com que o nó anterior à raiz na ordem e-r-d (esquerda – raiz – direita) assumo o papel de raiz;



Árvore Binária de Busca

- Remove nó qualquer:
 - A remoção dos demais nós segue a mesma ideia da remoção do nó raiz;
 - Ou seja, pode-se fazer uso da função que remove o nó raiz, passando como parâmetro o nó a ser removido;



Árvore Binária de Busca

- Remove nó qualquer:
 - Uma possível implementação seria:

```
noArvore * removeNo(noArvore * raiz, int valor) {
    noArvore * n = busca(raiz, valor); //checa se o noh existe
    if (n) {
        noArvore * pai = buscaPai(raiz, n); //descobre quem eh o noh pai
        if (pai) { //caso tenha noh pai
            if (pai->dir == n) //noh a ser removido eh filho a direita
                pai->dir = removeRaiz(n);
            else //noh a ser removido eh filho a esquerda
                pai->esq = removeRaiz(n);
        } else { //nao possui pai, logo, eh o proprio noh raiz
            raiz = removeRaiz(n);
        }
    }
    return raiz;
}
```

Referências Bibliográficas

- Estruturas de Dados e Seus Algoritmos. Szwarcfiter J. L.; Markenzon L.. 3a Edição. Editora LTC. 2010.
- Estruturas De Dados Usando C. Tenenbaum A. M.; Langsam Y.; Augenstein M. J.. 1a Edição. Editora Pearson. 1995.
- Introdução a Estruturas de Dados: Com Técnicas de Programação em C. Celes W.; Cerqueira R.; Rangel J.. 2a Edição. Editora Elsevier. 2017.
- <https://www.ime.usp.br/~pf/algoritmos/aulas/binst.html> , acesso em 09/11/2018.