ASSIGNMENT NO1

'''Problem Statement : Consider telephone book database of N ,

Clients make use of a hash table implementation to quickly look up clients tele number .

Make use of a two collsion handling techniques and compare them using number of comparison required .

Find set of telephone number '''

```
#Function to calculate the address of the element according to the hash function
def HashFunction(name,tele,N):
    return tele%N



#Function to display the hash table
def display(table):
    for i in range(len(table)):
        print(i,end =" ")

        for j in table[i]:
            print("-->", end = " ")
            print(j, end = " ")

        print()



def main():

    size = int(input("Enter the size of the database you want to store: "))
```

```
    print("_"*50)
    print()

    HashTable =[[]for _ in range (size)]

    for i in range (size):

        print("Enter the name of the : ",i+1," Person : ",end = " ")
        Name=input()
        print()
        print("Enter the telephone number for ", Name,":",end =" ")
        Telephone_No=int(input())
        print()

        #Adding the values to the hash table and sorting out the collision
        address=HashFunction(Name,Telephone_No,size)
        Details=" [ Name:"+Name+" , "+"Tele_No:"+str(Telephone_No)+" ]"

        HashTable[address].append(Details)
    print("_"*50)
    print()
    display(HashTable)

if __name__ == '__main__':
  main()
```

ASSIGNMENT NO2

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.

```
#include <iostream>
#include <limits>
#include <string>
#include <vector>
#define clear_buffer
cin.ignore(std::numeric_limits<std::streamsize>::max(),'\n');

using std::cout;
using std::cin;
using std::endl;
using std::string;
using std::vector;

namespace mine
{
   class node
   {
   public:
      string data;
      vector<node*> children;
      int chapter_counter;
      node()
      {
         data = "";
```

```
        chapter_counter = 0;
    }
    node(const string& data)
    {
        this->data = data;
        this->chapter_counter = 0;
    }
};

class general_tree
{
private:
    friend class node;
    node* root_ptr;
    node* temp_ptr;
public:
    general_tree();
    void create_tree();
    void insert_chapter(const int&);
    void insert_section(const int&, const string&);
    void insert_subsection(const int&, const int&, const string&);
    void display();
};
}

int main()
{
    using mine::general_tree;
```

```
    general_tree *book = new general_tree();
    book->create_tree();
    book->display();

    return 0;
}

mine::general_tree::general_tree()
{
    this->root_ptr = new node();
    this->temp_ptr = new node();
}

void mine::general_tree::create_tree()
{
    cout << "Creating tree." << endl;
    int chapter_counter;

    cout << "\nEnter the name of Book: ";
    std::getline(cin, this->root_ptr->data);
    cout << endl;

    cout << "How many chapters are there?: ";
    cin >> chapter_counter;
    cout << endl;

    clear_buffer;
```

```
    this->insert_chapter(chapter_counter);
}

void mine::general_tree::insert_chapter(const int& chapter_counter)
{
    for (int i = 0; i < chapter_counter; i++)
    {
        // chapter creation and pushing
        string chapter_name;
        cout << "Enter the name of chapter " << (i+1) << ": ";
        std::getline(cin, chapter_name);
        cout << endl;

        node* chapter = new node(chapter_name);
        this->root_ptr->children.push_back(chapter);

        this->insert_section(i, chapter_name);
    }
}

void mine::general_tree::insert_section(const int& i, const string&
chapter_name)
{
    // section creation corrwsponding to chapter[i]
    int section_count;
    cout << "How many sections are there in chapter " << chapter_name <<
": ";
    cin >> section_count;
    cout << endl;
```

```
    clear_buffer;

    // appending sections to chapter[i]
    for (int c = 0; c < section_count; c++)
    {
        string section_name;
        cout << "Enter the name of section " << (c+1) << ": ";
        std::getline(cin, section_name);
        cout << endl;

        node* section = new node(section_name);
        this->root_ptr->children[i]->children.push_back(section);

        this->insert_subsection(i, c, section_name);
    }
}

void mine::general_tree::insert_subsection(const int& i, const int& c, const
string& section_name)
{
    // sub-section creation corresponding to section[i]
    int sub_section_count = 0;
    cout << "How many sub-sections are there in section " << section_name
<< ": ";
    cin >> sub_section_count;
    cout << endl;
    clear_buffer

    // appending sub_section to section[i]
```

```
    for (int s = 0; s < sub_section_count; s++)
    {
        string sub_section_name;
        cout << "Enter the name of sub-section " << (s+1) << ": ";
        std::getline(cin, sub_section_name);
        cout << endl;

        node* sub_section = new node(sub_section_name);
        this->root_ptr->children[i]->children[c]-
>children.push_back(sub_section);
    }
}

void mine::general_tree::display()
{
    cout << "Book name: " << this->root_ptr->data << endl;
    for (auto chapter : this->root_ptr->children)
    {
        cout << "\tChapter name: " << chapter->data << endl;
        for (auto section : chapter->children)
        {
            cout << "\t\tSection name: " << section->data << endl;
            for (auto sub_section : section->children)
            {
                cout << "\t\t\tSub Section name: " << sub_section->data << endl;
            }
        }
    }
```

}

ASSIGNMENT NO 3

There are flight paths between cities. If there is a flight between city A and city B then there is an edge between the cities. The cost of the edge can be the time that flight take 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. Justify the storage representation used.

```
#include <iostream>
#include <string>
using std::cout;
using std::cin;
using std::string;
using std::endl;
const int max_cities = 10;
namespace mine
{
   class node
   {
      public:
         string air_name;
         int time;
         node* next;
      node()
      {
         this->air_name = "" ;
         this->time = -1 ;
         this->next = nullptr;
```

```
    }
    node(const string& airport)
    {
        this->air_name = airport;
        this->next = nullptr;
    }
    node(const string& airport,const int& mins)
    {
        this->air_name = airport;
        this->time=mins;
        this->next = nullptr;
    }
};
class Graf
{
    private:
        int matrix[max_cities][max_cities];
        string city[max_cities];
        node *head_ptr[max_cities];
        node* temp;
        int n_cities;
    public:
        Graf()
        {
            for(int i = 0 ; i < max_cities; i ++ )
            {
            for(int j = 0 ; j< max_cities ; j ++ )
            {
```

```
                if(i==j)
                    this->matrix[i][j] = -1;
            }
            }
        }
      void input_graph();
      void list_initialiser();
      void display_matrix();
      void display_list();
   };
}

int  main()
{
   using namespace mine;
   Graf *graf = new Graf();

   int choice;
   cout<<"Welcome"<<endl;
   int t = 0 ;
   while(t<7)
   {
   cout<<"Enter your Choice:\n1)Enter the Graph.\n2)Display Adjacency
Matrix.\n3)Display Adjecency List.\n4)Exit "<<endl;
   cin>>choice;
   switch (choice)
   {
   case 1:
```

```
        graf->input_graph();
        break;
    case 2:
        graf->display_matrix();
        break;
    case 3:
        graf->display_list();
        break;
    case 4:
        exit(0);
    default:
        cout<<"WOW"<<endl;
        t--;
        break;
    }
    t++;
    }



    return 0;
}
void mine::Graf::input_graph()
{
    cout<<"Enter the Number of cities : ";
    cin>>this->n_cities;
    cout<<"Enter the name of the cities(with space): ";
    for ( int i = 0 ; i < n_cities ; i ++ )
    {
```

```
        cin>>this->city[i];
    }
    for( int i = 0 ; i <n_cities ; i ++ )
    {
        for ( int j = 0; j<n_cities ; j++)
        {

            if ( i==j)
                continue;
            else
            {   char choice;
                cout<<"Is there a path between : "<<city[i]<<"-> "<<city[j]<<"
Enter y or n : ";
                cin>>choice;
                if(choice =='y')
                {
                    cout<<"How much time does it take ? :";
                    cin>>matrix[i][j];
                }
                else if(choice =='n')
                {
                    matrix[i][j]=0;
                }
                else
                {
                    cout<<"Invalid input"<<endl;
                }
```

```
        }
      }
   }
   this->list_initialiser();
}

void mine::Graf::list_initialiser()
{
   for(int i = 0 ; i < n_cities ; i++)
   {
      this->head_ptr[i] = new node(city[i]);
   }
   for ( int i = 0 ; i < n_cities ; i++)
   {
      for ( int j = 0 ; j < n_cities ; j++)
      {
         if(matrix[i][j]>0)
         {

            node *t = new node(city[j],matrix[i][j]);
            if(head_ptr[i]->next == NULL)
               head_ptr[i]->next=t;
            else
            {
               temp = head_ptr[i];
               while(temp->next !=NULL)
               {
                  temp= temp->next;
```

```
                }
                temp->next = t;
            }

        }
    }
}

void mine::Graf::display_matrix()
{
    cout<<"\n";
    for(int i = 0; i < n_cities ; i++)
    { cout<<"\t "<<city[i];}
    cout<<endl;
    for (int i = 0 ; i < n_cities;i++)
    {
        cout<<city[i];
        for(int j= 0 ; j < n_cities ; j++)
        {
        if(matrix[i][j]==-1)
        {
            cout<<"\t";
        }
        else
        {
```

```
            cout<<" \t";
        }
        cout<<matrix[i][j];

        }
        cout<<"\n";
    }
}

void mine::Graf::display_list()
{
    cout<<"Path and time taken to reach city \n";
    for(int i = 0 ; i< n_cities;i++)
    {
        if(this->head_ptr==NULL)
        {
            cout<<"Abey graph is MT ."<<endl;
        }
        else
        {


            for(temp = head_ptr[i]->next;temp !=NULL;temp=temp->next)
            {
                cout<<head_ptr[i]->air_name;
                cout<<"-> "<<temp->air_name<<"\t\t"<<"[ Time required :
"<<temp->time<<" ]"<<endl;
            }
```

```
      }
   }
}
```

ASSIGNMENT NO4

A Dictionary stores keywords & 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 Height balance tree and find the complexity

```
#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;
                        }
```

ASSIGNMENT NO5

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

```
#include<iostream>
#define SIZE 100
```

```
using namespace std;


int heap[SIZE];

void create(int heap[],int n);

void buildheapmax(int heap[],int i);

void create1(int heap[],int n);

void buildheapmin(int heap[],int i);


int main()


{

  int n,i,j,k;


cout<<"Enter the no of Students appeared For ADS online
examination"<<endl;

cin>>n;

heap[0]=n;

cout<<"\nEnter the Marks of the students"<<endl;

for(k=1;k<=n;k++)

 {

  cin>>heap[k];

 }

create(heap,n);

cout<<"\nDisplay max heap"<<endl;

for(k=0;k<=n;k++)

 {

   cout<<heap[k]<<"\t";

  }
```

```
cout<<"\nThe Maximum marks in the subject is ";
cout<<heap[1];
create1(heap,n);
cout<<"\nDisplay min heap"<<endl;
for(k=0;k<=n;k++)
 {
   cout<<heap[k]<<"\t";
  }

cout<<"\nThe Minimum marks in the subject is ";
cout<<heap[1];

return 0;
}

void create(int heap[],int n)
{
int i,k;

for(i=n/2;i>=1;i--)
{
    buildheapmax(heap,i);
}

}

void buildheapmax(int heap[],int i)
```

```
{



int j,temp,m;
int q=1;
m=heap[0];
while(2*i<=m&&q==1)
 {
  j=2*i;
  if(j+1<=m&&heap[j+1]>heap[j])
   {j=j+1;}

  if(heap[i]>heap[j])
    {q=0;}
  else
   {
    temp=heap[i];
    heap[i]=heap[j];
    heap[j]=temp;
     i=j;
    }

  }



}
void create1(int heap[],int n)
{
```

```
int i,k;

for(i=n/2;i>=1;i--)
{
    buildheapmin(heap,i);
}

}

void buildheapmin(int heap[],int i)
{

int j,temp,m;
int q=1;
m=heap[0];
while(2*i<=m&&q==1)
 {
  j=2*i;
  if(j+1<=m&&heap[j+1]<heap[j])
   {j=j+1;}

  if(heap[i]<heap[j])
    {q=0;}
  else
   {
    temp=heap[i];
    heap[i]=heap[j];
```

```
    heap[j]=temp;
     i=j;
    }

  }


}
```

ASSIGNMENT NO6

Department maintains a student information. The file contains roll number, name, division and address. Allow user to add, delete information of student. Display information of particular employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays the student details. Use sequential file to main the data.

```
#include<iostream>
#include<fstream>
#include<cstdio>
using namespace std;


class employee
{
    int admno;
    char name[50];
    char addr[50];
public:
    void setData()
    {
        cout << "\nEnter Roll NO : . ";
```

```
        cin >> admno;
        cout << "Enter name ";
        cin>>name;
        cout<<"enter the address of the student";
        cin>>addr;


    }

    void showData()
    {
        cout << "\n*Student Roll No  : " << admno;
        cout << "\n*Student Name : " << name;
        cout<<"\n*Address: "<<addr;
    }

    int retAdmno()
    {
        return admno;
    }
};


void write_record()
{
    ofstream outFile;
    outFile.open("employee.dat", ios::binary | ios::app);
```

```
    employee obj;
    obj.setData();

    outFile.write((char*)&obj, sizeof(obj));

    outFile.close();
}

void display()
{
    ifstream inFile;
    inFile.open("employee.dat", ios::binary);

    employee obj;

    while(inFile.read((char*)&obj, sizeof(obj)))
    {
        obj.showData();
    }

    inFile.close();
}

void search(int n)
{
    ifstream inFile;
    inFile.open("employee.dat", ios::binary);
```

```
    employee obj;

    while(inFile.read((char*)&obj, sizeof(obj)))
    {
        if(obj.retAdmno() == n)
        {
            obj.showData();
            break;
        }
    }

    inFile.close();
}

void delete_record(int n)
{
    employee obj;
    ifstream inFile;
    inFile.open("employee.dat", ios::binary);

    ofstream outFile;
    outFile.open("temp.dat", ios::out | ios::binary);

    while(inFile.read((char*)&obj, sizeof(obj)))
    {
        if(obj.retAdmno() != n)
        {
            outFile.write((char*)&obj, sizeof(obj));
```

```
        }
    }

    inFile.close();
    outFile.close();

    remove("employee.dat");
    rename("temp.dat", "employee.dat");
}

void modify_record(int n)
{
    fstream file;
    file.open("employee.dat",ios::in | ios::out);

    employee obj;

    while(file.read((char*)&obj, sizeof(obj)))
    {
        if(obj.retAdmno() == n)
        {
            cout << "\nEnter the new details of Student";
            obj.setData();

            int pos = -1 * sizeof(obj);
            file.seekp(pos, ios::cur);

            file.write((char*)&obj, sizeof(obj));
```

```
      }
   }

   file.close();
}

int main()
{
        int ch;
        do{
   cout<<"\n\n\n********************File
operations*************************
\n1.write\n2.display\n3.search\n4.delete\n5.modify";
   cout<<"\nEnter your choice";
   cin>>ch;
   switch(ch){

                case 1: cout<<"Enter number of records: ";
                int n;
                cin>>n;
                for(int i = 0; i <n; i++)
                  write_record();
                break;

                case 2:

                        cout << "\nList of records";
                        display();
                        break;
```

```
case 3:
    cout<<"Enter Student Roll No : ";
    int s;
    cin>>s;
    search(s);
    break;
case 4:
    cout<<"enter no to be deleted";
    int d;
    cin>>d;

delete_record(d);
cout << "\nRecord Deleted";
break;
case 5:



cout<<"enter rno to be modified";
int m;
cin>>m;



modify_record(m);
break;
case 6:
```

```
    return 0;
    }

}while(ch!=6);
}
```