1 - Conceito de Árvore Binária

Árvore binária (*binary tree* em inglês) é uma estrutura de dados dinâmica para manter elementos no formato de uma árvore, onde cada nó pode ter 0, 1 ou 2 filhos, assim como mostra a Figura 1.

Os elementos da árvore são chamados de nós e cada nó é formado por conteúdo e uma referência para dois outros nós, que serão os filhos esquerdo e direito. A Figura 2 mostra uma classe usada para criar objetos do tipo No da árvore.

Termos importantes sobre árvore binária:

- Nós: são todos os elementos mantidos na árvore;
- Raiz: o nó raiz é o topo da árvore, na Figura 1 é o nó 8;
- Folha (*leaf* em inglês) ou nó terminal: é um nó que não tem filhos, na Figura 1 são os nós 1, 4, 7 e 13;
- Nó interno ou nó não terminal: é um nó que não é folha;
- Pai e filho: um nó ***n*** abaixo de um nó ***r*** é chamado de filho de ***r***. De modo oposto, ***r*** é chamado de pai de ***n***. No exemplo da Figura 1, o nó 3 é pai do nó 1 e os nós 1 e 6 são filhos do nó 3;
- Profundidade: é a distância de um nó até a raiz;
- Nível e um nó: é a profundidade do nó. A raiz está no nível 0;
- Altura do nó: é a distância do nó até o seu descendente mais afastado. Então a altura da árvore é a distância do nó raiz até o seu descendente mais distante;
- Grau de um nó: é o número de filhos do nó. Numa árvore binária um nó pode ter grau 0, 1 ou 2;
- Grau de uma árvore: é a altura da árvore, ou seja, o número de níveis. A árvore da Figura 1 possui grau 3;
- Árvore binária cheia: uma árvore de grau *p* é uma árvore cheia se possui o número máximo de nós, isto é, todos os nós têm o número máximo de filhos exceto as folhas, e todas as folhas estão na mesma altura. A Figura 3 mostra uma árvore cheia. A quantidade máxima de nós em cada nível é dada por $2^{nível}$ e a quantidade máxima de nós de uma árvore é dada por $2^{nível+1} - 1$;
- Propriedade de uma árvore: só existe um caminho da raiz para qualquer nó;

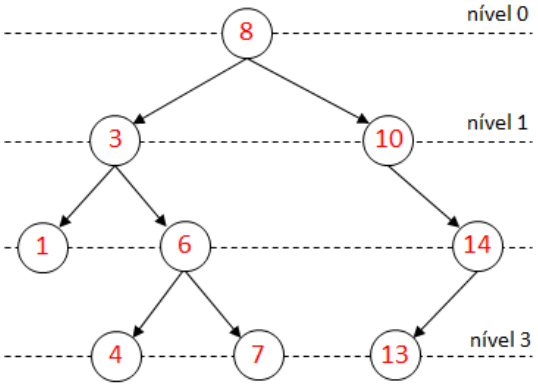

Figura 1 – Representação de uma árvore binária.

```
public class No {
    int conteudo;
    No esquerdo, direito;
}
```

Figura 2 – Classe que representa um nó da árvore.

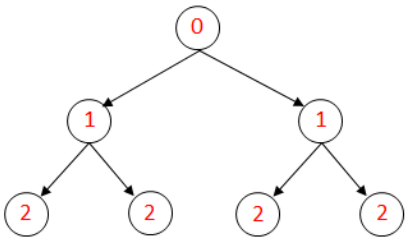

Figura 3 – Representação de uma árvore cheia de nível 2.

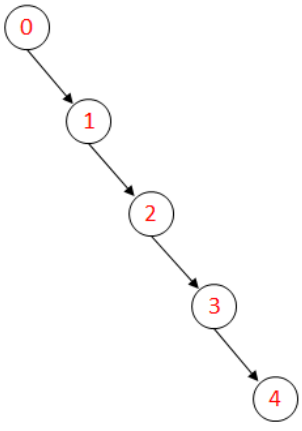

Figura 4 – Representação de uma árvore zigue-zague de nível 4.

- Árvore binária zigue-zague: qualquer nó interior possui uma subárvore esquerda ou direita vazia, a Figura 4 mostra um exemplo;
- Árvore perfeitamente balanceada: para todo nó da árvore, os números de nós das suas duas subárvores diferem no máximo em um;
- Árvore binária de busca: uma árvore binária é de busca se e somente se para cada nó ***n*** tem-se a seguinte propriedade:

  O conteúdo de ***n*** é
  - maior ou igual ao conteúdo de qualquer nó na sua subárvore esquerda e
  - menor ou igual ao conteúdo de qualquer nó na sua subárvore direita.

  Em outros termos, se ***n*** é um nó de uma árvore binária de busca, então

  n.esquerdo.conteudo ≤ ***n***.conteudo ≤ ***n***.direito.conteudo

2 - Implementação de uma Árvore Binária de Busca

A Figura 5 mostra uma implementação de Árvore Binária de Busca (ABB) usando a ideia de lista encadeada. O código de cada nó é apresentado na Figura 2. Percorrer a árvore é uma operação tipicamente recursiva, por este motivo todos os métodos programados nesta classe são recursivos.

O código da Figura 6 é usado para testar a classe Arvore da Figura 5, os valores inseridos criam a árvore representada na Figura 1.

A Figura 7 mostra os valores armazenadas na árvore testada na Figura 6. A diferença na ordem dos números acontece pelo fato de existirem diferentes formas de varrer recursivamente a árvore:

- Na varredura infixa, ou na ordem, ou simétrica os nós são visitados da seguinte forma:
  - Subárvore esquerda do nó;
  - O próprio nó;
  - Subárvore direita do nó.
- Na varredura prefixa, ou pré ordem os nós são visitados da seguinte forma:
  - O próprio nó;
  - Subárvore esquerda do nó;
  - Subárvore direita do nó.
- Na varredura posfixa, ou pós ordem os nós são visitados da seguinte forma:
  - Subárvore esquerda do nó;
  - Subárvore direita do nó;
  - O próprio nó.

A inserção de novos nós na ABB deve manter os valores ordenados. O método inserir(no:No, nro:int), da classe Arvore (Figura 5), recebe um nó e insere o valor em uma subárvore desse nó – lembre-se que cada nó possui as subárvores esquerda e direita - a 1ª chamada desse método recebe o nó raiz assim como acontece na classe Principal (Figura 6) em `arvore.inserir(arvore.raiz, 4)`, mas as demais chamadas recebem filhos do nó raiz, uma vez que o método `inserir` trabalha de forma recursiva.

```
public class Arvore {
   public No raiz;

   /* retorna a altura de uma árvore binária */
   public int altura(No no){
      if( no == null ){
         return -1; /* árvore vazia */
      }
      else{
         int altEsq = altura(no.esquerdo);
         int altDir = altura(no.direito);
         return altEsq < altDir ? altDir + 1 : altEsq + 1;
      }
   }

   /* libera a árvore que se encontra abaixo do no */
   public No limpar(No no){
      if( no != null ){
         no.esquerdo = limpar(no.esquerdo); /* limpa o lado esquerdo */
         no.direito = limpar(no.direito); /* limpa o lado direito */
      }
      return null;
   }

   public boolean buscar(No no, int nro){
      if( no == null ){
         return false; /* não encontrou */
      }
      else{
         return   nro == no.conteudo ||
               buscar(no.esquerdo, nro) ||
               buscar(no.direito, nro);
      }
   }

   /* Insere o nro mantendo os valores ordenados.
    * A inserção será em uma subárvore do no */
   public No inserir(No no, int nro){
      if( no == null ){ /* árvore vazia */
         no = new No();
         no.conteudo = nro;
      }
      else if( nro < no.conteudo ){ /* percorre a subárvore esquerda */
         no.esquerdo = inserir(no.esquerdo, nro);
      }
      else{ /* percorre a subárvore direita */
         no.direito = inserir(no.direito, nro);
      }
      return no;
   }

   /* Remove o nó que possui o nro mantendo os valores ordenados.
    * A busca será em uma subárvore do no */
   public No remover(No no, int nro){
      if( no == null ){
         return null; /* não existe o nó */
      }
      else if( nro < no.conteudo ){ /* o nro está em um filho à esquerda */
         no.esquerdo = remover(no.esquerdo, nro);
      }
      else if( nro > no.conteudo ){  /* o nro está em um filho à direita */
         no.direito = remover(no.direito, nro);
      }
      else{ /* o nro procurado está no nó */
         /* nó sem filhos */
         if( no.esquerdo == null && no.direito == null ){
            return null;
         }
         else if( no.esquerdo == null ){ /* nó só tem filho à direita */
            no = no.direito;
         }
         else if( no.direito == null ){ /* nó só tem filho à esquerda */
            no = no.esquerdo;
         }
         else{ /* nó tem os dois filhos */
```

```
            No aux = no.esquerdo;
            while( aux.direito != null ){
               aux = aux.direito;
            }
            no.conteudo = aux.conteudo; /* troca os conteúdos */
            aux.conteudo = nro;
            no.esquerdo = remover(no.esquerdo, nro);
         }
      }
      return no;
   }

   /* varredura infixa ou em ordem ou simétrica */
   public void imprimirEsqRaizDir(No no){
      if( no != null ){
         imprimirEsqRaizDir(no.esquerdo);
         System.out.print(no.conteudo + " ");
         imprimirEsqRaizDir(no.direito);
      }
   }

   /* varredura prefixa ou pré-ordem */
   public void imprimirRaizEsqDir(No no){
      if( no != null ){
         System.out.print(no.conteudo + " ");
         imprimirRaizEsqDir(no.esquerdo);
         imprimirRaizEsqDir(no.direito);
      }
   }


   /* varredura posfixa ou pós-ordem */
   public void imprimirEsqDirRaiz(No no){
      if( no != null ){
         imprimirEsqDirRaiz(no.esquerdo);
         imprimirEsqDirRaiz(no.direito);
         System.out.print(no.conteudo + " ");
      }
   }
}
```

Figura 5 – Código da classe Arvore.

```
public class Principal {

   public static void main(String[] args) {
      Arvore arvore = new Arvore();
      /* insere os valores na árvore a partir do nó raiz */
      arvore.raiz = arvore.inserir(arvore.raiz, 8);
      arvore.raiz = arvore.inserir(arvore.raiz, 10);
      arvore.raiz = arvore.inserir(arvore.raiz, 3);
      arvore.raiz = arvore.inserir(arvore.raiz, 6);
      arvore.raiz = arvore.inserir(arvore.raiz, 1);
      arvore.raiz = arvore.inserir(arvore.raiz, 14);
      arvore.raiz = arvore.inserir(arvore.raiz, 4);
      arvore.raiz = arvore.inserir(arvore.raiz, 13);
      arvore.raiz = arvore.inserir(arvore.raiz, 7);

      System.out.println("\nOrdem EsqRaizDir: ");
      arvore.imprimirEsqRaizDir(arvore.raiz);
      System.out.println("\nOrdem RaizEsqDir: ");
      arvore.imprimirRaizEsqDir(arvore.raiz);
      System.out.println("\nOrdem EsqDirRaiz: ");
      arvore.imprimirEsqDirRaiz(arvore.raiz);
   }
}
```

Figura 6 – Código da classe Principal para testar a classe Arvore da Figura 5.

```
Ordem EsqRaizDir:
1 3 4 6 7 8 10 13 14
Ordem RaizEsqDir:
8 3 1 6 4 7 10 14 13
Ordem EsqDirRaiz:
1 4 7 6 3 13 14 10 8
```

Figura 7 – Resultado do código da Figura 6.

Para remover um nó da ABB é necessário manter ela ordenada. O método remover(no:No, nro:int), da classe Arvore da Figura 5, possui o código para fazer essa operação. Veja que um nó a ser removido pode estar em uma das seguintes situações:

- O nó é folha ou seja ele não possui filhos: neste caso basta retirar a referência do nó pai para o nó a ser removido. No exemplo da Figura 8, foi retirada a referência `no.esquerdo` do nó 14 para remover o nó 13 da árvore;
- O nó possui um único filho: no exemplo da Figura 9 para remover o nó 10 é necessário fazer o nó assumir o valor do filho, que neste exemplo é `no = no.direito;`
- O nó possui dois filhos: o código a seguir mostra as operações:

```
No aux = no.esquerdo;
while( aux.direito != null ){
   aux = aux.direito;
}
no.conteudo = aux.conteudo; /* troca os conteúdos */
aux.conteudo = nro;
no.esquerdo = remover(no.esquerdo, nro);
```

A Figura 10 mostra a árvore após remover o nó 3, veja que o conteúdo do nó 3 foi trocado pelo de 1;

A Figura 11 mostra a árvore após remover o nó 8, veja que o conteúdo desse nó foi trocado pelo maior nó mais à direita (nó 7) do seu nó a esquerda (nó 3).

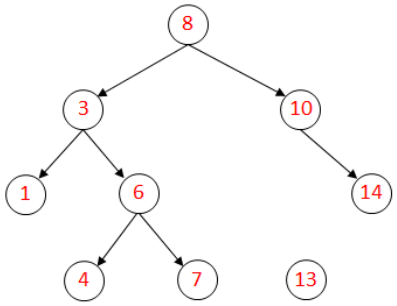

Figura 8 – Representação da ABB após remover o nó 13.

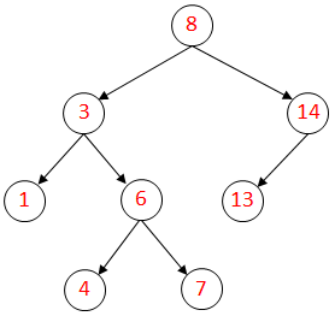

Figura 9 – Representação da ABB após remover o nó 10.

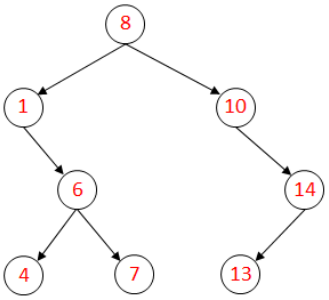

Figura 10 – Representação da ABB após remover o nó 3.

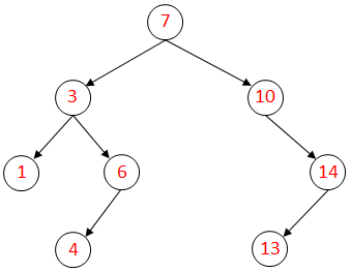

Figura 11 – Representação da ABB após remover o nó 8.

3 - Exercício

1 – Programar um método na classe Arvore da Figura 5 para retornar a quantidade de nós da árvore.