Lista de Exercícios: Árvores Binárias

Data da Entrega: 18/05/2017
Exercício individual
Entrega da solução dos exercícios pelo SIGAA

1. Qual a maior e menor quantidade de nós que podem existir em uma árvore binária completa de altura h?

   O maior número de nós em uma árvore binária completa é $2^{h+1} - 1$.

   O menor número de nós em uma árvore binária completa é 2^h.

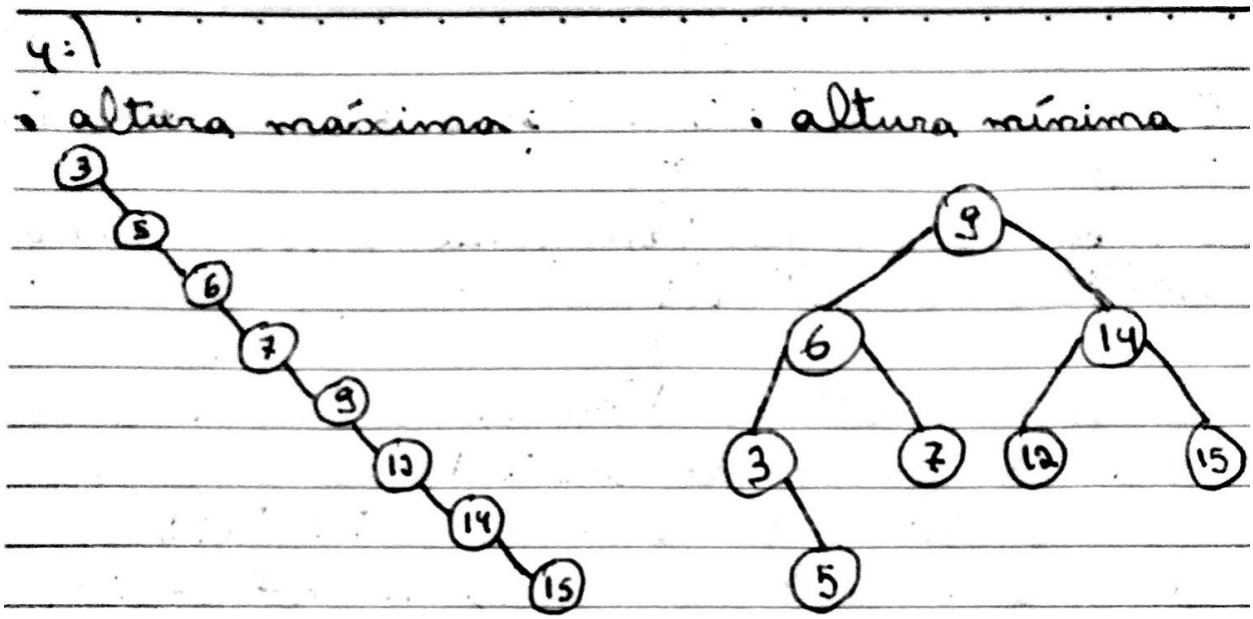
2. Uma árvore binária cuja altura é igual ao número de nós pode possuir nós cheios? Justifique.

   Não, pois a altura de uma árvore binária é o número de passos do mais longo caminho da raiz até a folha mais distante. Isso implica que se houver algum nó cheio, haverá algum nó que não fará parte desse caminho e consequentemente o números de nós da árvore será maior que a altura.

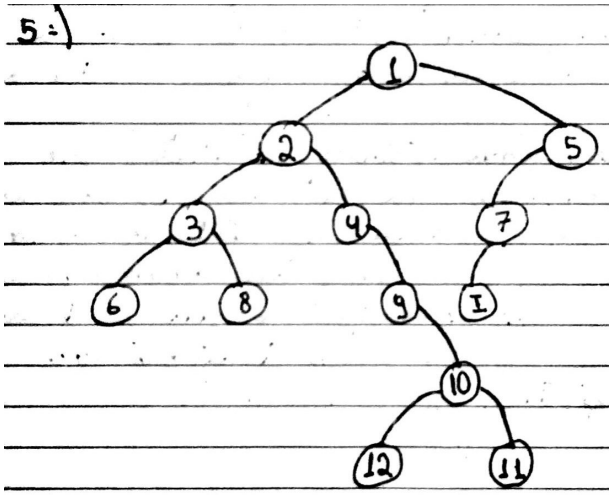
3. Toda árvore binária cheia é completa? Justifique.

   Sim, pois em uma árvore completa se n é um nó com alguma subárvore vazia então ele deve se localizar no último ou no penúltimo nível. Contudo, em uma árvore cheia, todo nó que tem subárvore vazia se localiza no último nível. Logo, toda árvore cheia é completa.

4. Represente a sequência abaixo na forma de árvores binárias de alturas mínima e máxima.
   s = { 3, 5, 9, 12, 14, 6, 7, 15 }

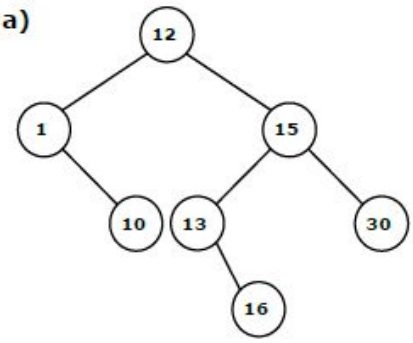


5. Dados os percursos abaixo, reconstruir a árvore
   original: pré-ordem: 1, 2, 3, 6, 8, 4, 9, 10, 12, 11, 5, 7, I
   simétrica (in-ordem) : 6, 3, 8, 2, 4, 9, 12, 10, 11, 1, I, 7, 5

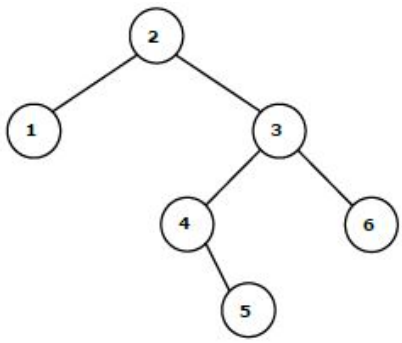


6. Verificar se as árvores abaixo são binárias de busca.

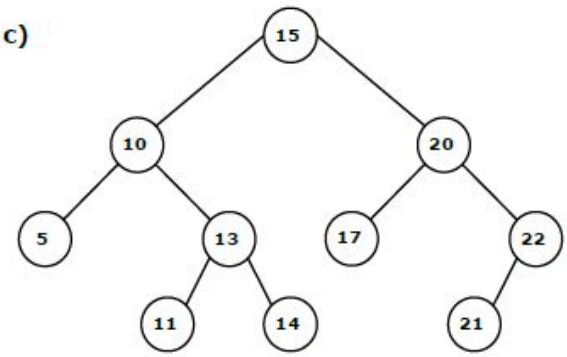
a)



b)



c)



a) **Não é uma ABB, pois, sendo 16 > 15, o nó de número 16 não poderia fazer parte da subárvore à esquerda do nó de número 15.**
b) **Não, pois sendo 4 > 3, o nó de número 4 não poderia está à esquerda do nó de número 3.**
c) **Sim, é uma ABB.**

8. Construir algoritmo para dada uma árvore n-ária, transformá-la em uma árvore binária.

Transforme o primeiro filho da raiz em seu filho a esquerda e em seguida transforme o próximo filho no filho a direita do filho a esquerda da raiz, e o próximo no filho a direita do último e assim sucessivamente até acabar os filhos da raiz. Faça o mesmo procedimento para os restantes dos nós na árvore.

9. Escrever programa em Scala para percorrer uma árvore binária qualquer, em nível, das folhas para a raiz.

```
def pecorrer(T: Array[Int], h: Int): Unit = {
    for(i <- h to 0){
       for(j <- 2^i to 2^(i+1) − 1 ){
              if(T(j).value != null){
                     println(T(j).value)
              }
       }
    }
}
```

10. Escrever programa Scala para buscar a informação em um nó de uma árvore binária de busca, sendo dada a sua chave. Faça atenção à eficiência do programa.

```
def busca(k: Int): Node = {
    var n: Node = root
    while(n.key != k){
       if(k < n.key){
              n = n.left
       }
       else{
              n = n.right
       }
    }
    n
}
```

11. Escrever em Scala um programa para achar o maior elemento (campo numérico) de uma árvore binária dada.

```
def maiorElemento(T: ArvoreBST): Int = {
    var n: Node = T.root
    while(n.right != null){
       n = n.right
    }
    n.key
}
```

12. Escreva, em Scala, uma função recursiva que permuta as subárvores esquerda e direita de todos os nós de uma árvore binária. A função deve receber como parâmetro o endereço do nó raiz da subárvore a ser processada. Explicite também a chamada externa.

```
def permuta(root: Node): Unit = {
    var node: Node = root
    if(node != null){
       var aux: Node = node.left
       node.left = node.right
       node.right = aux
       permuta(node.left)
       permuta(node.right)
    }
}
```

13. Considere uma árvore binária de busca representada com a seguinte estrutura de nó em C:

```
typedef struct bst_node BstNode;
struct _bst_node {
   void*    info;
   AvlNode* parent;
   AvlNode* left;
   AvlNode* right;
};
```

Escreva em Scala uma função que determina o antecessor de um dado nó, visitando o menor número de nós possível. Esta função um nó como parâmetro de entrada e retorna como saída um nó que é justamente o seu antecessor.

```
def antecessor(node: Node): Node = {
       var n: Node = node
       if(n.left != null){
                n = n.left
                while(n.right != null){
                        n = n.right
                }
                return n
       }
       else{
                var n2: Node = n.parent
                while(n2 != null && n == n2.left){
                        n = n2
                        n2 = n2.parent
                }
                return n2
       }
}
```