Árvores – Codificação 2

4. Códigos para Árvore Binária

Na estrutura de dados árvore binária cada elemento possui, além da carga útil de dados, dois ponteiros, um para a subárvore da esquerda e outro para a subárvore da direita.
Então a definição da estrutura pode ser codificada como no exemplo a seguir:

```
struct Elemento {
    int dado;
    struct Elemento *esq;
    struct Elemento *dir;
};

typedef struct Elemento *Arvore;
```

As operações mínimas definidas para uma árvore binária são:
- A criação de uma árvore vazia (criaArvore);
- A inserção de um elemento em uma árvore (insereArvore);
- A busca de um elemento em uma árvore (buscaArvore);
- A retirada de um elemento de uma árvore (retiraArvore);
- O percurso em pré-ordem (preOrdem);
- O percurso em in-ordem (inOrdem);
- O percurso em pós-ordem (posOrdem);

Opcionalmente, podem ser definidos também as seguintes operações:
- Teste se uma árvore está vazia (arvoreVazia);
- Remover todos os elementos de uma árvore (liberaArvore);

Continuando a codificação das operações sobre a árvore, hoje veremos a retirada de um elemento da árvore.

Os demais códigos (4.1 a 4.6 listados abaixo) já foram vistos na parte 1 da codificação.

4.1. A criação de uma árvore vazia (criaArvore)
4.2. A inserção de um elemento em uma árvore (insereArvore)
4.3. A busca de um elemento em uma árvore (buscaArvore)
4.4. O percurso em pré-ordem (preOrdem)
4.5. O percurso em in-ordem (inOrdem)
4.6. O percurso em pós-ordem (posOrdem)

4.7. A retirada de um elemento de uma árvore (retiraArvore)

A retirada de um elemento de uma árvore é o procedimento mais complexo que os já apresentados, e seus princípios já foram discutidos no item 3.2, do texto de conceitualização de árvore, e relembrado aqui para melhor compreensão do código.
Para retirar um elemento de uma árvore binária é preciso localizar o elemento, refazer as conexões da árvore e então liberar o espaço ocupado pelo elemento. Mas diferentemente das listas lineares, na árvore não há somente um prosseguimento. Então é preciso tomar uma decisão de qual elemento colocar no lugar do elemento retirado, de modo a manter a integridade da árvore, ou seja, manter a

mesma regra de formação, que diz que os elementos menores que a raiz são alocados na subárvore da esquerda e os elementos maiores que a raiz são alocados na subárvore da direita.
Dessa forma, a remoção precisa considerar os possíveis casos do nodo que será retirado.
1. Nodo sem filhos (nodo folha). Nesse caso, basta retirar o próprio nodo. Por exemplo, para remover o nodo 5, 13 ou 25 da figura 1, basta remover o próprio nodo.

2. Nodo com somente um filho. Nesse caso, o filho do nodo retirado passa a ser conectado diretamente ao pai do nodo retirado. Por exemplo, para retirar o nodo 9 da figura 1, basta fazer o nodo 5 se tornar o filho da esquerda do nodo 10.

3. Nodo com dois filhos. Nesse caso, a árvore tem elementos maiores e menores que o nodo a ser retirado. Para manter a integridade com o mínimo de movimentações, é preciso escolher o nodo com valor mais próximo (o antecessor (imediatamente menor) ou o sucessor (imediatamente maior)). Pode-se optar por colocar no lugar do nodo retirado o maior elemento da subárvore da esquerda (o antecessor) ou o menor elemento da subárvore da direita (sucessor). Isso porque esses dois elementos são os que estão numericamente mais próximos do valor do elemento a ser retirado. Na codificação o programador escolhe um dos lados para remover.

Por exemplo, para retirar o elemento 15 (raiz) da árvore da figura 2, podemos colocar no lugar do 15 o elemento 13, que é o antecessor do 15 (maior elemento da subárvore da esquerda).

Observe que o elemento 13 na raiz consegue manter a estrutura da árvore, pois todos os elementos da esquerda da raiz são menores que o 13 e os elementos à direita da raiz são maiores que o 13. A outra opção possível é o elemento 21 (menor elemento da subárvore da direita). Se fizermos essa escolha, o elemento 21 também consegue manter a estrutura da árvore, pois todos os elementos à esquerda da raiz são menores que o 21 e os da direita são maiores que o 21.

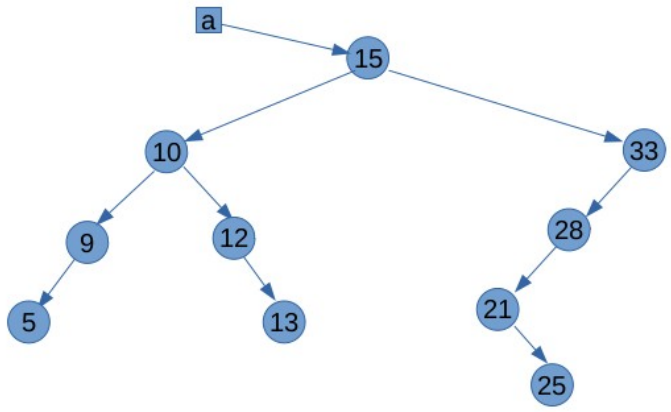

Figura 1. Exemplo de árvore binária de inteiros.

Em geral a codificação da retirada de um elemento da árvore binária é dividida em duas funções. A primeira é a retiraArvore, que busca o elemento na árvore (observe a semelhança das linhas 5 a 10 com o buscaArvore), e se encontrar, retira o elemento caso ele seja elemento folha (sem filhos), ou se tiver somente um filho (nas linhas 11 a 20). Para o caso de retirar um elemento que tenha dois filhos, é chamada uma função auxiliar, que precisa encontrar nesse caso o antecessor do elemento a ser retirado, que por sua vez é o maior elemento da subárvore da esquerda (linha 22).

Observe também que, para retirar um elemento de uma árvore `a`, é necessário passar a árvore por referência (ponteiro para `a`, e não `a`), pois `a` pode ser alterado.

```
// Função que retira o elemento valor da árvore apontada por a, se existir
1  void retiraArvore(Arvore *a, int valor)
2  {
3     Arvore aux;
4
5     if (*a != NULL) { // Árvore não nula
6        if (valor > ((*a)->dado)) { // Retira na sub-árvore da direita
7           retiraArvore(&((*a)->dir), valor);
8        } else if (valor < ((*a)->dado)) { // Retira na sub-árvore da esquerda
9           retiraArvore(&((*a)->esq), valor);
10       } else { // Achou elemento, vai remover
11          if ((*a)->dir == NULL) { // Elemento não tem filho da direita,
12                                   // promove filho da esquerda
13             aux = (*a);
14             *a = aux->esq;
15             free(aux);
16          } else if ((*a)->esq == NULL) { // Elemento não tem filho da esquerda,
17                                  // promove filho da direita
18             aux = (*a)->dir;
19             free(*a);
20             *a = aux;
21          } else { // Elemento tem dois filhos, promove o antecessor
22             antecessor(*a, &((*a)->esq));
23          }
24       }
25    }
26 }
```

A função antecessor recebe por parâmetro o endereço do elemento a ser retirado, e endereço do ponteiro para um candidato a substituir o elemento a ser retirado da árvore. Essa função deve buscar o maior elemento nessa subárvore (o elemento mais à direita), pois é o mais próximo numericamente ao elemento que será retirado.

Observe no código abaixo, que a função chama a si mesma recursivamente, sempre que o candidato a substituto tiver um filho da direita e, portanto, maior que o próprio candidato (linhas 7 a 9).

Quando não tiver mais filho da direita, a função encontrou o substituto, move o valor para a posição apontada por `a` (linha 13), promove o filho da esquerda do substituto para o lugar dele, se houver (linhas 14 e 15), e depois libera o espaço usado pelo substituto (linha 18).

```
1  void antecessor(Arvore a, Arvore *x)
2  {  // Coloca no elemento a o maior elemento da subárvore *x
3     // a é o endereço do elemento a ser retirado
4     // x é o endereço do ponteiro para o candidato a substituto de a
5     Arvore aux;
6
7     if ((*x)->dir != NULL) {
8        // *x tem filho da direita, logo não é o maior
9        antecessor(a, &((*x)->dir));
10    } else {
11       // *x não tem filho da direita, logo é o subsituto
12       // Move o dado subsituto para o retirado.
13       a->dado = (*x)->dado;
14       // Promove o filho da esquerda do susbituto ao lugar dele
15       aux = *x;
16       *x = aux->esq;
17       // Libera o espaço usado pelo substituto (já movido para a)
18       free(aux);
19    }
20 }
```

Observe também a necessidade de informar x por referência na passagem de parâmetros, pois o ponteiro que faz parte de um dos elementos da árvore deverá ser alterado.