Binary search tree

Used Node and Class/Interface in single link list

**public** **class** BTNode<T> {

**public** T t;

**public** BTNode<T> lt, rt, pt, next, prev; // next point to successor/prdecessor of node

**public** **int** h, ind;

**public** BTNode(T t) {

**this**.t = t;

**this**.lt = **this**.rt = **this**.pt = **this**.next = **this**.prev = **null**;

**this**.h = 0;

**this**.ind = 0;

}

}

-------------------------------------------------------------------------------------

Used interface

----------------------------------------------------

**public** **interface** IBasic<T> {

/\* 1. Binary Search Tree | Set 1 (Search and Insertion) \*/

/\* create bst \*/

**public** BTNode<T> insert(BTNode<T> node, T t);

/\* search node in bst \*/

**public** BTNode<T> search(BTNode<T> node, T t);

/\* 2. Binary Search Tree | Set 2 (Delete) \*/

**public** BTNode<T> delete(BTNode<T> node, T t);

/\*print inorder\*/

**public** **void** inOrder(BTNode<T> node);

}=====================================================================================Implementation of interface

-------------------------------------------------------------------------------------

**public** **class** BSTreeImpl **implements** IBSTree {

@Override

**public** BTNode create(BTNode node, **int** dt) {

**if** (node == **null**)

node = **new** BTNode(dt);

**else** {

**if** (dt <= node.dt) {

node.lt = create(node.lt, dt);

node.lt.pt = node;

} **else** {

node.rt = create(node.rt, dt);

node.rt.pt = node;

}

}

**return** node;

}

//this is also level order insertion

**public** BTNode insert(BTNode node, **int**[] arr, **int** start, **int** size) {

**int** left = 2 \* start + 1;

**int** right = 2 \* start + 2;

**if** (left > size || right > size) {

**return** **null**;

}

**if** (node == **null**) {

node = **new** BTNode(arr[start]);

}

**if** (node.lt == **null** && node.rt == **null**) {

**if** (left < size) {

node.lt = **new** BTNode(arr[left]);

}

**if** (right < size) {

node.rt = **new** BTNode(arr[right]);

}

}

insert(node.lt, arr, left, size);

insert(node.rt, arr, right, size);

**return** node;

}

// row wise ie 1 row, 2nd row etc

@Override

**public** **void** levelPrint(BTNode node) {

Queue<BTNode> que = **new** LinkedList<>();

que.add(node);

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

BTNode temp = que.poll();

System.***out***.print("(" + temp.dt + ")");

**if** (temp.lt != **null**)

que.add(temp.lt);

**if** (temp.rt != **null**)

que.add(temp.rt);

}

}

**public** **static** **void** main(String[] args) {

BSTreeImpl imp = **new** BSTreeImpl();

**int** arr[] = { 5, 4, 6, 3, 7, 2, 8, 1, 9 };

BTNode root = **null**;

**for** (**int** i = 0; i < arr.length; i++)

root = imp.create(root, arr[i]);

imp.levelPrint(root);

}

// root,left to right level,then right to left and so on

@Override

**public** **void** spirialPrintLeft(BTNode node) {

// **TODO** Auto-generated method stub

}

// root,right to left level,then left to right

@Override

**public** **void** spiralPrintRight(BTNode node) {

// **TODO** Auto-generated method stub

}

}

=====================================================================================

Test case:

-----------------------------------------------------------------

**public** **class** IBasicTest {

**public** IBasic<Integer> ib = **null**;

@Before

**public** **void** init() {

ib = **new** BasicImpl();

}

/\* 1. Binary Search Tree | Set 1 (Search and Insertion) \*/

/\* create bst \*/

**public** **void** insertTest() {

}

/\* search node in bst \*/

**public** **void** searchTest() {

}

/\* 2. Binary Search Tree | Set 2 (Delete) \*/

**public** **void** deleteTest() {

}

}