Universidade Federal do Ceará – UFC
Centro de Ciências – CC
Mestrado e Doutorado em Ciências da Computação - MDCC

Estruturas de Dados Avançadas

Exercício: Árvores binárias

NOME: Pedro Roger Magalhães Vasconcelos

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

O total de nós de uma árvore binária é a soma do número de nós a cada nível. Sendo que o nível 0 contém um nó (raiz), o nível 1 contém no máximo 2 nós e no nível h pode haver no máximo 2^L nós.

A árvore binária cheia de altura h tem exatamente 2^L nós em cada nível 0 <= h <= d.

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

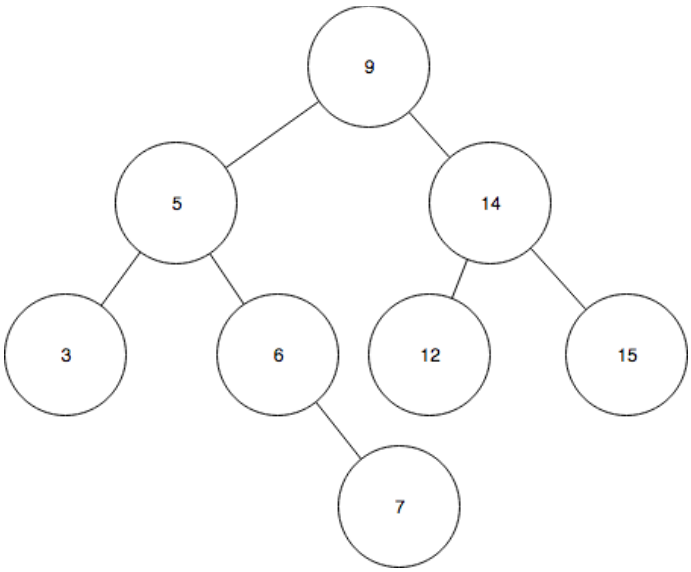
Para uma árvore ter altura igual a quantidade de nós, esta não poderá ter mais que um filho, sendo assim não tem como possuir um nó cheio sem deixar de passar pelo em um nó a mais que o tamanho da altura.

3) Toda árvore binária cheia é completa? Justifique.

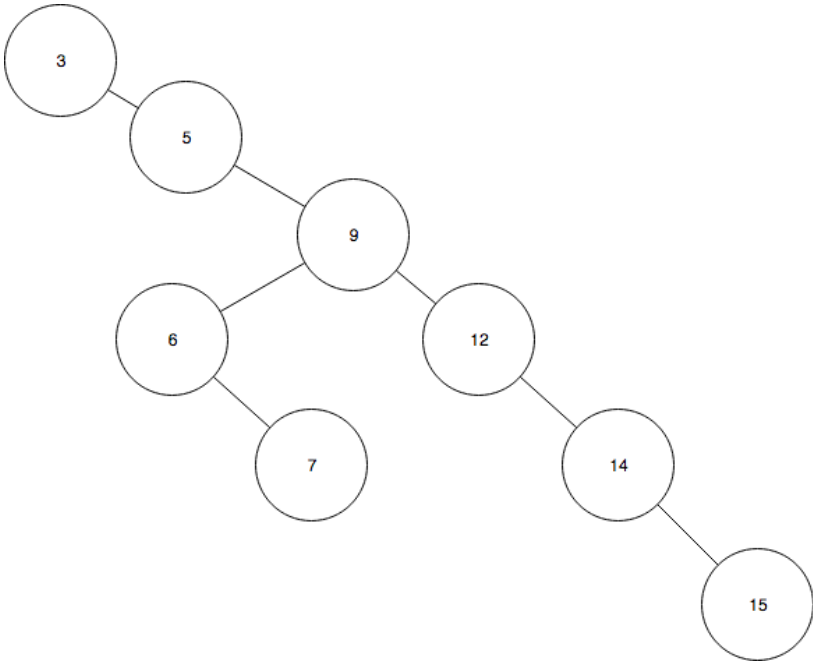
Sim. Uma árvore binária cheia é uma árvore em que se um nó tem alguma sub-árvore vazia então ele está no último nível.

Uma árvore completa é aquela em que se n é um nó com alguma sub-árvore vazia, então n se localiza no penúltimo ou no último nível. Portanto, toda árvore cheia é completa e necessariamente binária.

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



Altura mínima

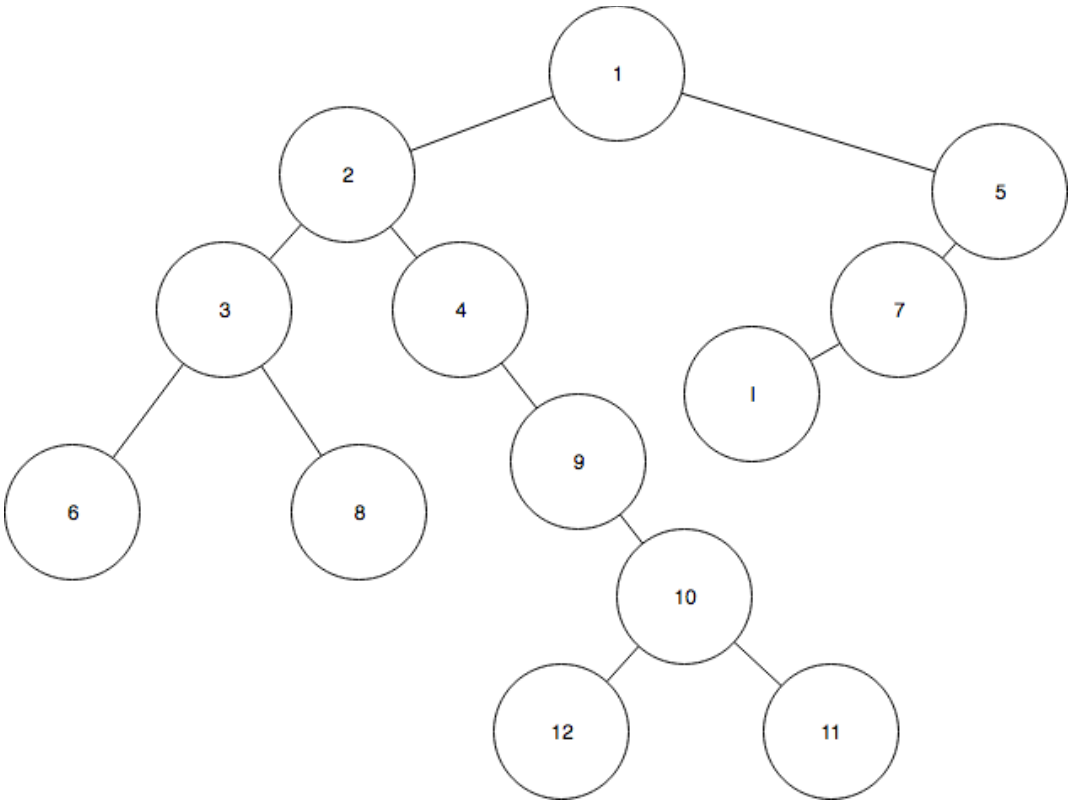


Altura máxima

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) Não, pois não é possível localizar o elemento 16

b) Não é pois não é possível localizar o 4 e o 5.

c) Sim, pois é possível localizar todos os nós.

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

Processo de Conversão
Seja T uma árvore qualquer.
T é convertida em uma árvore binária B(T) da tal qual:
B(T) possui um nó B(v), para cada nó v de T.
As raízes de T e B(T) coincidem.
O filho esquerdo de um nó B(v) em B(T) corresponde ao primeiro filho de v em T, caso exista.
Se não existir, a subárvore esquerda de B(v) é vazia.
O filho direito de um nó B(v) em B(T) corresponde ao irmão de v em T, localizado imediatamente à sua direita, caso exista.
Se não existir, a subárvore direita de B(v) é vazia.

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

```
def printTreeNodesToRoot(node: Node): Unit = {
      if(node.left != null){
            printTreeNodesToRoot(node.left)
      }
      if(node.right != null){
            printTreeNodesToRoot(node.right)
      }

      println("Value: "+ node.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 searchBTree(tree: Node, key: Int): Int = {
      if(tree == null){
            return null
      } else {
            if(key > tree.value){
                  if(tree.right == null)
                        return null
                  return searchBTree(tree.right, key)
            } else {
                  if(tree.left == null)
                        return null
                  return searchBTree(tree.left, key)
            }
      }
}
```

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

```
def findMaxValue(tree: Node): Int = {
      var max: Int = tree.value
      if(tree.left != null){
```

```
            max = findMaxValue(tree.left)
      }
      if(tree.right != null){
            max = findMaxValue(tree.right)
      }

      return max
}
```

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 swap(tree: Node): Unit = {
      if(tree.right != null){
            swap(tree.right)
      }
      if(tree.left != null){
            swap(tree.left)
      }

      change(tree.left, tree.right)
}

def change(left: Node, right: Node): Unit = {
      var rightNode = right
      var leftNode = left

      if(leftNode != null && rightNode != null){
            var tmp = leftNode.value
            leftNode.value = rightNode.value
            rightNode.value = tmp
      }
}
```

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 max(tree: AVLNode): AVLNode = {
	var r = tree

	if(r == null)
		return null
	while(r.right != null)
		r = r.right

	return r

}

def prevNode(n: AVLNode): AVLNode = {
	var node = n

	if(node == null)
		return null
	else
		if(node.left != null){
			node = node.right
			return node
		} else {
			var p = node.parent
			while(p.parent != null && node==p.left){
				node = p
				p = p.parent
			}

			return p
		}
}
```