4. Árvore binária de busca

Uma árvore binária é dita de busca se suportar operações eficientes de busca, inserção e remoção. Para ser eficiente deve-se ter um critério de ordenação de dados. Assim, uma árvore binária de busca poderá ter como critério que a partir de um nó R os elementos de sua sub-árvore esquerda tem somente elementos menores que R e a sua sub-árvore direita tem somente elementos maiores que R.

4.1. Inserção em Árvore Binária de Busca

Vejamos a rotina:

```
void insere_arvore(def_arvore *arvore, int valor)
{
     def_arvore p;

    if (*arvore!=NULL){
       if ((*arvore)->info > valor) insere_arvore(&((*arvore)->esq),valor);
       else if ((*arvore)->info < valor) insere_arvore(&((*arvore)->dir),valor);
       else printf("O numero ja existe\n");}
    else{
            p=(def_arvore)malloc(sizeof(struct no_arvore));
            p->info = valor;
            p->esq = NULL;
            p->dir = NULL;
            *arvore=p; }
}
```

Verifiquemos a árvore que será construída usando esse algoritmo e a entrada de dados = 14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17, 9, 14, 5.

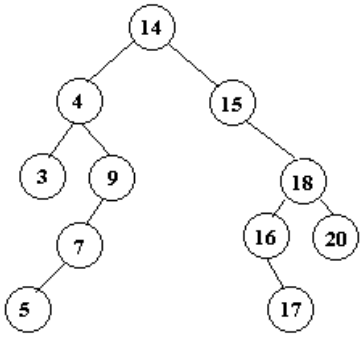

Figura 9. A árvore gerada pelo algoritmo

4.2. Busca em Árvores Binárias de Busca

O Problema da busca:

Seja S = {s_1, ..., s_n} o conjunto de chaves satisfazendo $s_1 < ... < s_n$. Seja x um valor dado. O objetivo é verificar se x E S ou não. Em caso positivo, localizar x em S, isto é, determinar o índice j tal que $x = s_j$.

Para resolver o problema usa-se uma árvore binária T de busca que possui as seguintes características:

(i) T possui n nós. Cada nó v corresponde a uma chave distinta sj E S e possui como rótulo o valor r(v) = sj.

(ii) Seja um nó v de T. Seja também v1 pertencente à subárvore esquerda de v. Então r(v1) < r(v). Analogamente, se v2 pertence à subárvore direita de v, r(v2) > r(v).

4.3. Remoção de nós em uma Árvore de Busca Binária

Quando queremos remover um nó temos três situações para estudar:

(a) o nó a ser removido não tem filhos, ou seja, ele é um nó folha
(b) o nó possui um único filho
(c) o nó possui dois filhos

Caso (a): remoção do nó folha

Basta retirar o nó. Não precisa fazer nenhum ajuste na árvore. Veja Figura 10.

Caso (b): remoção do nó que possui um filho

O filho deverá ser movido para cima para ocupar a posição do nó removido. Assim, deve-se fazer com que o pai do nó que vai ser removido passe a apontar para o filho do nó que está sendo removido. Veja Figura 11.

Caso (c): remoção do nó que possui dois filhos

Procura-se um elemento da subárvore a direita do nó que vai ser retirado. Um elemento que é uma folha ou um nó que não possua subárvore à esquerda, ou seja, o menor elemento do lado direito do nó a ser removido. Retira esse elemento de sua posição e o coloca na posição do nó a ser removido. Veja Figura 12.

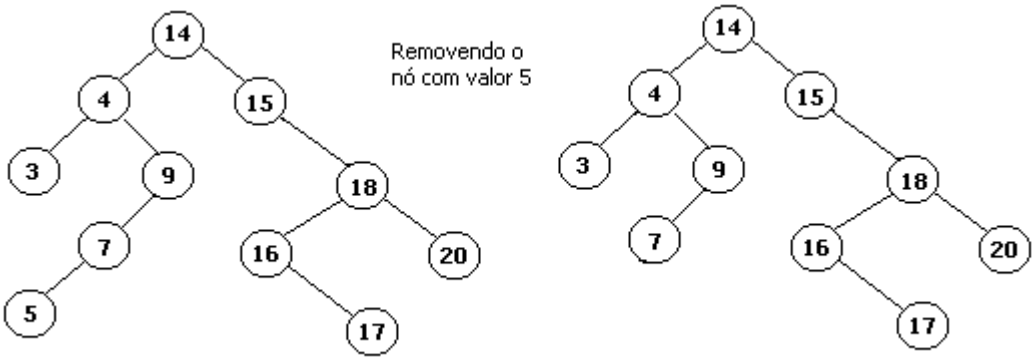

Figura 10. Exemplo de remoção do caso (a)

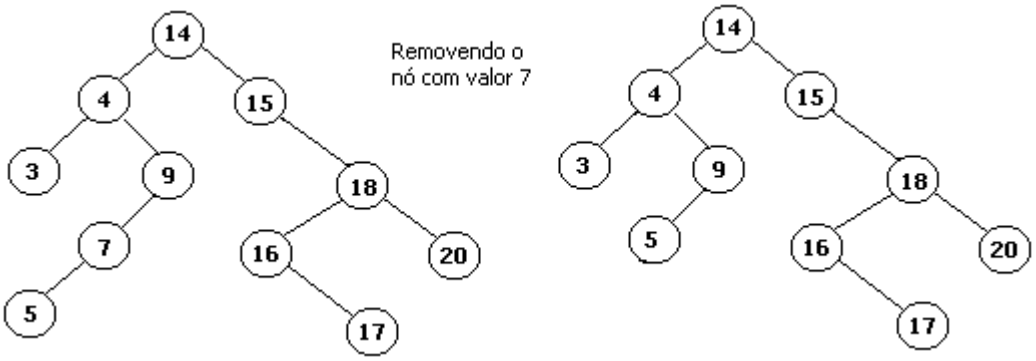

Figura 11. Exemplo de remoção do caso (b)

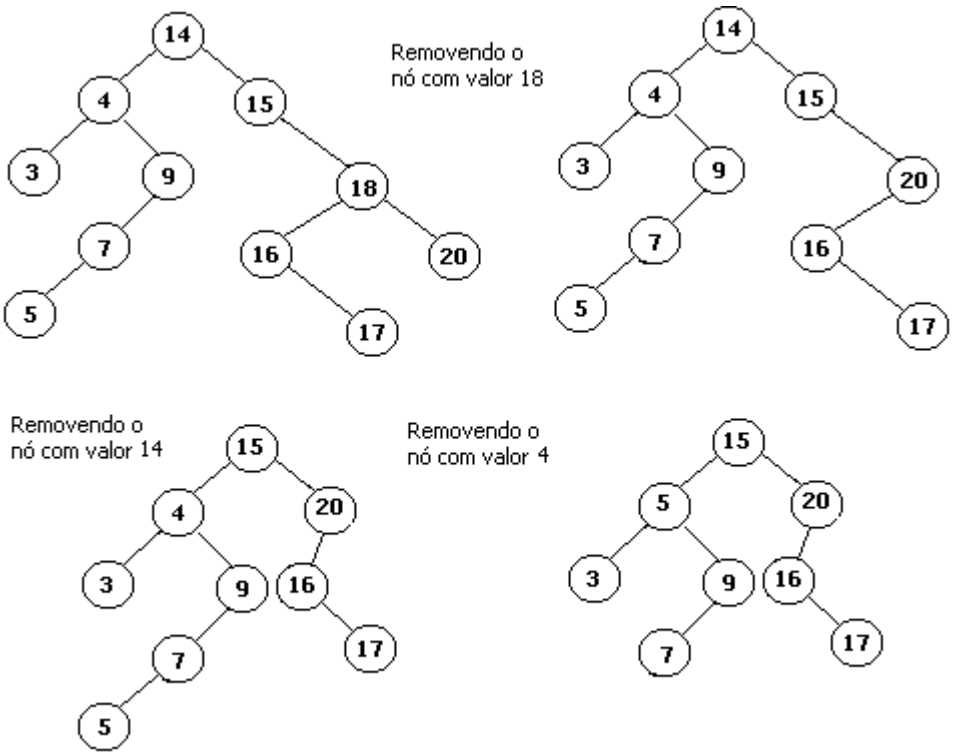

Figura 12. Exemplo de remoção do caso (c)

OBS:

Esse tipo de remoção de nós de uma árvore de busca binária baseou-se no algoritmo dado no livro do Tenenbaum et al. (1995, p.514), mas pode-se implementar a remoção de outras formas.

A remoção de nós nos casos (a) e (b) permanecem inalterados, mas a remoção no caso (c) onde o nó tem dois filhos pode aparecer de forma diferente em outros livros. Poderia pensar que nesse caso (c), o elemento substituto é o maior elemento que está na subárvore esquerda do nó que vai ser removido. Esse elemento pode ser uma folha ou um nó que não tenha subárvore direita. Veja Figura 13.

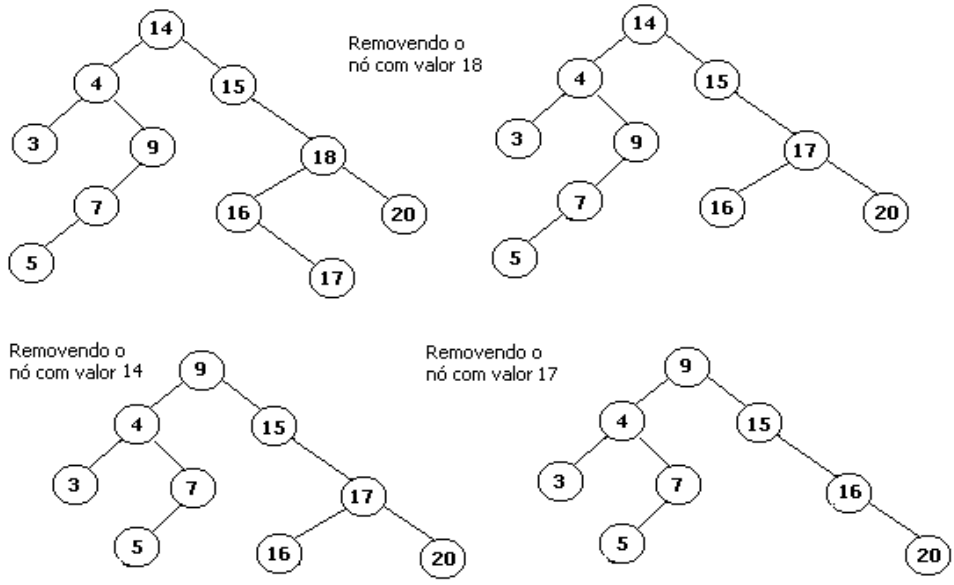

Figura 13. Exemplo de remoção do caso (c)