ASSIGNMENT 1

AIM:

To create ADT that implement the "set" concept.

a. Add (new Element) -Place a value into the set
b. Remove (element)
c. Contains (element) Return true if element is in collection
d. Size () Return number of values in collection
e. Intersection of two sets
f. Union of two sets
g. Difference between two sets
h. Subset

PROGRAM:

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

void create(int set[])
{
int n;
cout<<"\n enter the size of set : ";
cin>>n;
cout<<"\n enter the elements in the set : ";
for(inti=1;i<=n;i++)
cin>>set[i];

set[0]=n;
}

bool member(int set[],intnum)
{
for(inti=1;i<=set[0];i++)
if(set[i]==num)
return true;

return false;
}
```

```
void intersection(int set1[],int set2[],int set3[])
{
for(inti=1;i<=set2[0];i++)
   {
if(member(set1,set2[i])== true)
      {
set3[0]++;
set3[set3[0]]=set2[i];
      }

   }
}


void union1(int set1[],int set2[],int set4[])
{

for(inti=0;i<=set1[0];i++)
set4[i]=set1[i];

for(inti=1;i<=set2[0];i++)
   {
if(member(set1,set2[i])== false)
      {
set4[0]++;
set4[set4[0]]=set2[i];
      }
   }
}

void difference1(int set1[],int set2[],int set5[])
{
for(inti=1;i<=set1[0];i++)
   {
if((member(set2,set1[i]) == false))
      {
set5[0]++;
set5[set5[0]]=set1[i];
      }
 ;   }
}
```

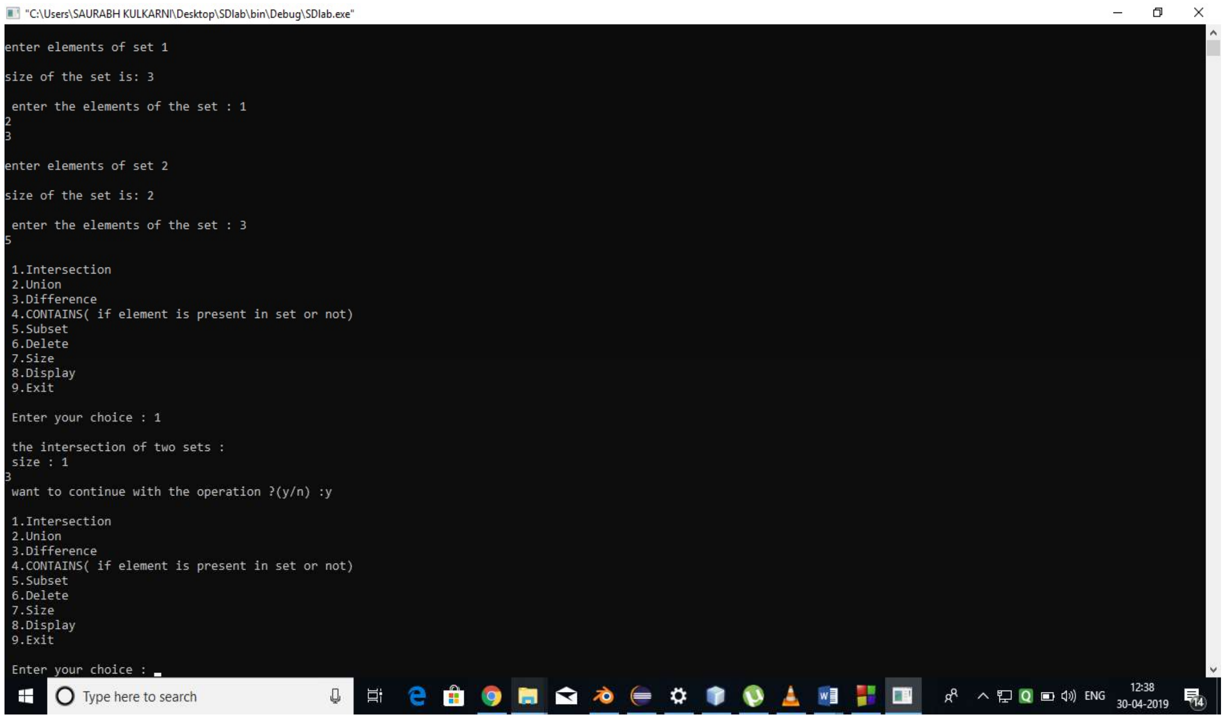
```
void contains(int set[])
{
intnum;
cout<<"\n enter the element to be searched ";
cin>>num;
if((member(set,num))== true)
cout<<"\n element is present ";
else
cout<<"\n element is not present ";
}

bool subset(int seta[],intsetb[])
{
for(inti=1;i<=setb[0];i++)
   {
if((member(seta,setb[i]))==true)
continue;
else
return false;
   }
return true;
}

void remove(int set[])
{
intpos;
cout<<"\n enter the position from which you want to remove the element : ";
cin>>pos;
if(pos<=set[0])
   {
if(pos<set[0])
      {
for(inti=pos;i<=set[0];i++)
         {
set[i]=set[i+1];
         }
set[0]--;
      }
else if(pos==set[0])
      {
set[0]--;
      }
   }
else
   {
cout<<"\n entered position exceeds the size of the set " ;
   }
}

void size(int set[])
{
cout<<set[0];

}


void display(int set[])
{
cout<<"\n size : "<<set[0]<<"\t";
for(inti=1;i<=set[0];i++)
   {
cout<<set[i]<<"    ";
   }
}



int main()
{
int set1[10];
cout<<"\n FOR SET 1 ";
create(set1);

int set2[10];
cout<<"\n FOR SET 2 ";
create(set2);

intch,c;
char choice;
do{
cout<<"\n\n -------------- OPERATION MENU ---------------- ";
cout<<"\n 1 for INTERSECTION ";
cout<<"\n 2 for UNION ";
cout<<"\n 3 for DIFFERENCE ";
cout<<"\n 4 for CONTAINS( if element is present in set or not)";
```

```
cout<<"\n 5 for SUBSET";
cout<<"\n 6 for REMOVE";
cout<<"\n 7 for SIZE";
cout<<"\n 8 for DISPLAY";
cout<<"\n 9 for EXIT";
cout<<"\n\n Enter your choice : ";
cin>>ch;
switch(ch)
{
case 1:
        {
int set3[1];
set3[0]=0;
cout<<"\n the intersection of two sets : \t";
intersection(set1,set2,set3);
display(set3);
break;
        }
case 2:
        {
int set4[set1[0]+1];
set4[0]=0;
cout<<"\n the union of two sets \t";
union1(set1,set2,set4);
display(set4);
break;
        }
case 3:
        {
int set5[1];
set5[0]=0;
cout<<"\n the difference of two sets \t";
difference1(set1,set2,set5);
display(set5);
break;
        }
case 4:
        {
cout<<"\n enter 1 for searching in set1 and 2 for searching in set2 ";
cin>>c;
switch(c)
            {
case 1: contains(set1); break;
case 2: contains(set2); break;
default: cout<<"\n wrong choice entered ";
        }

break;
      }

case 5:
      {
label:
cout<<"\n enter 1 for checking if set1 is subset of set2 else enter 2 ";
cin>>c;
switch(c)
        {
case 1:
           {
if(subset(set2,set1)==true )
cout<<"\n set1 is subset of set2";
else
cout<<"\n set1 is not a subset of set2";
break;
           }
case 2:
           {
if(subset(set1,set2)==true )
cout<<"\n set2 is subset of set1";
else
cout<<"\n set2 is not a subset of set1";
break;
           }
default:
cout<<"\n wrong choice entered ";
goto label;
        }
break;
      }
case 6:
      {
        label2:
cout<<"\n enter 1 for removing element from set 1 and 2 for removal from set 2 ";
cin>>c;
switch(c)
        {
```

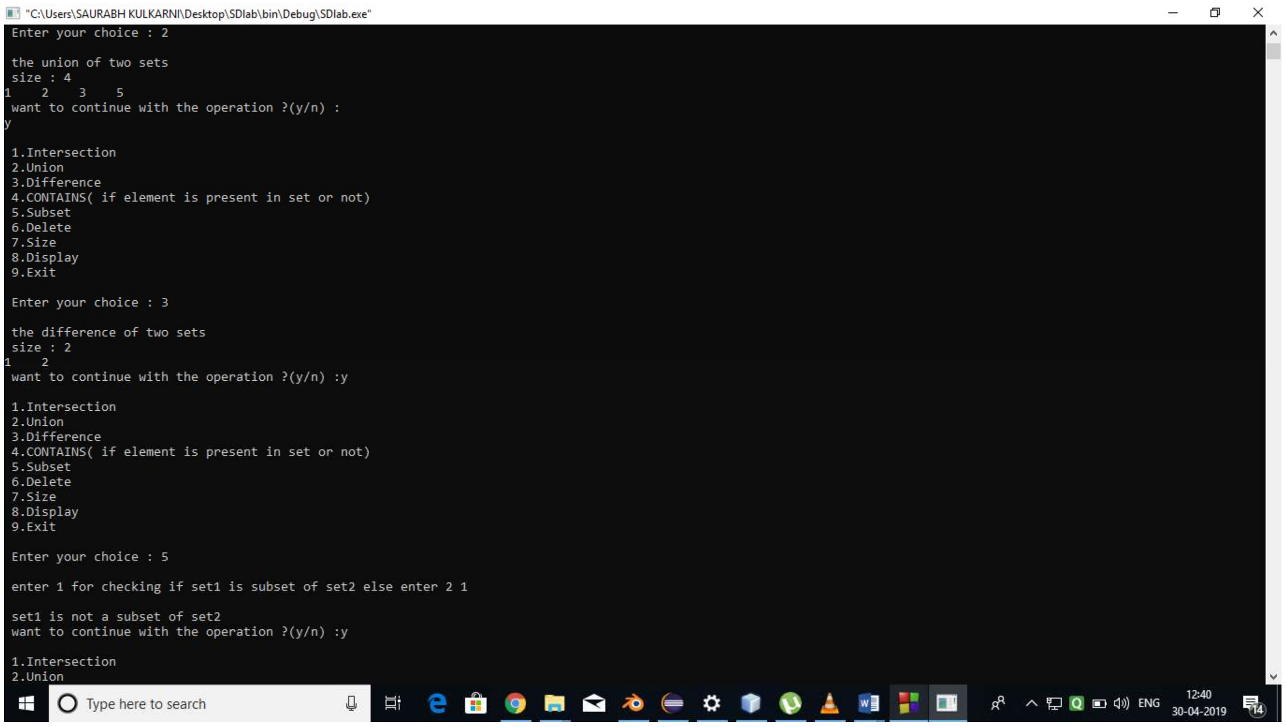
```
case 1:
            {
remove(set1);
break;
            }
case 2:
            {
remove(set2);
break;
            }
default:
cout<<"\n wrong choice entered ";
goto label2;
         }
break;
      }
case 7:
      {
cout<<"\n size of set 1 : ";
size(set1);
cout<<"\n size of set 2 : ";
size(set2);
break;
      }
case 8:
      {
display(set1);
display(set2);
break;
      }

case 9:
exit(0);
default:
cout<<"\n wrong choice entered ";
}

cout<<"\n want to continue with the operation ?(y/n) :";
cin>>choice;

}while((choice=='y')||(choice=='Y'));

return 0;
```

}

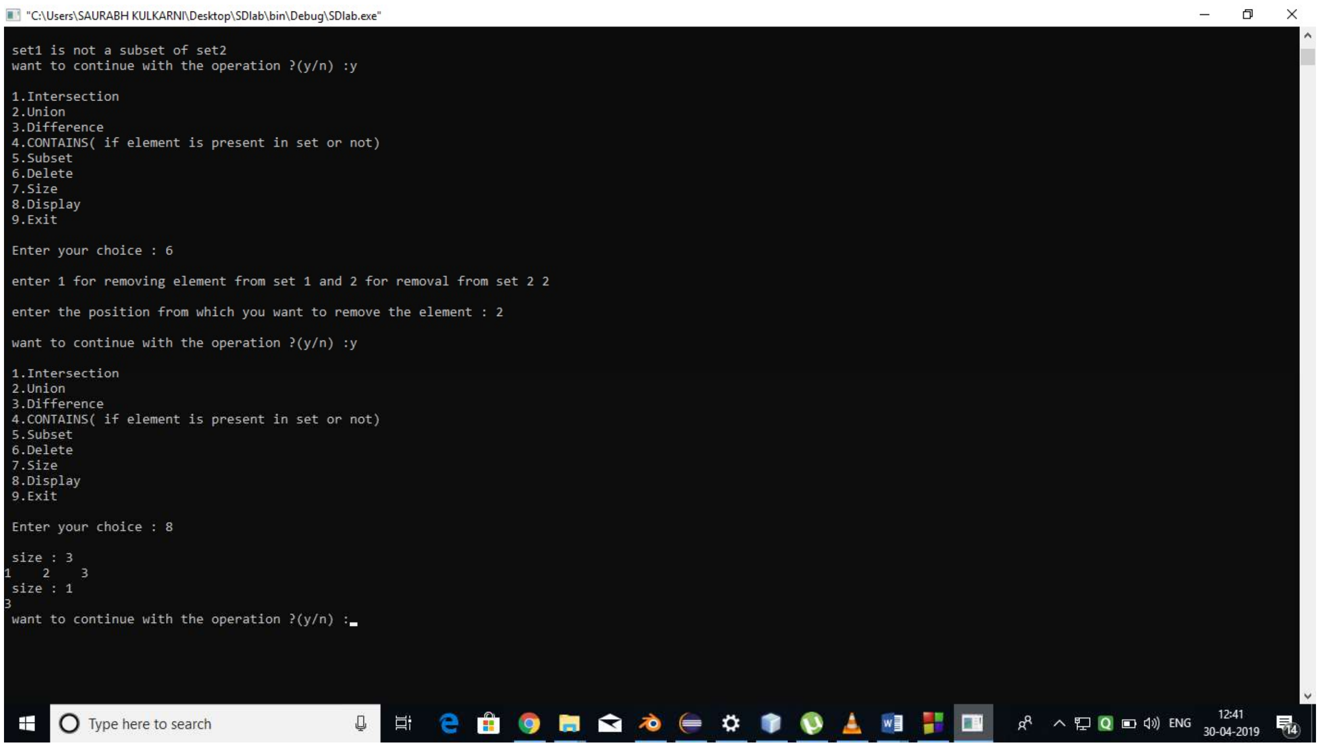
OUTPUT:



```
"C:\Users\SAURABH KULKARNI\Desktop\SDlab\bin\Debug\SDlab.exe"
enter elements of set 1

size of the set is: 3

 enter the elements of the set : 1
2
3

enter elements of set 2

size of the set is: 2

 enter the elements of the set : 3
5

 1.Intersection
 2.Union
 3.Difference
 4.CONTAINS( if element is present in set or not)
 5.Subset
 6.Delete
 7.Size
 8.Display
 9.Exit

 Enter your choice : 1

 the intersection of two sets :
 size : 1
3
 want to continue with the operation ?(y/n) :y

 1.Intersection
 2.Union
 3.Difference
 4.CONTAINS( if element is present in set or not)
 5.Subset
 6.Delete
 7.Size
 8.Display
 9.Exit

 Enter your choice : _
```



```
"C:\Users\SAURABH KULKARNI\Desktop\SDlab\bin\Debug\SDlab.exe"
 Enter your choice : 2

 the union of two sets
 size : 4
1    2    3    5
 want to continue with the operation ?(y/n) :
y

 1.Intersection
 2.Union
 3.Difference
 4.CONTAINS( if element is present in set or not)
 5.Subset
 6.Delete
 7.Size
 8.Display
 9.Exit

 Enter your choice : 3

 the difference of two sets
 size : 2
1    2
 want to continue with the operation ?(y/n) :y

 1.Intersection
 2.Union
 3.Difference
 4.CONTAINS( if element is present in set or not)
 5.Subset
 6.Delete
 7.Size
 8.Display
 9.Exit

 Enter your choice : 5

 enter 1 for checking if set1 is subset of set2 else enter 2 1

 set1 is not a subset of set2
 want to continue with the operation ?(y/n) :y

 1.Intersection
 2.Union
```



"C:\Users\SAURABH KULKARNI\Desktop\SDlab\bin\Debug\SDlab.exe"

```
 set1 is not a subset of set2
 want to continue with the operation ?(y/n) :y

 1.Intersection
 2.Union
 3.Difference
 4.CONTAINS( if element is present in set or not)
 5.Subset
 6.Delete
 7.Size
 8.Display
 9.Exit

 Enter your choice : 6

 enter 1 for removing element from set 1 and 2 for removal from set 2 2

 enter the position from which you want to remove the element : 2

 want to continue with the operation ?(y/n) :y

 1.Intersection
 2.Union
 3.Difference
 4.CONTAINS( if element is present in set or not)
 5.Subset
 6.Delete
 7.Size
 8.Display
 9.Exit

 Enter your choice : 8

 size : 3
1    2    3
 size : 1
3
 want to continue with the operation ?(y/n) :
```

Conclusion: Hence implemented the set operation on arrays.

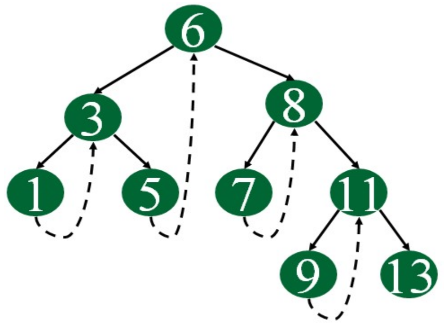
ASSIGNMENT 2

AIM:
Construct a threaded binary search tree by inserting values in the given order and traverse it in inorder traversal using threads.

THEORY:
Threaded Binary Tree

Inorder traversal of a Binary tree can either be done using recursion or with the use of a auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).
There are two types of threaded binary trees.
Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)
Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.
The threads are also useful for fast accessing ancestors of a node.

Following diagram shows an example Single Threaded Binary Tree. The dotted lines represent threads.



PROGRAM:

```
#include<iostream>
```

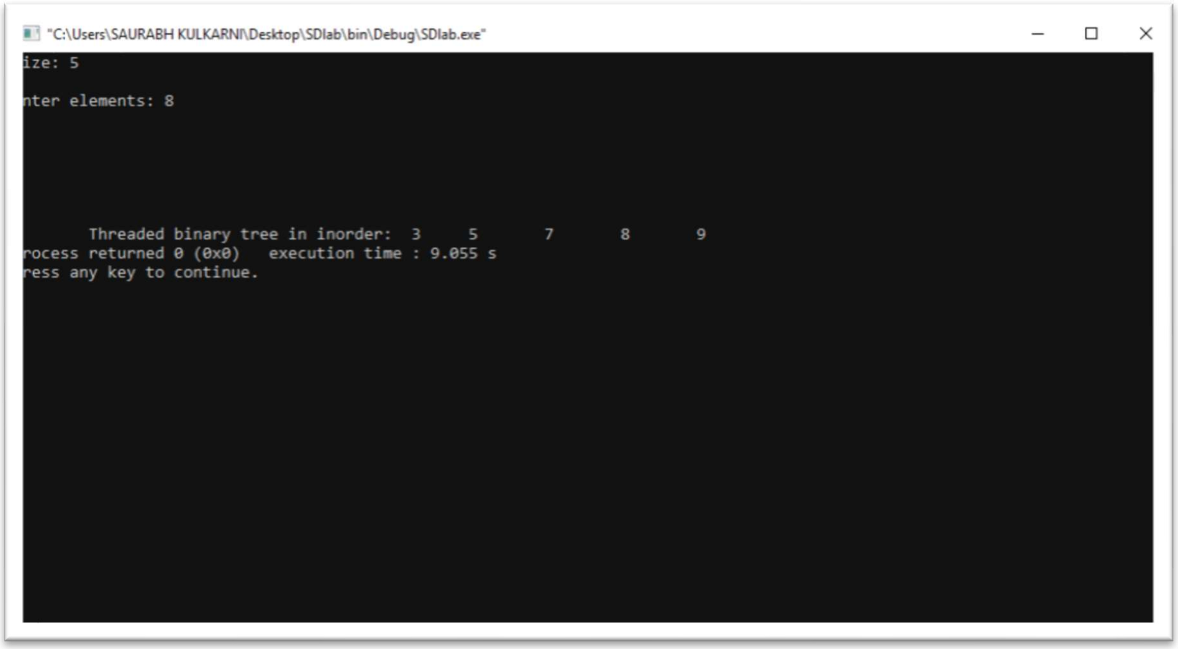
```
using namespace std;

class ttree
{
	private:
		struct thtree
		{
			int left;
			thtree *leftchild;
			int data;
			thtree *rightchild;
			int right;
		}*th_head;

	public:
		ttree();
		void insert(int num);
		void inorder();

};

ttree::ttree()
{
	th_head=NULL;
}
void ttree::insert(int num)
{


	thtree *head=th_head,*p,*z;

	z=new thtree;
	z->left=true;
	z->data=num;
	z->right=true;

	if(th_head==NULL)
	{
		head=new thtree;
```

```
        head->left=false;
        head->leftchild=z;
        head->data=-9999;
        head->rightchild=head;
        head->right=false;

        th_head=head;
        z->leftchild=head;
        z->rightchild=head;

    }
    else
    {
        p=head->leftchild;
        while(p!=head)
        {
            if(p->data > num)
            {
                if(p->left!=true)
                p=p->leftchild;
                else
                {
                    z->leftchild=p->leftchild;
                    p->leftchild=z;

                    p->left=false;
                    z->right=true;
                    z->rightchild=p;
                    return;
                }
            }
            else
            {
                if(p->data < num)
                {
                    if(p->right!=true)
                    p=p->rightchild;
                    else
                    {
```

```
                                                z->rightchild=p->rightchild;
                                                p->rightchild=z;

                                                p->right=false;
                                                z->left=true;
                                                z->leftchild=p;
                                                return;
                                        }
                                }
                        }
                }
        }
}
void ttree::inorder()
{
        thtree *a;
        a=th_head->leftchild;
        while(a!=th_head)
        {
                while(a->left==false)

                        a=a->leftchild;
                        cout<<a->data<<"\t";



                while(a->right==true)
                {
                        a=a->rightchild;

                        if(a==th_head)
                        break;

                        cout<<a->data<<"\t";
                }
                a=a->rightchild;
        }
}
```

```
int main()
{
        ttree th;
        int n,e;
        cout<<"Enter no. of elements: ";
        cin>>n;
        cout<<"\nEnter elements: ";
        for(int i=0;i<n;i++)
        {
                cin>>e;
                th.insert(e);
        }

        cout<<"\n\n\tThreaded binary tree in inorder:  ";
        th.inorder();
}
```

OUTPUT:



```
"C:\Users\SAURABH KULKARNI\Desktop\SDlab\bin\Debug\SDlab.exe"
ize: 5

nter elements: 8




        Threaded binary tree in inorder:  3     5       7       8       9
rocess returned 0 (0x0)   execution time : 9.055 s
ress any key to continue.
```

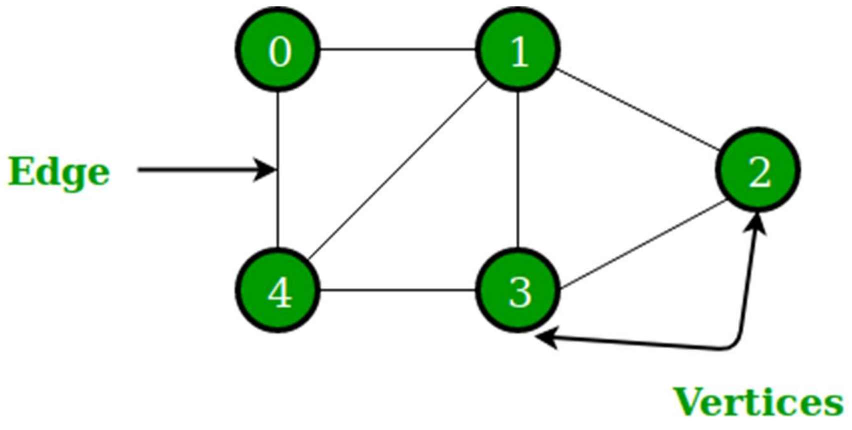
ASSIGNMENT 3

AIM: 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 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. Justify the storage representations used.

OBJECTIVE: To Construct an adjacency matrix in which time represent the cost of matrix of flights from city A to city B.

THEORY:

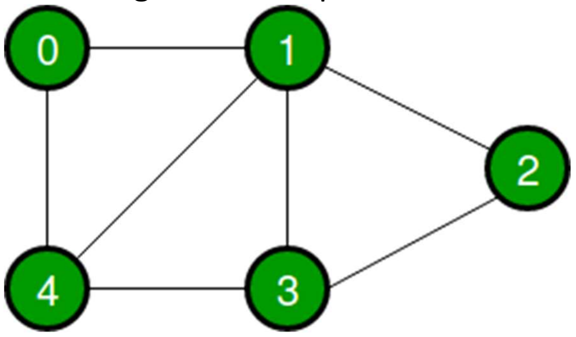
A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.



In the above Graph, the set of vertices V = {0,1,2,3,4} and the set of edges E = {01, 12, 23, 34, 04, 14, 13}.

Graphs are used to solve many real-life problems. Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender, locale etc.

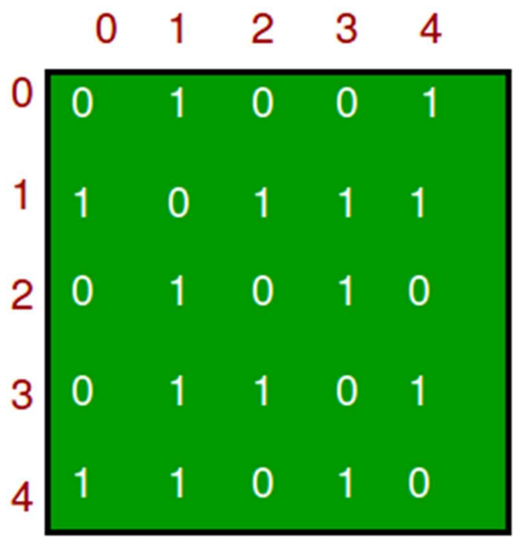
Graph is a data structure that consists of following two components:
1. A finite set of vertices also called as nodes.
2. A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.
Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network. Graphs are also used in social networks like linkedIn, Facebook. For example, in Facebook, each person is represented with a vertex(or node). Each node is a structure and contains information like person id, name, gender and locale. See this for more applications of graph.
Following is an example of an undirected graph with 5 vertices.



Following two are the most commonly used representations of a graph.
1. Adjacency Matrix

2. Adjacency List
There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.
Adjacency Matrix:
Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w.

The adjacency matrix for the above example graph is:



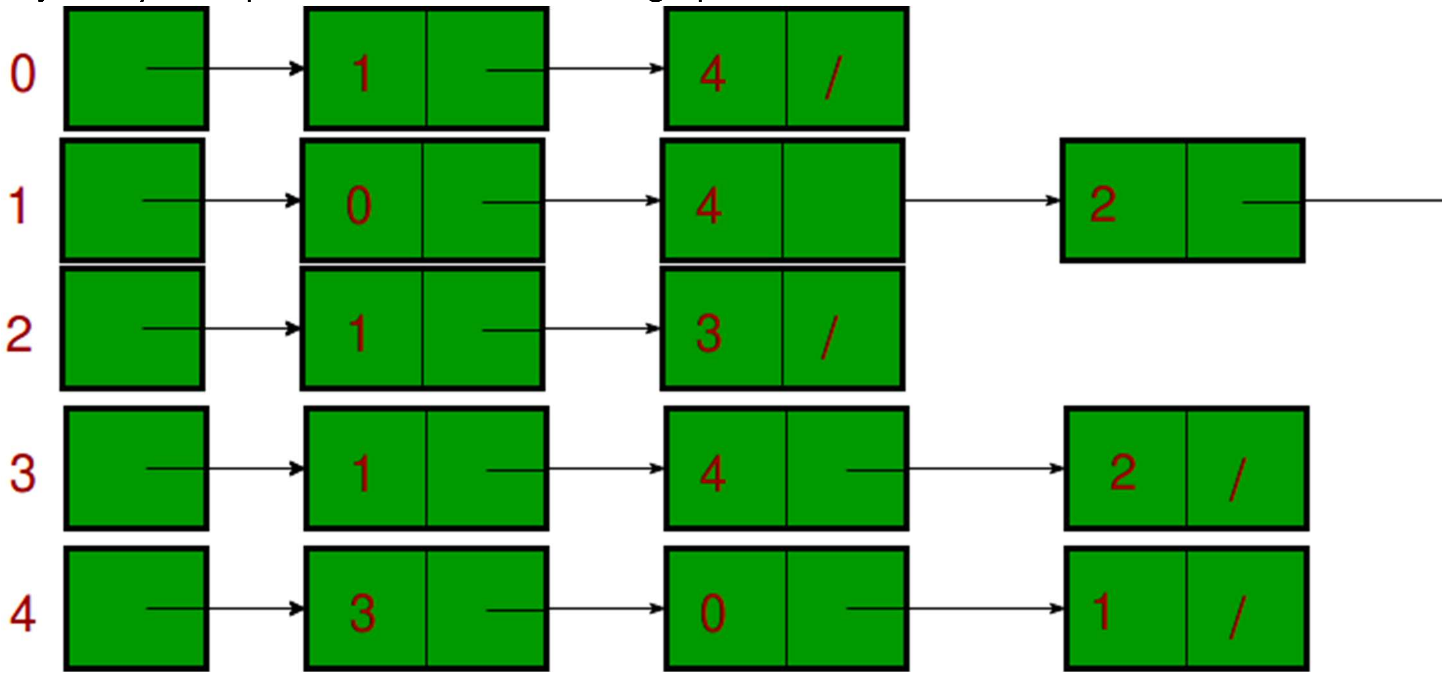
$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 2 & 0 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 & 0 & 1 \\ 4 & 1 & 1 & 0 & 1 & 0 \end{array}$$

Pros: Representation is easier to implement and follow. Removing an edge takes O(1) time. Queries like whether there is an edge from vertex ‘u’ to vertex ‘v’ are efficient and can be done O(1).

Cons: Consumes more space O(V^2). Even if the graph is sparse(contains less number of edges), it consumes the same space. Adding a vertex is O(V^2) time. Please see this for a sample Python implementation of adjacency matrix.

Adjacency List:

An array of lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the list of vertices adjacent to the **I** th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be represented as lists of pairs. Following is adjacency list representation of the above graph.



Recommended: Please solve it on "*PRACTIC*

PROGRAM:

#include<iostream>

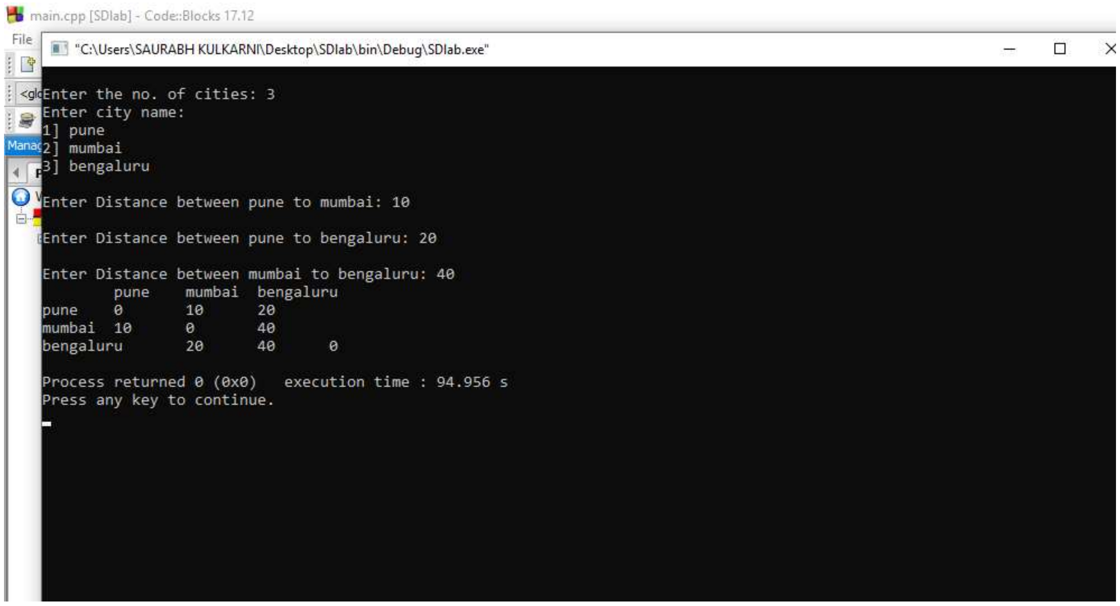
#define MAX 10

using namespace std;

```
class airport
{
        string city[MAX];
        int distance[10][10];
public :
   int n;
   airport();
   void read_city();
   void show_graph();
};
airport::airport()
{
        n=0;
        for(int i=0;i<MAX;i++)
        {
                for(int j=0;j<MAX;j++)
                        distance[i][j]=0;
        }
}
void airport::read_city()
{
        int k;
        cout<<"\nEnter the no. of cities: " ;
        cin>>n;
        cout<<"Enter city name:\n";
        for(int k=0;k<n;k++)
        {
           cout<<k+1<<"] ";
                cin>>city[k];
        }
        for(int i=0;i<n;i++)
        {
                for(int j=i+1 ; j<n ; j++)
                {
                        cout<<"\nEnter Distance between "<<city[i]<<" to
"<<city[j]<<": ";
                        cin>>distance[i][j];
                        distance[j][i]=distance[i][j];
                }
        }
}
void airport::show_graph()
{
  cout<<"\t";
  for(int k=0;k<n;k++)
  {
    cout<<city[k]<<"\t";
  }
  cout<<endl;
  for(int i=0;i<n;i++)
  {
    cout<<city[i]<<"\t";
    for(int j=0;j<n;j++)
    {
      cout<<distance[i][j]<<"\t";
    }
    cout<<endl;
  }
}
int main()
{
      airport obj;
      obj.read_city();
      obj.show_graph();
```

}

OUTPUT:


main.cpp [SDlab] - Code::Blocks 17.12
"C:\Users\SAURABH KULKARNI\Desktop\SDlab\bin\Debug\SDlab.exe"

```
Enter the no. of cities: 3
Enter city name:
1] pune
2] mumbai
3] bengaluru

Enter Distance between pune to mumbai: 10

Enter Distance between pune to bengaluru: 20

Enter Distance between mumbai to bengaluru: 40
        pune    mumbai  bengaluru
pune    0       10      20
mumbai  10      0       40
bengaluru       20      40      0

Process returned 0 (0x0)   execution time : 94.956 s
Press any key to continue.
```

CONCLUSION:

Hence created graph structure using **Adjacency matrix.**

ASSIGNMENT 4

AIM: For a weighted graph G, find the minimum spanning tree using Prims algorithm**.**

OBJECTIVE: Use Prims algorithm to the minimum spanning tree.

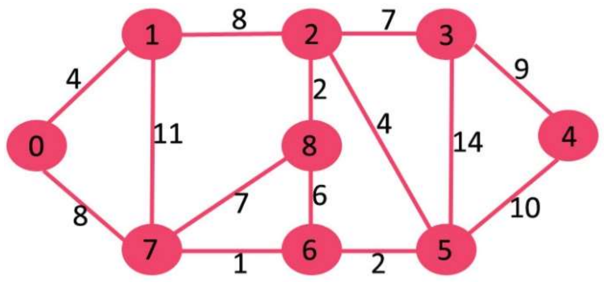
THEORY:

Prim's algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.
A group of edges that connects two set of vertices in a graph is called cut in graph theory. *So, at every step of Prim's algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the verices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).*
How does Prim's Algorithm Work? The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a *Spanning* Tree. And they must be connected with the minimum weight edge to make it a *Minimum* Spanning Tree.

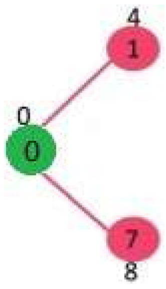
Algorithm
1) Create a set *mstSet* that keeps track of vertices already included in MST.
2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

3) While mstSet doesn’t include all vertices
….**a)** Pick a vertex *u* which is not there in *mstSet* and has minimum key value.
….**b)** Include *u* to mstSet.
….**c)** Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u*-*v* is less than the previous key value of *v*, update the key value as weight of *u*-*v*
The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

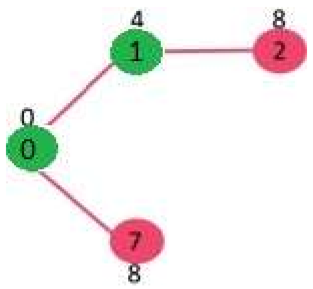
Let us understand with the following example:



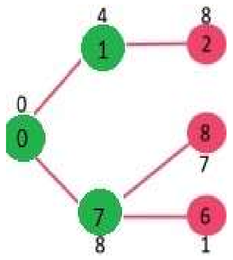
The set *mstSet* is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum key value. The vertex 0 is picked, include it in *mstSet*. So *mstSet* becomes {0}. After including to *mstSet*, update key values of adjacent vertices. Adjacent vertices of 0 are 1 and 7. The key values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



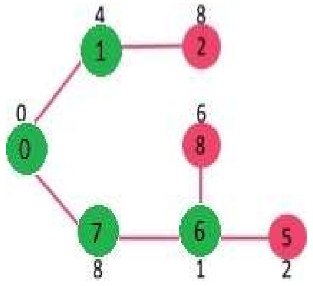
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



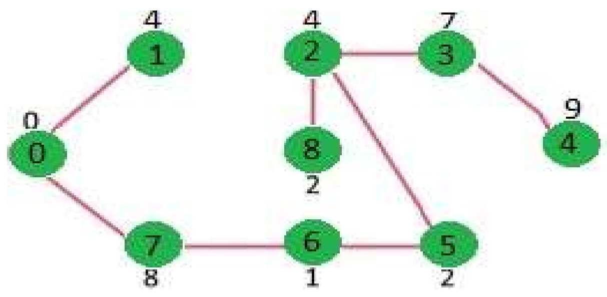
Pick the vertex with minimum key value and not already included in MST (not in mstSET). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstSet now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (1 and 7 respectively).



Pick the vertex with minimum key value and not already included in MST (not in mstSET). Vertex 6 is picked. So mstSet now becomes {0, 1, 7, 6}. Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.



We repeat the above steps until *mstSet* includes all vertices of given graph. Finally, we get the following graph.



PROGRAM:

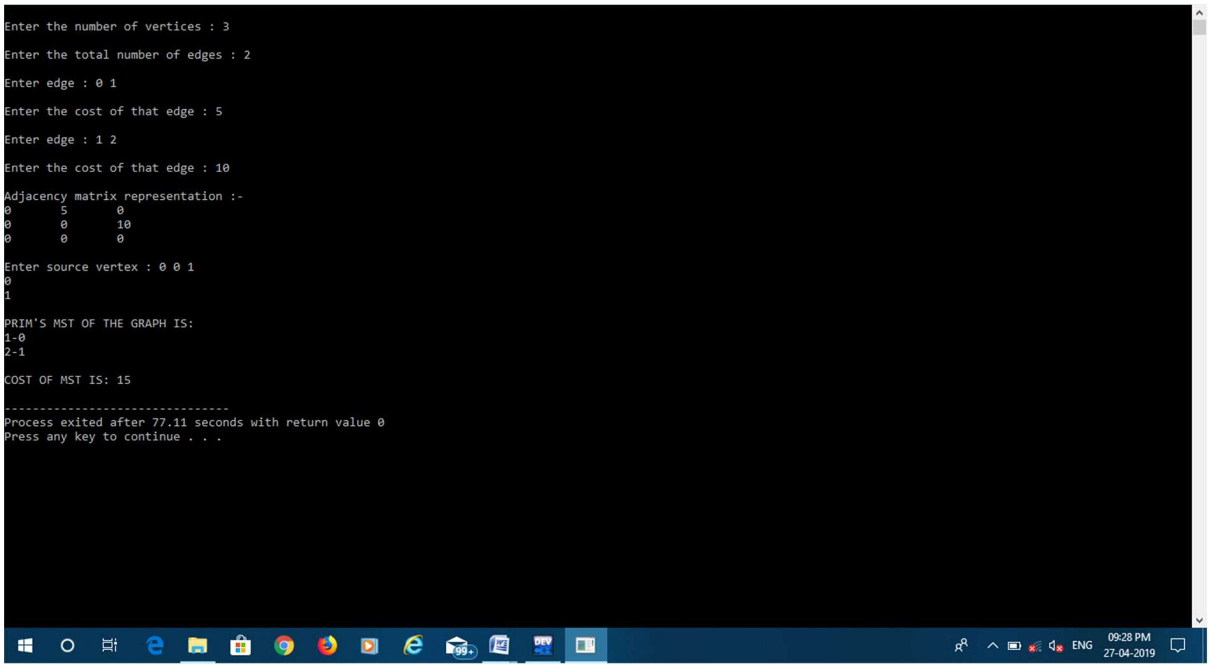
```
#include<iostream>
using namespace std;

void create(int mat[][10], int v)
{
int v1,v2,cost;
int edges;
cout<< "\nEnter the total number of edges : ";
cin>> edges;

for(inti=0;i<v;i++)
          {
                    for(int j=0;j<v;j++)
                    {
                              mat[i][j] = 0;
                    }
          }

for(inti=0;i<edges;i++)
   {
cout<< "\nEnter edge : ";
cin>> v1 >> v2;
cout<< "\nEnter the cost of that edge : ";
cin>> cost;
mat[v1][v2] = cost;
```

```
    }
}

voiddisplay_matrix(int mat[][10],int v)
{
cout<< "\nAdjacency matrix representation :- " <<endl;
for(inti=0;i<v;i++)
    {
for(int j=0;j<v;j++)
        {
cout<< mat[i][j] << "\t";
        }
cout<<endl;
    }
}

intmin_dist(intdist[],int visited[],int v)
{
int min = 32767;
intmin_index;
for(inti=0;i<v;i++)
    {
if(visited[i] == 0 &&dist[i] <= min)
        {
min = dist[i];
min_index = i;
        }
    }
cout<<min_index<<endl;
returnmin_index;
}

voidprint_dist(int mat[][10],intdist[],intv,int parent[])
{
int sum=0;
cout<<"\nPRIM'S MST OF THE GRAPH IS: ";
for(inti = 1; i<v; i++)
    {
cout<<"\n"<<i<<"-"<<parent[i];
```

```
sum = sum + mat[parent[i]][i];
    }
cout<<endl;
cout<<"\nCOST OF MST IS: "<<sum<<endl;
}

void prims(int mat[][10],ints,int v)
{
intdist[v];
int visited[v];
int parent[v];
for(inti=0;i<v;i++)
    {
dist[i] = 32767;
visited[i] = 0;
    }
dist[s] = 0;
int p=0;
for(int j=0;j<v-1;j++)
    {
        p = min_dist(dist,visited,v);
visited[p] = 1;
for(int q=0;q<v;q++)
        {
if(mat[p][q] != 0)
            {
if(visited[q] == 0 && mat[p][q] <dist[q])
                {
dist[q] = mat[p][q];
parent[q] = p;
                }
            }
        }
    }
print_dist(mat,dist,v,parent);
}
```

```
int main()
{
int v;
int s;
cout<< "\nEnter the number of vertices : ";
cin>> v;
int mat[v][10];
            create(mat,v);
display_matrix(mat,v);
cout<< "\nEnter source vertex : ";
cin>> s;
prims(mat,s,v);
return 0;
}
```

OUTPUT:



```
Enter the number of vertices : 3

Enter the total number of edges : 2

Enter edge : 0 1

Enter the cost of that edge : 5

Enter edge : 1 2

Enter the cost of that edge : 10

Adjacency matrix representation :-
0       5       0
0       0       10
0       0       0

Enter source vertex : 0 0 1
0
1

PRIM'S MST OF THE GRAPH IS:
1-0
2-1

COST OF MST IS: 15

--------------------------------
Process exited after 77.11 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION: Hence using the prim's algorithm ,found out the minimum spanning tree.

ASSIGNMENT 5

Aim: You have a business with several offices; you want to lease phone lines to connect them up with each other and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.

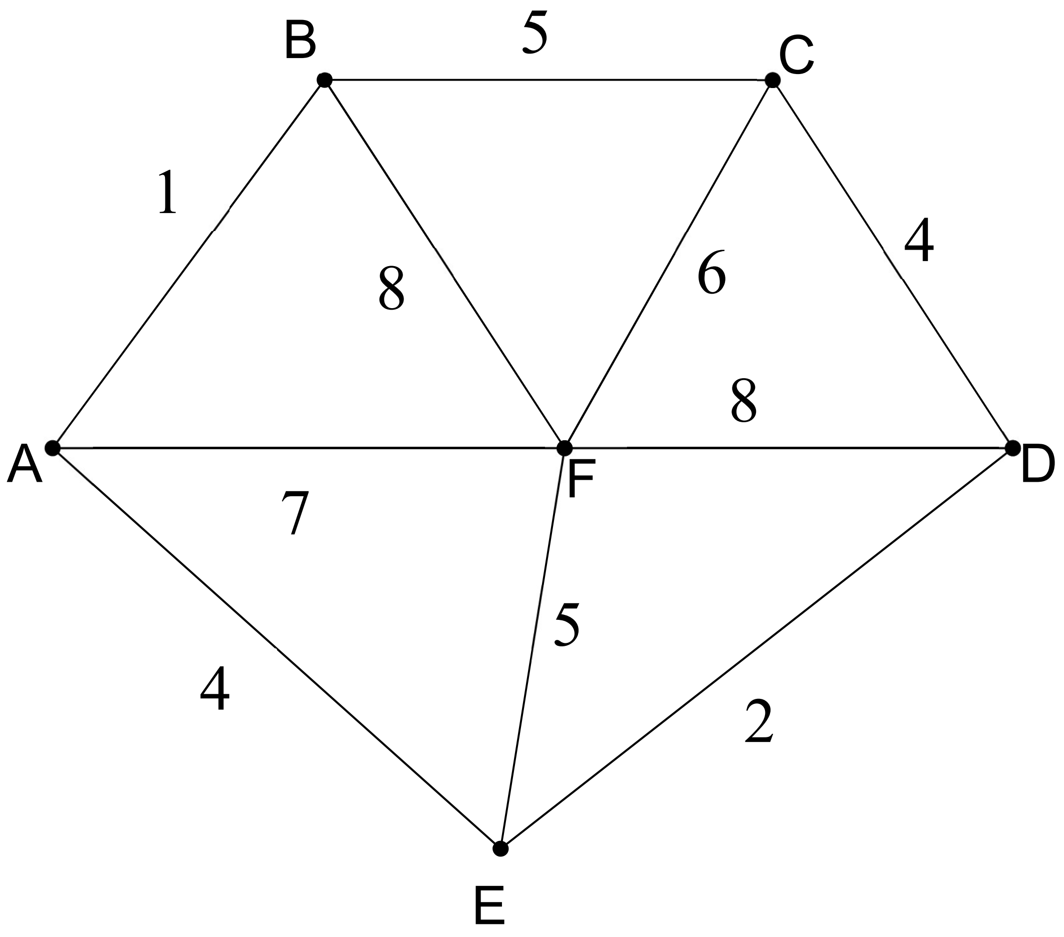
Objective: To understand the concept of minimum spanning tree and finding the minimum cost of tree using Kruskals algorithm.

Theory: A spanning tree of the graph is a connected (if there is at least one path between every pair of vertices in a graph) subgraph in which there are no cycle. Suppose you have a connected undirected graph with a weight (or cost) associated with each edge. The cost of a spanning tree would be the sum of the costs of its edges. A minimum-cost spanning tree is a spanning tree that has the lowest cost. There are two basic algorithms for finding minimum-cost spanning trees: 1. Prim's Algorithm 2. Kruskal's Algorithm .

Kruskals's algorithm: It tarts with no nodes or edges in the spanning tree, and repeatedly add the cheapest edge that does not create a cycle.

Steps of Kruskal's Algorithm to find minimum spanning tree:

1. Select the shortest edge in a network
2. Select the next shortest edge which does not create a cycle
3. Repeat step 2 untill spanning tree has n-1 edges.



Example:

The solution is

AB 1

ED 2

CD 4

AE 4

EF 5

Total weight of tree: 16

Algorithm:

- Algorithm kruskal(G,V,E,T)

```
{
 1.Sort E in increasing order of weight
 2.let G=(V,E) and T=(A,B),A=V ,B is null
    set and let n =count(V)
3.Initialize n set ,each containing a different element of v.
4.while(|B|<n-1) do
     begin
      e=<u,v>the shortest edge not yet considered
      U=Member(u)
       V=Member(v)
     if( Union(U,V))
          update in B and add the cost
      } }
   end
5.T is the minimum spanning tree
}
```

Program code:

```
include<iostream>
```

```
using namespace std;
#define MAX 30
typedef struct edge
{
   int u,v,w;
}edge;
typedef struct edgelist
{
   edge data[MAX];
   int count;
}edgelist;
edgelist elist;
int G[MAX][MAX],n;
edgelist spanlist;

void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();
int main()
{
   int i,j;
   cout<<"\nEnter number of city's:";
   cin>>n;
cout<<"\nEnter the adjacency matrix of city ID's:\n";
for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         cin>>G[i][j];
   kruskal();
   print();
}
void kruskal()
{
   int belongs[MAX],i,j,cno1,cno2;
   elist.count=0;
   for(i=1;i<n;i++)
      for(j=0;j<i;j++)
```

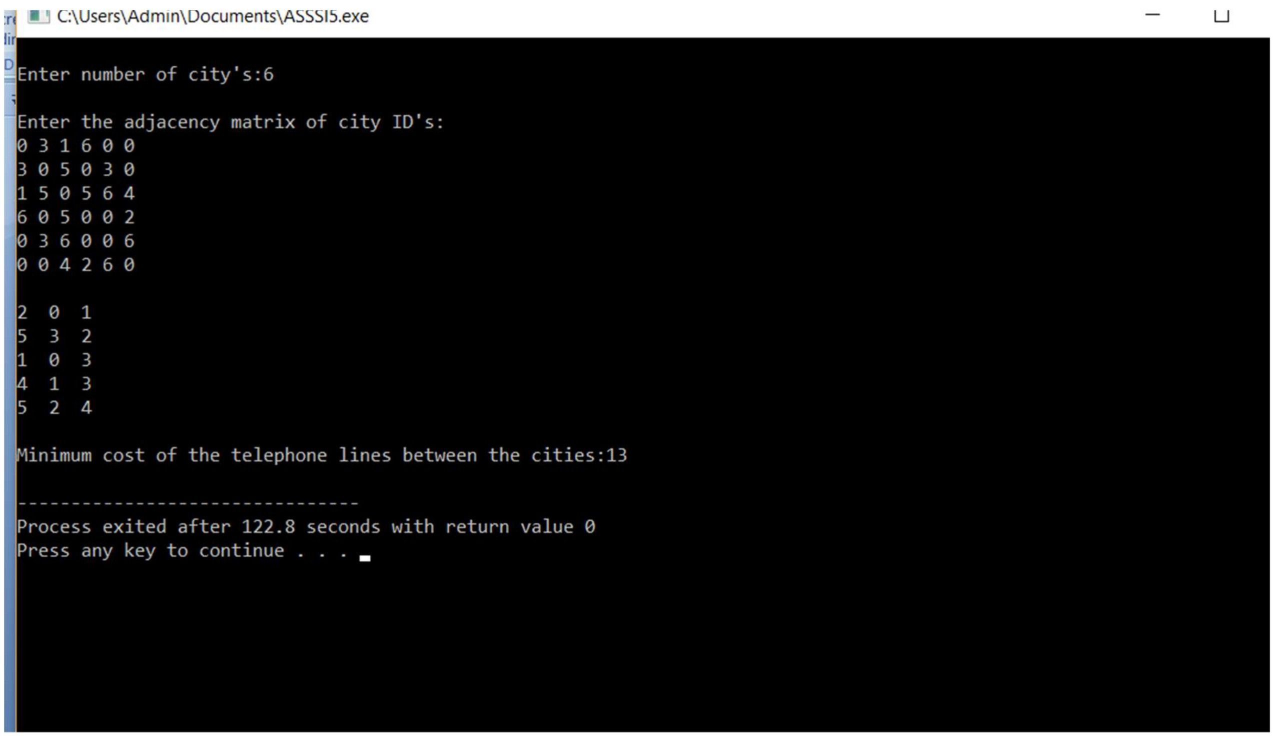
```
    {
       if(G[i][j]!=0)
       {
          elist.data[elist.count].u=i;
          elist.data[elist.count].v=j;
          elist.data[elist.count].w=G[i][j];
          elist.count++;
       }
    }
 sort();
 for(i=0;i<n;i++)
    belongs[i]=i;
 spanlist.count=0;
 for(i=0;i<elist.count;i++)
 {
    cno1=find(belongs,elist.data[i].u);
    cno2=find(belongs,elist.data[i].v);

    if(cno1!=cno2)
```

```
        {
            spanlist.data[spanlist.count]=elist.data[i];
            spanlist.count=spanlist.count+1;
            union1(belongs,cno1,cno2);
        }
    }
}
int find(int belongs[],int vertexno)
{
    return(belongs[vertexno]);
}
void union1(int belongs[],int c1,int c2)
{
    int i;
    for(i=0;i<n;i++)
       if(belongs[i]==c2)
            belongs[i]=c1;
}
void sort()
```

```
{
    int i,j;
    edge temp;
    for(i=1;i<elist.count;i++)
        for(j=0;j<elist.count-1;j++)
            if(elist.data[j].w>elist.data[j+1].w)
            {
                temp=elist.data[j];
                elist.data[j]=elist.data[j+1];
                elist.data[j+1]=temp;
            }
}
void print()
{
    int i,cost=0;
    for(i=0;i<spanlist.count;i++)
    {
        cout<<"\n"<<spanlist.data[i].u<<"
"<<spanlist.data[i].v<<"  "<<spanlist.data[i].w;
```

```
        cost=cost+spanlist.data[i].w;

    }



    cout<<"\n\nMinimum cost of the telephone lines between the cities:"<<cost<<"\n";

}
```

OUTPUT:



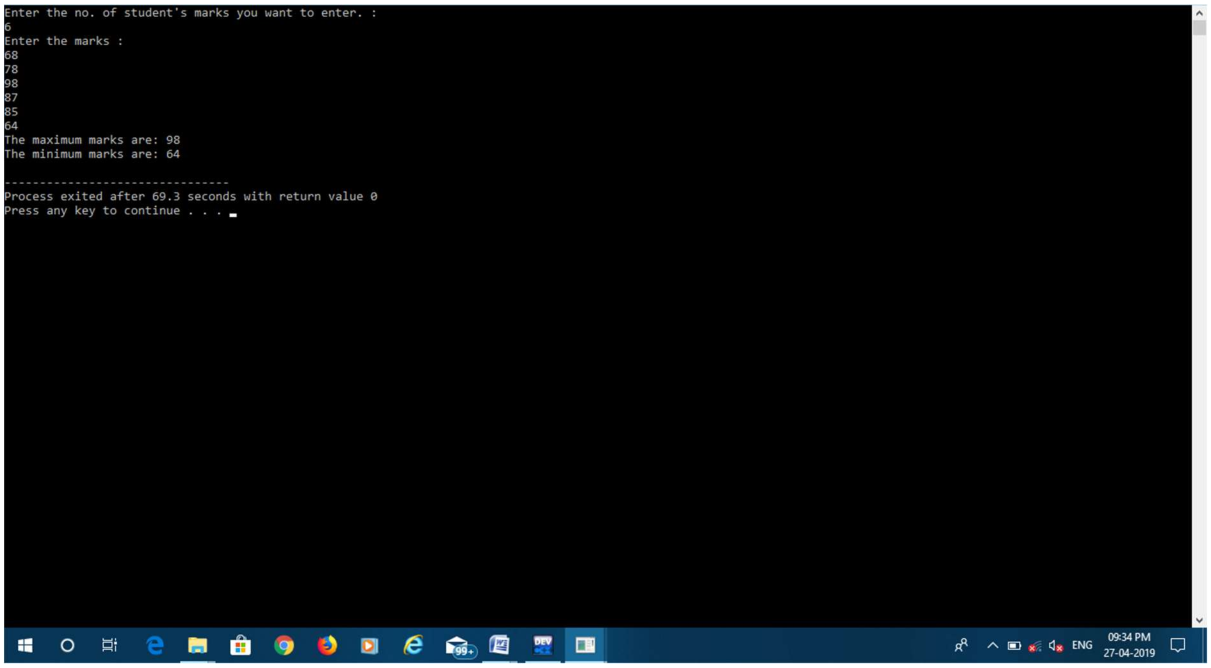
C:\Users\Admin\Documents\ASSSI5.exe

```
Enter number of city's:6

Enter the adjacency matrix of city ID's:
0 3 1 6 0 0
3 0 5 0 3 0
1 5 0 5 6 4
6 0 5 0 0 2
0 3 6 0 0 6
0 0 4 2 6 0

2  0  1
5  3  2
1  0  3
4  1  3
5  2  4

Minimum cost of the telephone lines between the cities:13

--------------------------------
Process exited after 122.8 seconds with return value 0
Press any key to continue . . .
```

Conclusion: Kruskal's algorithm can be shown to run in O(**E log E**) time, where E is the number of edges in the graph. Thus we have connected all the offices with a total minimum cost using kruskal's algorithm.

```
#include <iostream>
using namespace std;
// Toheapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
voidheapify(intarr[], int n, inti)
{
        int largest = i; // Initialize largest as root
        int l = 2*i + 1; // left = 2*i + 1
        int r = 2*i + 2; // right = 2*i + 2
        // If left child is larger than root
        if (l < n &&arr[l] >arr[largest])
                largest = l;
        // If right child is larger than largest so far
        if (r < n &&arr[r] >arr[largest])
                largest = r;
        // If largest is not root
        if (largest != i)
        {
                swap(arr[i], arr[largest]);
                // Recursively heapify the affected sub-tree
                heapify(arr, n, largest);
        }
}
// main function to do heap sort
voidheapSort(intarr[], int n)
{
        for (inti = n / 2 - 1; i>= 0; i--)
                heapify(arr, n, i);
        for (inti=n-1; i>=0; i--)
        {
                swap(arr[0], arr[i]);
                heapify(arr, i, 0);
        }
}
int main()
{
intn,arr[100];
        cout<<"Enter the no. of student's marks you want to enter. :\n";
        cin>>n;
        cout<<"Enter the marks :\n";
        for(inti=0;i<n;i++)
   {
cin>>arr[i];
```

```
    }
        heapSort(arr, n);
        cout<< "The maximum marks are: "<<arr[n-1]<<"\n";
        cout<<"The minimum marks are: "<<arr[0]<<"\n";
}
```



```
Enter the no. of student's marks you want to enter. :
6
Enter the marks :
68
78
98
87
85
64
The maximum marks are: 98
The minimum marks are: 64

--------------------------------
Process exited after 69.3 seconds with return value 0
Press any key to continue . . .
```

ASSIGNMENT 6

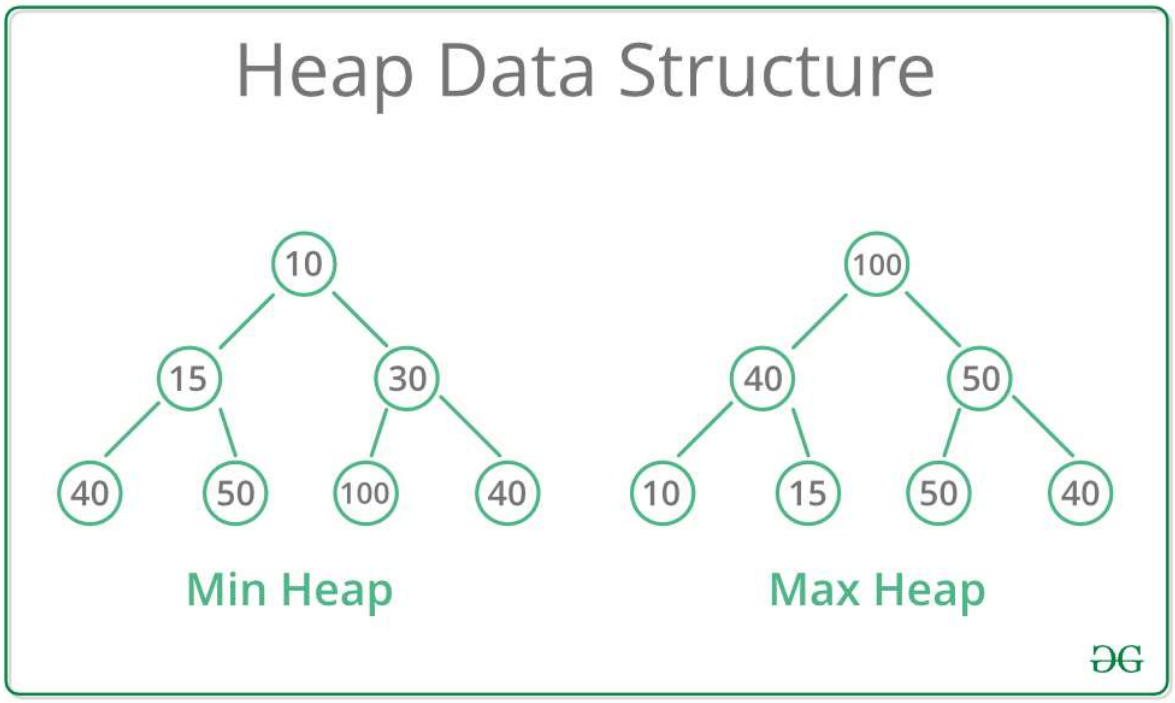
AIM: Read the marks obtained by students of second year in an online examination of particular subject. Find out maximum and minimum marks obtained in that subject using heap data structure.

OBJECTIVE: Use heap data structure to find out maximum and minimum marks obtained in that subject.

THEORY:

A Heap is a special Tree-based data structure in which the tree is a complete binary tree. Generally, Heaps can be of two types:

1. **Max-Heap**: In a Max-Heap the key present at the root node must be greatest among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.
2. **Min-Heap**: In a Min-Heap the key present at the root node must be minimum among the keys present at all of it's children. The same property must be recursively true for all sub-trees in that Binary Tree.



Binary Heap

A Binary Heap is a Binary Tree with following properties.
1) It’s a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.

2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.

Examples of Min Heap:

```
       10                   10
     /    \               /     \
   20      100       15       30
  /                 / \      / \
 30                40   50   100   40
```

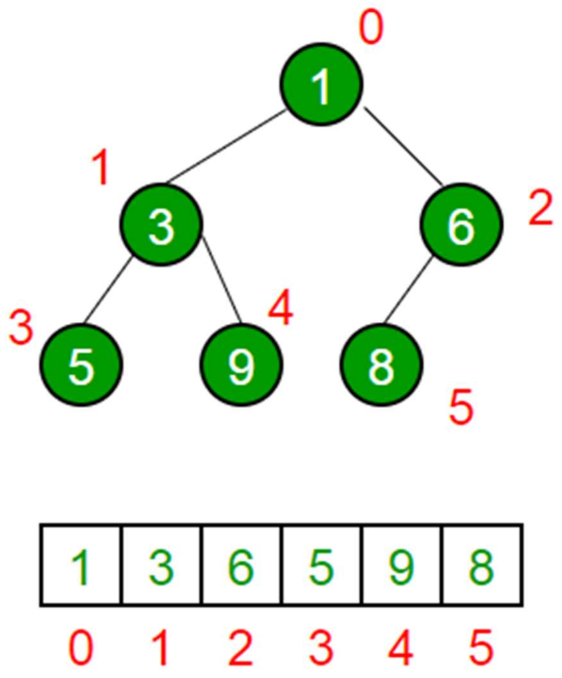
How is Binary Heap represented?

A Binary Heap is a Complete Binary Tree. A binary heap is typically represented as an array.

- The root element will be at Arr[0].
- Below table shows indexes of other nodes for the ith node, i.e., Arr[i]:

Arr[(i-1)/2]	Returns the parent node
Arr[(2*i)+1]	Returns the left child node
Arr[(2*i)+2]	Returns the right child node

The traversal method use to achieve Array representation is **Level Order**



Please refer Array Representation Of Binary Heap for details.

Applications of Heaps:
1) Heap Sort: Heap Sort uses Binary Heap to sort an array in O(nLogn) time.

2) Priority Queue: Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in O(logn) time. Binomoial Heap and Fibonacci Heap are variations of Binary Heap. These variations perform union also efficiently.
3) Graph Algorithms: The priority queues are especially used in Graph Algorithms like Dijkstra's Shortest Path and Prim's Minimum Spanning Tree.
4) Many problems can be efficiently solved using Heaps. See following for example.
a) K'th Largest Element in an array.

b) Sort an almost sorted array/
c) Merge K Sorted Arrays.

Operations on Min Heap:

1) getMini(): It returns the root element of Min Heap. Time Complexity of this operation is O(1).

2) extractMin(): Removes the minimum element from MinHeap. Time Complexity of this Operation is O(Logn) as this operation needs to maintain the heap property (by calling heapify()) after removing root.

3) decreaseKey(): Decreases value of key. The time complexity of this operation is O(Logn). If the decreases key value of a node is greater than the parent of the node, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

4) insert(): Inserting a new key takes O(Logn) time. We add a new key at the end of the tree. IF new key is greater than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

5) delete(): Deleting a key also takes O(Logn) time. We replace the key to be deleted with minum infinite by calling decreaseKey(). After decreaseKey(), the minus infinite value must reach root, so we call extractMin() to remove the key.

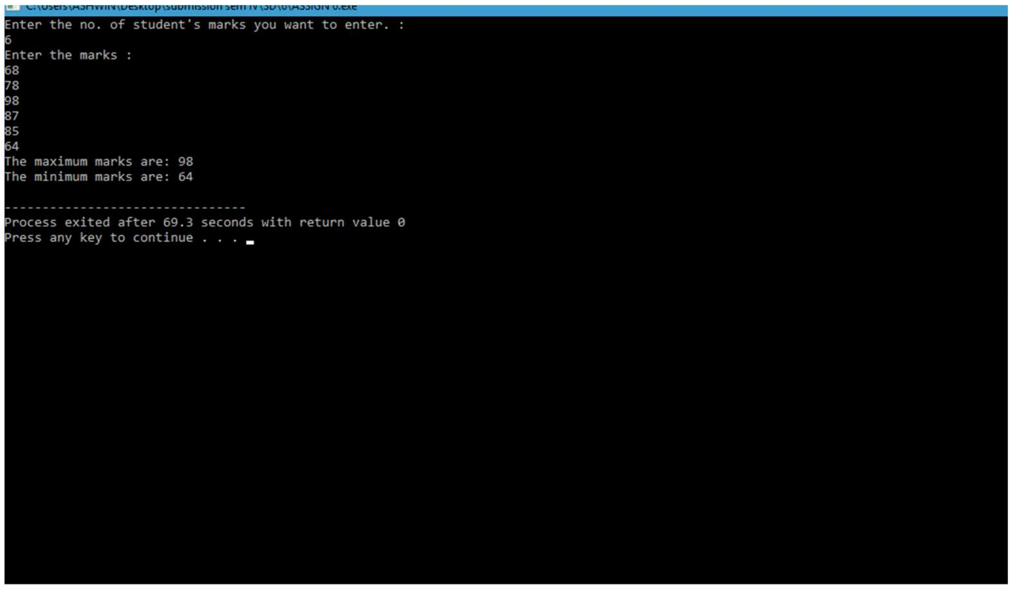
PROGRAM:

```
#include <iostream>

using namespace std;

// Toheapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

voidheapify(intarr[], int n, inti)
```

```
{
        int largest = i; // Initialize largest as root
        int l = 2*i + 1; // left = 2*i + 1
        int r = 2*i + 2; // right = 2*i + 2
        // If left child is larger than root
        if (l < n &&arr[l] >arr[largest])
                largest = l;
        // If right child is larger than largest so far
        if (r < n &&arr[r] >arr[largest])
                largest = r;
        // If largest is not root
        if (largest != i)
        {
                swap(arr[i], arr[largest]);
                // Recursively heapify the affected sub-tree
                heapify(arr, n, largest);
        }
}
// main function to do heap sort
voidheapSort(intarr[], int n)
{
        for (inti = n / 2 - 1; i>= 0; i--)
```

```
                heapify(arr, n, i);
        for (inti=n-1; i>=0; i--)
        {
                swap(arr[0], arr[i]);
                heapify(arr, i, 0);
        }
}
int main()
{
intn,arr[100];
        cout<<"Enter the no. of student's marks you want to enter. :\n";
        cin>>n;
        cout<<"Enter the marks :\n";
        for(inti=0;i<n;i++)
   {
cin>>arr[i];
   }
        heapSort(arr, n);
        cout<< "The maximum marks are: "<<arr[n-1]<<"\n";
        cout<<"The minimum marks are: "<<arr[0]<<"\n";
}
```

OUTPUT:



```
Enter the no. of student's marks you want to enter. :
6
Enter the marks :
68
78
98
87
85
64
The maximum marks are: 98
The minimum marks are: 64

--------------------------------
Process exited after 69.3 seconds with return value 0
Press any key to continue . . .
```

CONCLUSION: Hence using heap data structure we can find out the maximum and minimum element in the heap.

ASSIGNMENT 7

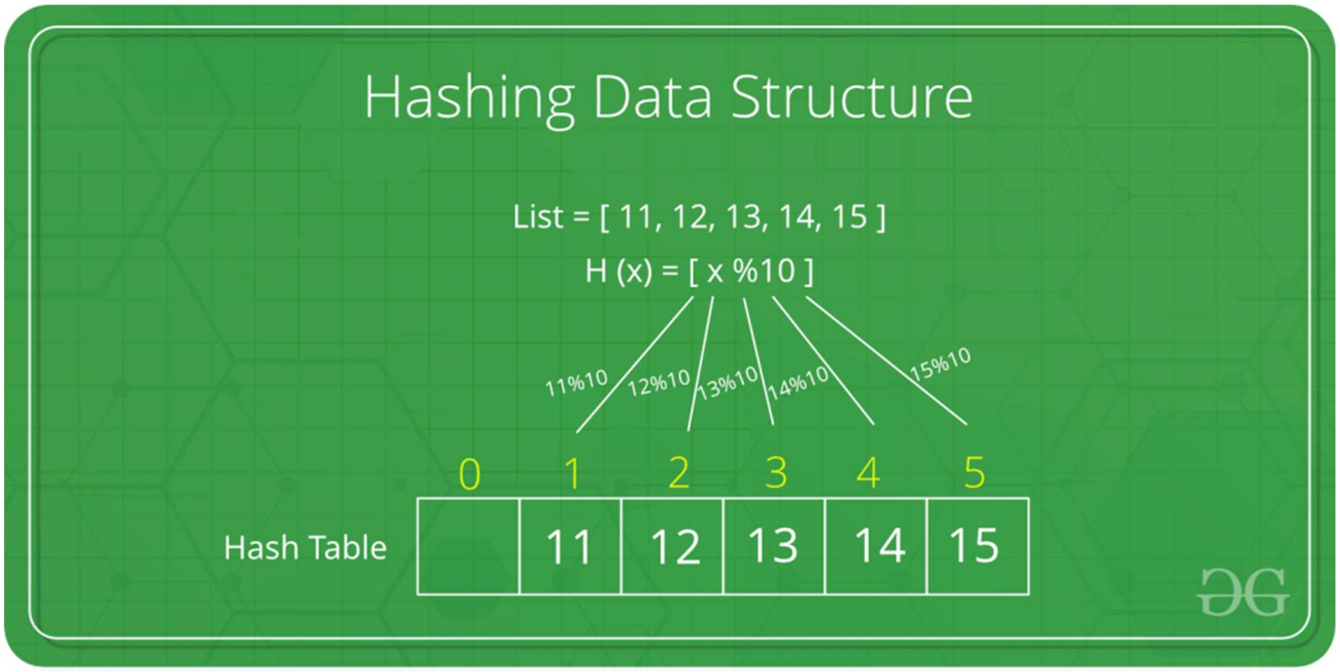
AIM: Insert the keys into a hash table of length m using open addressing using double hashing with h(k)=1+(k mod(m-1)).

OBJECTIVE: Use open addressing' double hashing method to insert a key in a Hash table.

THEORY:

Hashing is an important Data Structure which is designed to use a special function called the Hash function which is used to map a given value with a particular key for faster access of elements. The efficiency of mapping depends of the efficiency of the hash function used.

Let a hash function H(x) maps the value at the index **x%10** in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



Hash Function: A function that converts a given big number to a small practical integer value. The mapped integer value is used as an index in hash table. In simple terms, a hash function maps a big number or string to a small integer that can be used as index in hash table.
A good hash function should have following properties
1) Efficiently computable.
2) Should uniformly distribute the keys (Each table position equally likely for each key)
For example for phone numbers a bad hash function is to take first three digits. A better function is consider last three digits. Please note that this may not be the best hash function. There may be better ways.

Hash Table: An array that stores pointers to records corresponding to a given phone number. An entry in hash table is NIL if no existing phone number has hash function value equal to the index for the entry.
Collision Handling: Since a hash function gets us a small number for a big key, there is possibility that two keys result in same value. The situation where a newly inserted key maps to an already occupied slot in hash table is called collision and must be handled using some collision handling technique. Following are the ways to handle collisions:

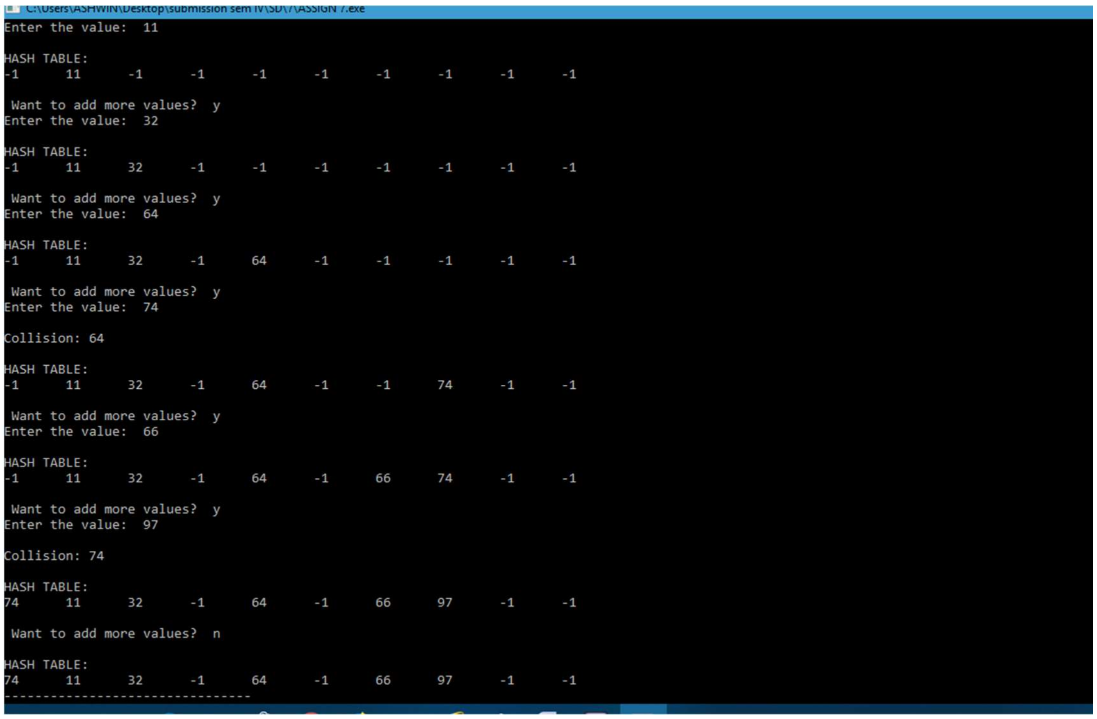
- **Chaining:**The idea is to make each cell of hash table point to a linked list of records that have same hash function value. Chaining is simple, but requires additional memory outside the table.
- **Open Addressing:** In open addressing, all elements are stored in the hash table itself. Each table entry contains either a record or NIL. When searching for an element, we one by one examine table slots until the desired element is found or it is clear that the element is not in the table.

PROGRAM:

```
#include<iostream>

#include<stdlib.h>

using namespace std;
```

```
int size=10;
void display(int hash[])
{
        int i;
        cout<<"\nHASH TABLE:\n";
        for(i=0;i<size;i++)
        {
                cout<<""<<hash[i]<<"\t";
        }
}
int main()
{
        int hash[size],val,i;
        char ch;
        for(i=0;i<size;i++)
        {
                hash[i]=-1;
        }
        do
        {
                cout<<"Enter the value:  ";
                cin>>val;
                int m=val%size;
                if(hash[m] == -1)
                {
                        hash[m]=val;
                        display(hash);
                }
        else
        {
                cout<<"\nCollision: ";
                cout<<""<<hash[m]<<"\n";
                if(hash[m]%10!=m)
                {
                        int h=hash[m];
                        hash[m]=val;
                        val=h;
                }
                int x=1+(val%(size-1));
                for(i=1;i<size;i++)
                {
                        int y=(val+i*x);
                        int z=y%size;
                        if(hash[z]==-1)
                {
                        hash[z]=val;
                        display(hash);
                        break;
                }
            }
        }
        cout<<"\n\n Want to add more values?  ";
        cin>>ch;
        int ss=0;
        for(i=0;i<size;i++)
        {
            if(hash[i]!= -1)
            ss++;
        }
        if(ss==10)
        {
            cout<<"\n\nHast table is full";
            exit(1);
        }
}while(ch=='y' || ch=='Y');
display(hash);
```

```
return 0;

}
```



```
C:\Users\ASHWIN\Desktop\submission sem IV\SD\7\ASSIGN 7.exe
Enter the value:  11

HASH TABLE:
-1      11      -1      -1      -1      -1      -1      -1      -1      -1

 Want to add more values?  y
Enter the value:  32

HASH TABLE:
-1      11      32      -1      -1      -1      -1      -1      -1      -1

 Want to add more values?  y
Enter the value:  64

HASH TABLE:
-1      11      32      -1      64      -1      -1      -1      -1      -1

 Want to add more values?  y
Enter the value:  74

Collision: 64

HASH TABLE:
-1      11      32      -1      64      -1      -1      74      -1      -1

 Want to add more values?  y
Enter the value:  66

HASH TABLE:
-1      11      32      -1      64      -1      66      74      -1      -1

 Want to add more values?  y
Enter the value:  97

Collision: 74

HASH TABLE:
74      11      32      -1      64      -1      66      97      -1      -1

 Want to add more values?  n

HASH TABLE:
74      11      32      -1      64      -1      66      97      -1      -1
--------------------------------
```

OUTPUT:

CONCLUSION: Hence using Hashing data structure we can insert the big numbers or data in a hash table.

ASSIGNMENT NO.8.

Aim :- 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

Objective:- To study the different data structure concepts to implement this program.

Theory:-

Input/output formatting

Writing to or reading from a file is similar to writing onto a terminal screen or reading from a keyboard. Differences are:

- File must be opened with an OPEN statement, in which the unit number and (optionally) the filename are given
- Subsequent writes (or reads) must refer to a known unit number (used for open)
- File should be closed at the end

File opening and closing

The syntax is:

```
OPEN([unit=]lunit,file='name' [,options])

CLOSE([unit=]lunit [,options])
```

For example:

```
OPEN(10, file='output.dat', status='new')

CLOSE(unit=10)
```

- The first parameter is the unit number and the keyword unit= can be omitted.
- The unit numbers 0,5 and 6 are predefined.
  - 0 is output for standard (system) error messages
  - 5 is for standard (user) input
  - 6 is for standard (user) output
  - These units are opened by default and should not be re-opened nor closed by users

Some options for opening a file:

- status: existence of the file ('old', 'new', 'replace', 'scratch', 'unknown')
- position: offset, where to start writing ('append')
- action: file operation mode ('write','read','readwrite')
- form: text or binary file ('formatted', 'unformatted')
- access: direct or sequential file access ('direct','sequential','stream')
- iostat: error indicator, (output) integer (non zero only upon an error)
- err: the label number to jump upon error
- recl: record length, (input) integer for direct access files only. Be careful, it can be in bytes or words...

Algorithm:-

```
#include <iostream>

#include <fstream>

#include <cstring>

#include <iomanip>

#include<cstdlib>

#define max 100

using namespace std;

class Student
```

```
{
    char name[max];
    int rollNo;
    int year;
    int division;
    char address[50];
    friend class file_operat;
    public:  Student()
            {
                    strcpy(name,"");
                    rollNo=year=division=0;
                    strcpy(address,"");
            }
            Student(char name[max],int rollNo,int year,int
division,char address[max])
            {
                strcpy(this->address,address);
                strcpy(this->name,name);
                this->division=division;
                this->rollNo=rollNo;
                this->year=year;
            }
            int getRollNo()
            {
                return rollNo;
            }
            void displayStudentData()
            {

    cout<<endl<<setw(3)<<rollNo<<setw(10)<<name<<setw(3)<
<year<<setw(2)<<division<<setw(10)<<address;
            }
};
class file_operat
{
    fstream file;
    public:file_operat(char *name)
             {


                 this-
>file.open(name,ios::in|ios::out|ios::ate|ios::binary);
```

```
		}
		void insertRecord(int rollNo,char name[max],int
year,int division,char address[max])
		{
			Student
s=Student(name,rollNo,year,division,address);
			file.seekp(0,ios::end);
			file.write((char*)&s,sizeof(Student));
			file.clear();
		}
		void displayAllRecords()
		{
			Student s;
			file.seekg(0,ios::beg);
			while(file.read((char *)&s,sizeof(Student)))
			{
				s.displayStudentData();
			}
			file.clear();
		}
		void displayRecord(int rollNo)
```

```
{
        Student s;
        file.seekg(0,ios::beg);
        void *p;
        while(file.read((char *)&s,sizeof(Student)))
        {
                if(s.rollNo==rollNo)
                {
                        s.displayStudentData();
                        break;
                }
        }
        if(p==NULL)
                throw "Element not present";
        file.clear();
}
void delete_record(int rollNo)
{
        ofstream new_file("new.txt",ios::binary);
        file.seekg(0,ios::beg);
```

```
bool flag=false;
Student s;
while(file.read((char *)&s,sizeof(s)))
{
        if(s.rollNo==rollNo)
        {
                flag=true;
                continue;
        }
        new_file.write((char *)&s,sizeof(s));
}
if(!flag)
{
        cout<<"Element Not Present";
}
file.close();
new_file.close();
remove("student.txt");
rename("new.txt","student.txt");
```

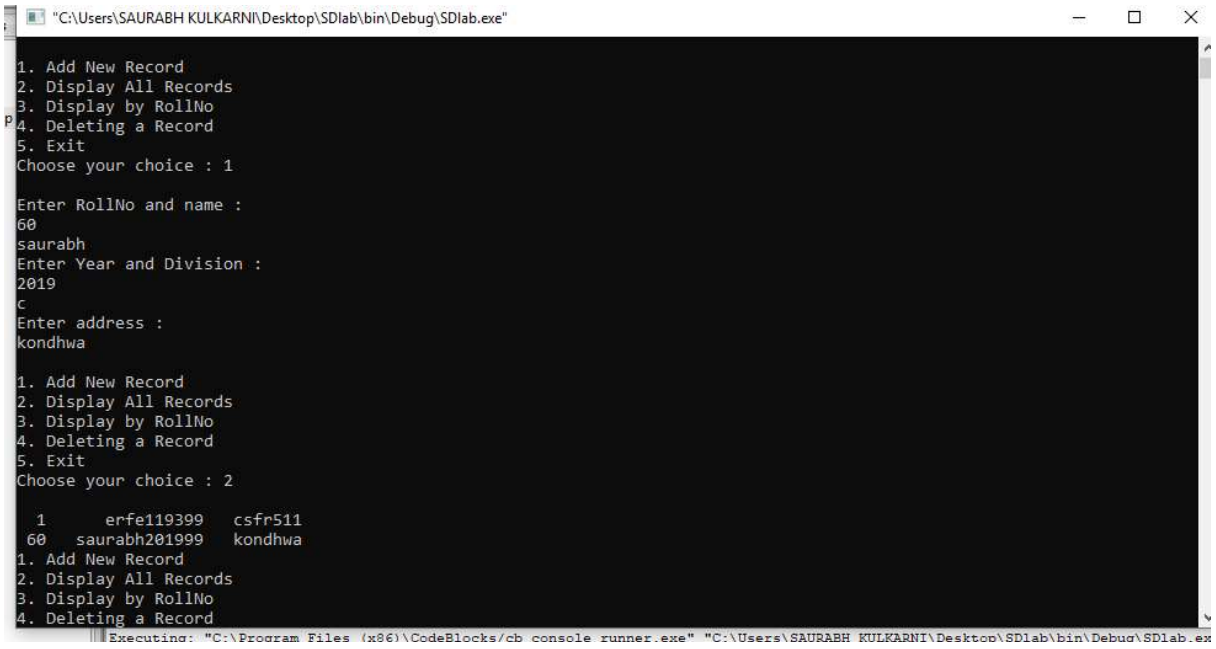
```
file.open("student.txt",ios::in|ios::ate|ios::out|ios::binary);
                }
                ~file_operat()
                {
                       file.close();
                       cout<<"Closing file..";
                }

};
int  main()
{
       ofstream new_file("student.txt",ios::app|ios::binary);
       new_file.close();
       file_operat file((char *)"student.txt");
    int rollNo,year,choice=0;
    char division;
    char name[max],address[max];
    while(choice!=5)
    {
```

```
cout<<"\n";
        cout<<"1. Add New Record\n";
        cout<<"2. Display All Records\n";
        cout<<"3. Display by RollNo\n";
        cout<<"4. Deleting a Record\n";
        cout<<"5. Exit\n";
        cout<<"Choose your choice : ";
        cin>>choice;
        switch(choice)
        {
            case 1 : //New Record
                        cout<<endl<<"Enter RollNo and name : \n";
                        cin>>rollNo>>name;
                        cout<<"Enter Year and Division : \n";
                        cin>>year>>division;
                        cout<<"Enter address : \n";
                        cin>>address;

        file.insertRecord(rollNo,name,year,division,address);
                        break;
```

```
case 2 :
        file.displayAllRecords();
        break;
case 3 :
        cout<<"Enter Roll Number";
        cin>>rollNo;
        try
        {
                file.displayRecord(rollNo);
        }
        catch(const char *str)
        {
                cout<<str;
        }
        break;
case 4:
        cout<<"Enter rollNo";
        cin>>rollNo;
        file.delete_record(rollNo);
        break;
```

```
            case 5 :break;
        }


    }
}
```

Output :-



"C:\Users\SAURABH KULKARNI\Desktop\SDlab\bin\Debug\SDlab.exe"

```
1. Add New Record
2. Display All Records
3. Display by RollNo
4. Deleting a Record
5. Exit
Choose your choice : 1

Enter RollNo and name :
60
saurabh
Enter Year and Division :
2019
c
Enter address :
kondhwa

1. Add New Record
2. Display All Records
3. Display by RollNo
4. Deleting a Record
5. Exit
Choose your choice : 2

  1      erfe119399   csfr511
 60   saurabh201999   kondhwa
1. Add New Record
2. Display All Records
3. Display by RollNo
4. Deleting a Record
Executing: "C:\Program Files (x86)\CodeBlocks/cb_console_runner.exe" "C:\Users\SAURABH KULKARNI\Desktop\SDlab\bin\Debug\SDlab.ex
```

Conclusion:- Using file organization we can create a permanent database where we can store the programs outputs and the entered data at the runtime.

ASSIGNMENT 9

AIM:-

Company maintain employee information as employee ID ,name,designation and salary.Allow user to add,delete information of employee.Display information of particular employee.If employee doesn't exists an appropriate message is displayed.Use index sequential file to maintain the data.

OBJECTIVE:-

To implement file handling and perform functions like insertion,deletion and display of data using index sequential file.

THEORY:-

Records in indexed sequential files are stored in the order that they are written to the disk. Records may be retrieved in sequential order or in random order using a numeric index to represent the record number in the file.The record size, specified when the file is created, may range from 1 to 8000 bytes.When an Internet Basic program opens an indexed sequential file, the Comet operating system assigns a unique record pointer to the file. Each user opening the file is assigned a unique pointer, allowing multiple users to access data from the same file at the same time. To avoid data integrity problems when more than one user is accessing a file, Comet provides a record locking mechanism. The EXTRACT statement is used to read and lock individual data records.When an indexed sequential file is opened, the record pointer is positioned at the first record. Subsequent I/O operations change the location of the pointer. Note: Some I/O operations do not move the pointer.

EXAMPLE:

For example, to read all the records from an indexed sequential file in order, you would open the file and read the records without specifying an index. This would move through the file in sequential order and end when the last record was read.To read a specific record from an indexed sequential file, you would include the KEY= parameter in the READ (or associated input) statement. The "key" in this case would be a specific record number (e.g., the number 35 would represent the 35th record in the file). The direct access to a record moves the record pointer, so that subsequent sequential access would take place from the new record pointer location, rather than the beginning of the file.

Application : Indexed sequential files are commonly used for transaction files because they take less disk space than keyed files, and are faster to read from beginning to end than a keyed file.

PROGRAM:

```
#include<iostream>

#include<fstream>

#include<string.h>

using namespace std;
```

```
typedef struct EMP_REC
 {
 char name[10];
 int emp_id;
 int salary;
 char des[10];
 }Rec;

typedef struct INDEX_REC
  {
  int emp_id;
  int position;
  }Ind_Rec;

class Employee
{

 Rec Records;
 Ind_Rec Ind_Records;
 public:
  void Create();
  void Display();
  void Search();
  void deletion();
};

void Employee::Create()
{
 char ch='y';
 ofstream seqfile;
 ofstream indexfile;
 int i=0;
 indexfile.open("IND.DAT",ios::out|ios::binary);
 seqfile.open("EMP.DAT",ios::out|ios::binary);
 do
 {
  cout<<"\n Enter Name: ";
  cin>>Records.name;
  cout<<"\n Enter Emp_ID: ";
  cin>>Records.emp_id;
  cout<<"\n Designation";
  cin>>Records.des;
  cout<<"\n Enter Salary: ";
  cin>>Records.salary;
  cout<<Records.name<<" "<<Records.emp_id<<" "<<Records.salary;

  seqfile.write((char*)&Records,sizeof(Records));


  Ind_Records.emp_id=Records.emp_id;
  Ind_Records.position=i;
  indexfile.write((char*)&Ind_Records,sizeof(Ind_Records));
  i++;
  cout<<"\nDo you want to add more records?";
  cin>>ch;
  }while(ch=='y');
  seqfile.close();
  indexfile.close();
```

```
}


void Employee::Display()
{
 ifstream seqfile;
 ifstream indexfile;

 seqfile.open("EMP.DAT",ios::in|ios::binary);
 indexfile.open("IND.DAT",ios::in|ios::binary);
 cout<<"\n The Contents of file are ..."<<endl;
 int i=0;
 while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))
 {
  i=Ind_Records.position*sizeof(Rec);
  seqfile.seekg(i,ios::beg);
  seqfile.read((char *)&Records,sizeof(Records));
  if(Records.emp_id!=-1)
  {
  cout<<"\nName: "<<Records.name<<flush;
```

```
   cout<<"\nEmp_ID: "<<Records.emp_id;
   cout<<"\nDesignation :"<<Records.des;
   cout<<"\nSalary: "<<Records.salary;
   cout<<"\n";
    }

 }
 seqfile.close();
 indexfile.close();
}
void Employee::Search()
{
 fstream seqfile;
 fstream indexfile;
 int id,pos,offset;
 cout<<"\n Enter the Emp_ID for searching the record ";
 cin>>id;
 indexfile.open("IND.DAT",ios::in|ios::binary);
 pos=-1;
```

```
while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))
{
 if(id==Ind_Records.emp_id)
 {
  pos=Ind_Records.position;
  break;
 }
}
 if(pos==-1)
 {
 cout<<"\n Record is not present in the file";
 return;
 }
 offset=pos*sizeof(Records);
 seqfile.open("EMP.DAT",ios::in|ios::binary);
 seqfile.seekg(offset,ios::beg);
 seqfile.read((char *)&Records,sizeof(Records));
 if(Records.emp_id==-1)
 {
```

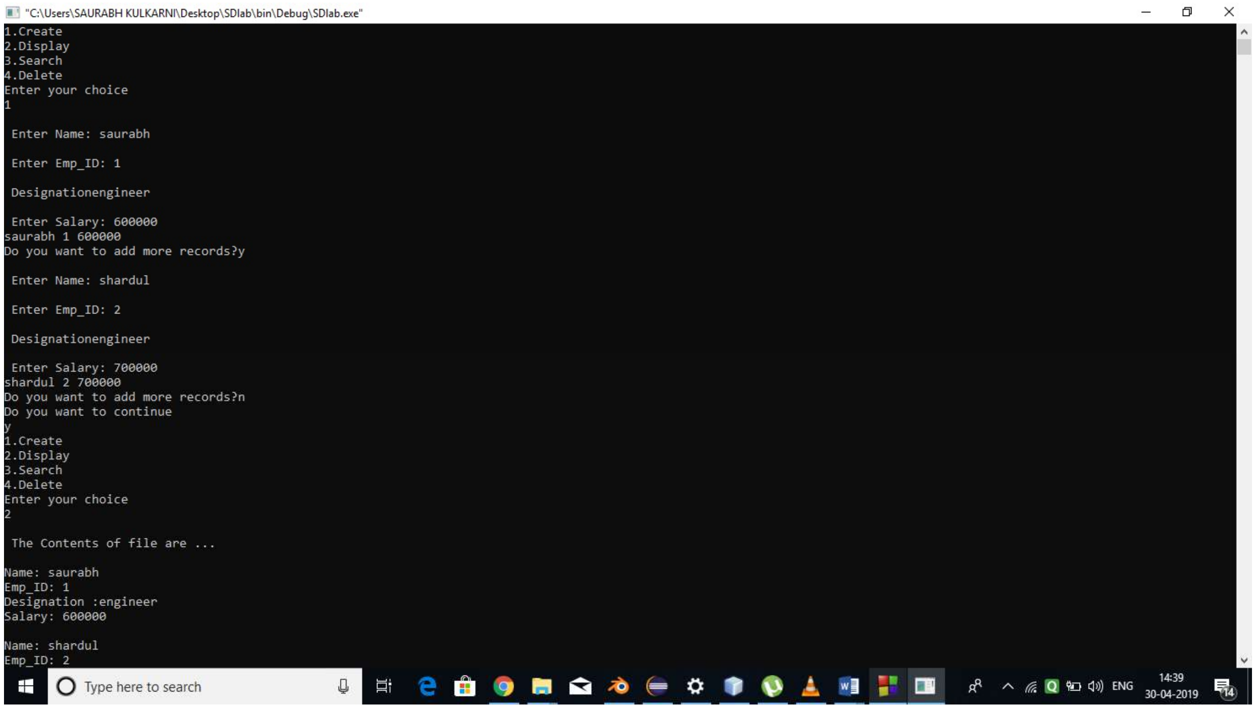
```
 cout<<"\n Record is not present in the file";
 return;
 }
 else
 {
 cout<<"\n The Record is present in the file and it is...";
 cout<<"\n Name: "<<Records.name;
 cout<<"\n Emp_ID: "<<Records.emp_id;
 cout<<"\n Designation: "<<Records.des;
 cout<<"\n Salary: "<<Records.salary;
 }
 seqfile.close();
 indexfile.close();
}
void Employee::deletion()
{
  int id,pos;
  cout<<"For deletion"<<endl;
  cout<<"\n Enter the employee id for searching"<<endl;
```

```
   cin>>id;
   fstream seqfile;
   fstream indexfile;
 seqfile.open("EMP.DAT",ios::in|ios::binary|ios::out);
indexfile.open("IND.DAT",ios::in|ios::binary|ios::out);
seqfile.seekg(0,ios::beg);
indexfile.seekg(0,ios::beg);
pos=-1;
while(indexfile.read((char *)&Ind_Records,sizeof(Ind_Records)))
{
 if(id==Ind_Records.emp_id)
 {
  pos=Ind_Records.position;
  Ind_Records.emp_id=-1;
  break;
 }
}
 if(pos==-1)
 {
```

```
 cout<<"\n Record is not present in the file";
 return;
 }
 int offset=pos*sizeof(Rec);
 seqfile.seekp(offset);
 strcpy(Records.name,"");
 Records.emp_id=-1;
 Records.salary=-1;
 strcpy(Records.des,"");
 seqfile.write((char *)&Records,sizeof(Records))<<flush;
 offset=pos*sizeof(Ind_Rec);
 indexfile.seekp(offset);
 Ind_Records.emp_id=-1;
 Ind_Records.position=pos;
 indexfile.write((char *)&Ind_Records,sizeof(Ind_Records));
 seqfile.seekg(0);
 indexfile.close();
 seqfile.close();
}
```

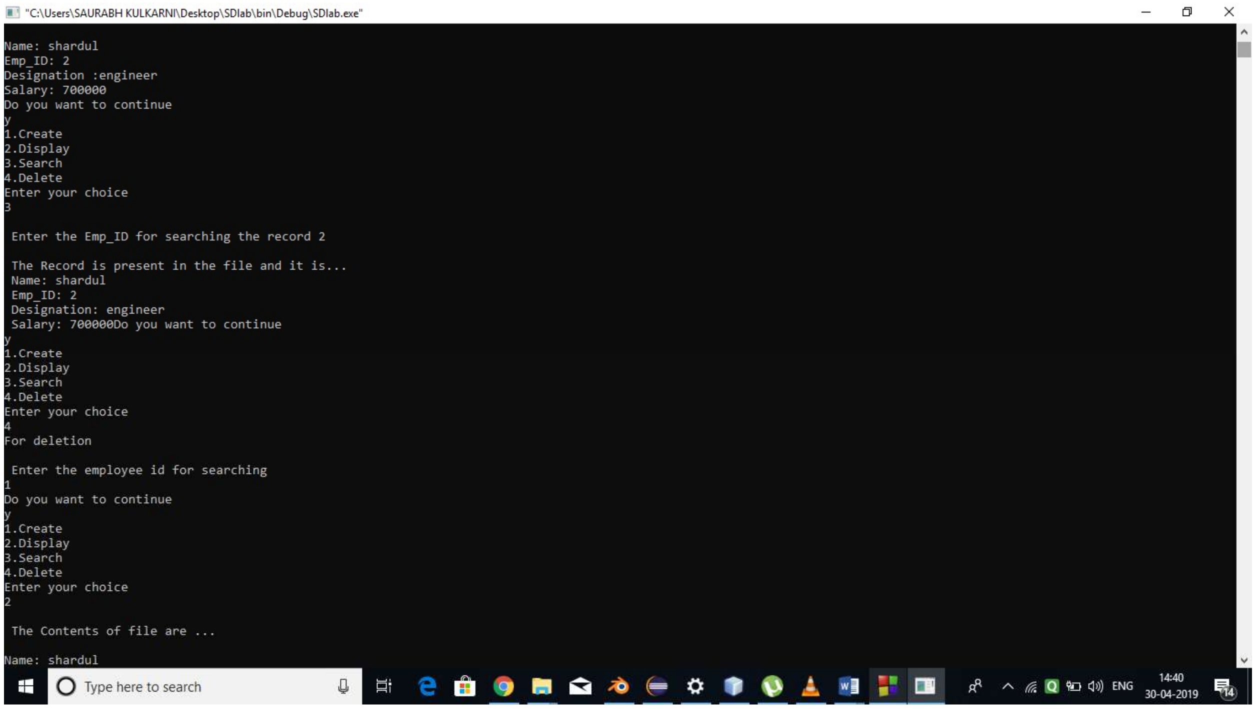
```
int main()
{
 Employee e;
 char ans='y';
 int choice,key;
 do
 {
    cout<<"1.Create"<<endl;
    cout<<"2.Display"<<endl;
    cout<<"3.Search"<<endl;
    cout<<"4.Delete"<<endl;
    cout<<"Enter your choice"<<endl;
    cin>>choice;
       switch(choice)
       {
       case 1:
          e.Create();
          break;
       case 2:
         e.Display();
         break;
       case 3:
           e.Search();
           break;
       case 4:
          e.deletion();
          break;
       }
       cout<<"Do you want to continue"<<endl;
       cin>>ans;
 }while (ans=='y');
return 0;
}
```

OUTPUT:-



"C:\Users\SAURABH KULKARNI\Desktop\SDlab\bin\Debug\SDlab.exe"

```
1.Create
2.Display
3.Search
4.Delete
Enter your choice
1

 Enter Name: saurabh

 Enter Emp_ID: 1

 Designationengineer

 Enter Salary: 600000
saurabh 1 600000
Do you want to add more records?y

 Enter Name: shardul

 Enter Emp_ID: 2

 Designationengineer

 Enter Salary: 700000
shardul 2 700000
Do you want to add more records?n
Do you want to continue
y
1.Create
2.Display
3.Search
4.Delete
Enter your choice
2

 The Contents of file are ...

Name: saurabh
Emp_ID: 1
Designation :engineer
Salary: 600000

Name: shardul
Emp_ID: 2
```



"C:\Users\SAURABH KULKARNI\Desktop\SDlab\bin\Debug\SDlab.exe"

```
Name: shardul
Emp_ID: 2
Designation :engineer
Salary: 700000
Do you want to continue
y
1.Create
2.Display
3.Search
4.Delete
Enter your choice
3

 Enter the Emp_ID for searching the record 2

 The Record is present in the file and it is...
 Name: shardul
 Emp_ID: 2
 Designation: engineer
 Salary: 700000Do you want to continue
y
1.Create
2.Display
3.Search
4.Delete
Enter your choice
4
For deletion

 Enter the employee id for searching
1
Do you want to continue
y
1.Create
2.Display
3.Search
4.Delete
Enter your choice
2

 The Contents of file are ...

Name: shardul
```

CONCLUSION:-

We have successfully implemented file handling and performed functions like insertion,deletion and display of employee data using index sequential file.