PICT, PUNE

Assignment-02

Name:- Shrikrushna S Zirape.
Roll No:- 21286.

Problem Statement:-
A dictionary stores keywords and meanings. provide facility for adding new keyword, deleting keyword, updating values. For any entry provide facility for displaying whole data in ascending & descending order Also find the no of comparisons required for finding a keyword use BST.

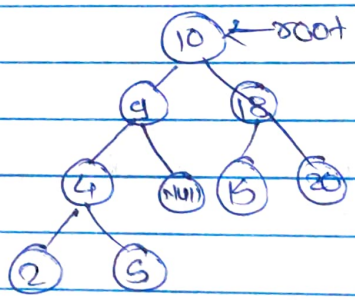
Learning objective:-
i) understand the concept of binary tree and it's properties.
ii) Implement functions to complete the req. takes for the BST.
iii) Impliment a dictionary using BST.

Leorning Outcomes:-
i) Impliment a dictionary using BST, which stores keywords & it's meaning. Perform the required operations on it.
ii) Write a menu driven function for the above in cpp using OOP-concept.

Theory:-
Binary Search Tree (BST) is a node based binary tree data structure. satisfy the following properties-

- left subtree contains nodes with value lesser than parent of that particular nodes.
- The right subtree of a particular node contains nodes with value greater than the parent node.
- The left and right subtree must also be binary tree.
- No two value can be same.

eg:-



Inorder traversal of the binary tree gives the data in ascending order.

Pseudo Code -

```
Class Node.
{
  string keyword, meaning;
  Node *left, *right;
  Node() {
  meaning = " ";
  keyword = " ";
  left = right = NULL;
  }
```

```
Node * (string k, string m)
{
   keyword = k;
   meaning = m
   right = NULL;
   left = NULL
  }
  friend class BST;
}

class BST
{ Node *parent = NULL
  Node *root = NULL
  algorithm SearchBST (string x)
   {
     parent = NULL
     Node *temp = root;
     int count = 0
    while (temp != NULL)
     {
       if (x > temp -> keyword)
          {
        parent = temp;
        temp = temp -> right;
         }
       if (x < temp -> keyword)
        {
          parent = temp
          temp = temp -> left;
        }
```

```
        else {
        Print("No of steps req" were:" (count) ;
        return temp'
        }
        count ++.
    }
    return NULL;
}
```

```
Algorithm insert (String k, String m)
  { // insert an entry.
  Node * n = Search BST (k);
    if (parent == NULL and n == NULL) then
     { root = new Node(k, m)
       return
     }
    if (n == NULL)
   { if (k > parent → keyword )
     {
       parent → right = new Node (k, m)
        return;
     }
     else {
     parent → left = new Node (k, m)
        return;
     }
   }
   Cout (" keyword already exist")
   return;
  }
```

```
Algorithm inorderDisplay (Node *temp)
{
  if (temp != NULL)
    { inorderDisplay (temp->left)
      Print (temp > keyword & meaning);
      inorderDisplay (temp->right)
    }
}
```

```
Algorithm descendingDisplay (Node *temp)
{
  if (temp != NULL)
    {
      descendingDisplay (temp->right)
      print (temp > keyword and temp->meaning)
      descendingDisplay (temp->left);
    }
}
```

```
Algorithm upDate (string k, string m)
{
  Node *n = SearchBST
   if (n == NULL) then
   {
     Print ("Keyword DNE");
     return
   }
   n-> meaning = m
}
```

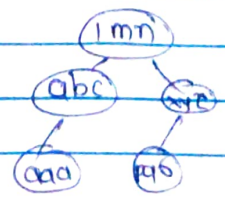
PICT, PUNE

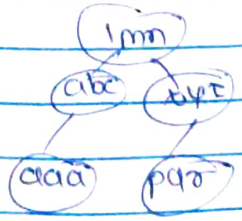
```
Algorithm deleteLeafNode (String x)
{
  Node *n = SearchBST(x);
  if (n == parent -> left)
  {
    parent -> left = NULL;
  }
  else { parent -> right = NULL;
  }
  delete(n);
  return;
}

Algorithm deleteNode (String x)
{ Node *n = SearchBST(x)
  if (n == NULL)
  { print ("keyword doesn't exist");
    return; }
  if (n -> left = NULL && n -> right = NULL)
  { deleteLeafNode(x)
    return;
  }
  if (n -> left = NULL OR n -> right = NULL)
  { if (n -> left = NULL)
  { if (parent -> keyword > n -> keyword)
    { parent -> left = n -> right; }
    else {
      parent -> right = n -> right; }
  }
```

```
else {
if (parent->keyword > n->keyword)
{ parent->left = n->left; }
else {
parent->right = n->left;
} }
delete(n);
return;
}
else {
Node *temp = inorderSuccessor(n->right);
String k = temp->keyword;
String m = temp->meaning;
deleteNode(temp->keyword);
n->keyword = k;
n->meaning = m;
return;
} } }
```

Test Cases	Input	output actual	output expected	Result
	Insert:			
①	lmn : lmn abc : abc xyz : xyz abc : abc zzz : zzz aaa : aaa par : par update (aaa : bbb)	abc already exist	abc already exist	pass





delete(xxx) zzz not found zzz not found. pass
Search(xxz) par found or par found
Search(par) after 25 p after 25 p

inorder:	inorder:
aaa:aaa	aaa:aaa
abc:abc	abc:abc
lmn:lmn	lmn:lmn
par:par	par:par
xyz:xyz	xyz:xyz.

* Time Complexities:-
class BST:-
- SearchBST(strings) - O(n)
- Insert(string, string) - O(n)
- descending display - O(n)
- update(string, string) - O(n)
- deleteNode(string) - O(n)
- search-passByRef(String, Node*, Node**) - O(n)

* Real World Application:-
- BST is used to maintain a sorted stream of data.
- A self Balancing BST is used to implement a double ended priority queue.
- BST support various searching & sorting algorithm.

Conclusion -

Thus we implimented a dictionary using the data structure BST. We implimented various functions like Insert, Search, delete, update, for dictionary storing keywords and their meaning we wrote a menu driven program for the same.