Assignment No: 12

Aim:

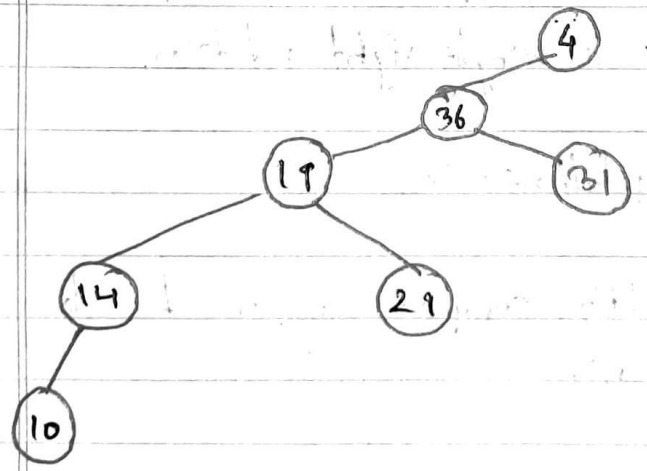
A dictionary stores keyword & its meaning provide facility for adding new keyword and its updating values of any entry provide facility to display whole class stored in ascending / descending order, also find how many max. comparisons may require for finding any keyword. Use height balance tree & find the complexity for finding keyword.

Theory:

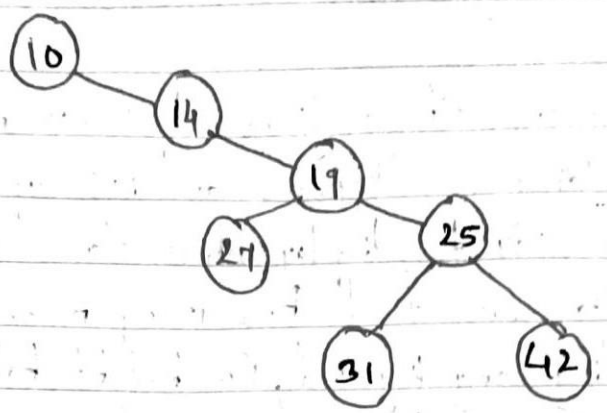
An empty tree is height balanced tree if T is a non-empty binary tree with T_L & T_R as its left & right sub-trees. T is height balance if and only if its balance factor 0-1-1.

The best time i.e $O(\log(N))$ search times an AVL tree is defined to be a well-balanced binary search tree in which each of its node has AVL property. The AVL property is that heights of the T_R & T_L of node are either equal or if they different only by 1.

This is observed that BST worst-case performance is closed to linear search algorithm that is $O(\log n)$ and their frequency so a need arise to balancer out the existing BST



if input appears non-increasing manner.



if input appears in non-decreasing manner

AVL Rotations ⇒

To balance itself an AVL-tree may perform following types of rotations.

Left Rotation ⇒

If tree becomes unbalanced when a node is into right structure / sub-tree then we perform a single left rotation.

Right Rotation ⇒

If tree becomes unbalanced when a node is inserted in left subtree then it needs a single right rotation.

Left-right / Right-left rotations ⇒

Double rotation are slightly complex version of already explained versions of rotations.

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)
```

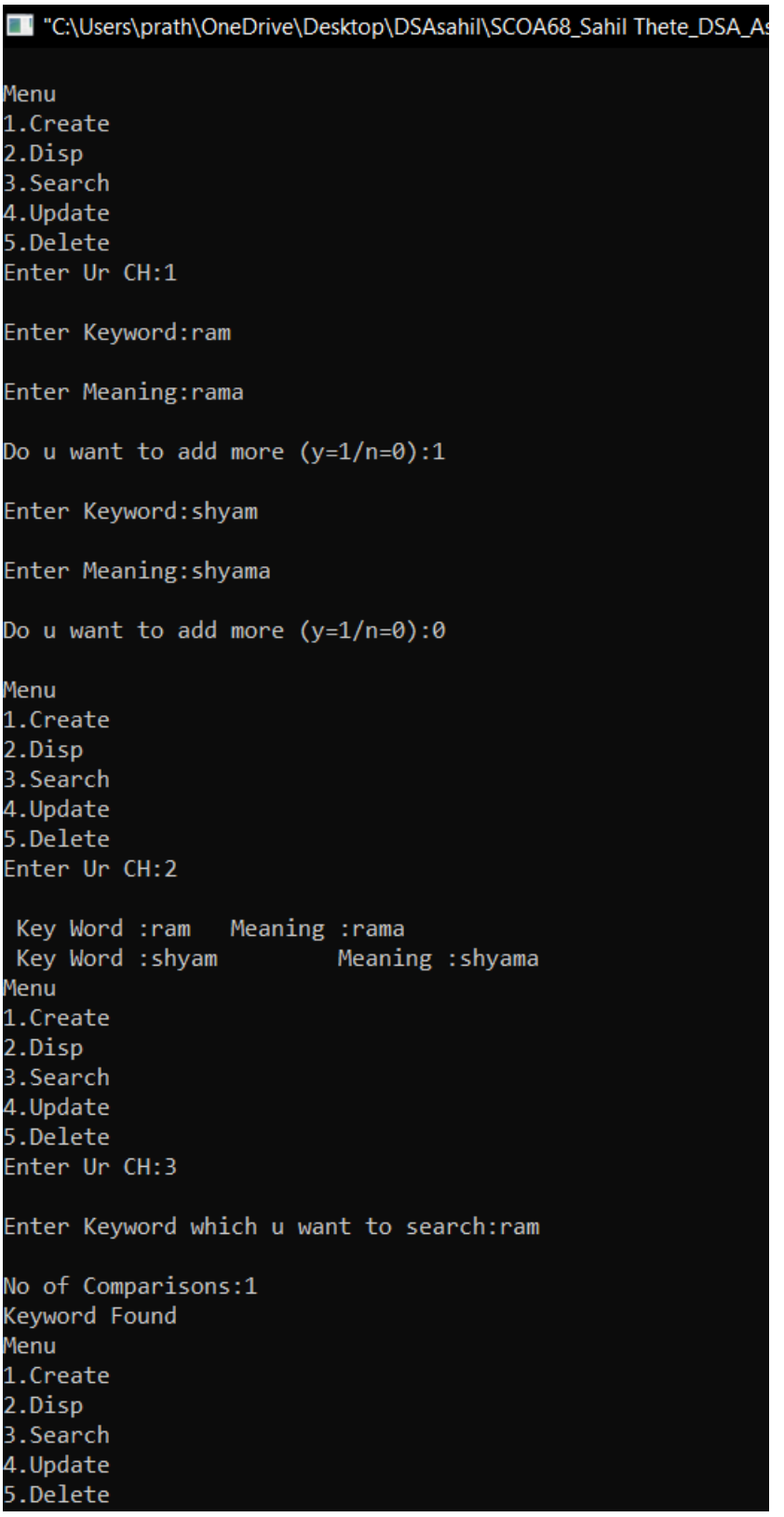
```
            root->left = temp;
        else
            insert(root->left, temp);
    }
    else
    {
        if (root->right == NULL)
            root->right = temp;
        else
            insert(root->right, temp);
    }
}
void dict::disp(node *root)
{
    if (root != NULL)
    {
        disp(root->left);
        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;
    }
```

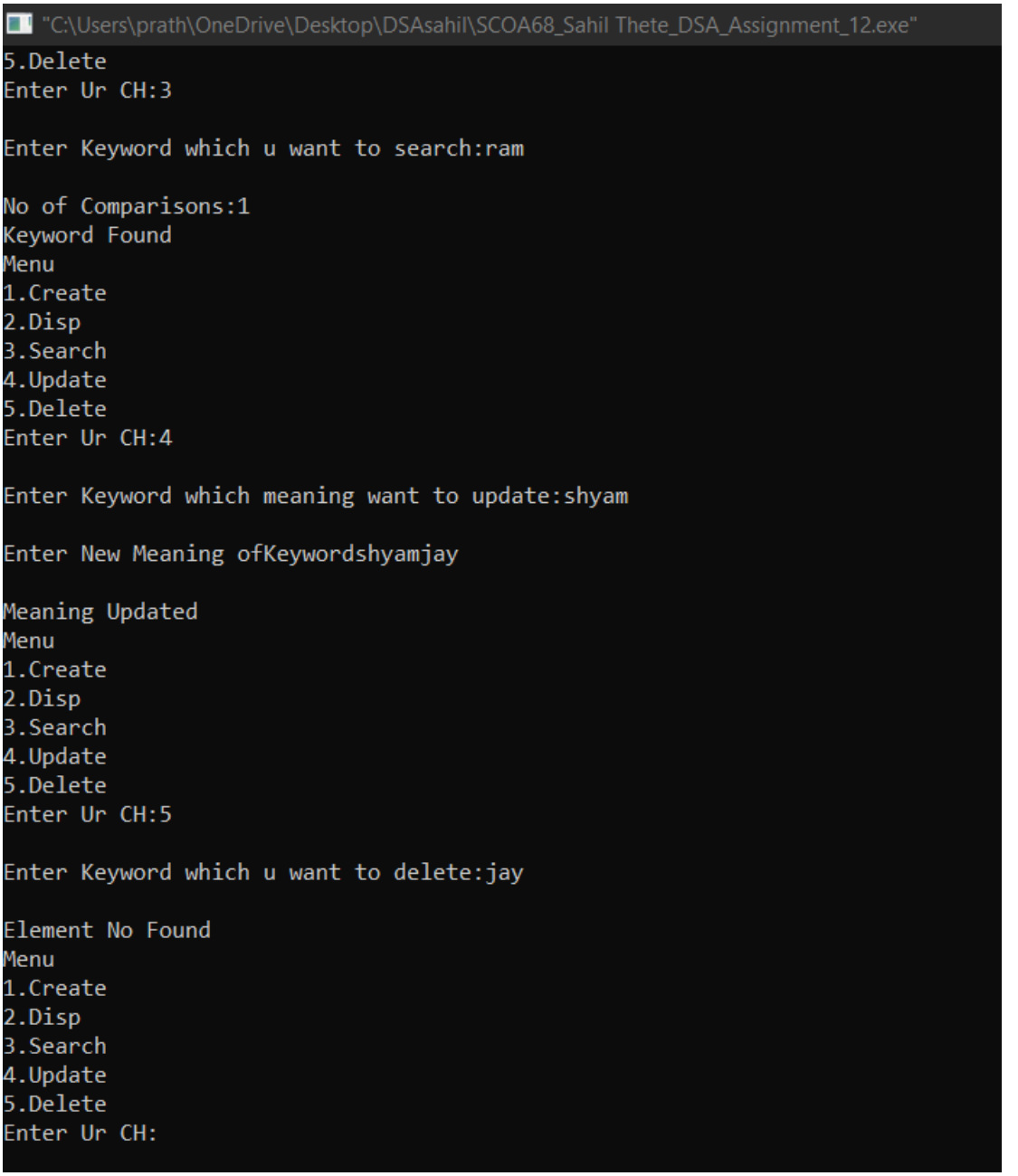
```
    return -1;
}
node *dict ::del(node *root, char k[20])
{
    node *temp;
    if (root == NULL)
    {
        cout << "\nElement No Found";
        return root;
    }
    if (strcmp(k, root->k) < 0)
    {
        root->left = del(root->left, k);
        return root;
    }
    if (strcmp(k, root->k) > 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()
{
```

```
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;
}
```

Program Output:-




"C:\Users\prath\OneDrive\Desktop\DSAsahil\SCOA68_Sahil Thete_DSA_Assignment_12.exe"
5.Delete
Enter Ur CH:3
Enter Keyword which u want to search:ram
No of Comparisons:1
Keyword Found
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:4
Enter Keyword which meaning want to update:shyam
Enter New Meaning ofKeywordshyamjay
Meaning Updated
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:5
Enter Keyword which u want to delete:jay
Element No Found
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH: