BST Breadth First Search

Breadth First search in Binary Search Tree means level wise traversal.

For this traversal we will use Queue to store the current root and append its childs at the end of Queue.

For the above tree traversal is as follows:

Insert 10.

In Queue |10|

**Poll it and Print it 10.**

Does 10 has left child? Yes insert 5.

In Queue |5|

Does 10 has right child? Yes insert 15.

In Queue |5|15|

**Poll 5 and Print 5.**

In Queue |15|

Does 5 has left child? Yes insert 2.

In Queue |15|2|

Does 5 has right child? Yes insert 7.

In Queue |15|2|7|

**Poll 15 and Print 15.**

In Queue |2|7|

Does 15 has left child? Yes insert 12.

In Queue |2|7|12|

Does 15 has right child? Yes insert 17.

In Queue |2|7|12|17|

**Poll 2 and Print 2.**

In Queue |7|12|17|

Does 2 has left child? No.

Does 2 has right child? No.

**Poll 7 Print 7.**

In Queue |12|17|

Does 7 has left child? No.

Does 7 has right child? No.

**Poll 12 Print 12.**

In Queue |17|

Does 12 has left child? No.

Does 12 has right child? No.

**Poll 17 Print 17.**

In Queue ||

Does 17 has left child? No.

Does 17 has right child? No.

Below is the program to print nodes level by level.

**public** **void** bfs(Node root) {

// Declare a Queue. Here we traverse in Breadth First Search

Queue<Node> q = **new** LinkedList<Node>();

**if** (root == **null**)

**return**;

// Add root so we can further process the children.

q.add(root);

// Run loop till queue is not empty

**while** (!q.isEmpty()) {

// remove the current root

root = q.poll();

System.***out***.println(root.iData);

// left child exists insert in queue

**if** (root.leftChild != **null**) {

q.add(root.leftChild);

}

// left child exists insert in queue

**if** (root.rightChild != **null**) {

q.add(root.rightChild);

}

}

}

Output:

10

5

15

2

7

12

17