|  |
| --- |
| 0hnQGAuNPo4wojnr77c2x8cWiaT70FGHBZ2h-3__FYQ.png |
| Binary Search Tree |
| Data Structures Made Easy |
|  |

dublin city university

# 1. *Binary Search Tree*

class binary\_Search\_Tree<T extends Comparable<T>>{

private static class Node<T>{

private T item;

private Node<T> left;

private Node<T> right;

Node(T item0, Node<T> left0, Node<T> right0){

item = item0;

left = left0;

right = right0;

}

}

private Node<T> root = null;

private int numItems = 0;

public int size(){

return numItems;

}

private boolean contains(Node<T> node, T t){

if(node == null)

return false;

else if((node.item).equals(t))

return true;

else if((node.item).compareTo(t) > 0)

return contains(node.left, t);

else

return contains(node.right, t);

}

public boolean contains(T t){

return contains(root, t);

}

private Node<T> add(Node<T> node, T t){

if(node == null){

numItems++;

return new Node<T>(t, null, null);

}

else if((node.item).compareTo(t) > 0){

node.left = add(node.left, t);

return node;

}

else if(t.compareTo(node.item) > 0){

node.right = add(node.right, t);

return node;

}

else

return node;

}

public boolean add(T t){

int num = numItems;

root = add(root, t);

return (numItems > num);

}

private Node<T> remove(Node<T> node, T t){

if(node == null)

return node;

else if((node.item).compareTo(t) > 0){

node.left = remove(node.left, t);

return node;

}

else if(t.compareTo(node.item) > 0){

node.right = remove(node.right, t);

return node;

}

else{

numItems--;

return mergeTrees(node.left, node.right);

}

}

public boolean remove(T t){

int num = numItems;

root = remove(root, t);

return (num > numItems);

}

private Node<T> mergeTrees(Node<T> a, Node<T> b){

if(b == null)

return a;

else if(b.left == null){

b.left = a;

return b;

}

else{

Node<T> p = b.left;

Node<T> p\_Parent = b;

while(p.left != null){

p\_Parent = p;

p = p.left;

}

p\_Parent.left = p.right;

p.left = a;

p.right = b;

return p;

}

}

private void preOrderTraversal(Node<T> node){

if(node != null){

System.out.print(node.item + " ");

preOrderTraversal(node.left);

preOrderTraversal(node.right);

}

}

public void preOrderTraversal(){

preOrderTraversal(root);

}

private void inOrderTraversal(Node<T> node){

if(node != null){

inOrderTraversal(node.left);

System.out.print(node.item + " ");

inOrderTraversal(node.right);

}

}

public void inOrderTraversal(){

inOrderTraversal(root);

}

private void postOrderTraversal(Node<T> node){

if(node != null){

postOrderTraversal(node.left);

postOrderTraversal(node.right);

System.out.print(node.item + " ");

}

}

public void postOrderTraversal(){

postOrderTraversal(root);

}

public static void main( String[ ] args ) {

binary\_Search\_Tree tree = new binary\_Search\_Tree();

int[] input = {94, 3, 65, 12, 44, 21, 76};

System.out.println('\n' + "INPUT");

System.out.println("=====");

for(int index : input) {

System.out.print(index + " ");

tree.add(index);

}

System.out.println();

System.out.println('\n' + "PRE-ORDER TRAVERSAL OF TREE");

System.out.println("=============");

tree.preOrderTraversal();

System.out.println();

System.out.println('\n' + "IN-ORDER TRAVERSAL OF TREE");

System.out.println("=============");

tree.inOrderTraversal();

System.out.println();

System.out.println('\n' + "POST-ORDER TRAVERSAL OF TREE");

System.out.println("=============");

tree.postOrderTraversal();

}

}