

ASSIGNMENT NO.1.

Aim :- To create ADT that implement the "set" concept.

- a. Add (newElement) -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 .

Objective:- to study the different set operations.

Theory:-

Set is a container implemented in C++ language in STL and has a concept similar to how set is defined in mathematics. The facts that separates set from the other containers is that is it contains only the distinct elements and elements can be traversed in sorted order. Having the strong hold on sets is useful in competitive programming and solving algorithmic problems. The insertion and deletion in STL sets are discussed in this article.Sets have the most impact in mathematical set theory. These theories are used in many kinds of proofs, structures, and abstract algebra. Creating relations from different sets and codomains are also an important applications of sets.

In computer science, set theory is useful if you need to collect data and do not care about their multiplicity or their order. In databases, especially for relational databases, sets are very useful. There are many commands that finds unions, intersections, and differences of different tables and sets of data.

Program Code:-

```
#include <iostream>

using namespace std;
```

```
const int MAX=50;

template<class T>

class SET

{
    T data[MAX];

    int n;

public:
    SET()
    {
        n=-1;
    }

    bool insert(T);

    bool remove(T);

    bool contains(T);

    int size();

    void print();

    void input(int num);

    SET unionS(SET,SET);

    SET intersection(SET,SET);

    SET difference(SET,SET);

};
```

```
template<class T>
void SET<T>::input(int num)
{
    T element;
    for(int i=0;i<num;i++)
    {
        cout<<"\nEnter Element: "<<i+1;
        cin>>element;
        insert(element);
    }
}

template<class T>
void SET<T>::print()
{
    for(int i=0;i<=n;i++)
        cout<<" "<<data[i];
}

template<class T>
SET<T> SET<T>::unionS(SET<T> s1,SET<T> s2)
{
    SET<T> s3;
```

```
int flag=0;

int i=0;

for(i=0;i<=s1.n;i++)
{
    s3.insert(s1.data[i]);
}

for(int j=0;j<=s2.n;j++)
{
    flag=0;

    for(i=0;i<=s1.n;i++)
    {
        if(s1.data[i]==s2.data[j])
        {
            flag=1;
            break;
        }
    }

    if(flag==0)
    {
        s3.insert(s2.data[j]);
    }
}
```

```
        }  
    }  
  
    return s3;  
}  
  
template<class T>  
SET<T> SET<T>::difference(SET<T> s1,SET<T> s2)  
{  
    SET<T> s3;  
    int flag=1;  
    for(int i=0;i<=s1.n;i++)  
    {  
        for(int j=0;j<=s2.n;j++)  
        {  
            if(s1.data[i]==s2.data[j])  
            {  
                flag=0;  
                break;  
            }  
            else flag=1;  
        }  
    }  
}
```

```
        if(flag==1)
        {
            s3.insert(s1.data[i]);

        }
    }
    return s3;
}

template<class T>
SET<T> SET<T>::intersection(SET<T> s1,SET<T> s2)
{
    SET<T> s3;
    for(int i=0;i<=s1.n;i++)
    {
        for(int j=0;j<=s2.n;j++)
        {
            if(s1.data[i]==s2.data[j])
            {
                s3.insert(s1.data[i]);
                break;
            }
        }
    }
}
```

```
        }

    }

    return s3;
}

template<class T>
bool SET<T>::insert(T element)
{
    if(n>=MAX)
    {
        cout<<"\nOverflow.SET is full.\n";
        return false;
    }
    data[++n]=element;
    return true;
}

template<class T>
bool SET<T>::remove(T element)
{
    if(n== -1)
    {
        cout<<"Underflow. Cannot perform delete operation on empty SET.";
```

```
        return false;
    }
    for(int i=0;i<=n;i++)
    {
        if(data[i]==element)
        {
            for(int j=i;j<=n;j++)
            {
                data[j]=data[j+1];
            }
            return true;
        }
    }
    //data[n--]=0;
    return false;
}

template<class T>
bool SET<T>::contains(T element)
{
    for(int i=0;i<=n;i++)
    {
        if(data[i]==element)
```



```
        return true;

    }

    return false;

}

template<class T>
int SET<T>::size()
{
    return n+1;
}

int main() {

    SET<int> s1,s2,s3;

    int choice;

    int element;

    cout<<"\nEnter number of elements in SET1:";
    cin>>element;//element is used for taking size
    s1.input(element);

    cout<<"\nEnter number of elements in SET2:";
    cin>>element;//element is used for taking size
    s2.input(element);

    do

    {
```

```
cout<<"\n***** SET OPERATIONS *****"

    <<"\n1.Insert"

    <<"\n2.Remove"

    <<"\n3.Search"

    <<"\n4.Size of Set"

    <<"\n5.Intersection"

    <<"\n6.Union"

    <<"\n7.Difference"

    <<"\n8.Check if Subset"

    <<"\nEnter Your Choice: ";

cin>>choice;

switch(choice)

{

case 1:

    cout<<"\nEnter Element: ";

    cin>>element;

    if(s1.insert(element))

    {

        cout<<element<<" inserted";

    }

    else

    {
```

```
        cout<<"Insertion Failed";  
    }  
    break;  
case 2:  
    cout<<"\nEnter Element: ";  
    cin>>element;  
    if(s1.remove(element))  
    {  
        cout<<element<<" deleted";  
    }  
    else  
    {  
        cout<<"Deletion Failed";  
    }  
    break;  
case 3:  
    cout<<"\nEnter Element: ";  
    cin>>element;  
    if(s1.contains(element))  
    {  
        cout<<element<<" is present";  
    }
```

```
        else
        {
            cout<<element<<"is not Present";
        }
        break;
case 4:
    cout<<"\nSize = "<<s1.size();
    break;
case 5:
    s3=s1.intersection(s1,s2);
    cout<<"\nSET 1's elements: ";
    s1.print();
    cout<<"\nSET 2's elements: ";
    s2.print();
    cout<<"\nIntersection: :";
    s3.print();
    break;

case 6:

    s3=s1.unionS(s1,s2);
    cout<<"\nSET 1's elements: ";
```

```
        s1.print();

        cout<<"\nSET 2's elements: ";

        s2.print();

        cout<<"\nUnion :";

        s3.print();

        break;

    case 7:

        s3=s1.difference(s1,s2);

        cout<<"\nSET 1's elements: ";

        s1.print();

        cout<<"\nSET 2's elements: ";

        s2.print();

        cout<<"\nDifference :";

        s3.print();

        break;

    }

    }while(choice!=0);

    return 0;

}
```

Output Screenshots:-

```
CAUsers\USER\Documents\skill1.exe
Enter number of elements in SET1:5
Enter Element: 1 67
Enter Element: 2 89
Enter Element: 3 12
Enter Element: 4 6
Enter Element: 5 51
Enter number of elements in SET2: 5
Enter Element: 16
Enter Element: 2 11
Enter Element: 3 67
Enter Element: 4 90
Enter Element: 5 10
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
5.Intersection
6.Union
7.Difference
8.Check if Subset
Enter Your Choice: 5
SET 1's elements: 67 89 12 6 51
SET 2's elements: 6 11 67 90 10
Intersection: : 67 6
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
```

```
CAUsers\USER\Documents\skill1.exe
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
5.Intersection
6.Union
7.Difference
8.Check if Subset
Enter Your Choice: 2
Enter Element: 12
12 deleted
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
5.Intersection
6.Union
7.Difference
8.Check if Subset
Enter Your Choice: 3
Enter Element: 11
11is not Present
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
5.Intersection
6.Union
7.Difference
8.Check if Subset
Enter Your Choice: 4
Size = 1
***** SET OPERATIONS *****
1.Insert
2.Remove
3.Search
4.Size of Set
5.Intersection
```

Conclusion:- Thus,we have studied different operations on set ADT.

ASSIGNMENT NO. 2

AIM: Construct a threaded binary search tree.

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

THEORY: Insertion in Binary threaded tree is similar to insertion in binary tree but we will have to adjust the threads after insertion of each element. 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). ALGORITHM: Insertion Operation Step 1: Start Step 2: If the root is null then create root node return Step 3: If the root exists then Comparer the data with root.data Step 4: While insert position is located If data is greater than node.data Go to right subtree Else Go to left subtree End while Step 5: Insert data Step 6: Stop.

Inorder Traversal Step 1: Start Step 2: current=leftmost->root Step 3: While current is not null OR current is not equal to head Print current->data.

Step 4: If current->rbit is true then current= current->right else current=leftmost->(current-right) Step 5: Stop.

PROGRAM:

```
#include<iostream>

using namespace std;

class TBT
{
    private:
        int data;
        int lbit,rbit;
        TBT*lchild,*rchild;
    public:
        TBT*create_TBT(TBT*,TBT*);
        void inorder_TBT(TBT*,TBT*);
        void preorder_TBT(TBT*,TBT*);
};

void TBT::inorder_TBT(TBT*root,TBT*header)
{
    int flag=0;
    TBT*trav;
    trav=root;

    while(trav!=header)
    {
        while(trav->lbit!=1&&flag==0)
```



```
        {  
            trav=trav->lchild; // go to left  
        }  
  
        cout<<" "<<trav->data; // print leftmost data  
  
        if(trav->rbit==0)  
        {  
            trav=trav->rchild;  
            flag=0;  
        }  
        else  
        {  
            trav=trav->rchild;  
            flag=1;  
        }  
    }  
}
```

```
TBT*TBT::create_TBT(TBT*root,TBT*header)
```

```
{  
    TBT*trav,*temp,*p;
```

```
int attached_flag=0;

char ans;

while(1)
{
    trav=root;
    if(root==NULL)
    {
        temp=new TBT();
        cout<<"\nEnter the data ";
        cin>>temp->data;
        temp->lbit=1;
        temp->rbit=1;
        temp->lchild=header;
        temp->rchild=header;
        root=temp;
    }
    else
    {
        temp=new TBT();
        cout<<"\nEnter the data ";
        cin>>temp->data;
```

```
temp->lbit=1;

temp->rbit=1;

attached_flag=0;


trav=root;

while(attached_flag==0)
{
    if(trav->data<temp->data&&trav->rbit==0)
    {
        trav=trav->rchild;
    }
    else
    if(trav->data<temp->data&&trav->rbit==1)
    {
        trav->rbit=0;
        p=trav->rchild;
        trav->rchild=temp;
        temp->rchild=p;
        temp->lchild=trav;
        attached_flag=1;
    }
}
```

```
        if(trav->data>temp->data&&trav->lbit==0)
        {
            trav=trav->lchild;
        }
        else
        if(trav->data>temp->data&&trav->lbit==1)
        {
            trav->lbit=0;
            p=trav->lchild;
            trav->lchild=temp;
            temp->lchild=p;
            temp->rchild=trav;
            attached_flag=1;
        }
    }

    attached_flag=0;
}

cout<<"\nDo you want to attach more nodes [y/n] ";
cin>>ans;

if(ans=='n' || ans=='N')

    break;
```

```
    }

    return root;
}

int main()
{
    int choice;

    TBT*root=NULL,obj,*header=NULL;

    do
    {
        cout<<"\n1.Create TBT";

        cout<<"\n2.Inorder TBT";

        cout<<"\n3.Exit";

        cout<<"\nEnter your choice :";

        cin>>choice;

        switch(choice)
        {

            case 1: root=obj.create_TBT(root,header);

                    break;

            case 2: obj.inorder_TBT(root,header);

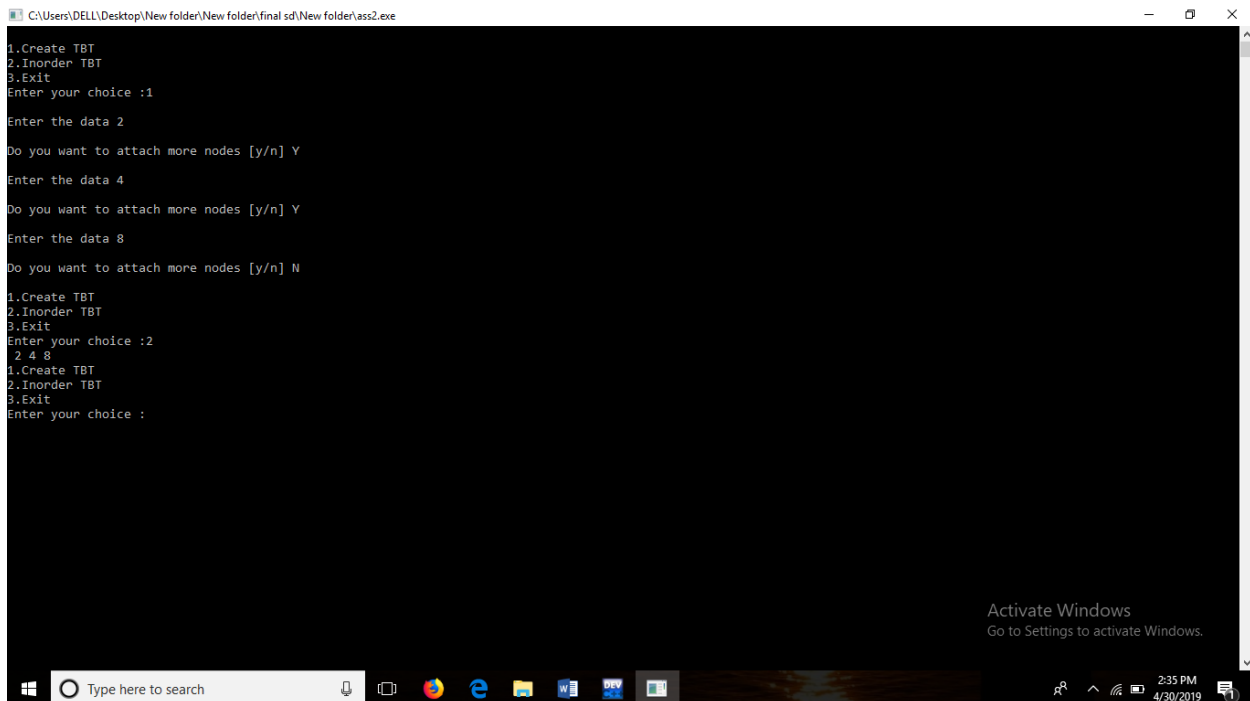
                    break;

        }

    }
```

```
    }  
  
    while(choice!=0);  
  
}
```

OUTPUT:



```
C:\Users\DELL\Desktop\New folder\New folder\final sd\New folder\ass2.exe  
1.Create TBT  
2.Inorder TBT  
3.Exit  
Enter your choice :1  
  
Enter the data 2  
  
Do you want to attach more nodes [y/n] Y  
  
Enter the data 4  
  
Do you want to attach more nodes [y/n] Y  
  
Enter the data 8  
  
Do you want to attach more nodes [y/n] N  
  
1.Create TBT  
2.Inorder TBT  
3.Exit  
Enter your choice :2  
2 4 8  
1.Create TBT  
2.Inorder TBT  
3.Exit  
Enter your choice :
```

CONCLUSION: Successfully implemented a TBT, inserted values in given order and traversed it in inorder traversal using threads.

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 Use adjacency list representation of the graph or use adjacency matrix representation of the graph. Justify the storage representations used.

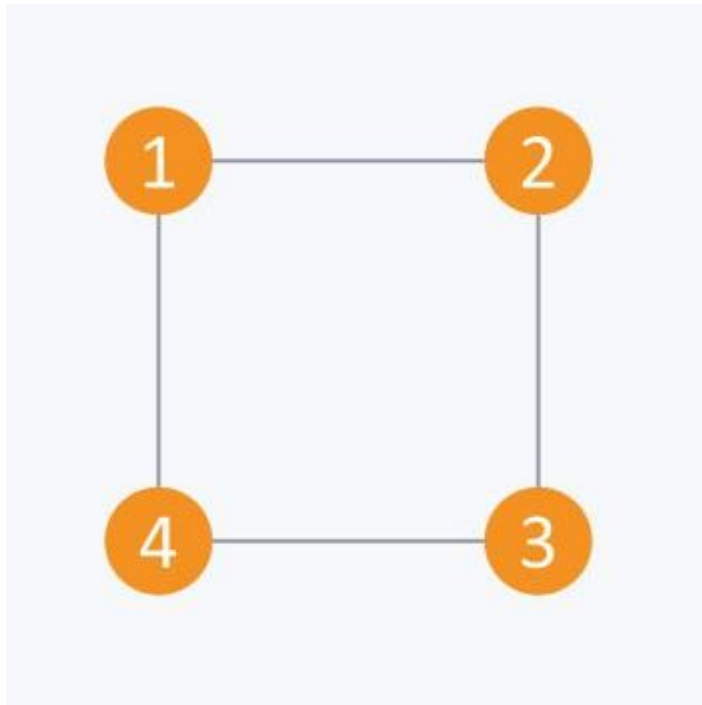
Theory:-

The other way to represent a graph is by using an adjacency list. An adjacency list is an array A of separate lists. Each element of the array A_i is a list, which contains all the vertices that are adjacent to vertex i .

For a weighted graph, the weight or cost of the edge is stored along with the vertex in the list using pairs. In an undirected graph, if vertex j is in list A_i then vertex i will be in list A_j .

The space complexity of adjacency list is $O(V + E)$ because in an adjacency list information is stored only for those edges that actually exist in the graph. In a lot of cases, where a matrix is sparse using an adjacency matrix may not be very useful. This is because using an adjacency matrix will take up a lot of space where most of the elements will be 0, anyway. In such cases, using an adjacency list is better.

Note: A sparse matrix is a matrix in which most of the elements are zero, whereas a dense matrix is a matrix in which most of the elements are non-zero.



Algorithm:-

1. Create an array A of size N and type of array must be list of vertices. Initially each list is empty so each array element is initialised with empty list.
2. Iterate each given edge of the form (u,v) and append v to the uth list of array A. Also, If graph is undirected append u to the vth list of array A

Code:-

```
#include<iostream>

#define MAX 10

using namespace std;

class airport
{
```



```
string city[MAX];

int distance[10][10];

public :

    int n;

    airport();

    void input_g();

    void output_g();

};

airport :: airport()
{
    n = 0;

    for(int i=0 ; i<MAX ; i++)
    {
        for(int j=0 ; j<MAX ; j++)

            distance[i][j] = 0;

    }

}

void airport :: input_g()
```

```
{  
    int k;  
  
    cout << "\nEnter the no. of cities: " ;  
    cin >> n;  
    cout<<"Enter city name:";  
    for(int k=0;k<n;k++)  
    {  
  
        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]<<":\n";
```

```
        cin >> distance[i][j];

        distance[j][i] = distance[i][j];

    }

}

}

void airport :: output_g()
{
    cout<<" ";
    for(int k=0;k<n;k++)
    {
        cout<<city[k]<<" ";

    }

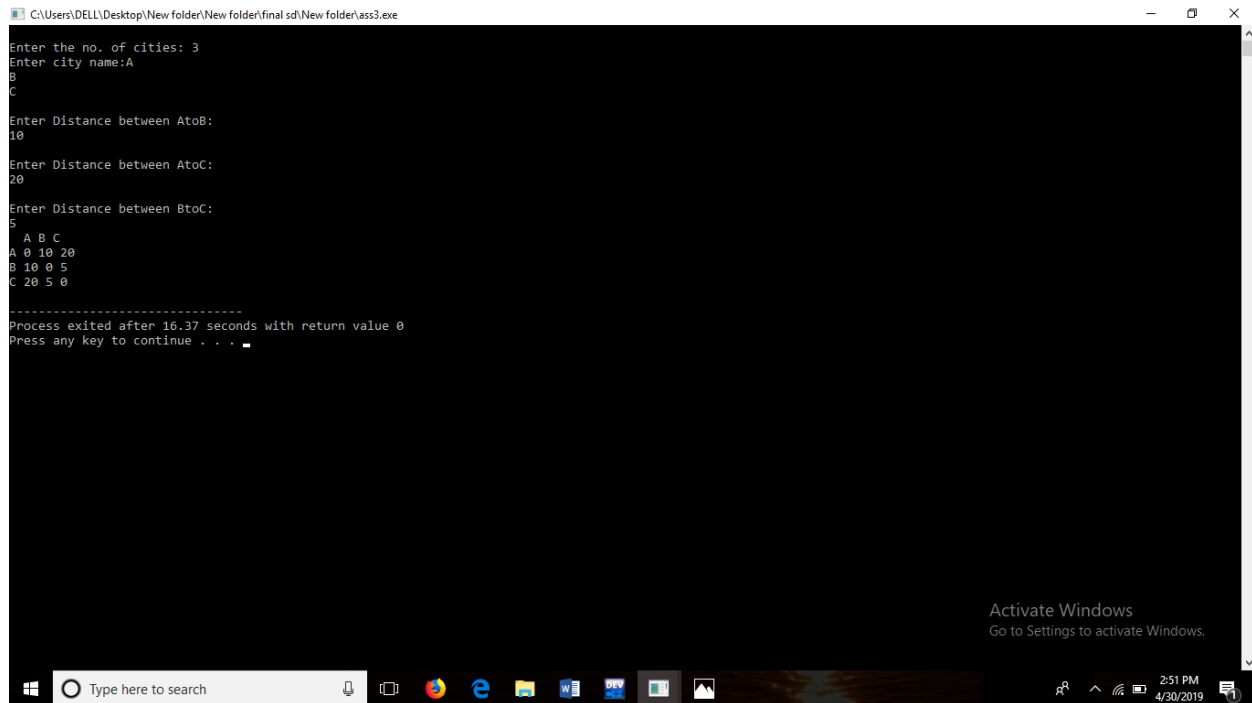
    cout << "\n";

    for(int i=0 ; i<n ; i++)
    {
        cout <<city[i] <<" ";

        for(int j=0 ; j<n ; j++)
```

```
        {  
            cout << distance[i][j] << " ";  
        }  
        cout << " ";  
  
        cout << "\n";  
    }  
  
}  
  
int main()  
{  
    airport obj;  
    obj.input_g();  
    obj.output_g();  
}
```

Output Screenshot:-



```
C:\Users\DELL\Desktop\New folder\New folder\final sd\New folder\ass3.exe
Enter the no. of cities: 3
Enter city name:A
B
C
Enter Distance between AtoB:
10
Enter Distance between AtoC:
20
Enter Distance between BtoC:
5
A B C
A 0 10 20
B 10 0 5
C 20 5 0
-----
Process exited after 16.37 seconds with return value 0
Press any key to continue . . .
```

Conclusion:-

We Conclude That We Can Use Adjacency List To Show If Route Exists Between Any Particular Cities Or Not.

Assignment 4

Aim:

For a weighted graph G, find the minimum spanning tree using Prims algorithm

Objective:

To find minimum distance between to nodes using Prims Algorithm.

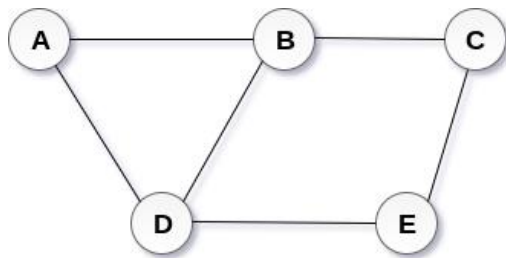
Theory:

Graph

A graph can be defined as group of vertices and edges that are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent child relationship.

Definition

A graph G can be defined as an ordered set $G(V, E)$ where $V(G)$ represents the set of vertices and $E(G)$ represents the set of edges which are used to connect these vertices.



Undirected Graph

Spanning Tree

Spanning tree can be defined as a sub-graph of connected, undirected graph G that is a tree produced by removing the desired number of edges from a graph. In other words, Spanning tree is a non-cyclic sub-graph of a connected and undirected graph G that connects all the vertices together. A graph G can have multiple spanning trees.

Minimum Spanning Tree

There can be weights assigned to every edge in a weighted graph. However, A minimum spanning tree is a spanning tree which has minimal total weight. In other words, minimum spanning tree is the one which contains the least weight among all other spanning tree of some particular graph.

Prim's Algorithm

Prim's Algorithm is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

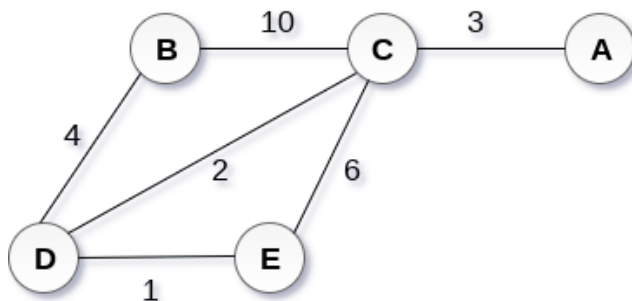
Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

Algorithm:

1. Step 1: Select a starting vertex
2. Step 2: Repeat Steps 3 and 4 until there are fringe vertices
3. Step 3: Select an edge e connecting the tree vertex and fringe vertex that has minimum weight
4. Step 4: Add the selected edge and the vertex to the minimum spanning tree T
[END OF LOOP]
5. Step 5: EXIT

Example :

Construct a minimum spanning tree of the graph given in the following figure by using prim's algorithm.



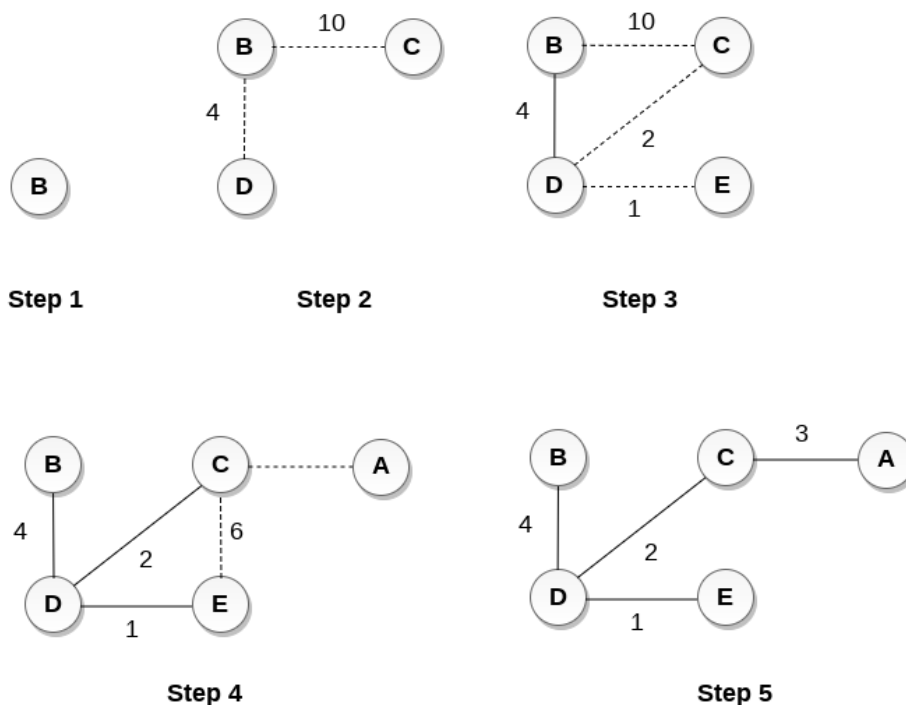
Solution

- Step 1 : Choose a starting vertex B.
- Step 2: Add the vertices that are adjacent to A. the edges that connecting the vertices are shown by dotted lines.
- Step 3: Choose the edge with the minimum weight among all. i.e. BD and add it to MST. Add the adjacent vertices of D i.e. C and E.
- Step 3: Choose the edge with the minimum weight among all. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C i.e. E and A.
- Step 4: Choose the edge with the minimum weight i.e. CA. We can't choose CE as it would cause cycle in the graph.

The graph produces in the step 4 is the minimum spanning tree of the graph shown in the above figure.

The cost of MST will be calculated as;

$$\text{cost(MST)} = 4 + 2 + 1 + 3 = 10 \text{ units.}$$



Program :


```
#include <iostream>
```

```
using namespace std;
```

```
class Graph
```

```
{
```

```
    private:
```

```
        int nVertices,choiceOfEdge,startVertex,endVertex;
```

```
        int **graph;
```

```
        int *visited;
```

```
    public :
```

```
        Graph(int nVertices)
```

```
        {
```

```
            this ->nVertices = nVertices;
```

```
            graph = new int*[nVertices];
```

```
            for(int i=0;i<nVertices;i++)
```

```
            {
```

```
                graph[i] = new int [nVertices];
```

```
            }
```

```
            //initialize graph matrix with 0
```

```
        for(int i=0;i<nVertices;i++)
        {
            for(int j=0;j<nVertices;j++)
            {
                (*(graph+i)+j)=0;
            }
        }

        visited = new int[nVertices];
    }

    int** createGraph(int nVertices)
    {
        bool hasNextEdge = true;

        //input edge weights
        while(hasNextEdge)
        {
            cout << "\t1.Enter a Edge in graph" << endl;

            cout << "\t2.Exit" << endl;

            cin >> choiceOfEdge;

            switch(choiceOfEdge)
            {
```

case 1:

```
        cout << "\tEnter first vertex" << endl;

        cin >> startVertex;

        cout << "\tEnter End vertex" << endl;

        cin >> endVertex;

        cout << "\tEnter edge Weight" << endl;

        cin >> *(* (graph+startVertex)+endVertex);

        *(* (graph+endVertex)+startVertex) =
*(* (graph+startVertex)+endVertex);

        hasNextEdge = true;

        break;
```

case 2:

```
        hasNextEdge = false;

    }

}

//assign 999 to not present edges

for(int i=0;i<nVertices;i++)

{

    for(int j=0;j<nVertices;j++)
```

```
        {  
            if(*(*(graph+i)+j) == 0)  
                *(*(graph+i)+j) = 999;  
        }  
    }  
    return graph;  
}
```

```
void display()  
{  
    for(int i=0;i<nVertices;i++)  
    {  
        for(int j=0;j<nVertices;j++)  
        {  
            cout << *(*(graph+i)+j) << " ";  
        }  
        cout << endl;  
    }  
}
```

```
int minKey(int bucket[],int *visited)
{
    int min=999,minIndex;
    for(int i=0;i<nVertices;i++)
    {
        if(*(visited+i)==0 && bucket[i]<min)
        {
            min = bucket[i];
            minIndex = i;
        }
    }
    return minIndex;
}
```

```
void primAlgo()
{
    int mst[nVertices];
    int bucket[nVertices];

    for(int i=0;i<nVertices;i++)
    {
```

```
        bucket[i]=999;

        *(visited+i)=0;

    }

    bucket[0]=0;

    mst[0]=-1;

    for(int j=0;j<nVertices-1;j++)

    {

        int min = minKey(bucket,visited);

        visited[min]=1;

        for(int k=0;k<nVertices;k++)

        {

            if(*(visited+k)==0 && (*(graph+min)+k) <

bucket[k])

            {

                mst[k]=min;

                bucket[k]=*(*(graph+min)+k);

            }

        }

    }
```

```
        int cost=0;

        cout << endl;

        cout << "\t EDGE" << "\tWEIGHT" << endl;

        for(int i=1;i<nVertices;i++)

        {

                cout << "\t" <<mst[i] << " - " << i << "\t " <<
*(*(graph+i)+mst[i]) << endl;

                cost += *(*(graph+i)+mst[i]);

        }

        cout << endl;

        cout << "\tThe cost of minimun spanning tree is " << cost <<
endl;

    }

};
```

```
int main()
```

```
{
```

```
    int nVertices,choice;
```

```
char ch='y';

cout << "\tEnter the number of vertices" << endl;

cin >> nVertices;

Graph graph(nVertices);

while(ch == 'y')
{
    cout << "\tMENU"<< endl;

    cout << "\t1.Create a graph" << endl;

    cout << "\t2.Prims Algorithm on created graph" << endl;

    cout << "\t3.Display" << endl;

    cin >> choice;

    switch(choice)
    {

        case 1:

            graph.createGraph(nVertices);

            break;

        case 2:

            graph.primAlgo();

            break;

        case 3:

            graph.display();
```



```
        break;

    default:

        cout << "\\tINVALID CHOICE" << endl;

    }

    cout << "\\tDo you wish to continue" << endl;

    cout << "\\tententer y if yes" << endl;

    cin >> ch;

}

}
```

Output:

```

E:\codeblocksprogram\PrimsJugal\bin\Debug\PrimsJugal.exe
Enter the number of vertices
5
MENU
1.Create a graph
2.Prims Algorithm on created graph
3.Display
1
1.Enter a Edge in graph
2.Exit
1
Enter first vertex
0
Enter End vertex
1
Enter edge Weight
34
1.Enter a Edge in graph
2.Exit
1
Enter first vertex
0
Enter End vertex
2
Enter edge Weight
56
1.Enter a Edge in graph
2.Exit
1
Enter first vertex
1
Enter End vertex
2
Enter edge Weight
3
1.Enter a Edge in graph
2.Exit
1
Enter first vertex
2
Enter End vertex
4
Enter edge Weight
23
1.Enter a Edge in graph
2.Exit
1
Enter first vertex
3
Enter End vertex
4
E:\codeblocksprogram\PrimsJugal\bin\Debug\PrimsJugal.exe
Enter edge Weight
1
1.Enter a Edge in graph
2.Exit
2
Do you wish to continue
enter y if yes
y
MENU
1.Create a graph
2.Prims Algorithm on created graph
3.Display
2
EDGE    WEIGHT
0 - 1    34
1 - 2     3
4 - 3     1
2 - 4    23
The cost of minimun spanning tree is 61
Do you wish to continue
enter y if yes
y
MENU
1.Create a graph
2.Prims Algorithm on created graph
3.Display
3
999 34 56 999 999
34 999 3 999 999
56 3 999 999 23
999 999 999 999 1
999 999 23 1 999
Do you wish to continue
enter y if yes
n
Process returned 0 (0x0)   execution time : 184.183 s
Press any key to continue.

```

Conclusion:

Graphs are nonlinear data structures that make operations on large data easier.

Minimum spanning trees are useful for many real time applications where a minimum of many possible outcomes is required.

ASSIGNMENT NO.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 study the use of kruskal's and prims algorithm in given problem.

Theory:- *What is Minimum Spanning Tree?*

Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

How many edges does a minimum spanning tree has?

A minimum spanning tree has $(V - 1)$ edges where V is the number of vertices in the given graph.

What are the applications of Minimum Spanning Tree?

See this for applications of MST.

Below are the steps for finding MST using Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are $(V-1)$ edges in the spanning tree.

The step#2 uses Union-Find algorithm to detect cycle.

The algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Algorithm:-

- create a forest F (a set of trees), where each vertex in the graph is a separate tree
- create a set S containing all the edges in the graph
- while S is nonempty and F is not yet spanning
 - remove an edge with minimum weight from S
 - if the removed edge connects two different trees then add it to the forest F , combining two trees into a single tree

At the termination of the algorithm, the forest forms a minimum spanning forest of the graph. If the graph is connected, the forest has a single component and forms a minimum spanning tree.

Program Code:-

```
#include <iostream>

#include<iomanip>

using namespace std;

const int MAX=10;

class EdgeList; //forward declaration

class Edge      //USED IN KRUSKAL

{

    int u,v,w;

public:

    Edge(){} //Empty Constructor

    Edge(int a,int b,int weight)
```

```
{
    u=a;
    v=b;
    w=weight;
}

friend class EdgeList;

friend class PhoneGraph;
};

//---- EdgeList Class -----

class EdgeList
{
    Edge data[MAX];
    int n;

public:
    friend class PhoneGraph;

    EdgeList()
    { n=0;}

    void sort();

    void print();

};

//----Bubble Sort for sorting edges in increasing weights' order ---//
```

```
void EdgeList::sort()
{
    Edge temp;
    for(int i=1;i<n;i++)
        for(int j=0;j<n-1;j++)
            if(data[j].w>data[j+1].w)
            {
                temp=data[j];
                data[j]=data[j+1];
                data[j+1]=temp;
            }
}

void EdgeList::print()
{
    int cost=0;
    for(int i=0;i<n;i++)
    {
        cout<<"\n"<<i+1<<" "<<data[i].u<<"--"<<data[i].v<<" = "<<data[i].w;
        cost=cost+data[i].w;
    }

    cout<<"\nMinimum cost of Telephone Graph = "<<cost;
}
```

//----- Phone Graph Class-----

class PhoneGraph

```
{  
    int data[MAX][MAX]={0, 28, 0, 0, 0,10,0},  
    {28,0,16,0,0,0,14},  
    {0,16,0,12,0,0,0},  
    {0,0,12,0,22,0,18},  
    {0,0,0,22,0,25,24},  
    {10,0,0,0,25,0,0},  
    {0,14,0,18,24,0,0},  
};  
    int n;
```

public:

```
    PhoneGraph(int num)  
{  
    n=num;  
}  
  
    void readgraph();  
    void printGraph();  
    int mincost(int cost[],bool visited[]);  
    int prim();
```

```
void kruskal(EdgeList &spanlist);

int find(int belongs[], int vertexno);

void unionComp(int belongs[], int c1,int c2);

};

void PhoneGraph::readgraph()
{
    cout<<"Enter Adjacency(Cost) Matrix: \n";
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n; j++)
            cin>>data[i][j];
    }
}

void PhoneGraph::printGraph()
{
    cout<<"\nAdjacency (COST) Matrix: \n";
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            cout<<setw(3)<<data[i][j];
        }
    }
}
```



```
        cout<<endl;

    }

}

int PhoneGraph::mincost(int cost[],bool visited[]) //finding vertex with minimum
cost
{

    int min=9999,min_index; //initialize min to MAX value(ANY) as temporary
    for(int i=0;i<n;i++)
    {

        if(visited[i]==0 && cost[i]<min)
        {

            min=cost[i];
            min_index=i;
        }

    }

    return min_index; //return index of vertex which is not visited and having
minimum cost

}

int PhoneGraph::prim()
{

    bool visited[MAX];
```

```
int parents[MAX]; //storing vertices

int cost[MAX]; //saving minimum cost

for(int i=0;i<n;i++)
{
    cost[i]=9999; //set cost as infinity/MAX_VALUE
    visited[i]=0; //initialize visited array to false
}

cost[0]=0; //starting vertex cost
parents[0]=-1; //make first vertex as a root

for(int i=0;i<n-1;i++)
{
    int k=mincost(cost,visited); //minimum cost elements index
    visited[k]=1; //set visited

    for(int j=0;j<n;j++)//for adjacent vertices comparison
    {
        if(data[k][j] && visited[j]==0 && data[k][j] < cost[j])
        {
            parents[j]=k;
            cost[j]=data[k][j];
        }
    }
}
```

```

        }
    }
}

cout<<"Minimum Cost Telephone Map:\n";
for(int i=1;i<n;i++)
{
    cout<<i<<" -- "<<parents[i]<<" = "<<cost[i]<<endl;
}

int mincost=0;
for (int i = 1; i < n; i++)
    mincost+=cost[i];           //data[i][parents[i]];

return mincost;
}

//----- Kruskal's Algorithm
void PhoneGraph::kruskal(EdgeList &spanlist)
{
    int belongs[MAX]; //Separate Components at start (No Edges, Only vertices)
    int cno1,cno2;    //Component 1 & 2
    EdgeList elist;
    for(int i=1;i<n;i++)
        for(int j=0;j<i;j++)
            {

```

```

        if(data[i][j]!=0)
        {
            elist.data[elist.n]=Edge(i,j,data[i][j]); //constructor for
initializing edge

            elist.n++;
        }
    }

    elist.sort(); //sorting in increasing weight order

    for(int i=0;i<n;i++)
        belongs[i]=i;

    for(int i=0;i<elist.n;i++)
    {
        cno1=find(belongs,elist.data[i].u); //find set of u
        cno2=find(belongs,elist.data[i].v); ////find set of v
        if(cno1!=cno2) //if u & v belongs to different sets
        {
            spanlist.data[spanlist.n]=elist.data[i]; //ADD Edge to spanlist
            spanlist.n=spanlist.n+1;
            unionComp(belongs,cno1,cno2); //ADD both components to
same set
        }
    }

```

```
        }
    }

void PhoneGraph::unionComp(int belongs[],int c1,int c2)
{
    for(int i=0;i<n;i++)
    {
        if(belongs[i]==c2)
            belongs[i]=c1;
    }
}

int PhoneGraph::find(int belongs[],int vertexno)
{
    return belongs[vertexno];
}

//----- MAIN PROGRAM-----

int main() {
    int vertices,choice;

    EdgeList spantree;

    cout<<"Enter Number of cities: ";

    cin>>vertices;

    PhoneGraph p1(vertices);
```

```
//p1.readgraph();  
  
do  
{  
  
    cout<<"\n1.Find Minimum Total Cost(By Prim's Algorithm)"  
        <<"\n2.Find Minimum Total Cost(by Kruskal's  
Algorithms)"  
  
        <<"\n3.Re-Read Graph(INPUT)"  
        <<"\n4.Print Graph"  
        <<"\n0. Exit"  
        <<"\nEnter your choice: ";  
  
    cin>>choice;  
    switch(choice)  
    {  
    case 1:  
        cout<<" Minimum cost of Phone Line to cities is: "<<p1.prim();  
        break;  
    case 2:  
        p1.kruskal(spantree);  
        spantree.print();  
        break;  
    case 3:  
        p1.readgraph();
```

```
                break;

            case 4:

                p1.printGraph();

                break;

            default:

                cout<<"\nWrong Choice!!!";

        }

    }while(choice!=0);


    return 0;

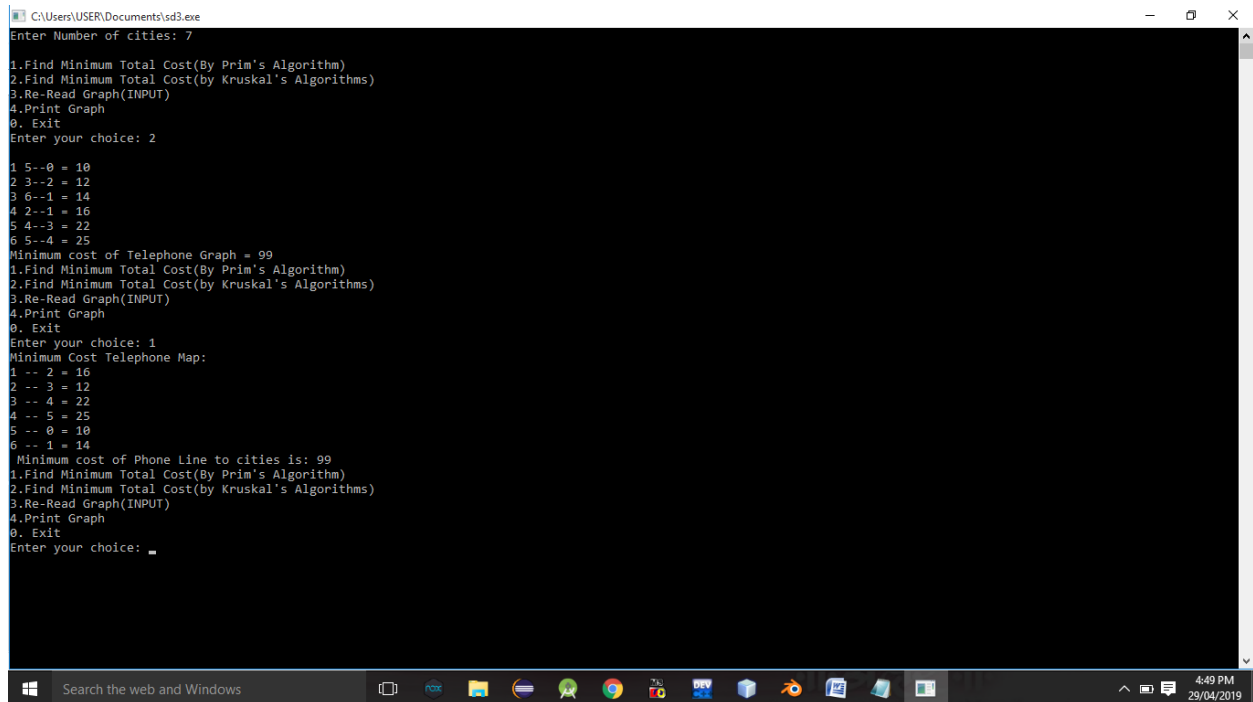
}

/*  Sample INPUT: vertices =7
*      {{0, 28, 0, 0, 0,10,0},
        {28,0,16,0,0,0,14},
        {0,16,0,12,0,0,0},
        {0,0,12,0,22,0,18},
        {0,0,0,22,0,25,24},
        {10,0,0,0,25,0,0},
        {0,14,0,18,24,0,0},
        };

    Minimum Cost: 99

*/
```

Output Screenshots:-



```
C:\Users\USER\Documents\sd3.exe
Enter Number of cities: 7
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 2
1 5--0 = 10
2 3--2 = 12
3 6--1 = 14
4 2--1 = 16
5 4--3 = 22
6 5--4 = 25
Minimum cost of Telephone Graph = 99
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: 1
Minimum Cost Telephone Map:
1 -- 2 = 16
2 -- 3 = 12
3 -- 4 = 22
4 -- 5 = 25
5 -- 0 = 10
6 -- 1 = 14
Minimum cost of Phone Line to cities is: 99
1.Find Minimum Total Cost(By Prim's Algorithm)
2.Find Minimum Total Cost(by Kruskal's Algorithms)
3.Re-Read Graph(INPUT)
4.Print Graph
0. Exit
Enter your choice: _
```

Conclusion:- Thus,we have studied implementation of kruskal's algorithm.

Assignment No. 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: Understand the heap data structure for heap sorting.

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

Since a Binary Heap is a Complete Binary Tree, it can be easily represented as array and array based representation is space efficient. If the parent node is stored at index l , the left child can be calculated by $2 * l + 1$ and right child by $2 * l + 2$ (assuming the indexing starts at 0).

EXAMPLE: $A = [7, 4, 3, 1, 2]$

Algorithm: 1. MAX-HEAPIFY(A, i, n) $l \leftarrow \text{LEFT}(i)$ $r \leftarrow \text{RIGHT}(i)$ if $l \leq n$ and $A[l] > A[i]$ then $\text{largest} \leftarrow l$ else $\text{largest} \leftarrow i$ if $r \leq n$ and $A[r] > A[\text{largest}]$ then $\text{largest} \leftarrow r$ if $\text{largest} \neq i$ then exchange $A[i] \leftrightarrow A[\text{largest}]$ MAX-HEAPIFY($A, \text{largest}, n$)
 2. BUILD-MAX-HEAP(A) $n = \text{length}[A]$ for $i \leftarrow \text{floor}(n/2)$ downto 1 do MAX-HEAPIFY(A, i, n)

3. HEAPSORT(A) BUILD-MAX-HEAP(A) for $i \leftarrow \text{length}[A]$ downto 2 do exchange $A[1] \leftrightarrow A[i]$ MAX-HEAPIFY($A, 1, i - 1$)

Code:

```
#include<iostream>

using namespace std;

class heap
{
public:
void printarray(int a[], int n);
void heapsort(int a[], int n);
void minimum(int a[],int n);
void maximum(int a[],int n);
};

void heapify(int a[],int n,int i);
void heap:: heapsort(int a[], int n)
{
    for(int i=(n/2)-1; i>=0;i--)
    {
        heapify(a,n,i);
    }
    for(int i=(n-1);i>=0;i--)
    {
```

```
    int temp= a[0];  
    a[0]= a[i];  
    a[i]= temp;  
    heapify (a,i,0);  
}  
}  
  
void heapify(int a[],int n, int i)  
{  
    int largest=i;  
    int l= (2*i)+1;  
    int r=(2*i)+2;  
    if(l<n && a[l]>a[largest])  
        largest=l;  
    if(r<n && a[r]>a[largest])  
        largest=r;  
  
    if(largest!=i)  
    {  
        int t= a[i];  
        a[i]=a[largest];  
        a[largest]=t;  
        heapify(a,n,largest);  
    }
```

```
    }  
}  
void heap:: printarray(int a[],int n)  
{  
    for(int i=0;i<n;i++)  
    {  
        cout<<a[i]<<" ";  
        cout<<"\n";  
    }  
}  
void heap::maximum(int a[],int n)  
{  
    cout<<"MAXIMUM MARKS:"<<a[n-1]<<endl;  
}  
void heap::minimum(int a[],int n)  
{  
    cout<<"MINIMUM MARKS:"<<a[0]<<endl;  
}  
int main()  
{  
    heap h;  
    int a[100],n;
```

```
cout<<"Enter number of students"<<endl;

cin>>n;

cout<<"enter the marks"<<endl;

for(int i=0;i<n;i++)

{

    cin>>a[i];

}

cout<<"HEAP SORT"<<endl;

h.heapsort(a,n);

cout<<"DISPLAY THE HEAP"<<endl;

h.printarray(a,n);

char ch;

int choice;

cout<<"DO YOU WANT TO SEE MAXIMUM OR MINIMUM MARKS(y/n)"<<endl;

cin>>ch;

while(ch=='y')

{

    cout<<"MENU"<<endl;

    cout<<"1.MAXIMUM MARKS"<<endl;

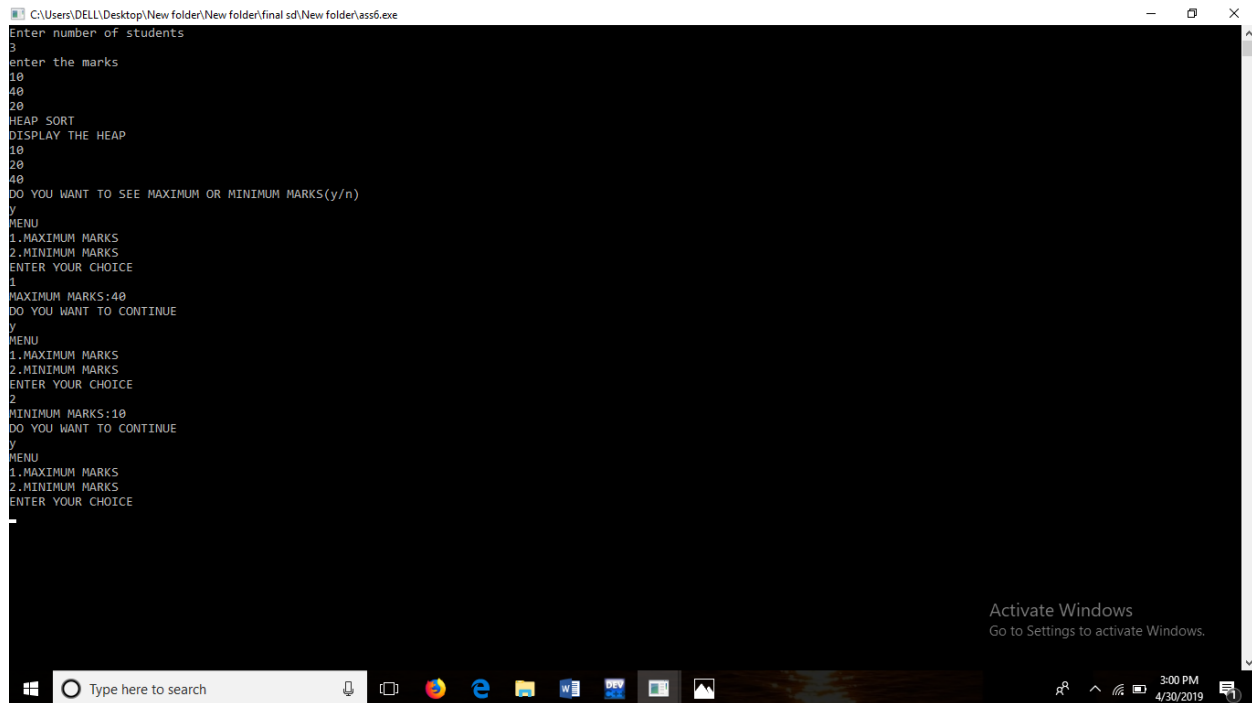
    cout<<"2.MINIMUM MARKS"<<endl;

    cout<<"ENTER YOUR CHOICE"<<endl;

    cin>>choice;
```

```
switch(choice)
{
    case 1:
        h.maximum(a,n);
        break;
    case 2:
        h.minimum(a,n);
        break;
    default:
        cout<<"SORRY!WRONG CHOICE"<<endl;
        break;
}
cout<<"DO YOU WANT TO CONTINUE"<<endl;
cin>>ch;
}
return 0;
}
```

Output:



```
C:\Users\DELL\Desktop\New folder\New folder\final sd\New folder\ass6.exe
Enter number of students
3
enter the marks
10
40
20
HEAP SORT
DISPLAY THE HEAP
10
20
40
DO YOU WANT TO SEE MAXIMUM OR MINIMUM MARKS(y/n)
y
MENU
1.MAXIMUM MARKS
2.MINIMUM MARKS
ENTER YOUR CHOICE
1
MAXIMUM MARKS:40
DO YOU WANT TO CONTINUE
y
MENU
1.MAXIMUM MARKS
2.MINIMUM MARKS
ENTER YOUR CHOICE
2
MINIMUM MARKS:10
DO YOU WANT TO CONTINUE
y
MENU
1.MAXIMUM MARKS
2.MINIMUM MARKS
ENTER YOUR CHOICE
1
```

Conclusion:- Thus,we have studied heap data structure,

ASSIGNMENT NO.7.

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

Objective:- to study the hashing concept and its techniques.

Theory:-

Open Addressing—

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).

Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.

Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

Delete(k): *Delete operation is interesting.* If we simply delete a key, then search may fail. So slots of deleted keys are marked specially as "deleted".

Insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Open Addressing is done following ways:

a) *Linear Probing:* In linear probing, we linearly probe for next slot. For example, typical gap between two probes is 1 as taken in below example also.

let $hash(x)$ be the slot index computed using hash function and S be the table size

If slot $hash(x) \% S$ is full, then we try $(hash(x) + 1) \% S$

If $(hash(x) + 1) \% S$ is also full, then we try $(hash(x) + 2) \% S$

If $(hash(x) + 2) \% S$ is also full, then we try $(hash(x) + 3) \% S$

Double Hashing:-

Double hashing is a collision resolving technique in Open Addressed Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.

Double hashing can be done using :

*$(hash1(key) + i * hash2(key)) \% TABLE_SIZE$*

Here $hash1()$ and $hash2()$ are hash functions and $TABLE_SIZE$ is size of hash table.

(We repeat by increasing i when collision occurs)

First hash function is typically $hash1(key) = key \% TABLE_SIZE$

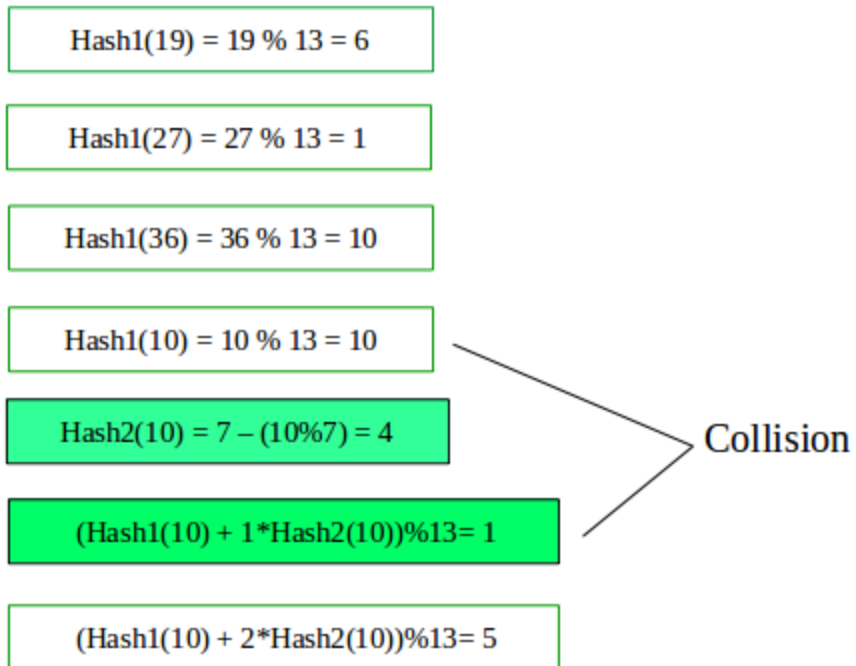
A popular second hash function is : $hash2(key) = PRIME - (key \% PRIME)$ where $PRIME$ is a prime smaller than the $TABLE_SIZE$.

A good second Hash function is:

- It must never evaluate to zero
- Must make sure that all cells can be probed

Lets say, $\text{Hash1}(\text{key}) = \text{key} \% 13$

$\text{Hash2}(\text{key}) = 7 - (\text{key} \% 7)$



Algorithm:-

Program Code:-

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
#define TABLE_SIZE 13
```

```
#define PRIME 7
```

```
class DoubleHash
```

```
{
```

```
    int *hashTable;
```

```
    int curr_size;
```

```
public:
```

```
    bool isFull()
```

```
    {
```

```
        return (curr_size == TABLE_SIZE);
```

```
    }
```

```
    int hash1(int key)
```

```
    {
```

```
        return (key % TABLE_SIZE);
```

```
    }
```

```
    int hash2(int key)
```

```
    {
```

```
    return (PRIME - (key % PRIME));  
}
```

DoubleHash()

```
{  
    hashTable = new int[TABLE_SIZE];  
    curr_size = 0;  
    for (int i=0; i<TABLE_SIZE; i++)  
        hashTable[i] = -1;  
}
```

void insertHash(int key)

```
{  
    if (isFull())  
        return;  
  
    int index = hash1(key);  
  
    if (hashTable[index] != -1)  
    {  
        int index2 = hash2(key);  
        int i = 1;
```

```
while (1)
{
    int newIndex = (index + i * index2) %
        TABLE_SIZE;

    if (hashTable[newIndex] == -1)
    {
        hashTable[newIndex] = key;
        break;
    }
    i++;
}

else
    hashTable[index] = key;
    curr_size++;
}

// function to display the hash table
void displayHash()
{
```

```
    for (int i = 0; i < TABLE_SIZE; i++)  
    {  
        if (hashTable[i] != -1)  
            cout << i << " --> "  
                << hashTable[i] << endl;  
        else  
            cout << i << endl;  
    }  
}  
};
```

// Driver's code

```
int main()
```

```
{
```

```
    int a[] = {19, 27, 36, 10, 64};
```

```
    int n = sizeof(a)/sizeof(a[0]);
```

```
    cout<<"inserted elements in hash table are as follows:";
```

```
    // inserting keys into hash table
```

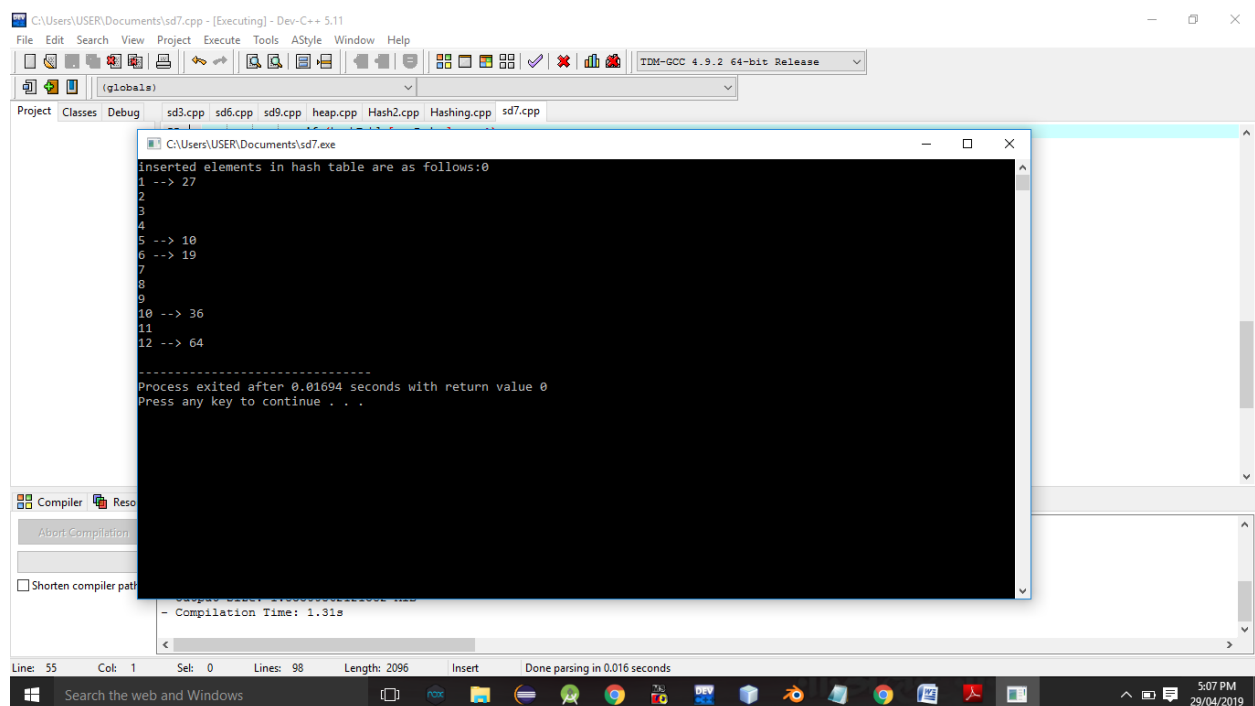
```
    DoubleHash h;
```

```
    for (int i = 0; i < n; i++)
```

```
        h.insertHash(a[i]);
```

```
// display the hash Table  
  
h.displayHash();  
  
return 0;  
  
}
```

Output Screenshots:-



```
C:\Users\USER\Documents\sd7.exe  
Inserted elements in hash table are as follows:  
1 --> 27  
2  
3  
4  
5 --> 10  
6 --> 19  
7  
8  
9  
10 --> 36  
11  
12 --> 64  
  
-----  
Process exited after 0.01694 seconds with return value 0  
Press any key to continue . . .  
  
Compilation Time: 1.31s  
Done parsing in 0.016 seconds
```

Conclusion:- Thus,we have studied double hashing and hashing technique.

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:-

Program Code:-

```
#include <iostream>

#include <fstream>

#include <cstring>

#include <iomanip>

#include<cstdlib>

#define max 50

using namespace std;

class Student

{

    char name[max];

    int rollNo;
```



```
int year;

int division;

char address[50];

friend class FileOperations;

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 FileOperations
{

    fstream file;

    public:FileOperations(char *name)

    {

        //strcpy(this->name,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 deleteRecord(int rollNo)  
{  
    ofstream newFile("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;  
        }  
        newFile.write((char *)&s,sizeof(s));  
    }  
    if(!flag)
```

```
        {
            cout<<"Element Not Present";
        }
        file.close();
        newFile.close();
        remove("student.txt");
        rename("new.txt","student.txt");
        file.open("student.txt",ios::in|ios::ate|ios::out|ios::binary);
    }
    ~FileOperations()
    {
        file.close();
        cout<<"Closing file..";
    }

};

int main()
{
    ofstream newFile("student.txt",ios::app|ios::binary);
    newFile.close();
    FileOperations file((char *)"student.txt");

    int rollNo,year,choice=0;
```

```
char division;

char name[max],address[max];

while(choice!=5)
{
    //clrscr();

    cout<<"\n*****Phone Book*****\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.deleteRecord(rollNo);

        break;
```

```
        case 5 :break;
    }

}

}
```

Output Screenshots:-

```
C:\Users\USER\Documents\sd3.exe
*****Phone Book*****
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 :
121 smita
Enter Year and Division :
2 c
Enter address :
bhor

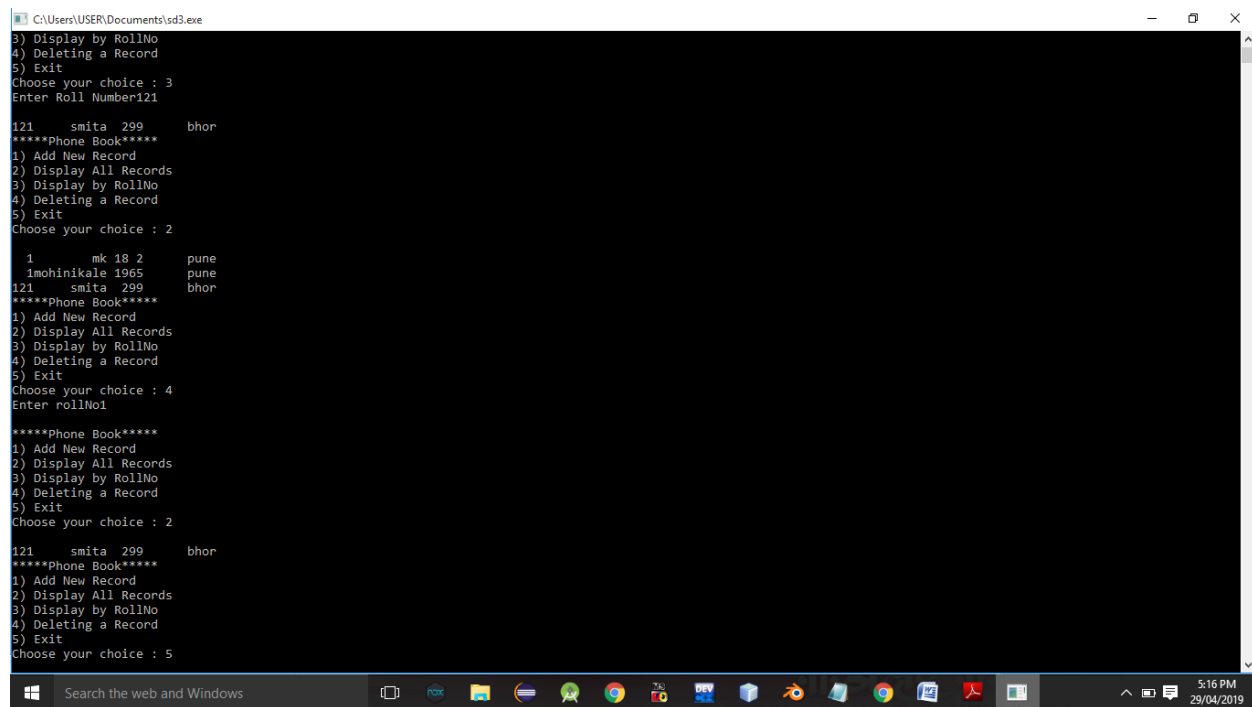
*****Phone Book*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 3
Enter Roll Number121

121 smita 299 bhor

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

1 mk 18 2 pune
1mohinikale 1965 pune
121 smita 299 bhor

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

```
C:\Users\USER\Documents\sd3.exe
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 3
Enter Roll Number:121

121    smita 299    bhor
*****Phone Book*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 2

1      mk 18 2      pune
1mohinikale 1965    pune
121    smita 299    bhor
*****Phone Book*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 4
Enter rollNo:1

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

121    smita 299    bhor
*****Phone Book*****
1) Add New Record
2) Display All Records
3) Display by RollNo
4) Deleting a Record
5) Exit
Choose your choice : 5
```

Conclusion:- Thus,this assignment implemented successfully.

ASSIGNMENT 9

Aim:-

Company maintains employee information as employee ID, name, designation and salary. Allow user to add, delete information of employee. Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.

Objective:-

To Display information of particular employee. If employee does not exist an appropriate message is displayed. If it is, then the system displays the employee details. Use index sequential file to maintain the data.

Theory:-

Indexing mechanisms are used to optimize certain accesses to data (records) managed in files. For example, the author catalog in a library is a type of index.

- Search Key (definition): attribute or combination of attributes used to look up records in a file.
- An Index File consists of records (called index entries) of the form search key value pointer to block in data file
- Index files are typically much smaller than the original file because only the values for search key and pointer are stored.
- There are two basic types of indexes: – Ordered indexes: Search keys are stored in a sorted order (main focus here in class). – Hash indexes: Search keys are distributed uniformly across “buckets” using a hash function.

Code:-

```
#include <iostream>

#include <fstream>

#include <cstring>

#include <iomanip>

#include<cstdlib>

#define max 50

using namespace std;

class Employee

{

    char name[max];

    int empid;

    int sal;
```

```
char de[50];

friend class FileOperations;

public:    Employee()
        {
            strcpy(name,"");
            empid=sal==0;
            strcpy(de,"");
        }

Employee(char name[max],int empid,int sal,char de[max])
{
    strcpy(this->de,de);
    strcpy(this->name,name);

    this->empid=empid;
    this->sal=sal;
}

int getEmpId()
{
    return empid;
}

void displayEmployeeData()
{
```

```

        cout<<endl<<empid<<"\t\t\t"<<name<<"\t\t\t"<<sal<<"\t\t\t"<<de;

    }

};

class FileOperations
{
    fstream file;

    public:FileOperations(char *name)
    {

        //strcpy(this->name,name);

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

    }

    void insertRecord(int empid,char name[max],int sal,char de[max])
    {

        Employee s=Employee(name,empid,sal,de);

        file.seekp(0,ios::end);

        file.write((char*)&s,sizeof(Employee));

        file.clear();

    }

    void displayAllRecords()
    {

```

```
Employee s;

file.seekg(0,ios::beg);

while(file.read((char *)&s,sizeof(Employee)))

{

    s.displayEmployeeData();

}

file.clear();

}

void displayRecord(int empid)

{

    Employee s;

    file.seekg(0,ios::beg);

    void *p;

    while(file.read((char *)&s,sizeof(Employee)))

    {

        if(s.empid==empid)

        {

            s.displayEmployeeData();

            break;

        }

    }

    if(p==NULL)
```

```
        throw "Element not present";

    file.clear();
}

void deleteRecord(int empid)
{
    ofstream newFile("new.txt",ios::binary);
    file.seekg(0,ios::beg);
    bool flag=false;
    Employee s;
    while(file.read((char *)&s,sizeof(s)))
    {
        if(s.empid==empid)
        {
            flag=true;
            continue;
        }
        newFile.write((char *)&s,sizeof(s));
    }
    if(!flag)
    {
        cout<<"Element Not Present";
    }
}
```

```
        file.close();

        newFile.close();

        remove("Employee.txt");

        rename("new.txt","Employee.txt");

        file.open("Employee.txt",ios::in|ios::ate|ios::out|ios::binary);

    }

    ~FileOperations()

    {

        file.close();

        cout<<"Closing file..";

    }

};

int main()

{

    ofstream newFile("Employee.txt",ios::app|ios::binary);

    newFile.close();

    FileOperations file((char *)"Employee.txt");

    int empid,sal,choice=0;

    char name[max],de[max];

    while(choice!=5)

    {
```

```
cout<<"\n\n1) 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 employee id and name : \n";
        cin>>empid>>name;
        cout<<"Enter sal \n";
        cin>>sal;
        cout<<"Enter designation : \n";
        cin>>de;
        file.insertRecord(empid,name,sal,de);
        break;

    case 2 :

        cout<<"Employee
ID"<<"\t\t"<<"Name"<<"\t\t"<<"Salary"<<"\t\t"<<"designation\n";
```



```
        cout<<"-----";

        file.displayAllRecords();

        break;

    case 3 :

        cout<<"Enter employee id";

        cin>>empid;

        try

        {

            file.displayRecord(empid);

        }

        catch(const char *str)

        {

            cout<<str;

        }

        break;

    case 4:

        cout<<"Enter employe id";

        cin>>empid;

        file.deleteRecord(empid);

        break;

    case 5 :break;

}
```

```
}  
  
}
```

OUTPUT:

```
C:\Users\USER\Documents\sd10.exe  
1) Add New Record  
2) Display All Records  
3) Display by RollNo  
4) Deleting a Record  
5) Exit  
Choose your choice : 1  
Enter employee id and name :  
213 Ramesh  
Enter sal  
90000  
Enter designation :  
manager  
1) Add New Record  
2) Display All Records  
3) Display by RollNo  
4) Deleting a Record  
5) Exit  
Choose your choice : 2  
Employee ID      Name      Salary      designation  
-----  
221      suresh      60000      cashier  
12       mk       10000      clerk  
213      Ramesh      90000      manager  
1) Add New Record  
2) Display All Records  
3) Display by RollNo  
4) Deleting a Record  
5) Exit  
Choose your choice : 4  
Enter employee id 12  
1) Add New Record  
2) Display All Records  
3) Display by RollNo  
4) Deleting a Record  
5) Exit  
Choose your choice : 4
```

Conclusion:- Thus, this assignment is completed successfully.