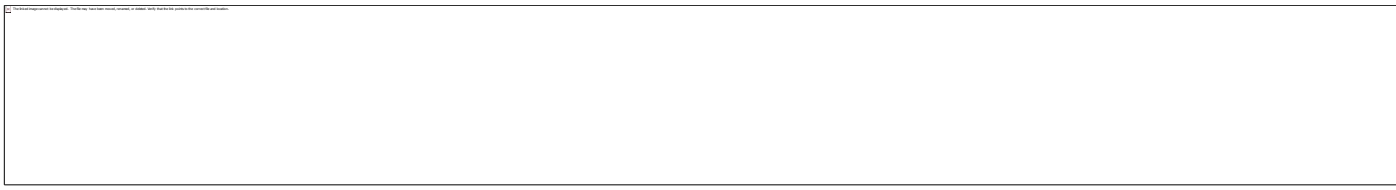

Java Program to Implement Binary Search Tree

This is a Java Program to implement Binary Search Tree. A binary search tree (BST), sometimes also called an ordered or sorted binary tree, is a node-based binary tree data structure which has the following properties:
i) The left subtree of a node contains only nodes with keys less than the node's key.
ii) The right subtree of a node contains only nodes with keys greater than the node's key.
iii) The left and right subtree must each also be a binary search tree.
iv) There must be no duplicate nodes.
Generally, the information represented by each node is a record rather than a single data element. However, for sequencing purposes, nodes are compared according to their keys rather than any part of their associated records. The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient. Binary search trees are a fundamental data structure used to construct more abstract data structures such as sets, multisets, and associative arrays.

Here is the source code of the Java program to implement Binary Search Tree. The Java program is successfully compiled and run on a Windows system. The program output is also shown below.

```java
/*
 *  Java Program to Implement Binary Search Tree
 */

import java.util.Scanner;

/* Class BSTNode */
class BSTNode
{
      BSTNode left, right;
      int data;

      /* Constructor */
      public BSTNode()
      {
          left = null;
          right = null;
          data = 0;
      }
      /* Constructor */
      public BSTNode(int n)
      {
          left = null;
          right = null;
          data = n;
      }
      /* Function to set left node */
      public void setLeft(BSTNode n)
      {
          left = n;
      }
      /* Function to set right node */
      public void setRight(BSTNode n)
      {
          right = n;
      }
      /* Function to get left node */
      public BSTNode getLeft()
      {
          return left;
      }
      /* Function to get right node */
      public BSTNode getRight()
      {
          return right;
      }
      /* Function to set data to node */
      public void setData(int d)
      {
          data = d;
      }
      /* Function to get data from node */
      public int getData()
      {
          return data;
      }
  }
```

```
58.
59.   /* Class BST */
60.   class BST
61.   {
62.       private BSTNode root;
63.
64.       /* Constructor */
65.       public BST()
66.       {
67.           root = null;
68.       }
69.       /* Function to check if tree is empty */
70.       public boolean isEmpty()
71.       {
72.           return root == null;
73.       }
74.       /* Functions to insert data */
75.       public void insert(int data)
76.       {
77.           root = insert(root, data);
78.       }
79.       /* Function to insert data recursively */
80.       private BSTNode insert(BSTNode node, int data)
81.       {
82.           if (node == null)
83.               node = new BSTNode(data);
84.           else
85.           {
86.               if (data <= node.getData())
87.                   node.left = insert(node.left, data);
88.               else
89.                   node.right = insert(node.right, data);
90.           }
91.           return node;
92.       }
93.       /* Functions to delete data */
94.       public void delete(int k)
95.       {
96.           if (isEmpty())
97.               System.out.println("Tree Empty");
98.           else if (search(k) == false)
99.               System.out.println("Sorry "+ k +" is not present");
100.          else
101.          {
102.              root = delete(root, k);
103.              System.out.println(k+ " deleted from the tree");
104.          }
105.      }
106.      private BSTNode delete(BSTNode root, int k)
107.      {
108.          BSTNode p, p2, n;
109.          if (root.getData() == k)
110.          {
111.              BSTNode lt, rt;
112.              lt = root.getLeft();
113.              rt = root.getRight();
114.              if (lt == null && rt == null)
```

```
                return null;
            else if (lt == null)
            {
                p = rt;
                return p;
            }
            else if (rt == null)
            {
                p = lt;
                return p;
            }
            else
            {
                p2 = rt;
                p = rt;
                while (p.getLeft() != null)
                    p = p.getLeft();
                p.setLeft(lt);
                return p2;
            }
        }
        if (k < root.getData())
        {
            n = delete(root.getLeft(), k);
            root.setLeft(n);
        }
        else
        {
            n = delete(root.getRight(), k);
            root.setRight(n);
        }
        return root;
    }
    /* Functions to count number of nodes */
    public int countNodes()
    {
        return countNodes(root);
    }
    /* Function to count number of nodes recursively */
    private int countNodes(BSTNode r)
    {
        if (r == null)
            return 0;
        else
        {
            int l = 1;
            l += countNodes(r.getLeft());
            l += countNodes(r.getRight());
            return l;
        }
    }
    /* Functions to search for an element */
    public boolean search(int val)
    {
        return search(root, val);
    }
    /* Function to search for an element recursively */
```

```
172.     private boolean search(BSTNode r, int val)
173.     {
174.         boolean found = false;
175.         while ((r != null) && !found)
176.         {
177.             int rval = r.getData();
178.             if (val < rval)
179.                 r = r.getLeft();
180.             else if (val > rval)
181.                 r = r.getRight();
182.             else
183.             {
184.                 found = true;
185.                 break;
186.             }
187.             found = search(r, val);
188.         }
189.         return found;
190.     }
191.     /* Function for inorder traversal */
192.     public void inorder()
193.     {
194.         inorder(root);
195.     }
196.     private void inorder(BSTNode r)
197.     {
198.         if (r != null)
199.         {
200.             inorder(r.getLeft());
201.             System.out.print(r.getData() +" ");
202.             inorder(r.getRight());
203.         }
204.     }
205.     /* Function for preorder traversal */
206.     public void preorder()
207.     {
208.         preorder(root);
209.     }
210.     private void preorder(BSTNode r)
211.     {
212.         if (r != null)
213.         {
214.             System.out.print(r.getData() +" ");
215.             preorder(r.getLeft());
216.             preorder(r.getRight());
217.         }
218.     }
219.     /* Function for postorder traversal */
220.     public void postorder()
221.     {
222.         postorder(root);
223.     }
224.     private void postorder(BSTNode r)
225.     {
226.         if (r != null)
227.         {
228.             postorder(r.getLeft());
```

```
            postorder(r.getRight());
            System.out.print(r.getData() +" ");
        }
    }
}

/* Class BinarySearchTree */
public class BinarySearchTree
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        /* Creating object of BST */
        BST bst = new BST();
        System.out.println("Binary Search Tree Test\n");
        char ch;
        /*  Perform tree operations  */
        do
        {
            System.out.println("\nBinary Search Tree Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. delete");
            System.out.println("3. search");
            System.out.println("4. count nodes");
            System.out.println("5. check empty");

            int choice = scan.nextInt();
            switch (choice)
            {
            case 1 :
                System.out.println("Enter integer element to insert");
                bst.insert( scan.nextInt() );
                break;
            case 2 :
                System.out.println("Enter integer element to delete");
                bst.delete( scan.nextInt() );
                break;
            case 3 :
                System.out.println("Enter integer element to search");
                System.out.println("Search result : "+ bst.search( scan.nextInt() ));
                break;
            case 4 :
                System.out.println("Nodes = "+ bst.countNodes());
                break;
            case 5 :
                System.out.println("Empty status = "+ bst.isEmpty());
                break;
            default :
                System.out.println("Wrong Entry \n ");
                break;
```

```
            }
            /*  Display tree  */
            System.out.print("\nPost order : ");
            bst.postorder();
            System.out.print("\nPre order : ");
            bst.preorder();
            System.out.print("\nIn order : ");
            bst.inorder();

            System.out.println("\nDo you want to continue (Type
   y or n) \n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');
    }
}
```

```
Binary Search Tree Test


Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
5
Empty status = true

Post order :
Pre order :
In order :
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
8

Post order : 8
Pre order : 8
In order : 8
Do you want to continue (Type y or n)

y
```

```
Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
5

Post order : 5 8
Pre order : 8 5
In order : 5 8
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
3

Post order : 3 5 8
Pre order : 8 5 3
In order : 3 5 8
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
7

Post order : 3 7 5 8
Pre order : 8 5 3 7
In order : 3 5 7 8
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
```

```
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
10

Post order : 3 7 5 10 8
Pre order : 8 5 3 7 10
In order : 3 5 7 8 10
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
15

Post order : 3 7 5 15 10 8
Pre order : 8 5 3 7 10 15
In order : 3 5 7 8 10 15
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
1
Enter integer element to insert
2

Post order : 2 3 7 5 15 10 8
Pre order : 8 5 3 2 7 10 15
In order : 2 3 5 7 8 10 15
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
```

```
5. check empty
4
Nodes = 7

Post order : 2 3 7 5 15 10 8
Pre order : 8 5 3 2 7 10 15
In order : 2 3 5 7 8 10 15
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
3
Enter integer element to search
24
Search result : false

Post order : 2 3 7 5 15 10 8
Pre order : 8 5 3 2 7 10 15
In order : 2 3 5 7 8 10 15
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
3
Enter integer element to search
7
Search result : true

Post order : 2 3 7 5 15 10 8
Pre order : 8 5 3 2 7 10 15
In order : 2 3 5 7 8 10 15
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
2
```

```
Enter integer element to delete
2
2 deleted from the tree

Post order : 3 7 5 15 10 8
Pre order : 8 5 3 7 10 15
In order : 3 5 7 8 10 15
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
8
8 deleted from the tree

Post order : 3 7 5 15 10
Pre order : 10 5 3 7 15
In order : 3 5 7 10 15
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
10
10 deleted from the tree

Post order : 3 7 5 15
Pre order : 15 5 3 7
In order : 3 5 7 15
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
2
```

```
Enter integer element to delete
5
5 deleted from the tree

Post order : 3 7 15
Pre order : 15 7 3
In order : 3 7 15
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
15
15 deleted from the tree

Post order : 3 7
Pre order : 7 3
In order : 3 7
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
3
3 deleted from the tree

Post order : 7
Pre order : 7
In order : 7
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
2
```

```
Enter integer element to delete
77
Sorry 77 is not present

Post order : 7
Pre order : 7
In order : 7
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
2
Enter integer element to delete
7
7 deleted from the tree

Post order :
Pre order :
In order :
Do you want to continue (Type y or n)

y

Binary Search Tree Operations

1. insert
2. delete
3. search
4. count nodes
5. check empty
5
Empty status = true

Post order :
Pre order :
In order :
Do you want to continue (Type y or n)

n
```

Sanfoundry Global Education & Learning Series – 1000 Java Programs.

If you wish to look at all Java Programming examples, go to Java Programs.

If you liked this Java Program, kindly share, recommend or like below!

- Java Program to Find the Minimum value of Binary Search Tree
- Java Program to Implement Self Balancing Binary Search Tree
- Java Program to Implement AA Tree
- Java Program to Implement Cartesian Tree
- Java Program to Implement Triply Linked List

About Manish

Manish Bhojasia, a technology veteran with 17+ years @ Cisco & Wipro, is Founder and CTO at Sanfoundry. He is Linux Kernel Developer and SAN Architect and is passionate about competency developments in these areas. He lives in Bangalore and delivers focused training sessions to IT professionals in Linux Kernel, Linux Debugging, Linux Device Drivers, Linux Networking, Linux Storage & Cluster Administration, Advanced C Programming, SAN Storage Technologies, SCSI Internals and Storage Protocols such as iSCSI & Fiber Channel. Stay connected with him below.

‹ Java Program to Implement a Binary Search Tree using Linked Lists
Java Program to Implement Cartesian Tree ›

Manish Bhojasia, a technology veteran with 17+ years @ Cisco & Wipro, is Founder and CTO at Sanfoundry. He is Linux Kernel Developer and SAN Architect and is passionate about competency developments in these areas. He lives in Bangalore and delivers focused training sessions to IT professionals in Linux Kernel, Linux Debugging, Linux Device Drivers, Linux Networking, Linux Storage & Cluster Administration, Advanced C Programming, SAN Storage Technologies, SCSI Internals and Storage Protocols such as iSCSI & Fiber Channel. Stay connected with him below.

Subscribe Newsletter & Posts

Name *
Email Address *
Subscribe

C MCQs
C++ MCQs
C# MCQs
Java MCQs
Linux MCQs
SAN MCQs
Javascript
PHP MCQs
Python MCQs
OS MCQs
COA MCQs
DBMS MCQs
SE MCQs
AI MCQs
LISP MCQs

C Programs
C++ Algorithms
Java Algorithms
Android Programs

Sanfoundry Internships

Follow Manish & Sanfoundry

Follow


Subscribe to Sanfoundry

Sanfoundry Trainings

- TOS
- License
- Technology Groups
- Interns
- Jobs

Read more:

C++ Program to Implement Set_Union in STL

Close