/\*\*

Retorna o número de nodos da árvore.

\*\*/

public int countNodes() {

return countNodes(root);

}

/\*\*

Versao protegida do metodo. Utilizada pela versao

publica que sempre inicia a busca pelo nodo raiz

\*\*/

public int countNodes(BSTNode n) {

if ( n == null ) {

return 0;

} else {

return 1 + countNodes( n.left ) + countNodes( n.right );

}

}

/\*\*

Retorna o número de nodos folha da árvore

\*\*/

public int countLeafNodes() {

return countLeafNodes(root);

}

/\*\*

Versao protegida do metodo. Utilizada pela versao

publica que sempre inicia a busca pelo nodo raiz

\*\*/

public int countLeafNodes(BSTNode n) {

if ( n == null ) {

return 0;

} else {

boolean isLeaf = (n.left == null && n.right == null);

return (isLeaf ? 1 : 0) + countLeafNodes( n.left ) + countLeafNodes( n.right );

}

}

/\*\*

Retorna a altura da árvore

\*\*/

public int height() {

return height(root);

}

/\*\*

Versao protegida do metodo. Utilizada pela versao

publica que sempre inicia a busca pelo nodo raiz

\*\*/

public int height(BSTNode n) {

if ( n == null ) return 0;

int leftHeight = height(n.left);

int rightHeight = height(n.right);

if ( leftHeight > rightHeight ) { return leftHeight + 1; }

else { return rightHeight + 1; }

}

/\*\*

Retornar true caso a árvore esteja cheia ou false caso contrário

\*\*/

public boolean isFull() {

int result = 2;

if ( isEmpty() ) { return false; }

if ( height() == 0 ) {

result = 1;

} else {

for ( int i = 1; i < height() ; i++) {

result \*= 2;

}

}

return countNodes() == (result - 1);

}