G. H. Raisoni College Of Engineering And Management, Wagholi Pune

2021- 2022

Assignment no :- 12

Department	CE [SUMMER 2022 (Online)]		
Term / Section	III/B	Date Of **submission**	14-12-2021
Subject Name /Code	Data Structures and Algorithms/ UCSL201/UCSP201		
Roll No.	SCOB77	Name	Pratham Rajkumar pitty
Registration Number	2020AC0E1100107		

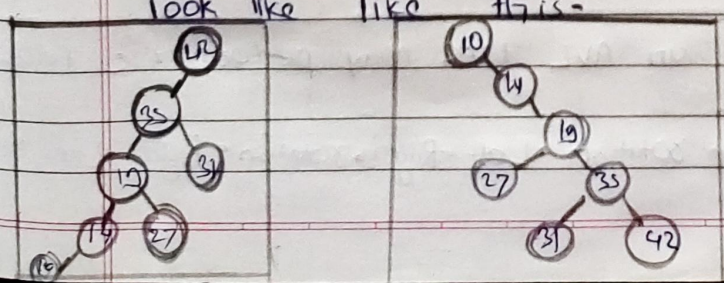
Experiment No. 12.

Aim :→ A Dictionary stores keywords & its meaning provide facility for adding new keyword, deleting keywords. Updating values of any entry. Provide facility to display whole data sorted in ascending / Desending order. Also find how many maximum comparisons may require for finding any keyword. Use height balance tree and find the complexity for finding a keyword.

Theory :→ An empty tree is height balanced tree if T is a nonempty binary tree with T_L and T_R as its left and right sub trees. The T is height balance if and only if its balance factor is 0, 1, -1.

AVL (Adelson - velskii and Landis) Tree :
A balance binary search tree. The best search time, that is O(log N) search times. An AVL tree is defined to be a well-balanced binary search tree in which each of its nodes has the AVL property. The AVL property is that the heights of left and right sub-trees of a node are either equal or if they differ only by 1

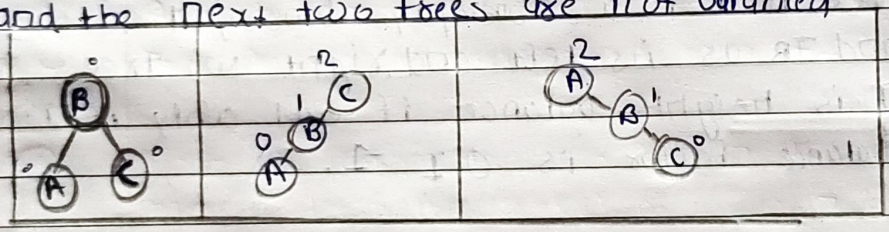
What if the input to binary search tree comes in a sorted (ascending or descending) manner ? It will then look like like this-



It is observed that BST's worst-case performance is closed to linear search algorithms, that is $O(n)$. In real-time data, we cannot predict data pattern and their frequencies so, a need arises to balance out the existing BST.

Named after their inventor Adelson, Velski & Landis, AVL trees are height balancing binary search trees ~~are height bg~~. AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the Balance factor.

Here we see that the first tree is balanced and the next two trees are not balanced.



In the second tree, the subtree of C has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of A has height 2 and the left is missing, so it is 0 and the difference is 2 again. AVL tree permits difference (balance factor) to be only one.

Balance factor = height(L.S) - height(R.S)

If the difference between in the height of Left Subtree and R.St is more than 1, the tree is balanced using some sorting techniques.

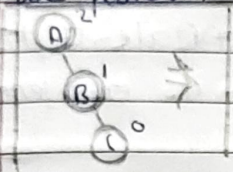
AVL Rotations

To balance itself, an AVL tree may perform the following 4 kinds of rotations.

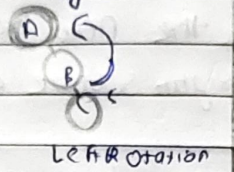
- Left rotation
- Right rotation
- Left-Right rotation
- Right-Left rotation

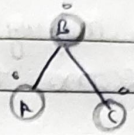
Left rotation

If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotations.



Right unbalanced tree



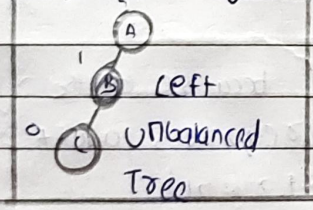


Balanced.

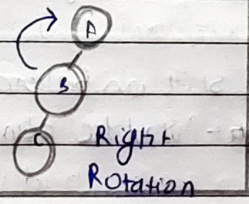
In our ex. node A become unbalanced as a node is inserted in the right subtree of A's right subtree we perform the left rotation by making A the left-subtree of B

Right Rotation

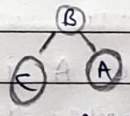
AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



Left unbalanced Tree

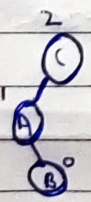


Right Rotation

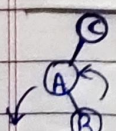


Balanced Tree

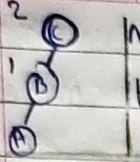
Left-Right Rotation



A node has been inserted into the right subtree of the left subtree. This makes C an unbalanced node These scenarios cause AVL tree to perform left-right rotn.

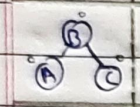


We first perform the left rotation on the left subtree of C. This makes A, the left subtree of B.



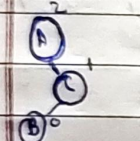
Node C is still unbalanced, however now, it is because of the left-subtree of the left-subtree.

We shall now right-rotate the tree, making B the tree, making B the new root node of this subtree C now becomes the right subtree of its own left subtree

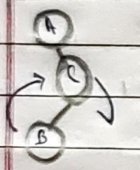


The tree is now balanced.

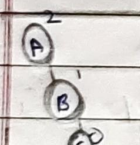
Right-Left Rotation



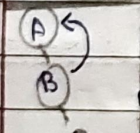
A node has been inserted into the left subtree of the right subtree. This makes A, an unbalanced node with balance factor 2.



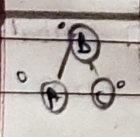
First, we perform the right rotation along C node, making C the right subtree of its own left subtree. Now, B becomes the right subtree of A.



Node A is still unbalanced because of the right subtree of its right subtree and requires a left rotation.



A left rotation is performed by making B the new root node of the subtree. A becomes the left subtree of its right subtree B.



The tree is now balanced.

Algorithm AVL

Tree: insert.

(1) If P is NULL then

1. P = new node
   i. P → left = NULL
   ii. P → right = NULL
   iii. P → element = y
   iv. P → height = 0

2. else if x > l = x < P → element
   a) insert (x, P → left)
   b. if height of P → left − height of P → right = 2
   i. insert (x, P → left)
   ii. insert (
   2. if height (P → left) − height (P → right) = 2 if
   x < P → left → element
   P = singlerotateleft (P)

3. else
   if x < P → element
   P = doublerotate left (P)
   a). insert (x, P → right)
   b). if height (P → right) − height (P → left) = 2
   if (x < P → right) → element
   P = singlerotateright (P)

4. else
   print already exits

5. int m, n, d
   else
6. P = doublerotate right (P)
7. m = AVL height (P → left)
8. n = AVL height (P → right
9. d = max (m, n)
10. P → height = d + 1
11. stop

```
Rotate WithLeftChild (AvlNode k2)
AvlNode k1 = k2.left;
k2.left = k1.right;
k1.right = k2;
k2.height = max(height(k2.left), height(k2.right)) +1;
Return k1;
```

Rotate with Right child (AvlNode k1

```
AvlNode k2 = k1.right;
k1.right = k2.left;
k2.left = k1;
k1.height = max (height (k1.left), height( k1.right)) +1;
k2.height = max (height (k2.right, k1.height) + 1;
return k2;
```

```
double with left child (AVLNode k3)
k3.left = rotate with Right child (k3.left);
return rotate with left child (k3);
```

```
doublewith Right child (AvlNode k1)
k1.right = rotatewith Left child (k1.right);
return rotate with Right child (k1);
```

Conclusion: This program gives us the knowledge about height balanced binary tree.

Program code :-

```
#include <iostream>
#include <string.h>
using namespace std;
typedef struct node
{
   char k[20];
   char m[20];
   class node *left;
   class node *right;
} node;
class dict
{
public:
   node *root;
   void create();
   void disp(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()
```

```
{
   class 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 (y=1/n=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::disp(node *root)
{
   if (root != NULL)
   {
      disp(root->left);
      cout << "\n Key Word :" << root->k;
      cout << "\t Meaning :" << root->m;
```

```
        disp(root->right);
    }
}
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 ofKeyword" << 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 No 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()
{
    cout << "\n\n SCOB77_Pratham_Pitty_Assignment_no_12 \n\n";

    int ch;
    dict d;
    d.root = NULL;
```

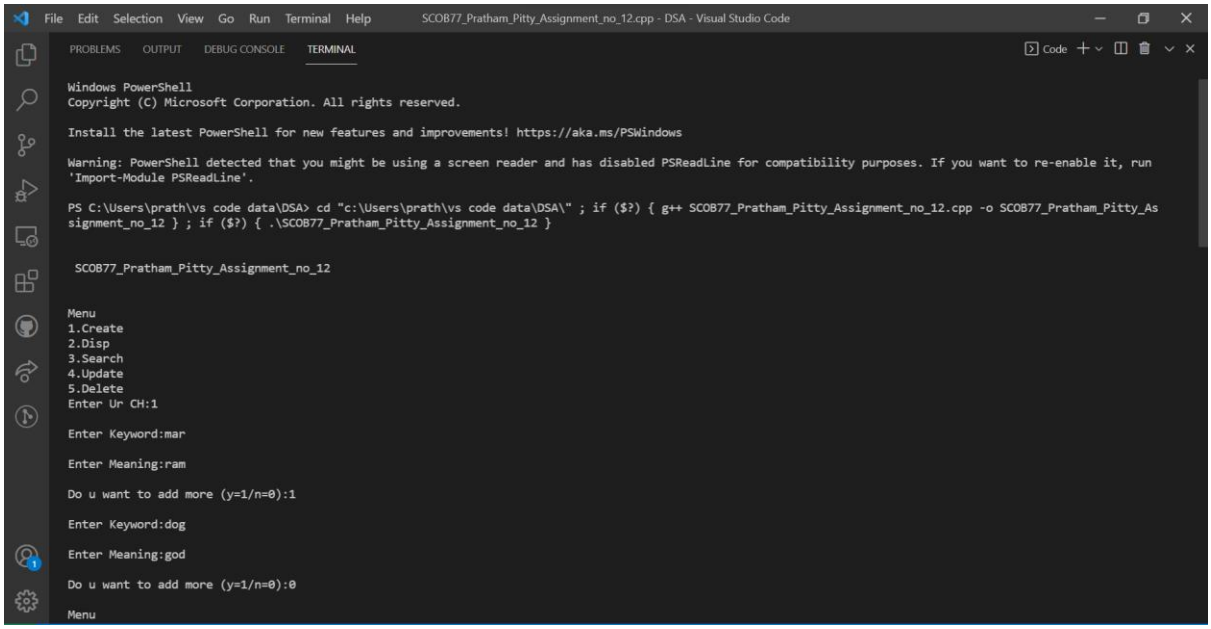
```
   do
   {
      cout <<
"\nMenu\n1.Create\n2.Disp\n3.Search\n4.Update\n5.Delete\nEnter
Ur CH:";
      cin >> ch;
      switch (ch)
      {
      case 1:
         d.create();
         break;
      case 2:
         if (d.root == NULL)
         {
            cout << "\nNo any Keyword";
         }
         else
         {
            d.disp(d.root);
         }
         break;
      case 3:
         if (d.root == NULL)
         {
```

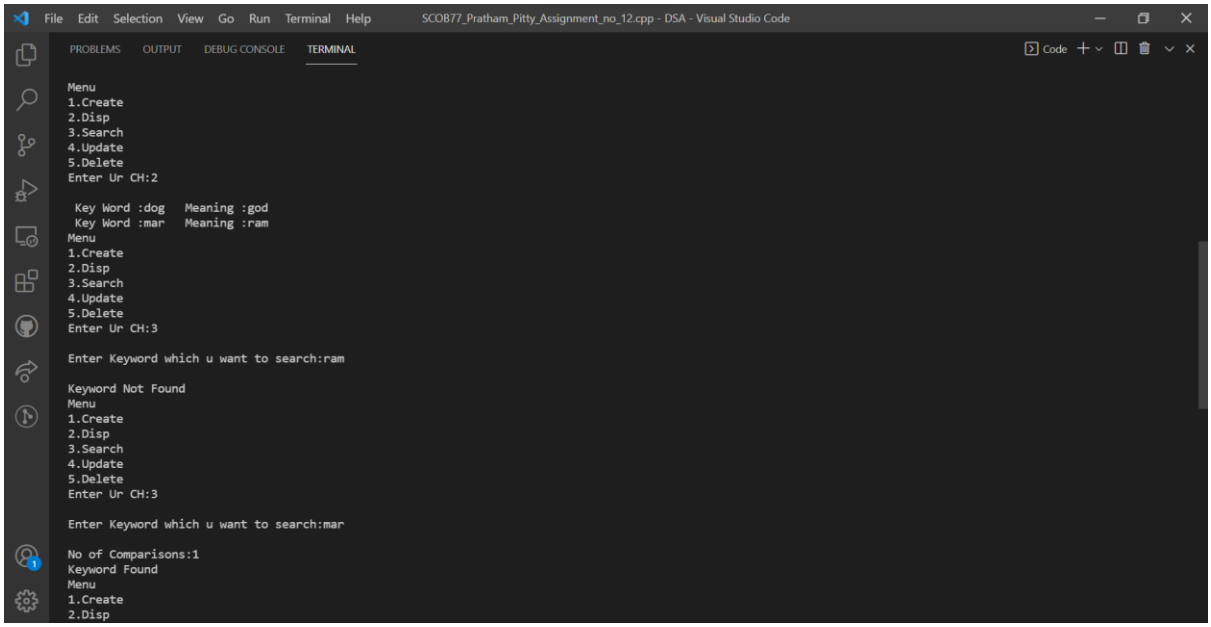
```
            cout << "\nDictionary is Empty. First add keywords then try
again ";
        }
        else
        {
            cout << "\nEnter Keyword which u want to search:";
            char k[20];
            cin >> k;
            if (d.search(d.root, k) == 1)
                cout << "\nKeyword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

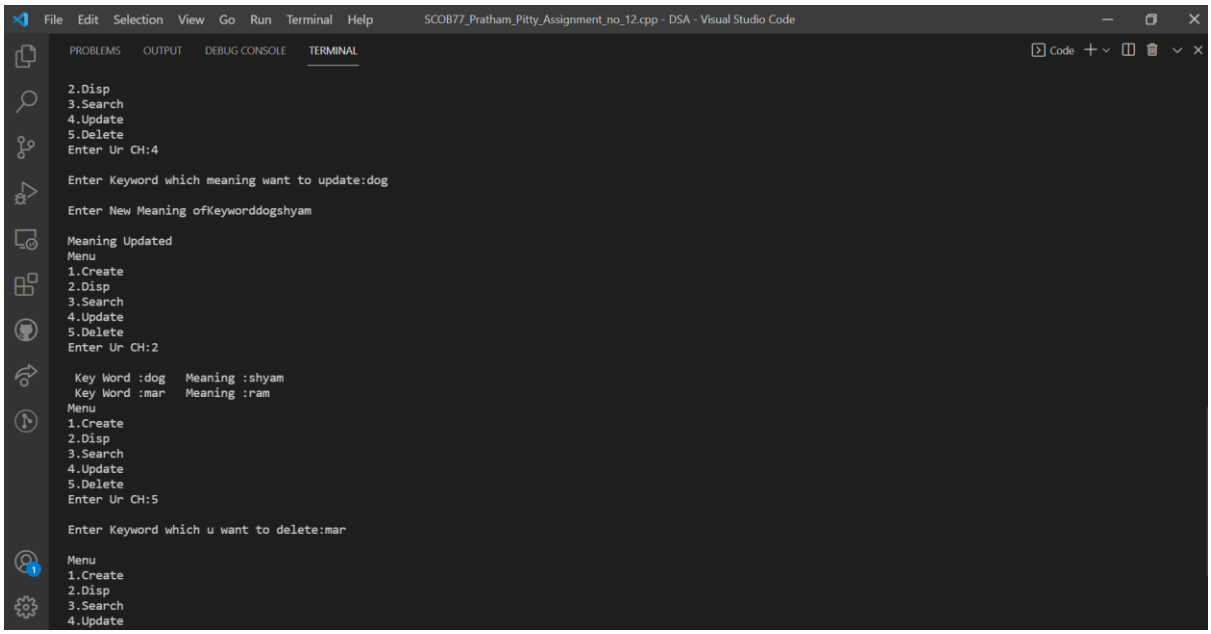
```
            cin >> k;
            if (d.update(d.root, k) == 1)
               cout << "\nMeaning Updated";
            else
               cout << "\nMeaning Not Found";
         }
         break;
      case 5:
         if (d.root == NULL)
         {
            cout << "\nDictionary is Empty. First add keywords then try
again ";
         }
         else
         {
            cout << "\nEnter Keyword which u want to delete:";
            char k[20];
            cin >> k;
            if (d.root == NULL)
            {
               cout << "\nNo any Keyword";
            }
            else
            {
```

```
                d.root = d.del(d.root, k);
            }
        }
    }
  } while (ch <= 5);
  return 0;
}
```

Output :-




File Edit Selection View Go Run Terminal Help
SCOB77_Pratham_Pitty_Assignment_no_12.cpp - DSA - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:2
Key Word :dog Meaning :god
Key Word :mar Meaning :ram
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:3
Enter Keyword which u want to search:ram
Keyword Not Found
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:3
Enter Keyword which u want to search:mar
No of Comparisons:1
Keyword Found
Menu
1.Create
2.Disp


File Edit Selection View Go Run Terminal Help
SCOB77_Pratham_Pitty_Assignment_no_12.cpp - DSA - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:4
Enter Keyword which meaning want to update:dog
Enter New Meaning ofKeyworddogshyam
Meaning Updated
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:2
Key Word :dog Meaning :shyam
Key Word :mar Meaning :ram
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:5
Enter Keyword which u want to delete:mar
Menu
1.Create
2.Disp
3.Search
4.Update