Experiment No.- 01

Consider telephone book database of N clients. Make use of a bash table implementation to quickly look up client's telephone number. Make use of two collision handling techniques and compare them. using number of comparisons required to find a set of telephone numbers

Program-

```
#include <iostream>
#include <string.h>
using namespace std;
struct node
{
   int value;
   node *next;
} * HashTable[10];
class hashing
{
public:
   hashing()
   {
      for (int i = 0; i < 10; i++)
      {
         HashTable[i] = NULL;
      }
   }
   int HashFunction(int value)
   {
      return (value % 10);
   }
   node *create_node(int x)
   {
      node *temp = new node;
      temp->next = NULL;
      temp->value = x;
      return temp;
   }
   void display()
   {
```

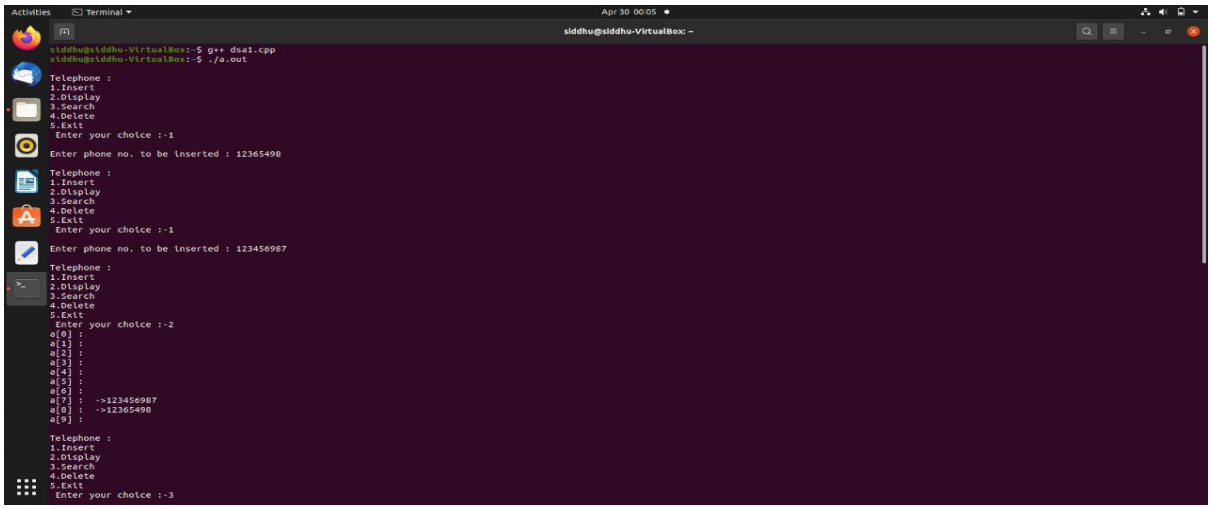
```
    for (int i = 0; i < 10; i++)
    {
        node *temp = new node;
        temp = HashTable[i];
        cout << "a[" << i << "] : ";
        while (temp != NULL)
        {
            cout << " ->" << temp->value;
            temp = temp->next;
        }
        cout << "\n";
    }
}
int searchElement(int value)
{
        {
    bool flag = false;
    int hash_val = HashFunction(value);
    node *entry = HashTable[hash_val];
    cout << "\nElement found at : ";
    while (entry != NULL)
    {
        if (entry->value == value)
        {
            cout << hash_val << " : " << entry->value << endl;
            flag = true;
        }
        entry = entry->next;
    }
    if (!flag)
        return -1;
        }
return 0;
}
void deleteElement(int value)
```

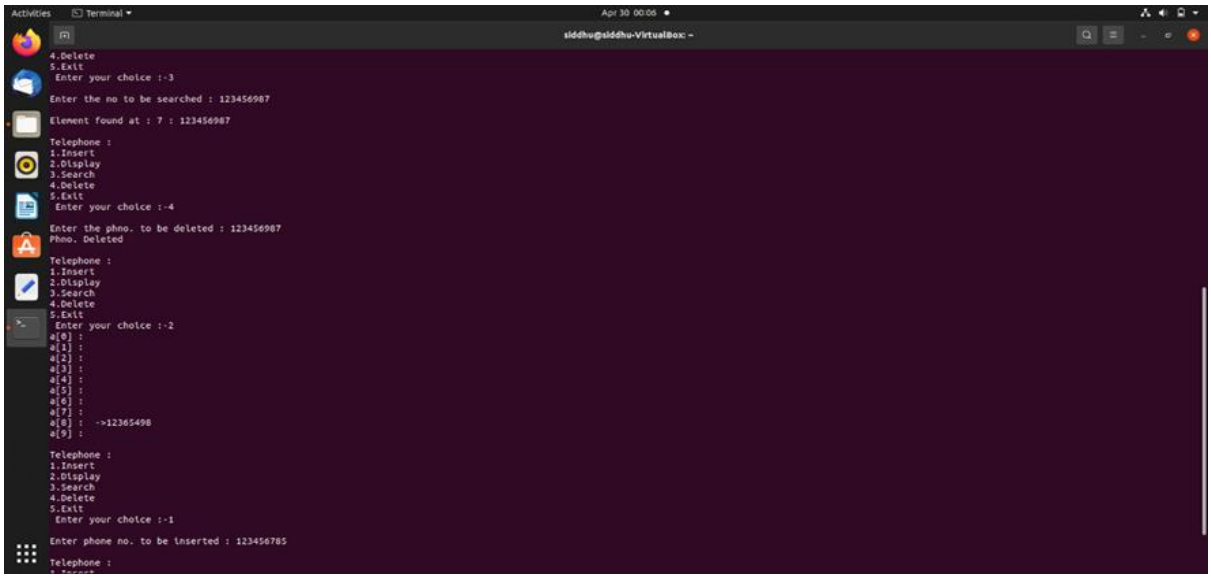
```
{
    int hash_val = HashFunction(value);
    node *entry = HashTable[hash_val];
    if (entry == NULL)
    {
        cout << "No Element found ";
        return;
    }
    if (entry->value == value)
    {
        HashTable[hash_val] = entry->next;
        return;
    }
    while ((entry->next)->value != value)
    {
        entry = entry->next;
    }
    entry->next = (entry->next)->next;
}
void insertElement(int value)
{
    int hash_val = HashFunction(value);
    // node* prev = NULL;
    // node* entry = HashTable[hash_val];
    node *temp = new node;
    node *head = new node;
    head = create_node(value);
    temp = HashTable[hash_val];
    if (temp == NULL)
    {
        HashTable[hash_val] = head;
    }
    else
    {
        while (temp->next != NULL)
```

```
            {
                temp = temp->next;
            }
            temp->next = head;
        }
    }
};
int main()
{
    int ch;
    int data, search, del;
    hashing h;
    do
    {
        cout << "\nTelephone : \n1.Insert \n2.Display \n3.Search \n4.Delete \n5.Exit\n Enter your
choice :-";
        cin >> ch;
        switch (ch)
        {
        case 1:
            cout << "\nEnter phone no. to be inserted : ";
            cin >> data;
            h.insertElement(data);
            break;
        case 2:
            h.display();
            break;
        case 3:
            cout << "\nEnter the no to be searched : ";
            cin >> search;
            if (h.searchElement(search) == -1)
            {
                cout << "No element found at key ";
                continue;
            }
            break;
```

```
        case 4:
            cout << "\nEnter the phno. to be deleted : ";
            cin >> del;
            h.deleteElement(del);
            cout << "Phno. Deleted" << endl;
            break;
        }
    } while (ch != 5);
    return 0;
}
```

Output-





Experiment No.- 02

Implement all the functions of a dictionary (ADT) using hashing and handle collisions using chaining with/without replacement. Data: Set of (key, value) pairs, Keys are mapped to values, Keys must be comparable, Keys must be unique. Standard Operations: Insert (key, value), Find (key), Delete(key).

Program-

```
#include<iostream>
#include<string.h>
using namespace std;
class HashFunction
 {
        typedef struct hash
       {
              long key;
              char name[10];
}hash;
 hash h[10];
        public:
 HashFunction();
              void insert();
              void display();
              int find(long);
 void Delete(long);
 };
HashFunction::HashFunction()
 {
       int i;
       for(i=0;i<10;i++)
        {
              h[i].key=-1;
              strcpy(h[i].name,"NULL");
        }
 }
void HashFunction::Delete(long k)
 {
 int index=find(k);
```

```
if(index==-1)
        {
 cout<<"\n\tKey Not Found";
        }
else
        {
                h[index].key=-1;
                strcpy(h[index].name,"NULL");
 cout<<"\n\tKey is Deleted";
        }
 }
int HashFunction::find(long k)
 {
        int i;
        for(i=0;i<10;i++)
        {
                 if(h[i].key==k)
                {
cout<<"\n\t"<<h[i].key<<" is Found at "<<i<<" Location With Name "<<h[i].name;
                 return i;
                 }
        }
 if(i==10)
        {
                return -1;
  }
  return 0;
 }
void HashFunction::display()
 {
        int i;
        cout<<"\n\t\tKey\t\tName";
        for(i=0;i<10;i++)
         {
                 cout<<"\n\th["<<i<<"]\t"<<h[i].key<<"\t\t"<<h[i].name;
```

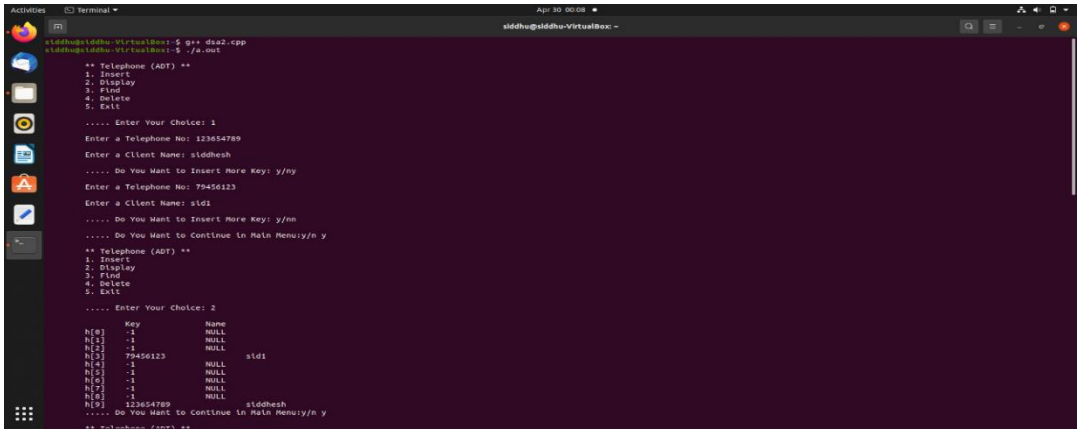
```
        }
 }
void HashFunction::insert()
 {
        char ans,n[10],ntemp[10];
        long k,temp;
 int v,hi,cnt=0,flag=0,i;
 do
        {
                if(cnt>=10)
                {
                        cout<<"\n\tHash Table is FULL";
                        break;
                }
                cout<<"\n\tEnter a Telephone No: ";
                cin>>k;
                cout<<"\n\tEnter a Client Name: ";
                cin>>n;
                hi=k%10;// hash function
                if(h[hi].key==-1)
                 {
                        h[hi].key=k;
                        strcpy(h[hi].name,n);
                }
                else
                {
                        if(h[hi].key%10!=hi)
                        {
                                temp=h[hi].key;
                                strcpy(ntemp,h[hi].name);
                                h[hi].key=k;
                                strcpy(h[hi].name,n);
                                for(i=hi+1;i<10;i++)
                                {
                                        if(h[i].key==-1)
```

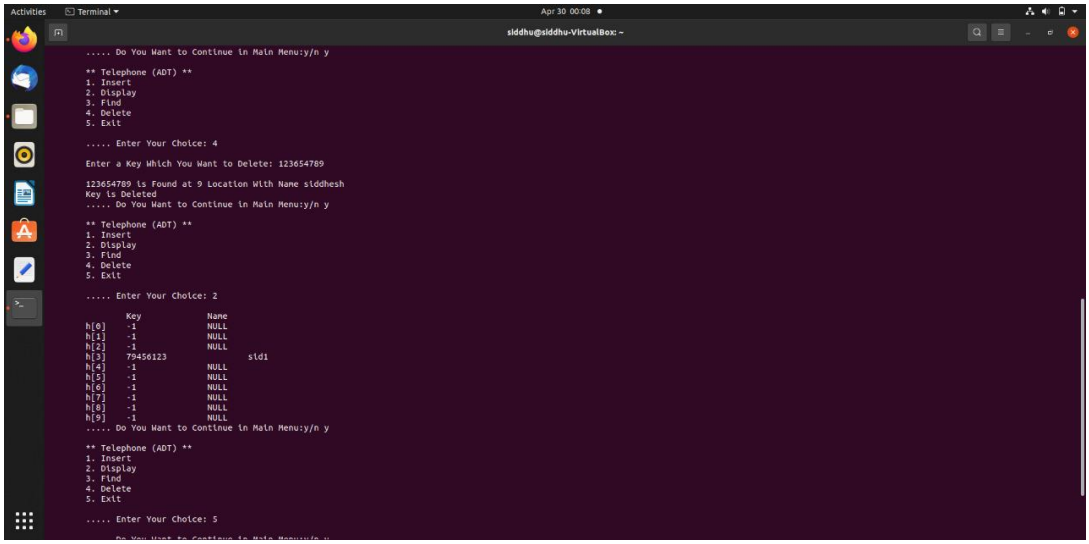
```
                {
                        h[i].key=temp;
                        strcpy(h[i].name,ntemp);
                        flag=1;
                        break;
                }
        }
        for(i=0;i<hi && flag==0;i++)
        {
                if(h[i].key==-1)
                {
                        h[i].key=temp;
                        strcpy(h[i].name,ntemp);
                        break;
                }
        }
}
else
{
        for(i=hi+1;i<10;i++)
        {
                if(h[i].key==-1)
                {
                        h[i].key=k;
                        strcpy(h[i].name,n);
                        flag=1;
                        break;
                }
        }
        for(i=0;i<hi && flag==0;i++)
        {
                if(h[i].key==-1)
                {
                        h[i].key=k;
                        strcpy(h[i].name,n);
```

```
                                        break;
                                    }
                                }
                            }
                        }
                        flag=0;
                        cnt++;
                        cout<<"\n\t..... Do You Want to Insert More Key: y/n";
                        cin>>ans;
            }while(ans=='y'||ans=='Y');
    }
int main()
    {
    long k;
    int ch,index;
    char ans;
    HashFunction obj;
            do
             {
                    cout<<"\n\t** Telephone (ADT) **";
                    cout<<"\n\t1. Insert\n\t2. Display\n\t3. Find\n\t4. Delete\n\t5. Exit\n\t";
                    cout<<"\n\t..... Enter Your Choice: ";
                    cin>>ch;
                    switch(ch)
                    {
                            case 1:  obj.insert();
                                    break;
                            case 2: obj.display();
                                     break;
                            case 3: cout<<"\n\tEnter a Key Which You Want to Search: ";
                                     cin>>k;
                                    index=obj.find(k);
                                    if(index==-1)
                                     {
                                            cout<<"\n\tKey Not Found";
```

```
                    }
                    break;
                case 4: cout<<"\n\tEnter a Key Which You Want to Delete: ";
                    cin>>k;
                    obj.Delete(k);
                    break;
                case 5:
                    break;
            }
            cout<<"\n\t..... Do You Want to Continue in Main Menu:y/n ";
            cin>>ans;
        }while(ans=='y'||ans=='Y');
}
```

Output-





Experiment No.- 03

A book consists of chapters, chapters consist of sections and sections consist of subsections. Construct a tree and print the nodes. Find the time and space requirements of your method.

Program-

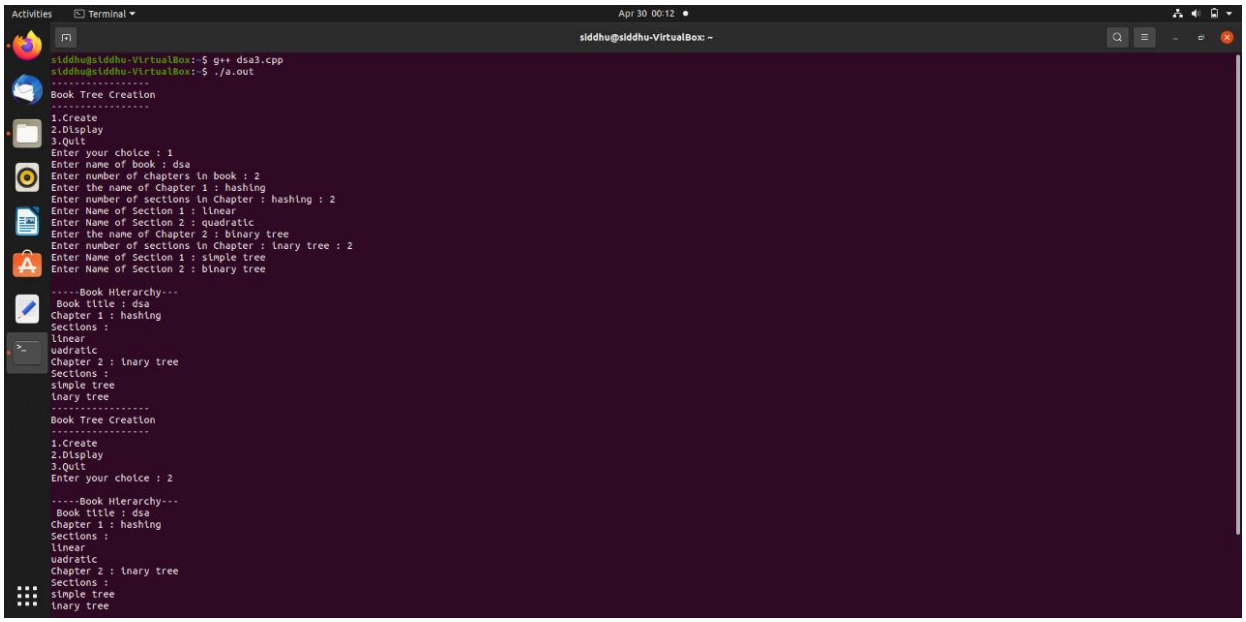
```
#include <iostream>
#include <string.h>
using namespace std;

struct node // Node Declaration
{
   string label;
   //char label[10];
   int ch_count;
   struct node *child[10];
} * root;

class GT // Class Declaration
{
public:
   void create_tree();
   void display(node *r1);

   GT()
   {
      root = NULL;
   }
};

void GT::create_tree()
{
   int tbooks, tchapters, i, j, k;
   root = new node;
   cout << "Enter name of book : ";
   cin.get();
   getline(cin, root->label);
   cout << "Enter number of chapters in book : ";
```

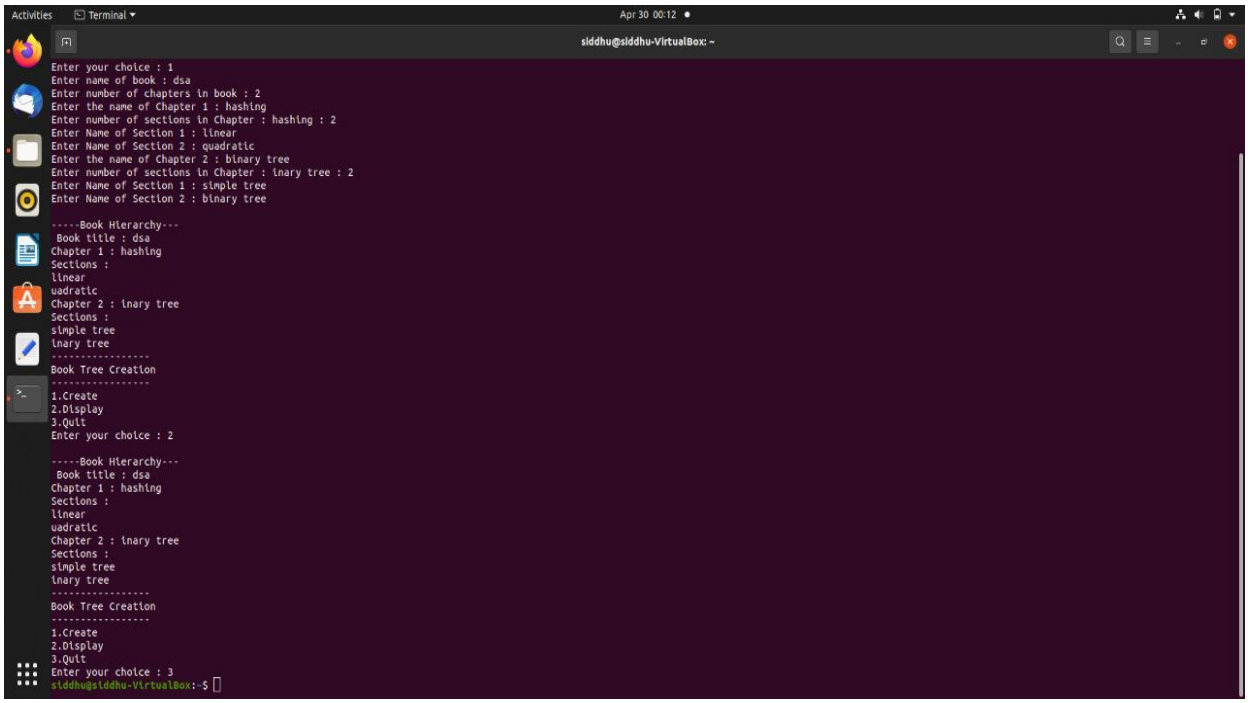
```
    cin >> tchapters;
    root->ch_count = tchapters;
    for (i = 0; i < tchapters; i++)
    {
        root->child[i] = new node;
        cout << "Enter the name of Chapter " << i + 1 << " : ";
        cin.get();
        getline(cin, root->child[i]->label);
        cout << "Enter number of sections in Chapter : " << root->child[i]->label << " : ";
        cin >> root->child[i]->ch_count;
        for (j = 0; j < root->child[i]->ch_count; j++)
        {
            root->child[i]->child[j] = new node;
            cout << "Enter Name of Section " << j + 1 << " : ";
            cin.get();
            getline(cin, root->child[i]->child[j]->label);
        }
    }
}

void GT::display(node *r1)
{
    int i, j, k, tchapters;
    if (r1 != NULL)
    {
        cout << "\n-----Book Hierarchy---";
        cout << "\n Book title : " << r1->label;
        tchapters = r1->ch_count;
        for (i = 0; i < tchapters; i++)
        {

            cout << "\nChapter " << i + 1;
            cout << " : " << r1->child[i]->label;
            cout << "\nSections : ";
            for (j = 0; j < r1->child[i]->ch_count; j++)
```

```
            {
                cout << "\n"<< r1->child[i]->child[j]->label;
            }
        }
    }
    cout << endl;
}

int main()
{
    int choice;
    GT gt;
    while (1)
    {
        cout << "-----------------" << endl;
        cout << "Book Tree Creation" << endl;
        cout << "-----------------" << endl;
        cout << "1.Create" << endl;
        cout << "2.Display" << endl;
        cout << "3.Quit" << endl;
        cout << "Enter your choice : ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            gt.create_tree();
        case 2:
            gt.display(root);
            break;
        case 3:
            cout << "Thanks for using this program!!!";
            exit(1);
        default:
            cout << "Wrong choice!!!" << endl;
        }
```

```
    }
    return 0;
}
```

Output-





Experiment No.- 04

Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree - i. Insert new node, ii. Find number of nodes in longest path from root, iii. Minimum data value found in the tree, iv. Change a tree so that the roles of the left and right pointers are swapped at every node, v. Search a value.

Program-

```
#include<iostream>
#include<math.h>
using namespace std;

struct Bstnode
{
 int data;
 Bstnode *left = NULL;
 Bstnode *right = NULL;

};

class Btree
{

  int n;
  int x;
  int flag;

public:
  Bstnode * root;
 Btree()
 {
  root = NULL;
 }

 Bstnode *GetNewNode(int in_data)
 {
  Bstnode * ptr = new Bstnode();
  ptr->data = in_data;
```

```
 ptr->left = NULL;
 ptr->right = NULL;
 return ptr;
}

Bstnode *insert( Bstnode *temp , int in_data)
{
 if( temp == NULL )
 {
  temp = GetNewNode(in_data);
 }
 else if( temp->data > in_data)
 {
  temp->left = insert(temp->left , in_data);
 }
 else
 {
  temp->right = insert( temp->right , in_data);
 }
 return temp;
}

void input()
{
 cout<<"ENTER NUMBER OF ELEMENTS IN THE BST : ";
 cin>>n;
 for(int i = 0 ; i < n ; i++)
 {
  cout<<"NUMBER = ";
  cin>>x;
  root = insert(root , x);
 }
}

int search(Bstnode *temp ,int in_data)
```

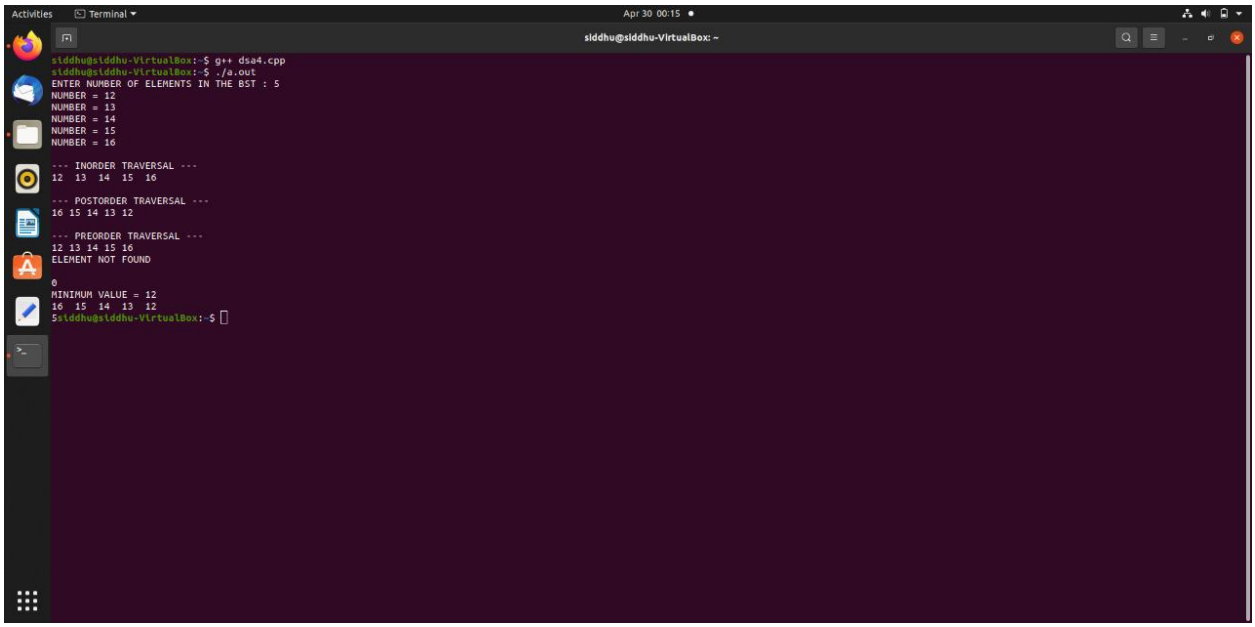
```
{
{
 if( temp != NULL)
 {
  if(temp->data == in_data)
  {
   cout<<":-- RECORD FOUND --:"<<endl;
   return 1;
  }
  else if(in_data < temp->data)
  {
   this->search(temp->left, in_data);
  }
  else if(in_data > temp->data)
  {
   this->search(temp->left , in_data);
  }
 }
 else
 {
   return 0;
 }
  return 0;
  }
}


void minvalue(Bstnode *temp)
{
 while(temp->left != NULL)
 {
  temp = temp->left;
 }
 cout<<"MINIMUM VALUE = "<<temp->data<<endl;
}
```

```
void mirror(Bstnode *temp)
{
 if(temp == NULL)
 {
  return;
 }
 else
 {
  Bstnode *ptr;
  mirror(temp->left);
  mirror(temp->right);
  ptr = temp->left;
  temp->left = temp->right;
  temp->right = ptr;
 }
}

void display()
{
 cout<<endl<<"--- INORDER TRAVERSAL ---"<<endl;
 inorder(root);
 cout<<endl;
 cout<<endl<<"--- POSTORDER TRAVERSAL ---"<<endl;
 postorder(root);
 cout<<endl;
 cout<<endl<<"--- PREORDER TRAVERSAL ---"<<endl;
 preorder(root);
 cout<<endl;
}
void inorder(Bstnode *temp)
{
 if(temp != NULL)
 {
  inorder(temp->left);
```

```
  cout<<temp->data<<"  ";
  inorder(temp->right);
 }
}
 void postorder(Bstnode *temp)
{
 if(temp != NULL)
 {
  postorder(temp->left);
  postorder(temp->right);
  cout<<temp->data<<" ";
 }
}
 void preorder(Bstnode *temp)
{
 if(temp != NULL)
 {
  cout<<temp->data<<" ";
  preorder(temp->left);
  preorder(temp->right);
 }
}
 int depth(Bstnode *temp)
{
 if(temp == NULL)
  return 0;
 return (max((depth(temp->left)),(depth(temp->right))) +1);
 }
};
int main()
{
 Btree obj;
 obj.input();
 obj.display();
 int a = 0;
```

```
a = obj.search(obj.root,10);
if( a == 0)
{
 cout<<"ELEMENT NOT FOUND"<<endl;
}
else
 cout<<"ELEMENT FOUND"<<endl;
cout<<endl<<a<<endl;
obj.minvalue(obj.root);
obj.mirror(obj.root);
obj.inorder(obj.root);
//int d ;
cout<<endl<<obj.depth(obj.root);
//cout<<endl<<d<<endl;
return 0;
}
```

Output-



Experiment No.- 05

A Dictionary stores keywords and its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide 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.

Program-

```
#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;
 }
```

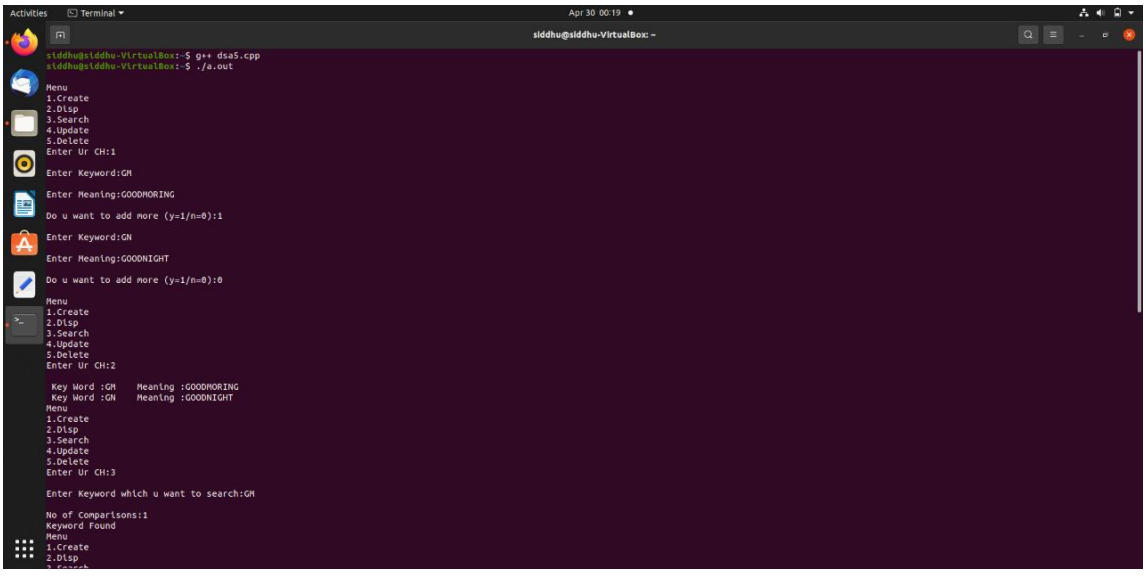
```
 if (strcmp(k,root->k) > 0)
 {
   root->right = del(root->right, k);
   return root;
 }

 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)
{
```

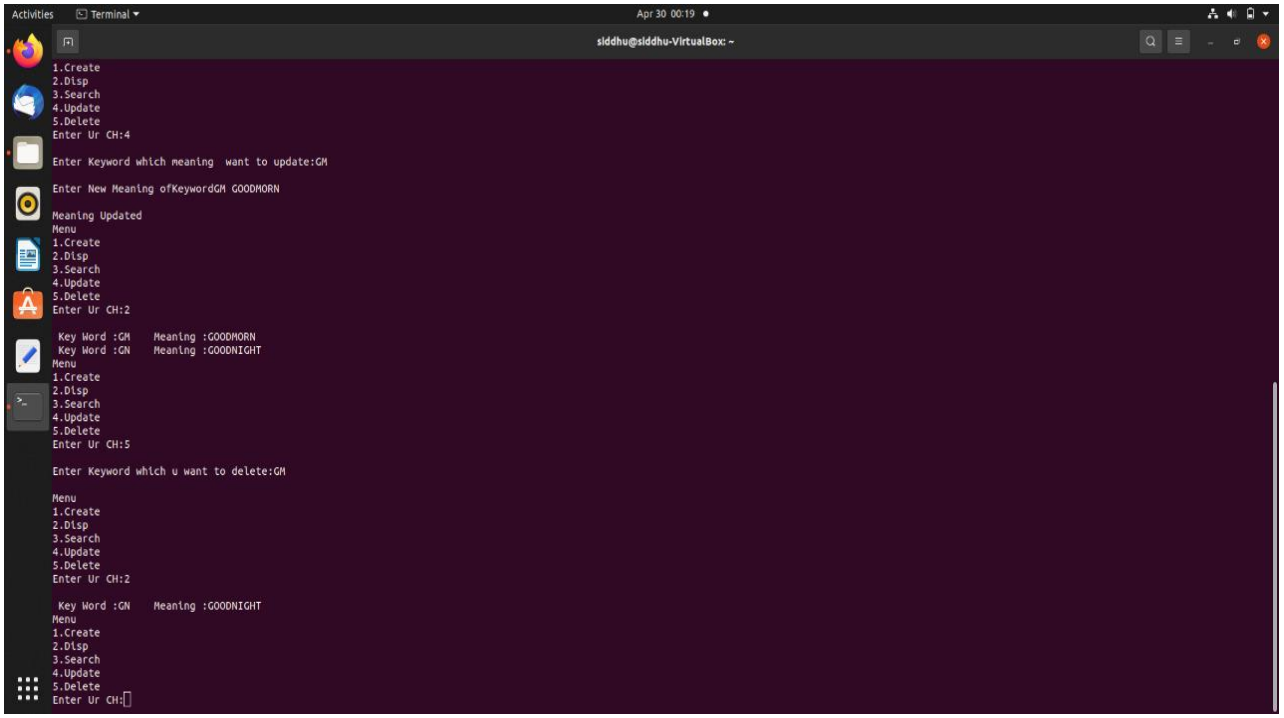
```
 while(q->left != NULL)
 {
  q = q->left;
 }
 return q;
}
int main()
{
 int ch;
 dict d;
 d.root = NULL;
 do
 {
  cout<<"\nMenu\n1.Create\n2.Disp\n3.Search\n4.Update\n5.Delete\nEnter Ur CH:";
  cin>>ch;

  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;

}
```

Output-




Activities
Terminal
Apr 30 00:19
siddhu@siddhu-VirtualBox: ~
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:4
Enter Keyword which meaning want to update:GM
Enter New Meaning ofKeywordGM GOODMORN
Meaning Updated
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:2
Key Word :GM Meaning :GOODMORN
Key Word :GN Meaning :GOODNIGHT
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:5
Enter Keyword which u want to delete:GM
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:2
Key Word :GN Meaning :GOODNIGHT
Menu
1.Create
2.Disp
3.Search
4.Update
5.Delete
Enter Ur CH:

Experiment No.- 06

There are flight paths between cities. If there is a flight between city A and city B then there is an t edge between the cities. The cost of the edge can be the time that flight. takes to reach city B from A or the amount of fuel used for the journey. Represent this as a graph. The node can be represented by airport name or name of the city. Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Check whether the graph is connected or not. Justify the storage representation used.

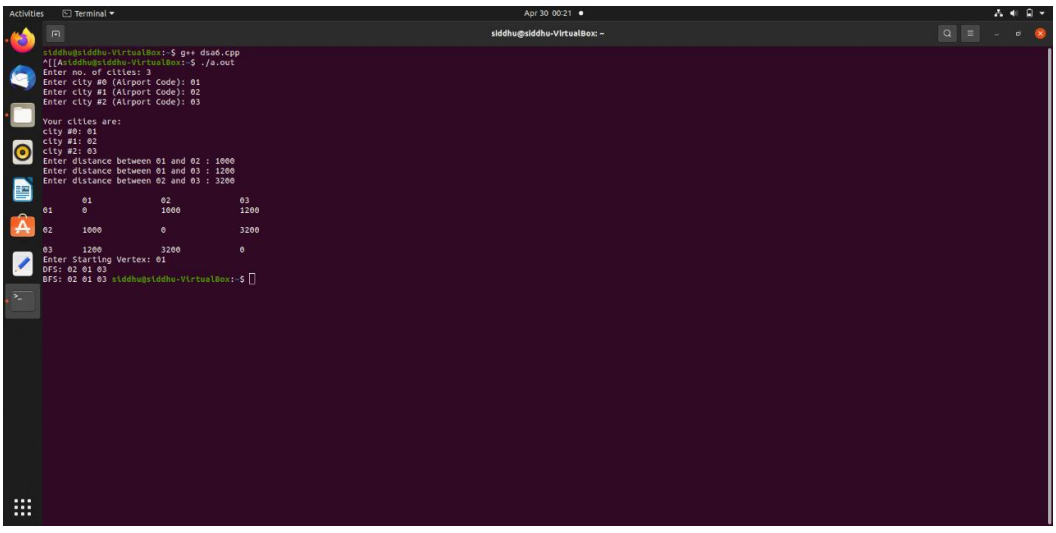
Program-

```
#include <iostream>
#include <queue>
using namespace std;

int adj_mat[50][50] = {0, 0};
int visited[50] = {0};

void dfs(int s, int n, string arr[])
{
   visited[s] = 1;
   cout << arr[s] << " ";
   for (int i = 0; i < n; i++)
   {
      if (adj_mat[s][i] && !visited[i])
         dfs(i, n, arr);
   }
}

void bfs(int s, int n, string arr[])
{
   bool visited[n];
   for (int i = 0; i < n; i++)
      visited[i] = false;
   int v;
   queue<int> bfsq;
   if (!visited[s])
   {
      cout << arr[s] << " ";
      bfsq.push(s);
```

```
        visited[s] = true;
        while (!bfsq.empty())
        {
            v = bfsq.front();
            for (int i = 0; i < n; i++)
            {
                if (adj_mat[v][i] && !visited[i])
                {
                    cout << arr[i] << " ";
                    visited[i] = true;
                    bfsq.push(i);
                }
            }
            bfsq.pop();
        }
    }
}

int main()
{
    cout << "Enter no. of cities: ";
    int n, u;
    cin >> n;
    string cities[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Enter city #" << i << " (Airport Code): ";
        cin >> cities[i];
    }

    cout << "\nYour cities are: " << endl;
    for (int i = 0; i < n; i++)
        cout << "city #" << i << ": " << cities[i] << endl;
    for (int i = 0; i < n; i++)
    {
```

```
        for (int j = i + 1; j < n; j++)
        {
            cout << "Enter distance between " << cities[i] << " and " << cities[j] << " : ";
            cin >> adj_mat[i][j];
            adj_mat[j][i] = adj_mat[i][j];
        }
    }
    cout << endl;
    for (int i = 0; i < n; i++)
        cout << "\t" << cities[i] << "\t";
    for (int i = 0; i < n; i++)
    {
        cout << "\n"
             << cities[i];
        for (int j = 0; j < n; j++)
            cout << "\t" << adj_mat[i][j] << "\t";
        cout << endl;
    }
    cout << "Enter Starting Vertex: ";
    cin >> u;
    cout << "DFS: ";
    dfs(u, n, cities);
    cout << endl;
    cout << "BFS: ";
    bfs(u, n, cities);
    return 0;
}
```

Output-



Experiment No.- 07

A Dictionary stores keywords & it meanings. Provide facility for adding new keywords, deleting keywords, updating values of any. entry. Provide facility to display whole data sorted in ascending/Descending 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.

Program-

```
#include<iostream>
#include<string.h>
using namespace std;
class dict
{
   dict *root,*node,*left,*right,*tree1;
   string s1,s2;
   int flag,flag1,flag2,flag3,cmp;
public:
   dict()
   {
      flag=0,flag1=0,flag2=0,flag3=0,cmp=0;
      root=NULL;
   }
   void input();
   void create_root(dict*,dict*);
   void check_same(dict*,dict*);
   void input_display();
   void display(dict*);
   void input_remove();
   dict* remove(dict*,string);
   dict* findmin(dict*);
   void input_find();
   dict* find(dict*,string);
   void input_update();
   dict* update(dict*,string);

};

                    void dict::input()
```

```
{
        node=new dict;
        cout<<"\nEnter the keyword:\n";
        cin>>node->s1;
        cout<<"Enter the meaning of the keyword:\n";
        cin.ignore();
        getline(cin,node->s2);
        create_root(root,node);
}


        void dict::create_root(dict *tree,dict *node1)
        {
           int i=0,result;
           char a[20],b[20];
           if(root==NULL)
           {
              root=new dict;
              root=node1;
              root->left=NULL;
              root->right=NULL;
              cout<<"\nRoot node created successfully"<<endl;
              return;
           }
           for(i=0;node1->s1[i]!='\0';i++)
           {
                 a[i]=node1->s1[i];
           }
           for(i=0;tree->s1[i]!='\0';i++)
           {
                 b[i]=tree->s1[i];
           }
           result=strcmp(b,a);
           check_same(tree,node1);
           if(flag==1)
```

```
        {
            cout<<"The word you entered already exists.\n";
            flag=0;
        }
        else
        {
    if(result>0)
    {
        if(tree->left!=NULL)
        {
            create_root(tree->left,node1);
        }
        else
        {
            tree->left=node1;
            (tree->left)->left=NULL;
    (tree->left)->right=NULL;
cout<<"Node added to left of "<<tree->s1<<"\n";
return;
      }
    }
    else if(result<0)
    {
 if(tree->right!=NULL)
 {
    create_root(tree->right,node1);
 }
 else
 {
    tree->right=node1;
    (tree->right)->left=NULL;
    (tree->right)->right=NULL;
    cout<<"Node added to right of "<<tree->s1<<"\n";
    return;
 }
```

```
                                        }
                                            }
                                        }


void dict::check_same(dict *tree,dict *node1)
{
        if(tree->s1==node1->s1)
        {
                flag=1;
                return;
        }
        else if(tree->s1>node1->s1)
    {
         if(tree->left!=NULL)
        {
         check_same(tree->left,node1);
        }
    }
    else if(tree->s1<node1->s1)
    {
         if(tree->right!=NULL)
         {
         check_same(tree->right,node1);
         }
    }
}


                void dict::input_display()
                {
                        if(root!=NULL)
                        {
                        cout<<"The words entered in the dictionary are:\n\n";
                        display(root);
```

```
                    }
                    else
                    {
                cout<<"\nThere are no words in the dictionary.\n";
                    }
            }


                        void dict::display(dict *tree)
                        {
                            if(tree->left==NULL&&tree->right==NULL)
                            {
                                cout<<tree->s1<<" = "<<tree->s2<<"\n\n";
                            }
                            else
                            {
                        if(tree->left!=NULL)
                        {
                            display(tree->left);
                        }
                        cout<<tree->s1<<" = "<<tree->s2<<"\n\n";
                        if(tree->right!=NULL)
                        {
                            display(tree->right);
                        }
                            }
                        }


void dict::input_remove()
{
        char t;
        if(root!=NULL)
        {
          cout<<"\nEnter a keyword to be deleted:\n";
```

```
            cin>>s1;
            remove(root,s1);
            if(flag1==0)
            {
                cout<<"\nThe word '"<<s1<<"' has been deleted.\n";
            }
            flag1=0;
          }
          else
          {
                cout<<"\nThere are no words in the dictionary.\n";
          }
}


                dict* dict::remove(dict *tree,string s3)
                {
                        dict *temp;
                   if(tree==NULL)
                   {
                        cout<<"\nWord not found.\n";
                        flag1=1;
                        return tree;
                   }
                   else if(tree->s1>s3)
                   {
                        tree->left=remove(tree->left,s3);
                        return tree;
                   }
                   else if(tree->s1<s3)
                          {
                        tree->right=remove(tree->right,s3);
                        return tree;
                   }
                   else
```

```
    {
        if(tree->left==NULL&&tree->right==NULL)
        {
            delete tree;
            tree=NULL;
        }
        else if(tree->left==NULL)
        {
            temp=tree;
            tree=tree->right;
            delete temp;
        }
        else if(tree->right==NULL)
        {
            temp=tree;
            tree=tree->left;
            delete temp;
        }
        else
        {
            temp=findmin(tree->right);
            tree=temp;
            tree->right=remove(tree->right,temp->s1);
        }
}
return tree;
}


            dict* dict::findmin(dict *tree)
            {
                while(tree->left!=NULL)
                {
                    tree=tree->left;
                }
```

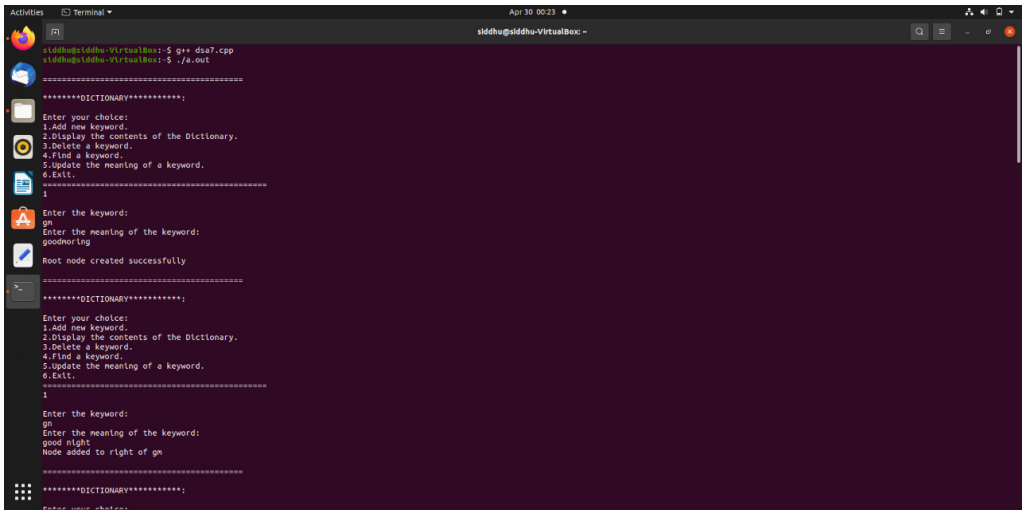
```
                return tree;
        }


void dict::input_find()
{
        flag2=0,cmp=0;
        if(root!=NULL)
        {
        cout<<"\nEnter the keyword to be searched:\n";
        cin>>s1;
   find(root,s1);
   if(flag2==0)
   {
                cout<<"Number of comparisons needed: "<<cmp<<"\n";
                cmp=0;
   }
        }
        else
        {
                cout<<"\nThere are no words in the dictionary.\n";
        }
}


                dict* dict::find(dict *tree,string s3)
                {
                        if(tree==NULL)
                        {
                                cout<<"\nWord not found.\n";
                                flag2=1;
                                flag3=1;
                                cmp=0;
                        }
                        else
```

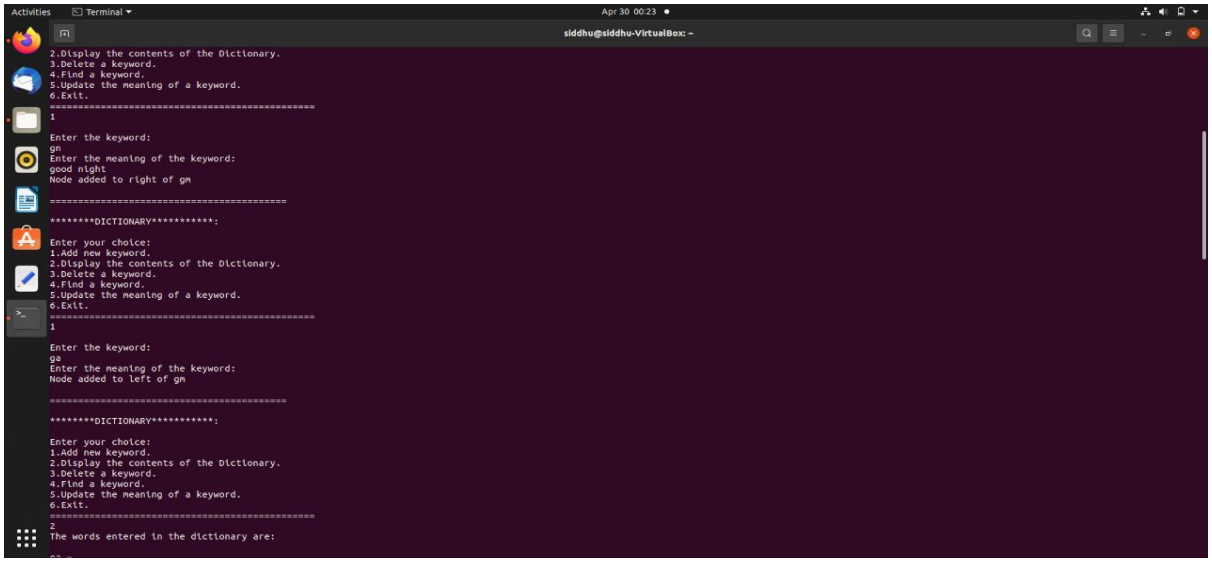
```
                {
                        if(tree->s1==s3)
                        {
                                cmp++;
                                cout<<"\nWord found.\n";
                                cout<<tree->s1<<": "<<tree->s2<<"\n";
                                tree1=tree;
                                return tree;
                        }
                        else if(tree->s1>s3)
                        {
                                cmp++;
                                find(tree->left,s3);
                        }
                        else if(tree->s1<s3)
                        {
                                cmp++;
                                find(tree->right,s3);
                        }
                }
                return tree;
        }


void dict::input_update()
{
        if(root!=NULL)
        {
        cout<<"\nEnter the keyword to be updated:\n";
        cin>>s1;
   update(root,s1);
        }
        else
        {
                cout<<"\nThere are no words in the dictionary.\n";
```

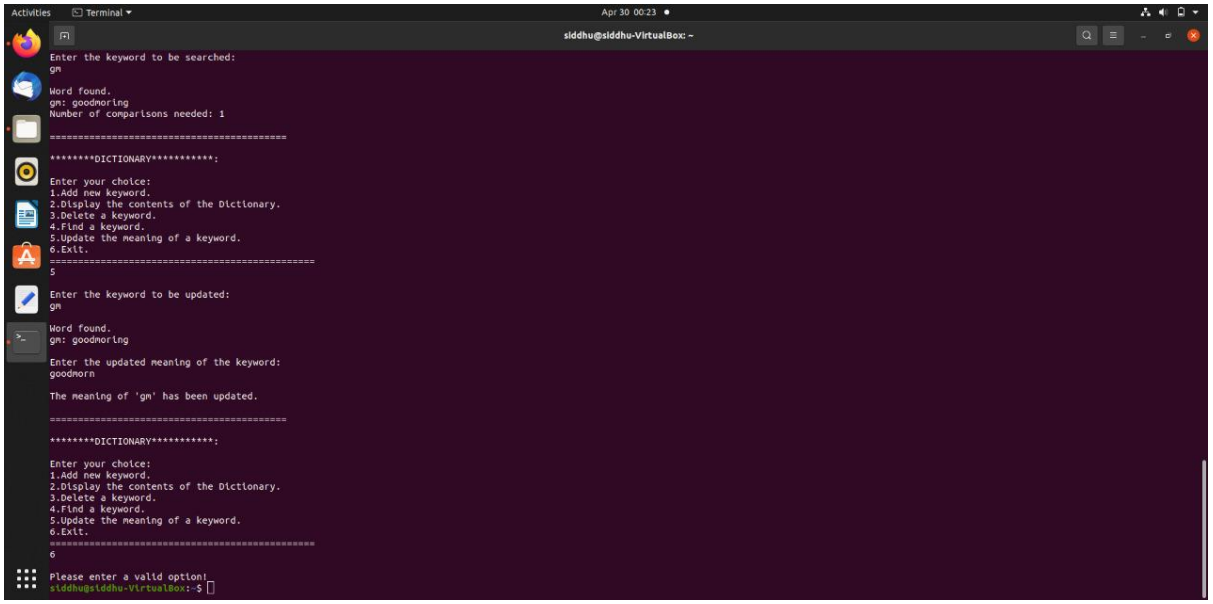
```
        }
}


dict* dict::update(dict *tree,string s3)
{
        flag3=0;
        find(tree,s3);
        if(flag3==0)
        {
   cout<<"\nEnter the updated meaning of the keyword:\n";
   cin.ignore();
   getline(cin,tree1->s2);
   cout<<"\nThe meaning of '"<<s3<<"' has been updated.\n";
        }
   return tree;
}


int main()
 {
   int ch;
   dict d;
   do
   {

cout<<"\n=========================================\n"
                "\n********DICTIONARY***********:\n"
                "\nEnter your choice:\n"
               "1.Add new keyword.\n"
               "2.Display the contents of the Dictionary.\n"
                "3.Delete a keyword.\n"
                "4.Find a keyword.\n"
```

```
                                        "5.Update the meaning of a keyword.\n"
                                        "6.Exit.\n"

"================================================\n";
                                    cin>>ch;
                                    switch(ch)
                                    {
                                       case 1:d.input();
                                              break;
                                       case 2:d.input_display();
                                                 break;
                                       case 3:d.input_remove();
                                              break;
                                       case 4:d.input_find();
                                              break;
                                       case 5:d.input_update();
                                                 break;
                                       default:cout<<"\nPlease enter a valid option!\n";
                                                  break;
                                    }
                                     }while(ch!=6);
                                    return 0;
                                    }
```

Output-




Activities
Terminal
Apr 30 00:23
siddhu@siddhu-VirtualBox: ~
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
1
Enter the keyword:
gm
Enter the meaning of the keyword:
good night
Node added to right of gm
==========
********DICTIONARY**********:
Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
1
Enter the keyword:
ga
Enter the meaning of the keyword:
Node added to left of gm
==========
********DICTIONARY**********:
Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
2
The words entered in the dictionary are:


Activities
Terminal
Apr 30 00:23
siddhu@siddhu-VirtualBox: ~
Enter the keyword to be searched:
gm
Word found.
gm: goodmorning
Number of comparisons needed: 1
==========
********DICTIONARY**********:
Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
5
Enter the keyword to be updated:
gm
Word found.
gm: goodmorning
Enter the updated meaning of the keyword:
goodnorn
The meaning of 'gm' has been updated.
==========
********DICTIONARY**********:
Enter your choice:
1.Add new keyword.
2.Display the contents of the Dictionary.
3.Delete a keyword.
4.Find a keyword.
5.Update the meaning of a keyword.
6.Exit.
==========
6
Please enter a valid option!
siddhu@siddhu-VirtualBox:~$

Experiment No.- 08

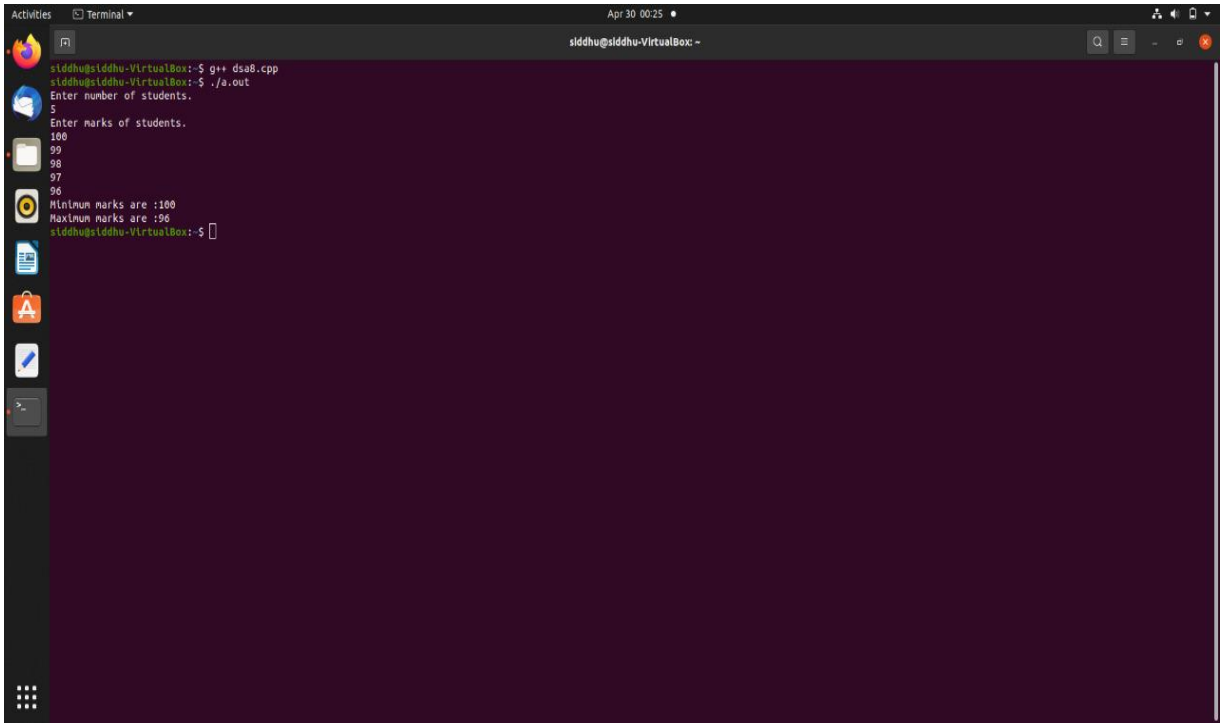
Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject. Use heap data structure. Analyze the algorithm.

Program-

```
#include<iostream>
using namespace std;
class Heap
{
        int n;
        int *minheap,*maxheap;
        public:
        void get();
        void displayMin(){cout<<"Minimum marks are :"<<minheap[0]<<endl;}
        void displayMax(){cout<<"Maximum marks are :"<<maxheap[0]<<endl;}
        void upAdjust(bool,int);
};
void Heap::get()
{
        cout<<"Enter number of students."<<endl;
        cin>>n;
        int k;
        minheap=new int[n];
        maxheap=new int[n];
        cout<<"Enter marks of students."<<endl;
        for(int i=0;i<n;i++)
        {
                cin>>k;
                minheap[i]=k;
                upAdjust(0,i);
                maxheap[i]=k;
                upAdjust(1,i);
        }
}
void Heap::upAdjust(bool m,int l)
{
```

```
        int s;
        if(!m)
        {
                while(minheap[(l-1)/2]<minheap[l])
                {
                        s=minheap[l];
                        minheap[l]=minheap[(l-1)/2];
                        minheap[(l-1)/2]=s;
                        l=(l-1)/2;
                        if(l==-1)
                                break;
                }
        }
        else
        {
                while(maxheap[(l-1)/2]>maxheap[l])
                {
                        s=maxheap[l];
                        maxheap[l]=maxheap[(l-1)/2];
                        maxheap[(l-1)/2]=s;
                        l=(l-1)/2;
                        if(l==-1)
                                break;
                }
        }
}
 int main()
{
        Heap H;
        H.get();
        H.displayMin();
        H.displayMax();
        return(0);
}
```

Output-


Activities
Terminal
Apr 30 00:25
siddhu@siddhu-VirtualBox: ~
siddhu@siddhu-VirtualBox:~$ g++ dsa8.cpp
siddhu@siddhu-VirtualBox:~$ ./a.out
Enter number of students.
5
Enter marks of students.
100
99
98
97
96
Minimum marks are :100
Maximum marks are :96
siddhu@siddhu-VirtualBox:~$