MIT Art Design and Technology University

MIT School of Computing, Pune

Department of Computer Science and Engineering

Second Year B. Tech

Academic Year 2022-2023. (SEM-II)

Subject: Advance Data Structures Laboratory

Assignment 2

Assignment Title: A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide a facility to display whole data sorted in ascending/ Descending order. Also, find how many maximum comparisons may require for finding any keyword. Use Binary Search Tree for implementation

Aim: Implement Binary Search tree and its operations.

Prerequisite:

1. Basic concepts of Binary Search tree and its representation
2. Operations of Binary Search Tree and its traversals

Objectives:

1. Understand and use Binary Search Trees (BST) and concepts.
2. Recognize and define the basic attributes of a binary tree
3. Process BST using fast insertion, removal, and lookup of items while offering an efficient way to iterate them in sorted order
4. To build a dictionary application with BST.

Outcomes:

Upon Completion of the assignment the students will be able to

1. Create and implement BST
2. Understand and analyse various operation of the BST
3. Utilize BST ADT to create dictionary like application programmes.

Theory:

A binary search tree, also known as an ordered binary tree is a variant of binary tree in which the nodes are arranged in order. In a binary search tree, all the nodes in the left sub-tree have a value less than that of the root node. Correspondingly, all the nodes in the right sub-tree have a value either equal to or greater than the root node. The same rule is applicable to every sub-tree in the tree (Ref. Figure 1)

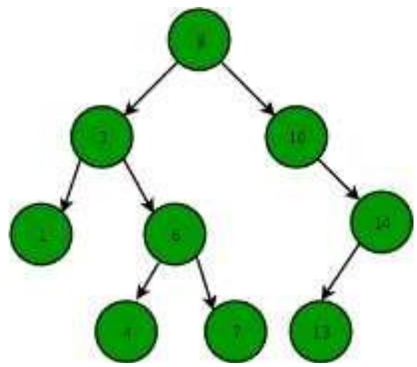


Figure 1: Example of Binary Search Tree

The average running time of a search operation is $O(\log_2 n)$ as at every step, we eliminate half of the sub-tree from the search process. Due to its efficiency in searching elements, binary search trees are widely used in dictionary problems where the code always inserts and searches the elements that are indexed by some key value.

The properties of the Binary Search Tree provide an ordering among keys so that the operations like search, minimum and maximum can be done fast. If there is no order, then we may have to compare every key to search for a given key.

1. Insertion in binary search tree:

A new key in BST is always inserted at the leaf.

- To insert an element in BST, we have to start searching from the root node; if the node to be inserted is less than the root node, then search for an empty location in the left subtree. Else, search for the empty location in the right subtree and insert the data.

2. Deletion in binary search tree:

When we delete a node, three possibilities arise.

1. Node to be deleted is the leaf (no child)
2. Node to be deleted has only one child
3. Node to be deleted has two children

I. **Node to be deleted is the leaf:** Simply remove it from the tree.
   - It is the simplest case to delete a node in BST. Here, we have to replace the leaf node with NULL and simply free the allocated space.

II. **Node to be deleted has only one child:**
   - In this case, we have to replace the target node (node which u want to delete) with its child, and then delete the child node.

III. **Node to be deleted has two children: Use inorder successor/inorder predecessor**

This case of deleting a node in BST is a bit complex among other two cases.

In such a case, the steps to be followed are listed as follows -

- First, find the inorder successor/inorder predecessor of the node to be deleted.

- After that, replace that node with the inorder successor/inorder predecessor until the target node is placed at the leaf of tree.
- And at last, replace the node with NULL and free up the allocated space.
- The inorder successor is required when the right child of the node is not empty. We can **obtain the inorder successor by finding the minimum element in the right subtree.**
- The inorder predecessor is required when the left child of the node is not empty. We can **obtain the inorder predecessor by finding the maximum element in the left subtree.**

3. Searching in binary search tree:

In Binary search tree, searching a node is easy because elements in BST are stored in a specific order.

The steps of searching a node in Binary Search tree are listed as follows -

- First, compare the element to be searched with the root element of the tree.
  If root is matched with the target element, then return the node's location.
  If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.
  If it is larger than the root element, then move to the right subtree.
- Repeat the above procedure recursively until the match is found.
- If the element is not found or not present in the tree, then return NULL.

Ascending and Descending order of BST

- Inorder traversal of BST prints it in ascending order.
- In-order means left-root-right.
- You can also do right-root-left. : It's just the way of traversal, that's it!!
- This is called reverse in-order by some people. So, in a BST, if you do reverse in-order, you get descending order sorted array.

Conclusion:

- Insertion, deletion, searching of an element is faster in BINARY SEARCH TREE than BINARY TREE due to the ordered characteristics
- In BINARY SEARCH TREE the left subtree has elements less than the nodes element and the right subtree has elements greater than the nodes element.
- Binary Search Tree does not allow duplicate values

Questions:

1. Create a binary search trees using the following data values (Step wise)
   45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67
2. Write an algorithm to create, Insert, Search, Delete and Modify an element in a BST
3. List applications of BST.

Algorithm for modify:

1) modify (root, value, newValue)
2) node = search (root, value)
   if node != NULL
      delete (root, value)
      insert (root, newvalue)
3) End.

3) List applications of BST.

Ans
- Dictionary look-up
- Sorting
- File systems.
- Graphs
- Expression trees.

Conclusion:

- Insertion, deletion, searching of an element is faster in a BST than in binary tree.
- In BST, the left node value is lesser than the parent node & right node value is greater than parent node.
- BST doesn't allow duplicate values.

Algorithm to search :

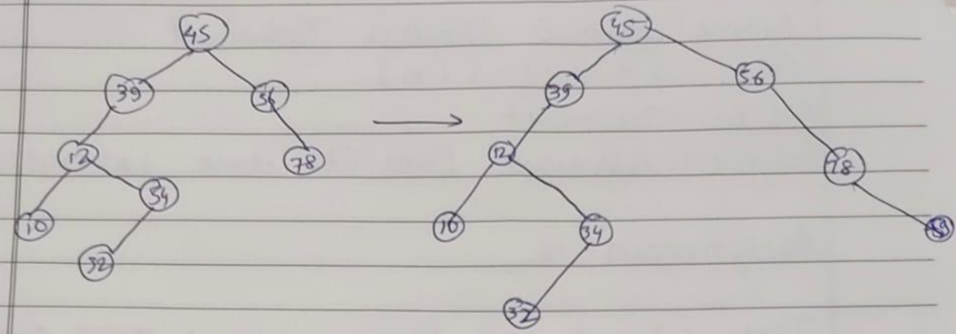
1) Search (root, item)
   i) if item = root → data or NULL
      return root
   ii) else if (item < root → data)
      search (root → left, item)
   iii) else
      return search (root → right, item)
2) END.

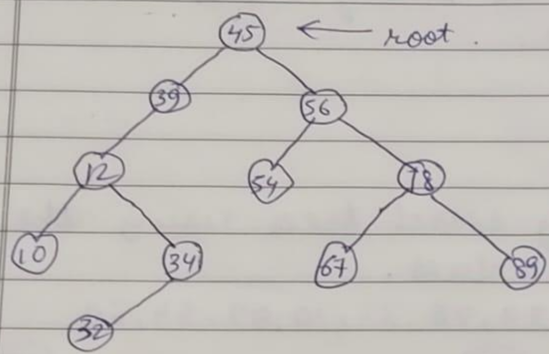
Algorithm to delete :

1) Delete (Tree, item)
   i) if item < Tree → data
      Delete (Tree → left, item)
   ii) else if item > tree → data
      Delete (Tree → right, item)
   iii) else if Tree → left & Tree → right
      set temp = findlargestnode (Tree → left)
      set tree → data = Temp → data
      Delete (Tree → Left, Temp → data)
   iv) else
      set temp = treee.
      - If tree → Left = NULL & Tree → Right = NULL
         set Tree = NULL.
      - Else if tree → left != NULL
         set tree = tree → Left
      - Else
         set tree → tree - right.
2) END.

MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

MIT-ADT UNIVERSITY PUNE, INDIA





2. Write an algorithm to create, insert, search, Delete and modify an element in a BST

Ans Algorithm for create and insert:

1) Create a new BST node and assign values to it
2) insert (node, key)
   i) if root == NULL,
      return the new node to calling function
   ii) if root → data < key,
      call the insert function with root → right & assign
   iii) if root → data > key,
      call the insert function with root → left & assign
3) Return original root pointer to the calling function

MIT SCHOOL OF ENGINEERING
Rajbaug, Loni-Kalbhor, Pune

MIT-ADT UNIVERSITY
PUNE, INDIA

Name: Sourav Shailesh Toshniwal
Class: SY CSE-1 (B)
Roll No.: 2213047
Subject: Advance Data Structure Laboratory.

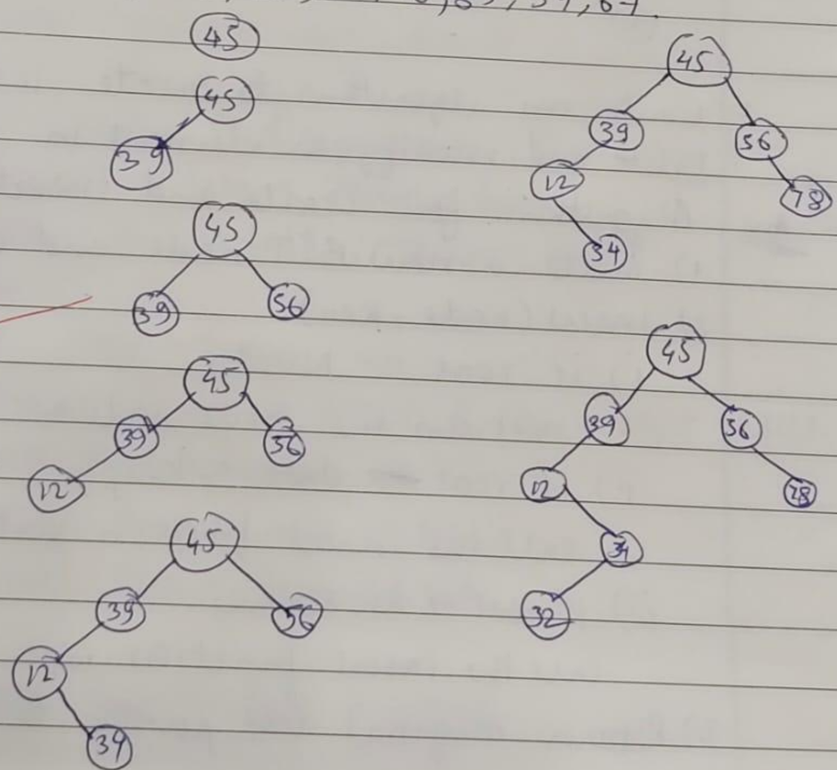
Assignment-2.

Title: Implement Binary search tree and its operation.

Questions:

1. Create a binary search trees using the following data values.
45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67.

Ans



Code:

```
#include<iostream>
#include<cstring>
```

```cpp
using namespace std;

typedef struct node
{
    char k[20];
    char m[20];
    struct node *left, *right;

}Node;


class dict
{
    public:
        Node *root;
        void create();
        void display_asc(Node *);
        void display_des(Node *);
        void insert(Node *root, Node *temp);
        int search(Node *, char[]);
        int update(Node *, char[]);
        Node *del(Node *, char[]);
        Node *min(Node *);
};


void dict :: create()
{
    Node *temp;
    int ch;


    do
    {
        temp = new Node;
        cout << "\nEnter Keyword:";
        cin >> temp->k;
        cout << "\nEnter Meaning:";
        cin >> temp->m;


        temp->left = NULL;
        temp->right = NULL;


        if (root == NULL)
        {
```

```
            root = temp;
        }
        else
        {
            insert(root, temp);
        }
        cout << "\nDo u want to add more (yes = 1/no = 0):";
        cin >> ch;
    } while (ch == 1);
}



void dict :: insert(node *root, node *temp)
{
    if (strcmp(temp->k, root->k) < 0)
    {
        if (root->left == NULL)
            root->left = temp;
        else
            insert(root->left, temp);
    }
    else
    {
        if (root->right == NULL)
            root->right = temp;
        else
            insert(root->right, temp);
    }
}


void dict :: display_asc(Node *root)
{
    if (root != NULL)
    {
        display_asc(root->left);
        cout << "\n Key Word :" << root->k;
        cout << "\t Meaning :" << root->m <<endl;
        display_asc(root->right);
    }
}


void dict :: display_des(Node *root)
{
    if (root != NULL)
    {
        display_des(root->right);
```

```
            cout << "\n Key Word :" << root->k;
            cout << "\t Meaning :" << root->m <<endl;
            display_des(root->left);
        }
    }



    int dict ::search(node *root, char k[20])
    {
        int c = 0;
        while (root != NULL)
        {
            c++;
            if (strcmp(k, root->k) == 0)
            {
                cout << "\nNo of Comparisons :" << c;
                return 1;
            }
            if (strcmp(k, root->k) < 0)
                root = root->left;
            if (strcmp(k, root->k) > 0)
                root = root->right;
        }


        return -1;
    }


    int dict ::update(Node *root, char k[20])
    {
        while (root != NULL)
        {
            if (strcmp(k, root->k) == 0)
            {
                cout << "\nEnter New Meaning of Keyword " << root->k<<" : ";
                cin >> root->m;
                return 1;
            }
            if (strcmp(k, root->k) < 0)
                root = root->left;
            if (strcmp(k, root->k) > 0)
                root = root->right;
        }
        return -1;
    }
```

```
Node *dict ::del(Node *root, char k[20])
{
    Node *temp;


    if (root == NULL)
    {
        cout << "\nElement Not Found";
        return root;
    }


    if (strcmp(k, root->k) < 0)
    {
        root->left = del(root->left, k);
        return root;
    }
    if (strcmp(k, root->k) > 0)
    {
        root->right = del(root->right, k);
        return root;
    }


    if (root->right == NULL && root->left == NULL)
    {
        temp = root;
        delete temp;
        return NULL;
    }
    if (root->right == NULL)
    {
        temp = root;
        root = root->left;
        delete temp;
        return root;
    }
    else if (root->left == NULL)
    {
        temp = root;
        root = root->right;
        delete temp;
        return root;
    }
    temp = min(root->right);
    strcpy(root->k, temp->k);
    root->right = del(root->right, temp->k);
    return root;
}
```

```
Node *dict ::min(Node *q)
{
    while (q->left != NULL)
    {
        q = q->left;
    }
    return q;
}


int main()
{
    int ch;
    dict d;
    d.root = NULL;


    while(true)
    {
        cout << "\nMenu \n1.Create \n2.Display ascending/descending \n3.Search
\n4.Update \n5.Delete \n6.Exit \nEnter your choice:";
        cin >> ch;


        switch (ch)
        {
            case 1:
                d.create();
                break;


            case 2:
                if (d.root == NULL)
                {
                    cout << "\nNo Keyword in the dictionary";
                }
                else
                {
                    int ord;
                    cout<<"\n1 - Ascending \n2 - Descending \nEnter choice for
displaying:";
                    cin>>ord;
                    if(ord==1)
                    {
                        d.display_asc(d.root);
```

```
                }
                else{
                    d.display_des(d.root);


                }

            }
            break;



        case 3:
            if (d.root == NULL)
            {
                cout << "\nDictionary is Empty. First add keywords then try
again ";
            }
            else
            {
                char k[20];
                cout << "\nEnter Keyword which you want to search:";
                cin >> k;


                if (d.search(d.root, k) == 1)
                    cout << "\nKeyword Found";
                else
                    cout << "\nKeyword Not Found";
            }
            break;



        case 4:
            if (d.root == NULL)
            {
                cout << "\nDictionary is Empty. First add keywords then try
again ";
            }
            else
            {
                char k[20];
                cout << "\nEnter Keyword whose meaning you want to update:";
                cin >> k;
                if (d.update(d.root, k) == 1)
                    cout << "\nMeaning Updated";
                else
                    cout << "\nMeaning Not Found";
```

```
                }
                break;



            case 5:
                if (d.root == NULL)
                {
                    cout << "\nDictionary is Empty. First add keywords then try
again ";
                }
                else
                {
                    cout << "\nEnter Keyword which you want to delete:";
                    char k[20];
                    cin >> k;
                    if (d.root == NULL)
                    {
                        cout << "\nKeyword is not present in the dictionary ";
                    }
                    else
                    {
                        d.root = d.del(d.root, k);
                        cout<<"\nKeyword is deleted";
                    }
                }
                break;


            case 6:
                exit(1);
                break;


            default:
                cout<<"Oops!... Invalid choice"<<endl;
                break;
        }
    }
    return 0;
}
```

Output:

```
PS C:\SOURAV\CODE\C++ language codes\ADS assignment> cd "c:\SOURAV\CODE\C++ language codes\ADS assignment\" ;

Menu
1.Create
2.Display ascending/descending
3.Search
4.Update
5.Delete
6.Exit
Enter your choice:1

Enter Keyword:Sourav

Enter Meaning:Name

Do u want to add more (yes = 1/no = 0):1

Enter Keyword:Toshniwal

Enter Meaning:Surname

Do u want to add more (yes = 1/no = 0):0

Menu
1.Create
2.Display ascending/descending
3.Search
4.Update
5.Delete
6.Exit
Enter your choice:2

1 - Ascending
2 - Descending
Enter choice for displaying:1

 Key Word :Sourav        Meaning :Name

 Key Word :Toshniwal     Meaning :Surname
```