KJSCE/IT/SY BTech/SEMIII/DS/2022-23

# Experiment 5

## Title: Implementation of Dynamic Binary Search Tree using Doubly Linked List (DLL)

(A Constituent College of Somaiya Vidyavihar University)

KJSCE/IT/SY BTech/SEMIII/DS/2022-23

## Batch:A3 Roll No.:16010421075 Experiment No.: 5

**Aim:** Write a program for following operations on Binary Search Tree using Doubly Linked List (DLL).

1. Create empty BST,
2. Insert a new element on the BST
3. Search for an element on the BST
4. Delete an element from the BST
5. Display all elements of the BST



**Resources needed:** C/C++/JAVA editor and compiler

## Theory Binary Tree :-

A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. The topmost node in the tree is called the root. Every node (excluding a root) in a tree is connected by a directed edge from exactly one other node. This node is called a parent. On the other hand, each node can be connected to arbitrary number of nodes, called children. Nodes with no children are called leaves, or external nodes. Nodes which are not leaves are called internal nodes. Nodes with the same parent are called siblings.

## Example

Figure(a): Binary Tree Example

## Traversals :

A traversal is a process that visits all the nodes in the tree. Since a tree is a nonlinear data structure, there is no unique traversal. We will consider several traversal algorithms with we group in the following two kinds.

(A Constituent College of Somaiya Vidyavihar University)

KJSCE/IT/SY BTech/SEMIII/DS/2022-23

* 1. depth-first traversal
  2. breadth-first traversal

There are three different types of depth-first traversals, :

* PreOrder traversal - visit the parent first and then left and right children; ∙ InOrder traversal - visit the left child, then the parent and the right child; ∙ PostOrder traversal - visit left child, then the right child and then the parent.

There is only one kind of breadth-first traversal--the level order traversal. This traversal visits nodes by levels from top to bottom and from left to right.

Consider an example the following tree and its four traversals:

## Figure 5.3: Example of Tree Traversals

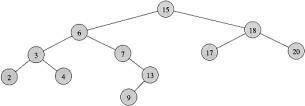
PreOrder - 8, 5, 9, 7, 1, 12, 2, 4, 11, 3

InOrder - 9, 5, 1, 7, 2, 12, 8, 4, 3, 11

PostOrder - 9, 1, 2, 12, 7, 5, 3, 11, 4, 8

LevelOrder - 8, 5, 4, 9, 7, 11, 1, 12, 3, 2

**Binary Search Tree(BST):** It is a binary tree where elements are stored in a particular order of left child is less than the parent and parent is less than the right child. This small modification makes BST efficient for searching. It is also called as BST property. **Example of BST** following fig is the example BST whose root is 8 and follows the BST property.



# Methods for storing binary trees

Binary trees can be constructed from programming language primitives in several ways. **Nodes and references**

(A Constituent College of Somaiya Vidyavihar University)

KJSCE/IT/SY BTech/SEMIII/DS/2022-23

In a language with records and references, binary trees are typically constructed by having

a tree node structure which contains some data and references to its left child and its right child. Sometimes it also contains a reference to its unique parent. If a node has fewer than two children, some of the child pointers may be set to a special null value, or to a special sentinel node.

Binary Search Tree can be implemented as a linked data structure in which each node is an object with two pointer fields. The two pointer fields *left and right* points to the nodes corresponding to the left child and the right child NULL in any pointer field signifies that there exists no corresponding child.

Figure : Binary Search Trees example



**Algorithm :**

**1. createBST()** – This function should create a ROOT pointer with NULL value as empty BST. **2. insert( typedef newelement )** – This void function should take a newelement as an argument to be

### inserted on an existing BST following the BST property.

1. **typedef search(typedef element )** – This function should search for specified element on the non empty BST and return a pointer to it.
2. **typedef delete(typedef element)** – This function searches for an element and if found deletes it from the BST and returns the same.
3. **typedef getParent(typedef element)** - This function searches for an element and if found, returns its parent element.

### **DisplayInorder( )** – This is a void function which should go through non- empty BST, traverse it in inorder fashion and print the elements on the way.



**NOTE : All functions should be able to handle boundary(exceptional) conditions.**

**Activity:** Write pseudocode for each method and implement the same.

(A Constituent College of Somaiya Vidyavihar University)

KJSCE/IT/SY BTech/SEMIII/DS/2022-23

**Results:** A program depicting the correct behaviour of BST and capable of handling all possible exceptional conditions and the same is reflecting clearly in the output.



## Program with output:

/\*

\* C program to implement the Binary Search Tree

\*/

#include <stdio.h> #include <stdlib.h>

// structure of a node struct node

{

int data;

struct node \*left; struct node \*right;

};

// globally initialized root pointer struct node \*root = NULL;

// function prototyping

struct node \*create\_node(int); void insert(int);

struct node \*delete (struct node \*, int); int search(int);

void inorder(struct node \*); void postorder();

void preorder();

struct node \*smallest\_node(struct node \*); struct node \*largest\_node(struct node \*); int get\_data();

int main()

{

int userChoice;

int userActive = 'Y'; int data;

struct node\* result = NULL;

while (userActive == 'Y' || userActive == 'y')

{

printf("\n\n------- Binary Search Tree \n");

printf("\n1. Insert");

printf("\n2. Delete"); printf("\n3. Search");

printf("\n4. Get Larger Node Data"); printf("\n5. Get smaller Node data"); printf("\n\n-- Traversals --"); printf("\n\n6. Inorder ");

printf("\n7. Post Order "); printf("\n8. Pre Oder "); printf("\n9. Exit");

printf("\n\nEnter Your Choice: "); scanf("%d", &userChoice); printf("\n");

switch(userChoice)

{

case 1:

data = get\_data(); insert(data); break;

case 2:

data = get\_data();

root = delete(root, data); break;

case 3:

data = get\_data();

if (search(data) == 1)

{

printf("\nData was found!\n");

}

else

{

printf("\nData does not found!\n");

}

break;

case 4:

result = largest\_node(root); if (result != NULL)

{

printf("\nLargest Data: %d\n", result->data);

}

break;

case 5:

result = smallest\_node(root); if (result != NULL)

{

printf("\nSmallest Data: %d\n", result->data);

}

break;

case 6:

inorder(root); break;

case 7:

postorder(root); break;

case 8:

preorder(root); break;

case 9:

printf("\n\nProgram was terminated\n"); break;

default:

printf("\n\tInvalid Choice\n"); break;

}

printf("\n \nDo you want to continue? "); fflush(stdin);

scanf(" %c", &userActive);

}

return 0;

}

// creates a new node

struct node \*create\_node(int data)

{

struct node \*new\_node = (struct node \*)malloc(sizeof(struct node));

if (new\_node == NULL)

{

printf("\nMemory for new node can't be allocated"); return NULL;

}

new\_node->data = data; new\_node->left = NULL; new\_node->right = NULL;

return new\_node;

}

// inserts the data in the BST void insert(int data)

{

struct node \*new\_node = create\_node(data);

if (new\_node != NULL)

{

// if the root is empty then make a new node as the root node if (root == NULL)

{

root = new\_node;

printf("\n\* node having data %d was inserted\n", data); return;

}

struct node \*temp = root; struct node \*prev = NULL;

// traverse through the BST to get the correct position for insertion while (temp != NULL)

{

prev = temp;

if (data > temp->data)

{

temp = temp->right;

}

else

{

temp = temp->left;

}

}

// found the last node where the new node should insert if (data > prev->data)

{

prev->right = new\_node;

}

else

{

prev->left = new\_node;

}

printf("\n\* node having data %d was inserted\n", data);

}

}

// deletes the given key node from the BST struct node \*delete (struct node \*root, int key)

{

if (root == NULL)

{

return root;

}

if (key < root->data)

{

root->left = delete (root->left, key);

}

else if (key > root->data)

{

root->right = delete (root->right, key);

}

else

{

if (root->left == NULL)

{

struct node \*temp = root->right; free(root);

return temp;

}

else if (root->right == NULL)

{

struct node \*temp = root->left; free(root);

return temp;

}

struct node \*temp = smallest\_node(root->right); root->data = temp->data;

root->right = delete (root->right, temp->data);

}

return root;

}

// search the given key node in BST int search(int key)

{

struct node \*temp = root;

while (temp != NULL)

{

if (key == temp->data)

{

return 1;

}

else if (key > temp->data)

{

temp = temp->right;

}

else

{

temp = temp->left;

}

}

return 0;

}

// finds the node with the smallest value in BST struct node \*smallest\_node(struct node \*root)

{

struct node \*curr = root;

while (curr != NULL && curr->left != NULL)

{

curr = curr->left;

}

return curr;

}

// finds the node with the largest value in BST struct node \*largest\_node(struct node \*root)

{

struct node \*curr = root;

while (curr != NULL && curr->right != NULL)

{

curr = curr->right;

}

return curr;

}

// inorder traversal of the BST void inorder(struct node \*root)

{

if (root == NULL)

{

return;

}

inorder(root->left); printf("%d ", root->data); inorder(root->right);

}

// preorder traversal of the BST void preorder(struct node \*root)

{

if (root == NULL)

{

return;

}

printf("%d ", root->data); preorder(root->left); preorder(root->right);

}

// postorder travsersal of the BST void postorder(struct node \*root)

{

if (root == NULL)

{

return;

}

postorder(root->left); postorder(root->right); printf("%d ", root->data);

}

// getting data from the user int get\_data()

{

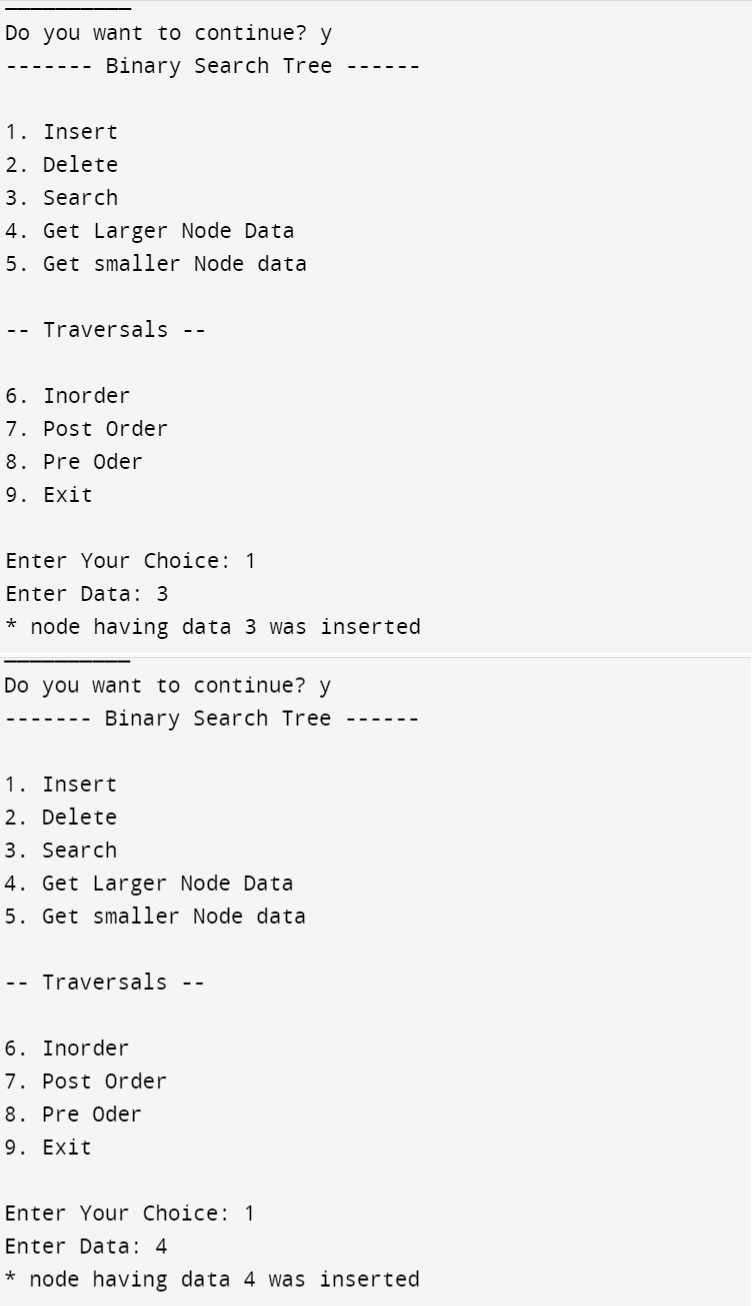
int data;

printf("\nEnter Data: "); scanf("%d", &data); return data;

}

## Output:



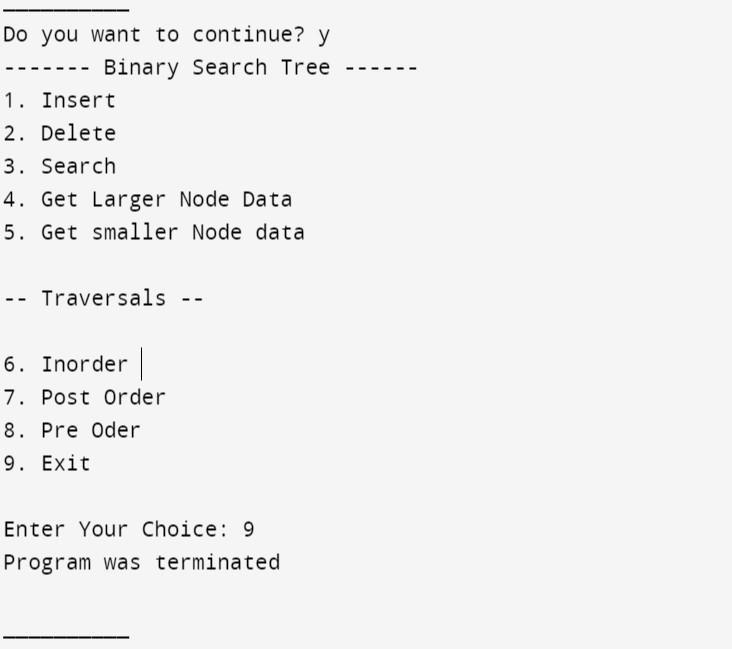












**Outcomes:**

**CO2. Apply linear and non-linear data structure in application development.**

## Conclusion:

Through this experiment, we learned how to implement a Dynamic Binary Search Tree using a Doubly Linked List (DLL) and perform various operations on it.

## Grade: AA / AB / BB / BC / CC / CD /DD

**Signature of faculty in-charge with date References:**

## Books/ Journals/ Websites:

* Y. Langsam, M. Augenstin and A. Tannenbaum, “Data Structures using C”, Pearson Education Asia, 1st Edition, 2002.
* https://ds1-iiith.vlabs.ac.in/exp/binary-search-trees/index.html