DATA STRUCTURES LABORATORY

Course Code: 1BCSL306

VTU 2025 Scheme – Semester 3

PART – B

Open-Ended Experiment

EXPERIMENT – 10

Hierarchical Data Management

Using Binary Search Tree

Simple Code • Easy Search • Easy to Write in Exam

SEARCH CREATORS

Your VTU Study Partner

Experiment 10

1. Aim

Aim

To develop a C program using Binary Search Tree to manage hierarchical information having numeric keys and perform insertion, traversal and quick search operations.

2. VTU Prescribed Experiment

VTU Prescribed Experiment

Develop a C program by selecting suitable data structure to manage hierarchical data/information having keys (numeric).
The program shall provide appropriate operations such as:

- Insertion of information.
- Traversal (display of all information).
- Quick search for information based on the key.

Demonstrate the solution using relevant test cases.

3. Suitable Data Structure

The suitable data structure is:

$$\boxed{\text{Binary Search Tree (BST)}}$$

A BST is suitable because it stores data in hierarchical form.

For every node:

$$\boxed{Left < Root < Right}$$

Hence searching can be performed quickly.

4. Simple Theory

A **Binary Search Tree** is a binary tree in which:

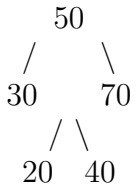
- Smaller keys are stored in the left subtree.

- Larger keys are stored in the right subtree.

For example, insert:

$$50,\ 30,\ 70,\ 20,\ 40$$

BST becomes:



5. Node Structure

Each node contains:

$$\boxed{Key + Left\ Pointer + Right\ Pointer}$$

C structure:

```c
struct Node
{
    int key;
    struct Node *left;
    struct Node *right;
};
```

6. Operations

The program performs:

$$\boxed{Insertion}$$

$$\boxed{Traversal}$$

$$\boxed{Search}$$

For displaying all information, we use:

$$\boxed{Inorder\ Traversal}$$

Inorder traversal of a BST displays keys in:

$$\boxed{Ascending\ Order}$$

7. Algorithm

7.1. Insertion

1. Create a new node.
2. If root is NULL, new node becomes root.
3. If key is smaller than root, go to left subtree.
4. If key is larger than root, go to right subtree.
5. Repeat until correct position is found.

7.2. Traversal

For inorder traversal:

1. Traverse left subtree.
2. Display root key.
3. Traverse right subtree.

Thus:

$$\boxed{Left \rightarrow Root \rightarrow Right}$$

7.3. Search

1. Compare search key with root.
2. If equal, key is found.
3. If key is smaller, search left subtree.
4. If key is larger, search right subtree.
5. Repeat until found or NULL is reached.

Easy Exam Memory

Remember only:

$$\boxed{Smaller \rightarrow Left}$$

$$\boxed{Larger \rightarrow Right}$$

$$\boxed{Inorder = Left\ Root\ Right}$$

$$\boxed{Search = Compare\ and\ Move}$$

8. C Program – Easy Student Version

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

struct Node
{
    int key;
    struct Node *left;
    struct Node *right;
};

struct Node* newNode(int key)
{
    struct Node *p;

    p=(struct Node*)malloc(sizeof(struct Node));

    p->key=key;
    p->left=NULL;
    p->right=NULL;

    return p;
}

struct Node* insert(struct Node *root,int key)
{
    if(root==NULL)
        return newNode(key);

    if(key<root->key)
        root->left=insert(root->left,key);

    else if(key>root->key)
        root->right=insert(root->right,key);

    return root;
}

void inorder(struct Node *root)
{
    if(root!=NULL)
    {
        inorder(root->left);

        printf("%d ",root->key);

        inorder(root->right);
    }
}

struct Node* search(struct Node *root,int key)
```

```
{
    if(root==NULL || root->key==key)
        return root;

    if(key<root->key)
        return search(root->left,key);

    return search(root->right,key);
}

int main()
{
    struct Node *root=NULL;
    struct Node *result;

    int ch,key;

    do
    {
        printf("\n1.Insert");
        printf("\n2.Display");
        printf("\n3.Search");
        printf("\n4.Exit");

        printf("\nEnter Choice: ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:

                printf("Enter Numeric Key: ");
                scanf("%d",&key);

                root=insert(root,key);

                printf("Key Inserted\n");

                break;

            case 2:

                if(root==NULL)
                    printf("Tree is Empty\n");
                else
                {
                    printf("BST Elements: ");
                    inorder(root);
                    printf("\n");
                }

                break;

```

```
            case 3:

                printf("Enter Key to Search: ");
                scanf("%d",&key);

                result=search(root,key);

                if(result!=NULL)
                    printf("Key Found\n");
                else
                    printf("Key Not Found\n");

                break;

            case 4:

                printf("Program Ended\n");
                break;

            default:

                printf("Invalid Choice\n");
        }

    }while(ch!=4);

    return 0;
}
```


9. Simple Program Explanation

9.1. newNode()

Creates a new BST node using:

```
malloc()
```

and initializes:

```
p->left=NULL;
p->right=NULL;
```

9.2. insert()

Important logic:

```
if(key<root->key)
    root->left=insert(root->left,key);

else if(key>root->key)
```

```
5       root->right=insert(root->right,key);
```

Remember:

$$\boxed{Small = Left}$$

$$\boxed{Big = Right}$$

9.3. inorder()

Code:

```
1  inorder(root->left);
2
3  printf("%d ",root->key);
4
5  inorder(root->right);
```

Therefore:

$$\boxed{Left \rightarrow Root \rightarrow Right}$$

For BST, inorder gives:

$$\boxed{Ascending\ Order}$$

9.4. search()

First check:

```
1  if(root==NULL || root->key==key)
2      return root;
```

If smaller:

```
1  return search(root->left,key);
```

Otherwise:

```
1  return search(root->right,key);
```

Easy Exam Memory

The three main functions are:

$$\boxed{insert()}$$

$$\boxed{inorder()}$$

$$\boxed{search()}$$

Remember:

$$\boxed{Insert = Compare + Move}$$

$$\boxed{Display = Inorder}$$

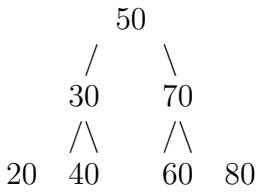
$$\boxed{Search = Compare + Left/Right}$$

10. Example

Insert the following numeric keys:

$$50,\ 30,\ 70,\ 20,\ 40,\ 60,\ 80$$

The BST becomes:



11. Traversal Example

Inorder traversal:

$$Left \rightarrow Root \rightarrow Right$$

Output:

$$\boxed{20\ 30\ 40\ 50\ 60\ 70\ 80}$$

Hence, inorder traversal of BST gives keys in sorted order.

12. Search Example

Search for:

$$\boxed{60}$$

Start:

$$50$$

Since:

$$60 > 50$$

move right to:

$$70$$

Since:

$$60 < 70$$

move left to:

$$60$$

Therefore:

$$\boxed{Key\ Found}$$

13. Sample Input and Output

```
1.Insert
2.Display
3.Search
4.Exit

Enter Choice: 1
Enter Numeric Key: 50
Key Inserted

Enter Choice: 1
Enter Numeric Key: 30
Key Inserted

Enter Choice: 1
Enter Numeric Key: 70
Key Inserted

Enter Choice: 1
Enter Numeric Key: 20
Key Inserted

Enter Choice: 1
Enter Numeric Key: 40
Key Inserted

Enter Choice: 2

BST Elements: 20 30 40 50 70

Enter Choice: 3

Enter Key to Search: 40
Key Found

Enter Choice: 3

```

```
Enter Key to Search: 100
Key Not Found

Enter Choice: 4

Program Ended
```

14. Relevant Test Cases

Test	Operation	Expected Result
1	Insert 50	50 becomes root.
2	Insert 30	30 becomes left child of 50.
3	Insert 70	70 becomes right child of 50.
4	Display using Inorder	Keys are displayed in ascending order.
5	Search existing key 30	Key Found.
6	Search missing key 90	Key Not Found.

15. Performance

In a balanced BST:

$$\boxed{Insertion = O(\log n)}$$

$$\boxed{Search = O(\log n)}$$

Traversal visits every node:

$$\boxed{Traversal = O(n)}$$

In the worst case, an unbalanced BST can take:

$$\boxed{O(n)}$$

for insertion or search.

16. Result

Result

Thus, a C program using Binary Search Tree was successfully implemented to manage hierarchical information having numeric keys.
The program performs:

- Insertion of numeric keys.
- Traversal and display of all information.
- Quick search using numeric key.
- Demonstration using suitable test cases.

17. Important Viva Questions

Important Viva Questions

1. What is a Binary Search Tree?
A Binary Search Tree is a binary tree in which smaller values are stored on the left and larger values are stored on the right.

2. What is the BST property?

$$\boxed{Left < Root < Right}$$

3. Why is BST suitable for hierarchical information?
Because BST stores information in hierarchical tree form and supports efficient searching.

4. What is the root node?
The first or topmost node of the tree.

5. Where is a smaller key inserted?
In the left subtree.

6. Where is a larger key inserted?
In the right subtree.

7. Which traversal is used to display BST in sorted order?
Inorder traversal.

8. What is the order of inorder traversal?

$$\boxed{Left \rightarrow Root \rightarrow Right}$$

9. What happens when search key equals root key?
The key is found.

10. Where do we search if key is smaller than root?
Left subtree.

11. Where do we search if key is larger than root?
Right subtree.

12. Which function is used for dynamic memory allocation?

`malloc()`

13. What does NULL indicate?
It indicates that a child node does not exist.

14. What is the average search complexity in a balanced BST?

$$\boxed{O(\log n)}$$

15. What is the worst-case search complexity of BST?

$$\boxed{O(n)}$$

18. 1-Minute Exam Revision

Easy Exam Memory

Data Structure

$$\boxed{Binary\ Search\ Tree}$$

BST Rule

$$\boxed{Left < Root < Right}$$

Insert Smaller

$$\boxed{Go\ Left}$$

Insert Larger

$$\boxed{Go\ Right}$$

Display

$$\boxed{Inorder}$$

Inorder

$$\boxed{Left \rightarrow Root \rightarrow Right}$$

Quick Search

$$\boxed{Compare \rightarrow Left/Right}$$

19. Easy Code Memory Map

Remember

Remember the code in this order:

$$\boxed{Node}$$

$$\downarrow$$

$$\boxed{newNode()}$$

$$\downarrow$$

$$\boxed{insert()}$$

$$\downarrow$$

$$\boxed{inorder()}$$

$$\downarrow$$

$$\boxed{search()}$$

$$\downarrow$$

$$\boxed{main()}$$

Most important exam rule:

$$\boxed{Small = Left, \qquad Big = Right}$$

SEARCH CREATORS

1BCSL306 – Data Structures Laboratory

PART – B

EXPERIMENT 10
COMPLETED

Hierarchical Data Management

Using Binary Search Tree

Insert → Traverse → Search → Test → Result

Your VTU Study Partner