**ASSIGNMENT 1**

**AIM:-**

To create ADT that implement the "set" concept.

1. Add (newElement) -Place a value into the set
2. Remove (element)
3. Contains (element) Return true if element is in collection
4. Size () Return number of values in collection
5. Intersection of two sets
6. Union of two sets
7. Difference between two sets
8. Subset

**OBJECTIVE**:

To get the thorough understanding of the concepts of sets and the various operations performed on it

**THEORY:-**

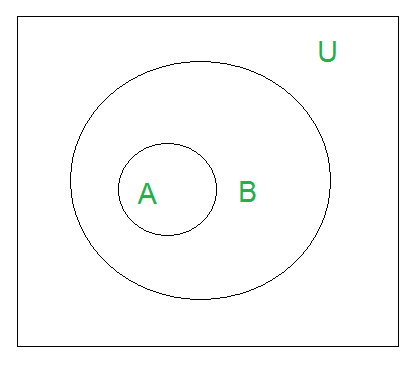
* **SET:-**

A **Set** is an unordered collection of objects, known as elements or members of the set.  
 An element ‘a’ belong to a set A can be written as ‘a ∈ A’,  ‘a∉ A’ denotes that a is not an element of the set A.

* **EQUAL SETS:-**

Two sets are said to be equal if both have same elements. For example A = {1, 3, 9, 7} and B = {3, 1, 7, 9} are equal sets.

* **SUBSET:-**

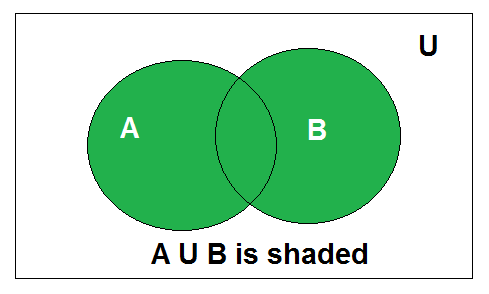
 A set A is said to be **subset** of another set B if and only if every element of set A is also a part of other set B.  
 Denoted by ‘**⊆**‘. ‘A ⊆ B ‘ denotes A is a subset of B.

* **SIZE OF A SET:-**  
   Size of a set can be finite or infinite.

Size of the set S is known as Cardinality number, denoted as |S|.

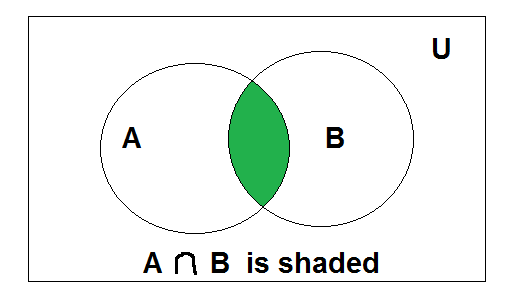
Example: Let A be a set of odd positive integers less than 10.  
 Solution : A = {1,3,5,7,9}, Cardinality of the set is 5, i.e.,|A| = 5.

* **UNION :** Union of the sets A and B, denoted by A ∪ B, is the set of distinct element belongs to set A or set B, or both.

****

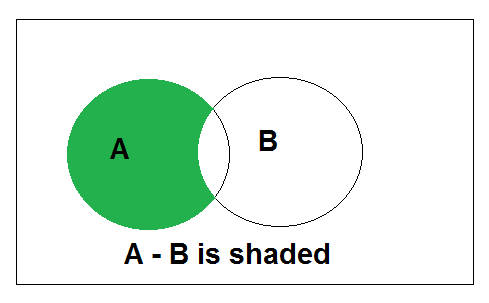
* **INTERSECTION:**

The intersection of the sets A and B, denoted by A∩ B, is the set of elements belongs to both A and B i.e. set of the common element in A and B

****

* **SET DIFFERENCE:-**

Difference between sets is denoted by ‘A – B’, is the set containing elements of set A but not in B. i.e all elements of A except the element of B.



**ALGORITHM:**

* FOR INTERSECTION:

*Step 1: Take an empty set (intersection set)*

*Step 2: pass each element of set-2 and the entire set-1 to the function member()*

*Step 3:if it returns true,*

*Add that element to the intersection set*

* FOR UNION:

*Step 1: Take an empty set (union set)*

*Step 2: copy all the elements of set1 to this new set*

*Step 3: traverse through the set2 and pass each element of set-2 along with the entire set-1 to the function member(),*

*and if it returns false then add that specified element to the union set*

* FOR CONTAINS:

*Step1 : take the number as input which you want to search*

*Step 2 : enter 1 for searching in set-1 or 2 for searching in set-2*

*Step 3 : initialise i=0*

*Step 4: traverse the set-1 or set-2 till the end depending on whether the input was 1 or 2 after passing the element and that set to the function member()*

*Step 5: if element found then display element is present*

* FOR SUBSET:

*Step 1: enter 1 if you want to check if set 2 is subset of 1, or enter 2 if you want to check if set-1 is subset of set-2*

*Step 2: depending on input we will traverse the set(which has to be the subset) until its end, by passing each element of this set and other set to the function member()*

*Step 3: if member() returns true, then continue*

*else return false*

*Step 4: if false, then display “it is a subset”*

*else display “it is not”*

* FOR DIFFERENCE :

*Step 1: Initialise the difference set to 0, difference set contains all the element which are in set-1 but not in set-2*

*Step 2: traverse the entire set-1 and, pass each element of this set and the the set-2 to the function member()*

*Step 3: if it returns false then add this element to the difference set*

* FOR REMOVE:

*Step 1: enter 1 or 2 for removing element from set-1 or set-2 respectively*

*Step 2: enter the index from which you want to remove the element*

*Step 3: if ,*

*entered position is less than the size of the set then move all the elements to their left from the position at which you want to remove the element and just decrease the size of the set else,*

*entered position is equal to the size of the set then just decrease the size*

* FOR SIZE:

*step 1: show the 0th index of the set which contains the size of our set*

**CPP SOURCE 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;i<=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();

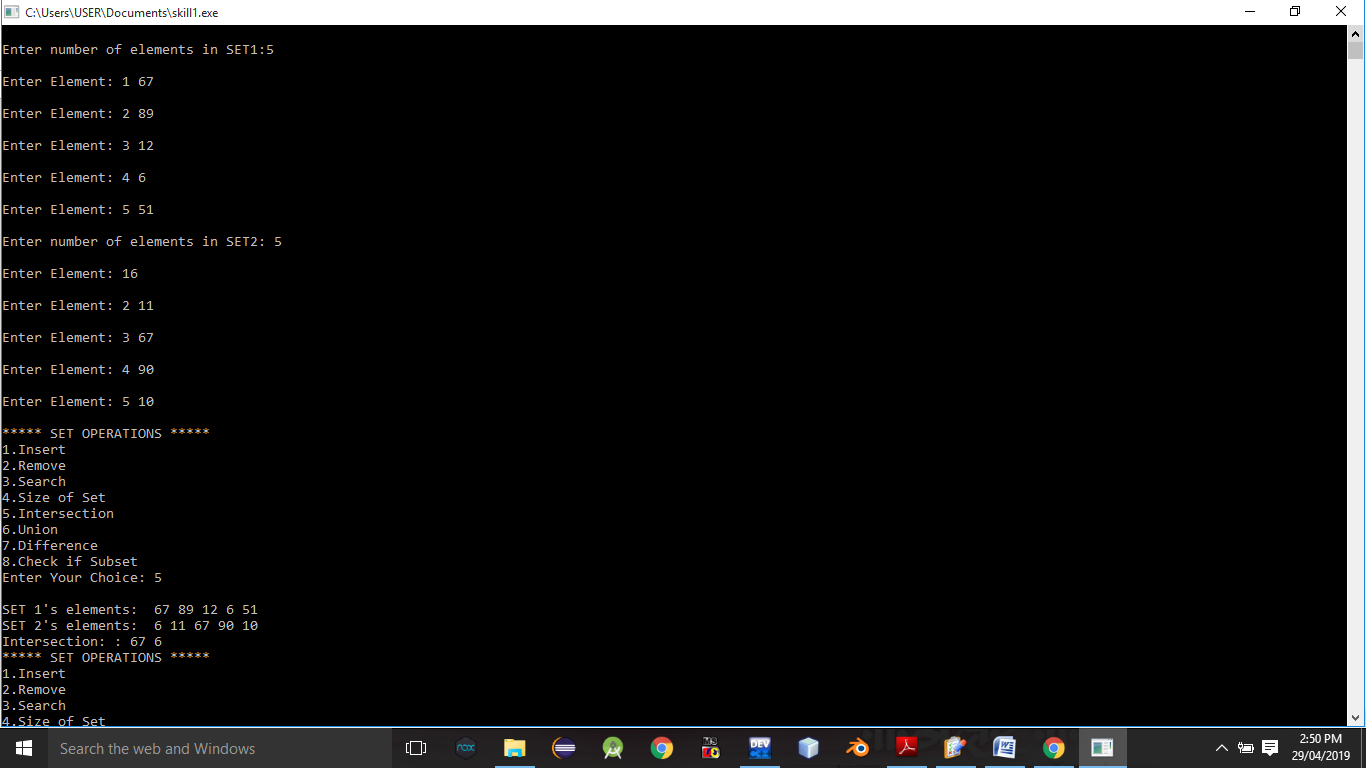
break;

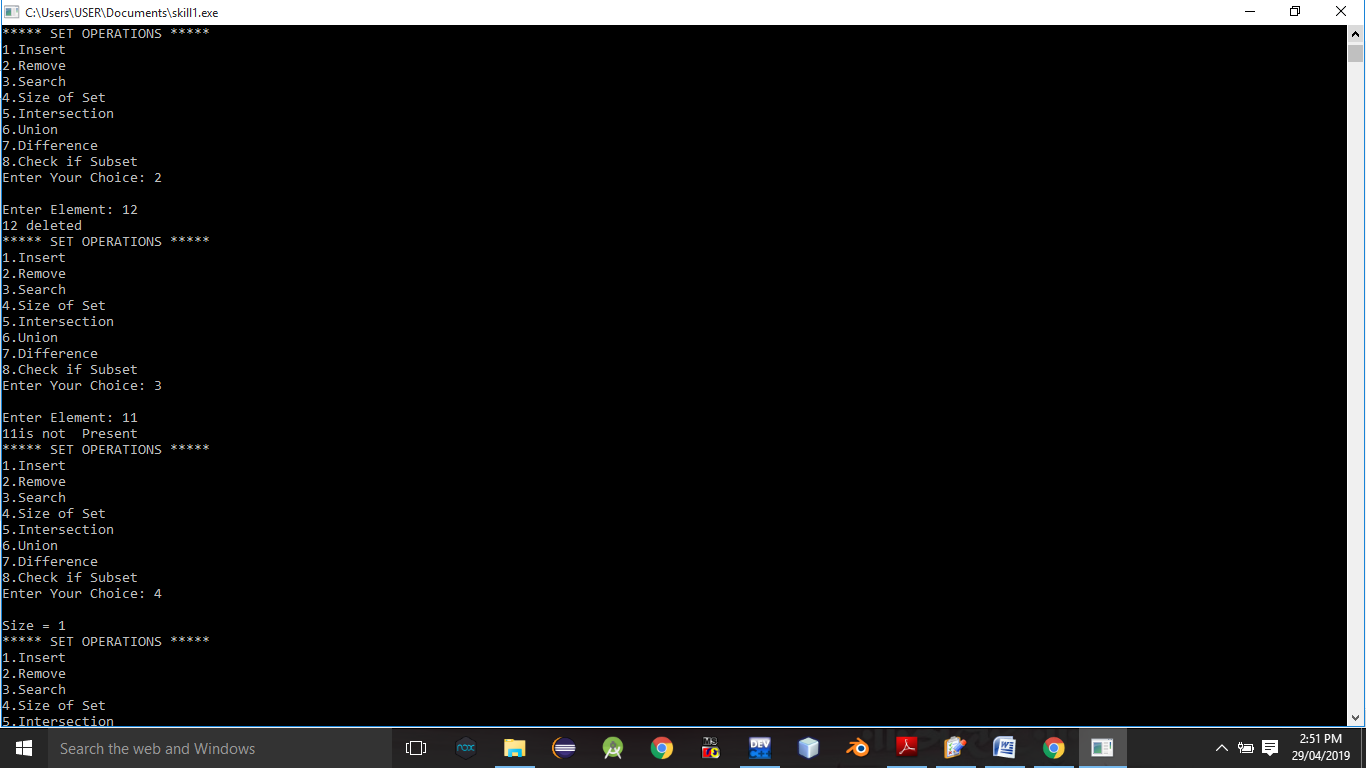
}

}while(choice!=0);

return 0;

}

**Output :-**

****

**CONCLUSION**:

We understood the concepts of sets and the various operations performed on them, and were able to apply those concepts through programming.

**ASSIGNMENT NO :2**

**Aim:**

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

**Objective:**

To understand the following Concepts of Threaded Binary Search Tree (TBT):

1. Creating a TBT using tree data structure.
2. Inorder traversal using threads.

**THEORY:**

# *Threaded Binary Tree:*

[Inorder traversal of a Binary tree](https://www.geeksforgeeks.org/618/) can either be done using recursion or [with the use of a auxiliary stack](https://www.geeksforgeeks.org/inorder-tree-traversal-without-recursion/). 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).

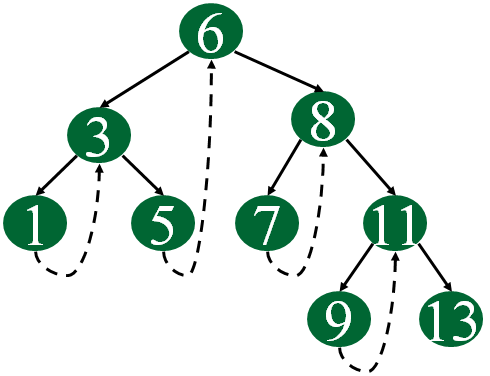
There are two types of threaded binary trees.

**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.



**ALGORITHM:**

**Non recursive Inorder traversal for a Threaded Binary Tree**

1. curr-node node leftmost (root)

2. While (curr\_node != Null)

a. print (curr\_node)

b. If (curr\_node.RTag == 0) then

curr\_node<- curr\_node.right

go to step 2.

c. else curr\_node<- leftmost(curr\_node.right)

go to step 2.

**CODE:**

#include <iostream>

using namespace std;

class TBT;

class node

{

node \*left,\*right;

int data;

boolrbit,lbit;

public:

node()

{

left=NULL;

right=NULL;

rbit=lbit=0;

}

node(int d)

{

left=NULL;

right=NULL;

rbit=lbit=0;

data=d;

}

friend class TBT;

};

class TBT

{

node \*root; //acts as a dummy node

public:

TBT() //dummy node initialization

{

root=new node(9999);

root->left=root;

root->rbit=1;

root->lbit=0;

root->right=root;

}

void create();

void insert(int data);

node \*inorder\_suc(node \*);

voidinorder\_traversal();

node \* preorder\_suc(node \*c);

voidpreorder\_traversal();

};

//--------------------------------------------

void TBT::preorder\_traversal()

{

node \*c=root->left;

while(c!=root)

{

cout<<" "<<c->data;

c=preorder\_suc(c);

}

}

void TBT::inorder\_traversal()

{

node \*c=root->left;

while(c->lbit==1)

c=c->left;

while(c!=root)

{

cout<<" "<<c->data;

c=inorder\_suc(c);

}

}

node\* TBT::inorder\_suc(node \*c)

{

if(c->rbit==0)

return c->right;

else

c=c->right;

while(c->lbit==1)

{

c=c->left;

}

return c;

}

node \*TBT::preorder\_suc(node \*c)

{

if(c->lbit==1)

{

return c->left;

}

while(c->rbit==0)

{

c=c->right;

}

return c->right;

}

//-------- Create Method

void TBT::create()

{

int n;

if(root->left==root&&root->right==root)

{

cout<<"\nEnter number of nodes:";

cin>>n;

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

{

int info;

cout<<"\nEnter data: ";

cin>>info;

this->insert(info);

}

}

else

{

cout<<"\nTree is Already created.\n";

}

}

void TBT::insert(int data)

{

if(root->left==root&&root->right==root) //no node in tree

{

node \*p=new node(data);

p->left=root->left;

p->lbit=root->lbit; //0

p->rbit=0;

p->right=root->right;

root->left=p;

root->lbit=1;

cout<<"\nInserted start"<<data;

return;

}

node \*cur=new node;

cur=root->left;

while(1)

{

if(cur->data<data) //insert right

{

node \*p=new node(data);

if(cur->rbit==0)

{

p->right=cur->right;

p->rbit=cur->rbit;

p->lbit=0;

p->left=cur;

cur->rbit=1;

cur->right=p;

//cout<<"\nInserted right "<<data;

cout<< data<<" Inserted right"<<" of "<< cur->data<<endl;

return;

}

else

cur=cur->right;

}

if(cur->data>data) //insert left

{

node \*p=new node(data);

if(cur->lbit==0)

{

p->left=cur->left;

p->lbit=cur->lbit;

p->rbit=0;

p->right=cur; //successor

cur->lbit=1;

cur->left=p;

cout<<data <<" Inserted left "<<" of "<<cur->data<<endl;

return;

}

else

cur=cur->left;

}

}

}

int main() {

TBT t1;

int value;

int choice;

do

{

cout<<"\n1.Create Tree\n2.Insert into tree\n3.Preorder\n4.Inorder\n0.Exit\nEnter your choice: ";

cin>>choice;

switch(choice)

{

case 1:

t1.create();

break;

case 2:

cout<<"\nEnter Number(data): ";

cin>>value;

t1.insert(value);

break;

case 3:

cout<<"\nPreorder traversal of TBT\n";

t1.preorder\_traversal();

break;

case 4:

cout<<"\nInoder Traversal of TBT\n";

t1.inorder\_traversal();

break;

default:

cout<<"\nWrong choice";

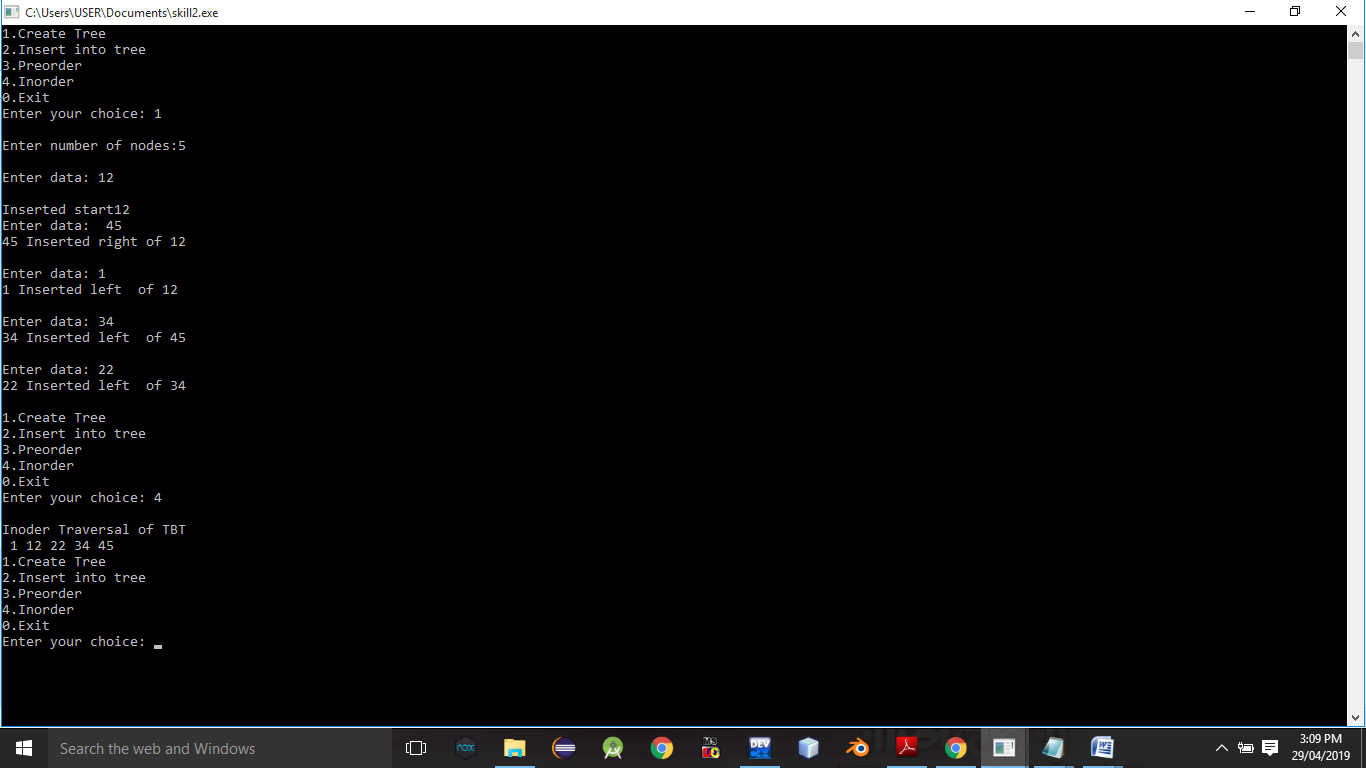
}

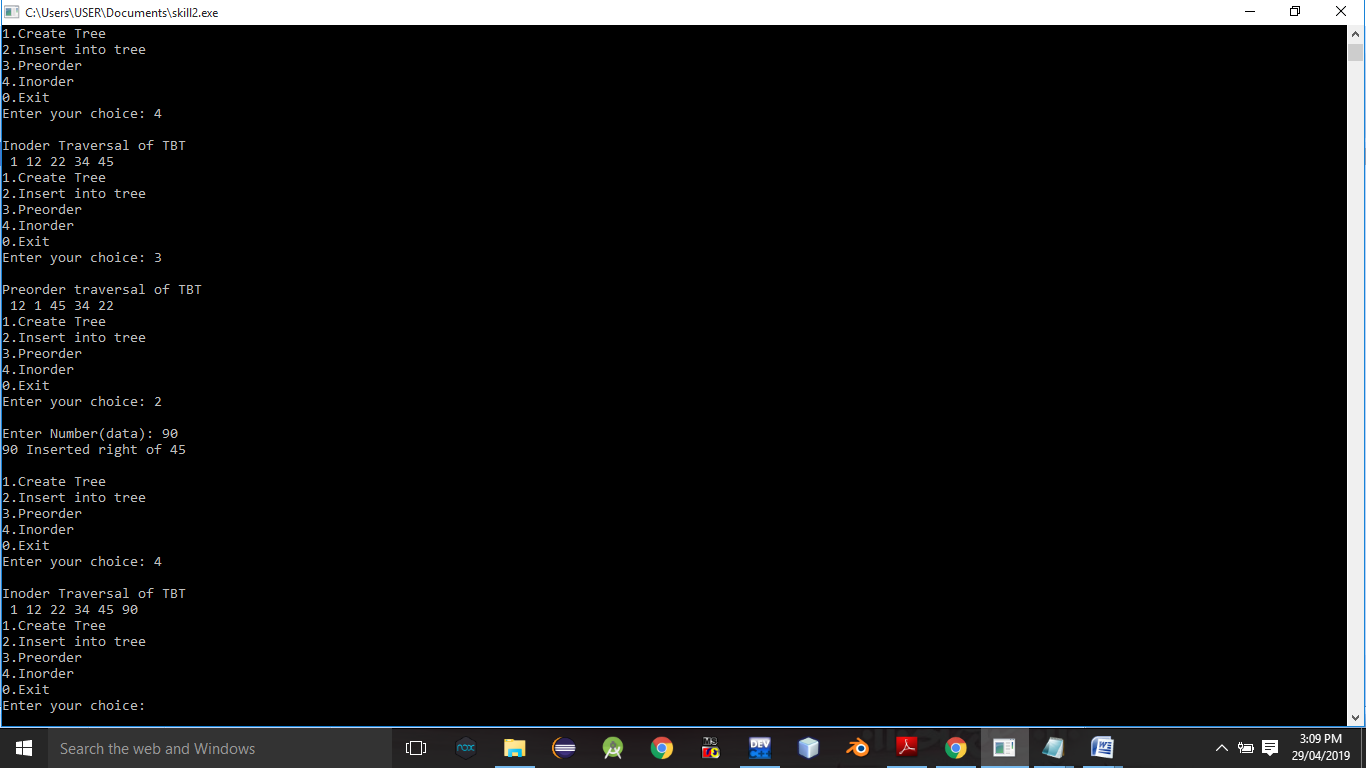
}while(choice!=0);

return 0;

}

**OUTPUT**

****



**CONCLUSION:**

Study and operations like creating TBT and Inorder traversal using threads was performed successfully.

**Assignment No: 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.

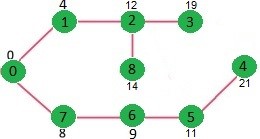
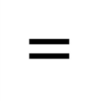
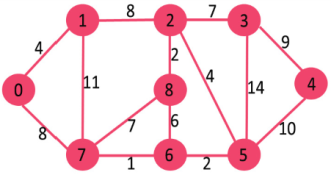
**Objectives:**

To understand the various operations on graphs.

**Theory:**

Dijkstra’s algorithm is very similar to Prim’s algorithm for minimum spanning tree. Like Prim’s MST, we generate ashortest path treewith given source as root. We maintain two sets, one set contains vertices included in shortest path tree, and other set includes vertices not yet included in shortest path tree.

For Example:



At every step of the algorithm, we find a vertex which is in the other set and has a minimum distance from the source. Below are the detailed steps used in Dijkstra’s algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

**Algorithm:**

1. Create priority queue pq
2. Enqueue(pq,s)
3. For(i=1;i<=g->v;i++)

Distance[i]=-1

1. Distance[s]=0
2. while(!isemptyqueue(pq))

{

5.1v=deletemin(pq);

5.2 for all adjacent vertices w to v

{

Compute new distance d=distance[v]+weight[v][w];

If(Distance[w]==-1)

{

Distance[w]=new distance d;

Insert w in priorityqueue with priority d

Path[w]=v}

If(Distance[w]>newdistance d)

{ distance[w]=new disance d;

Update priority 0f vertex w to be d;

Path[w]=v;

}

}}}

**Program:**

#include<iostream> #define MAX 20

using namespace std;

classdijkstra

{

int city; int distance[MAX][MAX]; int d[MAX]; int visited[MAX]; public:

voidcity\_no(); intminvertex(); void matrix\_fill(); void dijkstra\_code();

void display();

};

voiddijkstra::city\_no()

{

cout<<"\n enter the number of cities (including cities A and B) : "; cin>>city;

}

intdijkstra::minvertex()

{

intmvertex=-1;

for(inti=0;i<city;i++)

{

if(visited[i]==0 && (mvertex==-1 || d[i]<d[mvertex])) mvertex=i;

}

returnmvertex;

}

voiddijkstra::matrix\_fill()

{

cout<<"\n enter the distances between the cities : ";

for(inti=0;i<city;i++)

{

cout<<"\n For city "<<i<<endl;

for(int j=0;j<city;j++)

{

if(i==j) distance[i][j]=0;

cin>>distance[i][j];

}

d[i]=INT\_MAX;

visited[i]=0;

}

}

voiddijkstra::dijkstra\_code()

{

d[0]=0;

for(inti=0;i<city-1;i++)

{

intmvertex=minvertex(); visited[mvertex]=1;

for(int j=0;j<city;j++)

{

if((distance[mvertex][j]!=0)&&(visited[j]==0))

{

intdist=d[mvertex]+distance[mvertex][j];

if(dist<d[j]) d[j]=dist;

}

}

}

}

voiddijkstra::display()

{

cout<<"\n distance of cities from city 0 \n"; cout<<"city Distance\n";

for(inti=0;i<city;i++)

cout<<i<<"\t"<<d[i]<<endl;

}

int main()

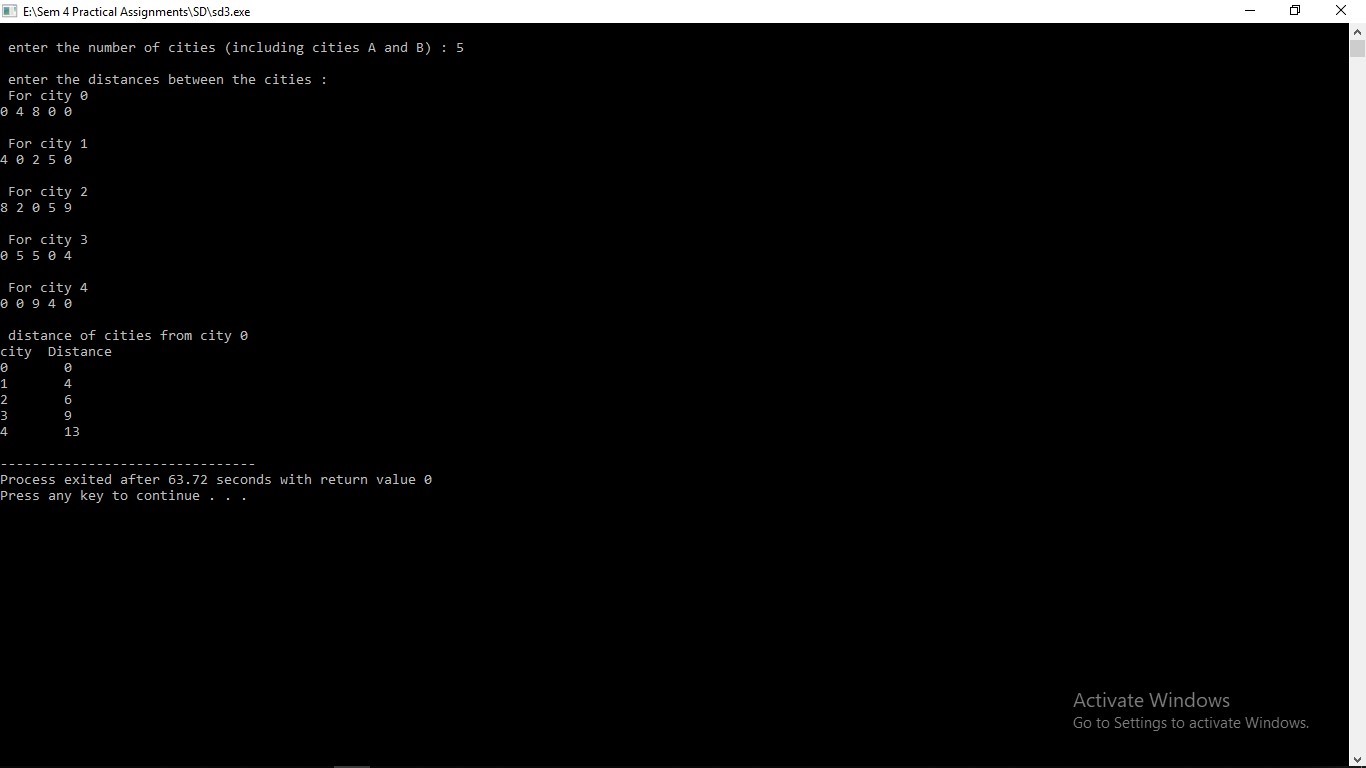
{

dijkstrasp; sp.city\_no(); sp.matrix\_fill(); sp.dijkstra\_code(); sp.display();

return 0;

}

**Output:**



**Conclusion:**

From above experiment we learnt how to use shortest path algorithm using graph operation.

**Assignment 4**

**Aim:**

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

**Objective:**

Understand the concepts of prims algorithm

**Theory:**

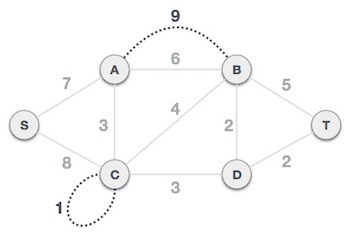
Prim's algorithm to find minimum cost spanning tree (as Kruskal's algorithm) uses the greedy approach. Prim's algorithm shares a similarity with the **shortest path first** algorithms.

Prim's algorithm, in contrast with Kruskal's algorithm, treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.

To contrast with Kruskal's algorithm and to understand Prim's algorithm better, we shall use the same example −



Step 1 - Remove all loops and parallel edges



Remove all loops and parallel edges from the given graph. In case of parallel edges, keep the one which has the least cost associated and remove all others.

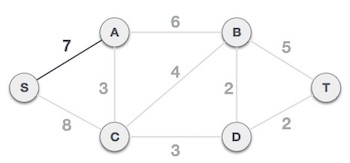


Step 2 - Choose any arbitrary node as root node

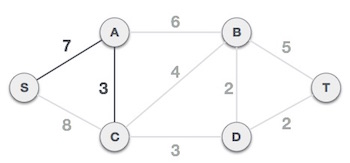
In this case, we choose **S** node as the root node of Prim's spanning tree. This node is arbitrarily chosen, so any node can be the root node. One may wonder why any video can be a root node. So the answer is, in the spanning tree all the nodes of a graph are included and because it is connected then there must be at least one edge, which will join it to the rest of the tree.

Step 3 - Check outgoing edges and select the one with less cost

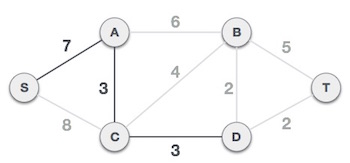
After choosing the root node **S**, we see that S,A and S,C are two edges with weight 7 and 8, respectively. We choose the edge S,A as it is lesser than the other.



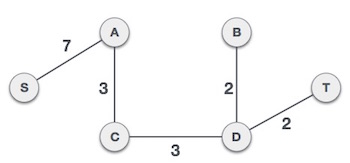
Now, the tree S-7-A is treated as one node and we check for all edges going out from it. We select the one which has the lowest cost and include it in the tree.



After this step, S-7-A-3-C tree is formed. Now we'll again treat it as a node and will check all the edges again. However, we will choose only the least cost edge. In this case, C-3-D is the new edge, which is less than other edges' cost 8, 6, 4, etc.



After adding node **D** to the spanning tree, we now have two edges going out of it having the same cost, i.e. D-2-T and D-2-B. Thus, we can add either one. But the next step will again yield edge 2 as the least cost. Hence, we are showing a spanning tree with both edges included.



We may find that the output spanning tree of the same graph using two different algorithms is same.

**Algorithm:**

Algorithm Prims(E,cost,n,t)

{1.Let (k,l) be the edge of minimum cost

2.mincost=cost(k,l)

3.t[1,1]=k;t[1,2]=l;

4.fori=1 to n do

If(cost[i,l]<cost[i,k] then near[i]=l

Else near[i]=k;

5.near[k]=near[l]=0

6.fori=2 to n-1 do

6.1Let j be the index such that near[j]!=0 and

Cost[j,near[j]] is minimum

6.2t[i,1]=j ;t[i ,2]=near[j]

6.3 mincost=mincost+cost[j,near[j]];

6.4 near[j]=0

6.5for k=1 to n do

if ((near[k]!=0) and (cost[k,near[k]]>cost[k,j]))

then near[k]=j}

Return mincost

}

**C++ Code:**

#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**

**PRIM'S MST OF THE GRAPH IS:**

**1-0**

**2-1**

**COST OF MST IS: 15**

**Conclusion:**

This assignment is used how prims algorithm is used in solving the example using vertex edge .

**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 application of Kruskal’s algorithm to find the minimum spanning tree.

**Theory:**

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. This algorithm treats the graph as a forest and every node it has as an individual tree. A tree connects to another only and only if, it has the least cost among all available options and does not violate MST properties.

The standard application is to a problem like phone network design. 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. It should be a spanning tree, since if a network isn’t a tree you can always remove some edges and save money.

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.
3. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
4. Repeat step#2 until there are (V-1) edges in the spanning tree.

**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.

**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 elemets index

visited[k]=1; //set visited

for(int j=0;j<n;j++)//for adjacent verices comparision

{

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},

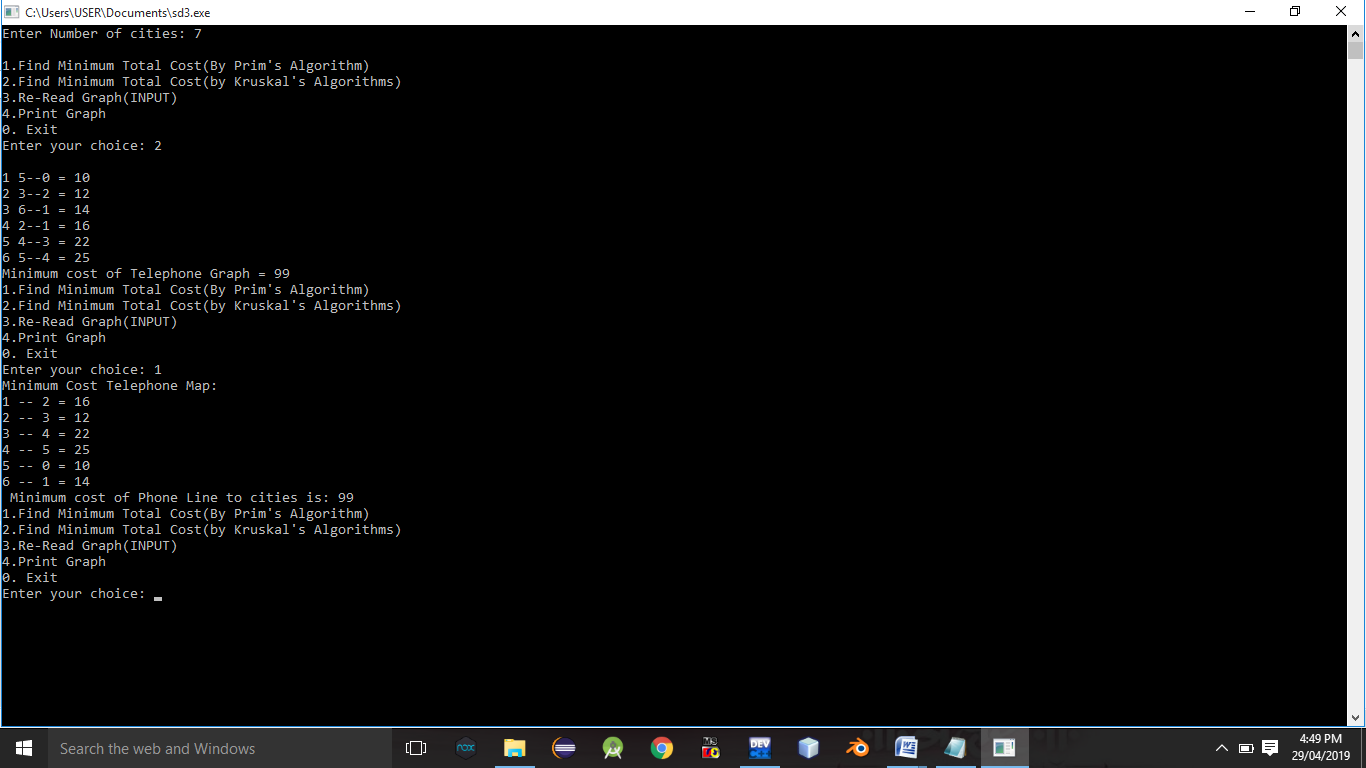
{0,14,0,18,24,0,0},

};

Minimum Cost: 99

\*/

**Output :-**

****

**Conclusion:**

We understood the implementation of Kruskal’s algorithm in real life problems.

**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** :

To find maximum and minimum marks obtained by the students in second year in a particular subject using a binary heap (either max heap or min heap) and then sorting the heap using heap sort algorithm for desired output.

**Theory :**

A binary heap is a complete binary tree which satisfies the heap ordering property.

The ordering can be one of two types:

* the min-heap property: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.
* the max-heap property: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.

We create a heap by adding numbers from left to right level by level. Heap can be implemented using an array or a priority queue.

For sorting the heap, after it’s creation, the first position of the array would contain either the smallest or the largest element depending on whether max heap or min heap is created ,heap sort algorithm swaps the first element in the heap with the last one and heapify the heap excluding the last element and reduce the size of the array by one. Repeat the steps until the complete heap is sorted.

**Code :**

#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];

}

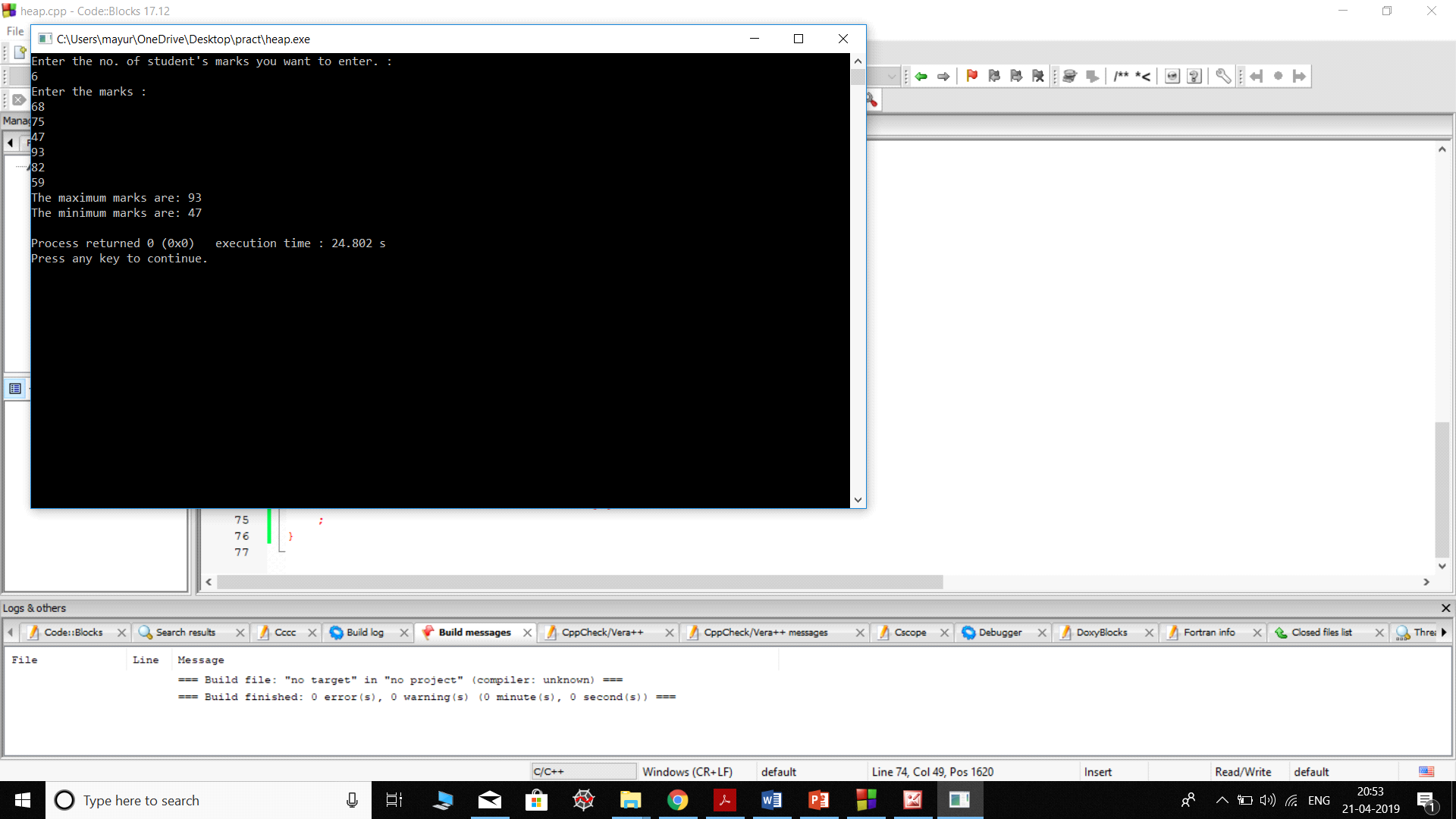
heapSort(arr, n);

cout<< "The maximum marks are: "<<arr[n-1]<<"\n";

cout<<"The minimum marks are: "<<arr[0]<<"\n";

}

**Output:**



**Conclusion-:**

The objective of creating a heap from an array was completed. The heap was then sorted to give out the maximum and minimum marks obtained by students using heap sort algorithm.

**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:**

To understand :

1. How keys can be mapped to the corresponding values , in a hash table, in order to have the lowest time complexity.

2. How collisions can be resolved , in a hash table , using a second hash function.

**Theory:**

Double hashing is a computer programming technique , used in hash tables to resolve hash collisions, in cases when two different values to be searched for produce the same hash key. It is a popular collision -resolution technique in open-addressed hash tables. Double hashing is implemented in many popular libraries.

Like linear probing, it uses one hash value as a starting point and then repeatedly steps forward an interval until the desired value is located, an empty location is reached, or the entire table has been searched; but this interval is decided using a second, independent hash function (hence the name double hashing). Unlike linear probing and quadratic probing , the interval depends on the data, so that even values mapping to the same location have different bucket sequences; this minimizes repeated collisions and the effects of clustering.

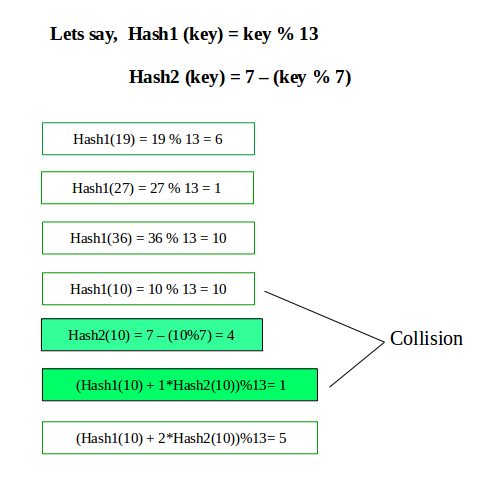
First hash function is typically hash1(key) = key % TABLE\_SIZE

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

**Example:**



**Algorithm:**

1. Start.
2. Accept the size of the table.
3. Initialize the hash table array to any negative integer value say “-111”(Provided negative keys are not accepted in the table).
4. Map the key to it’s value, using first hash function: hash1(key) = key % Table\_size.
5. If collision occurs use the second hash function: hash2(key) = 1+(key mod (size-1)).
6. Do: Hi(key)=((Hash(key) + i \* hash2(key)) mod size) , using a for loop, for i from 1 to (size-1), untill the key gets mapped to it’s appropriate value.
7. Stop.

**Code:**

#include<iostream>

#include<stdlib.h>

using namespace std;

int size=10;

void display(int hash[])

{

inti;

cout<<"HASH TABLE:\n";

for(i=0;i<size;i++)

{

cout<<""<<hash[i]<<"\t";

}

}

int main()

{

int hash[size],val,i;

charch;

for(i=0;i<size;i++)

{

hash[i]=-111;

}

do

{

cout<<"Enter the value\n";

cin>>val;

int m=val%size;

if(hash[m] == -111)

{

hash[m]=val;

display(hash);

}

else

{

cout<<"Collision\n";

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]==-111)

{

hash[z]=val;

display(hash);

break;

}

}

}

cout<<"\nWant to add more values?\n";

cin>>ch;

intss=0;

for(i=0;i<size;i++)

{

if(hash[i]!= -111)

ss++;

}

if(ss==10)

{

cout<<"Hast table is full\n";

exit(1);

}

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

display(hash);

return 0;

}

**OUTPUT:**

Enter the value

11

HASH TABLE:

-111 11 -111 -111 -111 -111 -111 -111 -111 -111

Want to add more values?

y

Enter the value

32

HASH TABLE:

-111 11 32 -111 -111 -111 -111 -111 -111 -111

Want to add more values?

y

Enter the value

64

HASH TABLE:

-111 11 32 -111 64 -111 -111 -111 -111 -111

Want to add more values?

y

Enter the value

74

Collision

64

HASH TABLE:

-111 11 32 -111 64 -111 -111 74 -111 -111

Want to add more values?

y

Enter the value

66

HASH TABLE:

-111 11 32 -111 64 -111 66 74 -111 -111

Want to add more values?

y

Enter the value

97

Collision

74

HASH TABLE:

74 11 32 -111 64 -111 66 97 -111 -111

Want to add more values?

n

HASH TABLE:

74 11 32 -111 64 -111 66 97 -111 -111

**Conclusion:**

Hence Double Hashing technique can be success

**Assignment 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

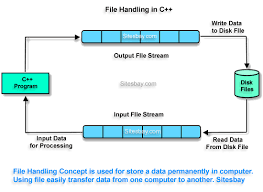
student does not exist an appropriate message is displayed. If it is, then the system displays the student details Use Sequential file to maintain data

**Objective:**

Understand the concepts of sequential file handling

**Theory:**

A file is a collection of related data stored in a particular area on the disk



A File can be opened in the following ways

|  |  |
| --- | --- |
| **File mode parameter** | **Meaning** |
| ios::app | Append to end of file |
| ios::ate | go to end of file on opening |
| ios::binary | file open in binary mode |
| ios::in | open file for reading only |
| ios::out | open file for writing only |
| ios::nocreate | open fails if the file does not exist |
| ios::noreplace | open fails if the file already exist |
| ios::trunc | delete the contents of the file if it exist |

When we want to move file pointer to desired position then use these function to manage the file pointers.

Seekg () = moves get pointer (input) to a

specified location

Seekp () = moves put pointer (output) to a

specified location

tellg () = gives the current position of the get pointer

tellp () = gives the current position of the put pointer

file .read ((char \*)&V , sizeof (V));

file . Write ((char \*)&V , sizeof (V));

These function take two arguments. The first is

the address of the variable V , and the second is

the length of that variable in bytes . The address

of variable must be cast to type char \* (i.e pointer

to character type) .

**Algorithm:**

1. Take the count of number of students from the user

2. Make an array of object of the student class which stores the information of the students

3. Open a file by using the ofstream object

4. Take the information of the student from the user and write it to the file

5. User can perform 1.Search 2.Delete 3.Display operations

6. For Search

1. Input the Roll number to be searched

2. Open the file using Ifstream object in input mode

3. Read the contents of the file in an object sequentially and check it with the roll number to be searched if found Display found message and the details of the students

4. If not found continue till end of file

5. If eof is reached display the message Not found

7.For Delete

1.Input the roll number to be deleted

2. Open the Main file in input mode and a temporary file in output mode

3.Sequentially search through the main file and copy the contents to the temp file except the roll number to be deleted

4. Delete the contents of the Main file

5. Rename the temp file with the name of the main file

8. For Display

1. Open the file in input mode and display the details of all the students sequentially

**C++ Code:**

#include<iostream>

#include<fstream>

using namespace std;

class student

{

introll\_num;

char div;

string name;

string address;

public:

voidgetdata()

{

cout<<"\n Enter the Roll Number";

cin>>roll\_num;

cout<<"\n Enter the division ";

cin>>div;

cout<<"\n Enter the Name";

fflush(stdin);

getline(cin,name);

cout<<"\n Enter the Address";

fflush(stdin);

getline(cin,address);

}

voidputdata(int n)

{

studentst[n];

ifstreaminfile;

infile.open("student.dat",ios::binary|ios::in);

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

{

infile.read((char \*)&st[i],sizeof(st[i]));

cout<<"\n Roll Number: "<<st[i].roll\_num;

cout<<"\n Division: "<<st[i].div;

fflush(stdin);

cout<<"\n Name: "<<st[i].name;

fflush(stdin);

cout<<"\n Address: "<<st[i].address;

cout<<"\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n";

}

infile.close();

}

void search\_(int n)

{

studentst[n];

ifstreaminfile;

cout<<"\n Enter the Roll Number to be searched";

int r;

cin>>r;

infile.open("student.dat",ios::in|ios::binary);

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

{

infile.read((char \*)&st[i],sizeof(st[i]));

if(st[i].roll\_num==r)

{

cout<<"\n Found";

cout<<"\n Details: "<<endl;

cout<<"\n Roll Number: "<<st[i].roll\_num;

cout<<"\n Division: "<<st[i].div;

fflush(stdin);

cout<<"\n Name: "<<st[i].name;

fflush(stdin);

cout<<"\n Address: "<<st[i].address;

cout<<"\n \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \n";

infile.close();

return;

}

}

cout<<"\n Not Found";

infile.close();

}

void del(int n)

{

studentst[n];

int r;

cout<<"\n Enter the roll number to be deleted ";

cin>>r;

ifstreaminfile;

ofstreamoutfile;

infile.open("student.dat",ios::binary|ios::in);

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

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

{

infile.read((char \*)&st[i],sizeof(st[i]));

if(st[i].roll\_num==r)

{

continue;

}

else

{

outfile.write((char \*)&st[i],sizeof(st[i]));

}

}

outfile.close();

infile.close();

remove("student.dat");

int re=rename("temp.dat","student.dat");

}

};

int main()

{

int n;

cout<<"\n Enter the Number of Students";

cin>>n;

student s[n];

ofstreamoutfile;

outfile.open("student.dat",ios::out|ios::binary);

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

{cout<<"\n Enter the Number of Students";

s[i].getdata();

outfile.write((char \*)&s[i],sizeof(s[i]));

}

outfile.close();

int c;

student d;

do

{

cout<<"\n 1.Search";

cout<<"\n 2.Delete";

cout<<"\n 3.Display";

cout<<"\n 4.Exit";

cout<<"\n Enter Your Choice";

cin>>c;

switch(c)

{

case 1:d.search\_(n);break;

case 2:d.del(n);n=n-1;break;

case 3:d.putdata(n);break;

case 4:break;

}

}

while(c!=4);

}

**Output:**

Enter the Number of Students3

Enter the details of Student 1

Enter the Roll Number34

Enter the division A

Enter the NameRam J

Enter the AddressPune 14

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Enter the details of Student 2

Enter the Roll Number89

Enter the division B

Enter the NameSham K

Enter the AddressKondhwa

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Enter the details of Student 3

Enter the Roll Number56

Enter the division A

Enter the NameDavid M

Enter the AddressPune

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1.Search

2.Delete

3.Display

4.Exit

Enter Your Choice1

Enter the Roll Number to be searched89

Found

Details:

Roll Number: 89

Division: B

Name: Sham K

Address: Kondhwa

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Conclusion:**

Through this assignment ,we learned and performed how to store information in a sequential file and perform various operations on the file like search,delete etc.

# **ASSIGNMENT 09**

**Aim**:

Department maintains a employee information. The file contains 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 main the data.

**Objective:**

To make use of index sequential files to maintain and operation on data.

**Theory:**

Index Sequential File:

This is basically a mixture of sequential and indexed file organisation techniques. Records are held in sequential order and can be accessed randomly through an index. Thus, these files share the merits of both systems enabling sequential or direct access to the data.

The index to these files operates by storing the highest record key in given cylinders and tracks. Note how this organisation gives the index a tree structure. Obviously this type of file organisation will require a direct access device, such as a hard disk.

Indexed sequential file organisation is very useful where records are often retrieved randomly and are also processed in (sequential) key order. Banks may use this organisation for their auto-bank machines i.e. customers randomly access their accounts throughout the day and at the end of the day the banks can update the whole file sequentially.

Advantages of Indexed Sequential Files:

1.Allows records to be accessed directly or sequentially.

2.Direct access ability provides vastly superior (average)

access times.

Disadvantages of Indexed Sequential Files:

1.The fact that several tables must be stored for the index makes for a considerable storage overhead.

2.As the items are stored in a sequential fashion this adds complexity to the

addition/deletion of records. Because frequent updating can be very inefficient, especially for large files, batch updates are often performed.

**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;

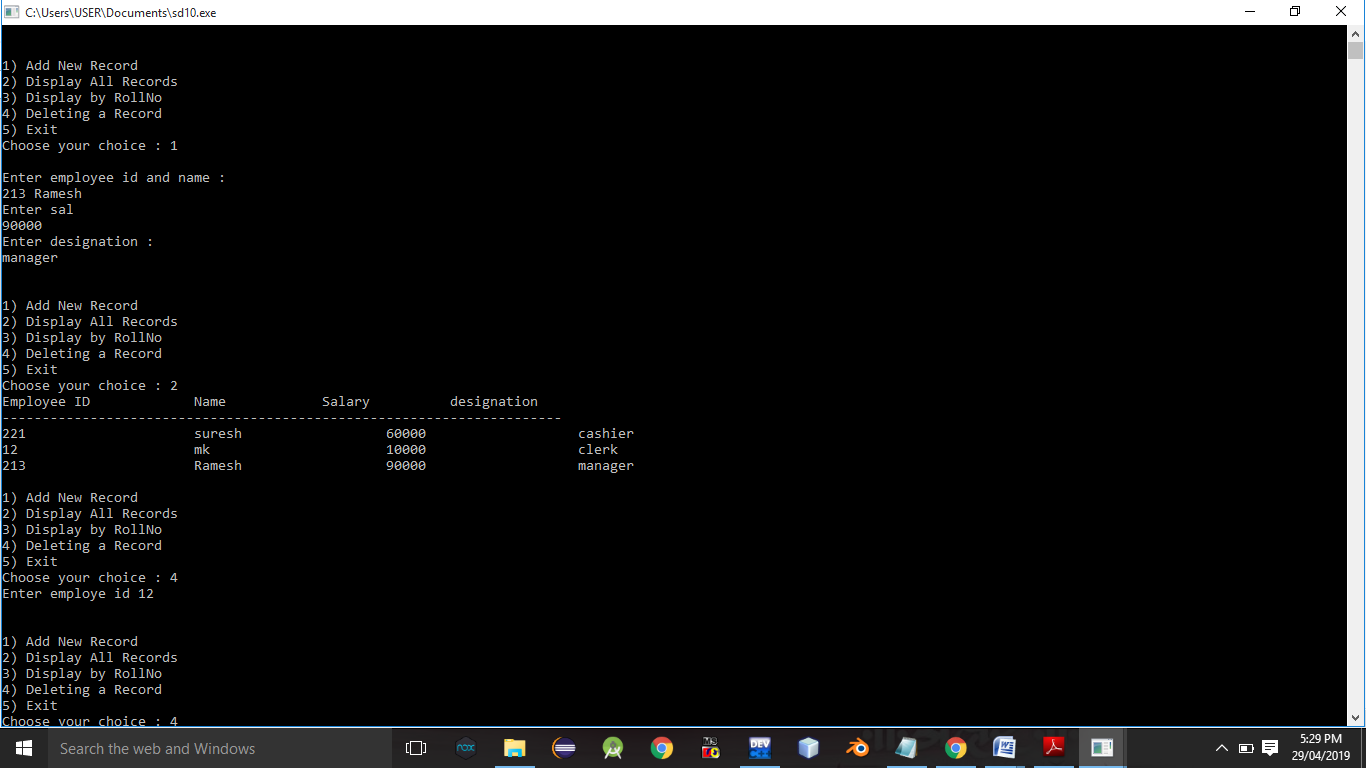
case 5 :break;

}

}

}

**Output Screenshots:-**

****

**Conclusion:**

In above assignment, we made the use of index sequential files to operate on employee data.