

ASSIGNMENT 1

AIM: TO CREATE ADT TO PERFORM THE FOLLOWING SET OPERATIONS:

1. ADD (NEW ELEMENT) PLACE A VALUE IN A SET.
2. REMOVE(ELEMENT).
3. RETURNS TRUE IF ELEMENT IS IN COLLECTION.
4. SIZE () RETURNS NUMBER OF VALUES IN A COLLECTION.
5. INTERSECTION OF TWO SETS.
6. UNION OF TWO SETS.
7. DIFFERENCE BETWEEN TWO SETS
8. SUBSET.

OBJECTIVE: TO IMPLEMENT THE "SET" CONCEPT.

THEORY : A **set** is an abstract data type that can store unique values, without any particular order. It is a computer implementation of the mathematical concept of a finite set. Unlike most other collection types, rather than retrieving a specific element from a set, one typically tests a value for membership in a set. One may define the operations of the algebra of sets:

- `union(S, T)`: returns the union of sets S and T .
- `intersection(S, T)`: returns the intersection of sets S and T .
- `difference(S, T)`: returns the difference of sets S and T .
- `subset(S, T)`: a predicate that tests whether the set S is a subset of set T .

•

ALGORITHM:

Union:

- 1) Initialize union U as empty.
- 2) Copy all elements of first array to U .
- 3) Do following for every element x of second array:
.... a) If x is not present in first array, then copy x to U .
- 4) Return U .

Intersection:

- 1) Initialize intersection I as empty.
- 2) Do following for every element x of first array

.... a) If x is present in second array, then copy x to l.

4) Return l.

CODE:

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

void create (int *s1, int *s2);
void display (int *s);
void intersection (int *s1, int *s2);
void insert (int *s);
void remove (int *s);
void contain (int *s);
void set_size (int *s);
void intersection (int *s1, int *s2);
int linear (int *s,int e);
#define SIZE 20

int main ()
{
    int s1[SIZE], s2[SIZE];
    int element,ch,c,i,r;

    do{
        cout<<"\n***MENU***";
        cout<<"\n1:CREATE \n2:ADD ELEMENT \n3:REMOVE ELEMENT \n4:CONTAIN
ELEMENT \n5:SIZE OF ELEMENT \n6:INTERSECTION";
        cout<<"\n Enter your choice:";
        cin>>ch;
        switch(ch)
        {
            case 1:create(s1,s2);
                        break;
            case 2: cout<<"\n IN WHICH SET YOU WANT TO INSERT
ELEMENT(1/2):";
                        cin>>c;
                        if(c==1)
                            insert(s1);
                        else
                            insert(s2);
                        break;
            case 3:cout<<"\n IN WHICH SET YOU WANT TO REMOVE
ELEMENT(1/2):";
                        cin>>c;
                        if(c==1)
                            remove(s1);
                        else
                            remove(s2);
```

```

        break;
    case 4:cout<<"\n IN WHICH SET YOU WANT TO CHECK THE
ELEMENT(1/2):";
        cin>>c;
        if(c==1)
            contain(s1);
        else
            contain(s2);
        break;
    case 5:cout<<"\n IN WHICH SET YOU WANT TO CHECK THE SIZE(1/2):";
        cin>>c;
        if(c==1)
            set_size(s1);
        else
            set_size(s2);
        break;
    case 6:intersection(s1,s2);
    default: cout<<"\n WRONG CHOICE!!!";
}

}while(ch<6);
return 0;
}
int linear(int *s, int e)
{
    int f;
    for(int i=1;i<=s[0];i++)
    {
        if(s[i]==e)
        {
            f=1;
            return f;
        }
    }
    if(f==0)
        return f;
}
void intersection(int *s1,int *s2)
{
    int s3[SIZE],i,j=1;
    for( i=1;i<=s1[0];i++)
    {
        if(linear(s2,s1[i])==1)
        {
            s3[j]=s1[i];
            j++;
        }
    }
}

```

```

void set_size(int *s)
{
    cout<<"\n SIZE OF SET:"<<s[0];
}
void contain(int *s)
{
    int element;
    cout<<"\n Enter element to check:";
    cin>>element;
    if(linear(s,element)==1)
        cout<<"\n ELEMENT PRESENT!";
    else
        cout<<"\n ELEMENT NOT PRESENT!!!";

}
void remove(int *s)
{
    int element,i,j;
    cout<<"\n Enter element to remove:";
    cin>>element;
    for(i=1;i<=s[0];i++)
    {
        if(s[i]==element)
        {
            for(int j=i;j<=s[0];j++)
            {
                s[j]=s[j+1];
            }
            s[0]-=1;
            cout<<"\n SIZE:"<<s[0]<<"\n";
            display(s);
            return;
        }
    }
    cout<<"\n ELEMENT NOT FOUND!!!";
}

void insert (int *s)
{
    int element;
    cout<<"\n Enter the element:";
    cin>>element;
    int size=s[0];
    s[++size]=element;
    s[0]=size;
    display(s);
}

void create(int *s1,int *s2)
{

```

```

    int n,i;
    cout<<"\n enter size of set1:";
    cin>>n;
    s1[0]=n;
    cout<<"\n enter elements:";
    for(i=1;i<=n;i++)
    {
        cin>>s1[i];
    }
    cout<<"\n ELEMENTS OF SET1:";
    display(s1);
    cout<<"\n enter size of set2:";
    cin>>n;
    s2[0]=n;
    cout<<"\n enter elements:";
    for(i=1;i<=n;i++)
    {
        cin>>s2[i];
    }
    cout<<"\n ELEMENTS OF SET2:";
    display(s2);
}

void display(int *s)
{
    int i;
    for(i=1;i<=s[0];i++)
    {
        cout<<" "<<s[i];
    }
}

```

OUTPUT:

```

1: **MENU**
2: 1. CREATE
3: 2. ADD ELEMENT
4: 3. REMOVE ELEMENT
5: 4. CONTAIN ELEMENT
6: 5. SIZE OF ELEMENT
7: 6. INTERSECTION
8: Enter your choice:1
Enter size of set1:3
Enter elements:1
2
3
ELEMENTS OF SET1: 1 2 3
Enter size of set2:3
Enter elements:3
4
5
ELEMENTS OF SET2: 3 4 5
1: **MENU**
2: 1. CREATE
3: 2. ADD ELEMENT
4: 3. REMOVE ELEMENT
5: 4. CONTAIN ELEMENT
6: 5. SIZE OF ELEMENT
7: 6. INTERSECTION
8: Enter your choice:2
IN WHICH SET YOU WANT TO INSERT ELEMENT(1/2):1
Enter the element:12
1 2 3 12
1: **MENU**
2: 1. CREATE
3: 2. ADD ELEMENT
4: 3. REMOVE ELEMENT
5: 4. CONTAIN ELEMENT
6: 5. SIZE OF ELEMENT
7: 6. INTERSECTION
8: Enter your choice:3
IN WHICH SET YOU WANT TO REMOVE ELEMENT(1/2):2
Enter element to remove:5
SIZE:2

```

```

C++11 Set Operations
1. WHICH SET YOU WANT TO REMOVE ELEMENT[1/2]:2
Enter element to remove:5
SIZE:2
1 4
****MENU****
1. CREATE
2. ADD ELEMENT
3. REMOVE ELEMENT
4. CONTAIN ELEMENT
5. SIZE OF ELEMENT
6. INTERSECTION
Enter your choice:4
2. WHICH SET YOU WANT TO CHECK THE ELEMENT[1/2]:1
Enter element to check:1
ELEMENT PRESENT!
****MENU****
1. CREATE
2. ADD ELEMENT
3. REMOVE ELEMENT
4. CONTAIN ELEMENT
5. SIZE OF ELEMENT
6. INTERSECTION
Enter your choice:5
3. WHICH SET YOU WANT TO CHECK THE SIZE[1/2]:1
SIZE OF SET:4
****MENU****
1. CREATE
2. ADD ELEMENT
3. REMOVE ELEMENT
4. CONTAIN ELEMENT
5. SIZE OF ELEMENT
6. INTERSECTION
Enter your choice:6
*****CHECKING*****
Process returned 0 (0x0)   execution time : 53.154 s
Press any key to continue.

```

CONCLUSION: We saw all the algorithms the STL offers to operate on sets, that are collections of sorted elements, in the general sense.

ASSIGNMENT 2

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

OBJECTIVE:-Traverse a threaded binary search tree

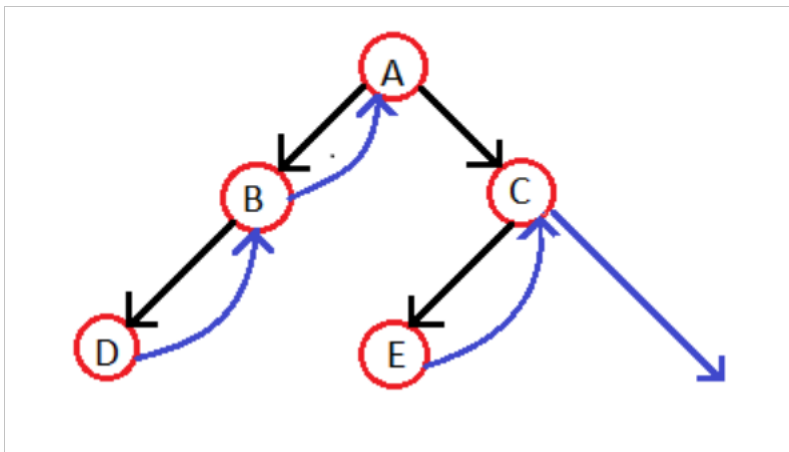
THEORY: - Inorder traversal of a Binary tree can either be done using recursion or with the use of an auxiliary stack. The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion. A binary tree is made threaded by making all right child pointers that would normally be NULL point to the inorder successor of the node (if it exists).

There are two types of threaded binary trees.

Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)

Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively. The predecessor threads are useful for reverse inorder traversal and postorder traversal.

The threads are also useful for fast accessing ancestors of a node.



ALGORITHM: - void inOrder(struct Node *root)

```

{
  struct Node *cur = leftmost(root);
  while (cur!= NULL)
  {
    printf("%d ", cur->data);

    // If this node is a thread node, then go to
    // inorder successor
    if (cur->rightThread)
  
```

```

        cur = cur->right;
    else // Else go to the leftmost child in right subtree
        cur = leftmost(cur->right);
    }
}

```

CODE:-

```

#include<iostream>
using namespace std;

struct treeNode
{
    int data;
    treeNode* left;
    treeNode* right;
    bool lthread;
    bool rthread;
};

class Tree
{
private :
    treeNode* headnode = NULL;
public :
    Tree(){
        headnode = new treeNode();
        headnode->lthread = false;
        headnode->rthread = true;
        headnode->left = headnode->right = headnode;
    }
    void insert(int data){
        treeNode* nn = new treeNode;
        nn->left = NULL;
        nn->right = NULL;
        nn->data = data;
        if(headnode == headnode->left && headnode == headnode->right)
        {
            nn->left = headnode;
            headnode->left = nn;
            nn->lthread = headnode->lthread;
            headnode->lthread = true;
            nn->right = headnode;
        }
        else{
            treeNode* current = headnode->left;
            while(true){
                if(current->data > nn->data){
                    if(current->lthread == false){
                        nn->left = current->left;
                        current->left = nn;
                    }
                }
            }
        }
    }
}

```



```

        nn->lthread = current->lthread;
        current->lthread = true;
        nn->right = current;
        break;
    }else
        current = current->left;
    }
    else {
        if(current->rthread == false){
            nn->right = current->right;
            current->right = nn;
            nn->rthread = current->rthread;
            current->rthread = true;
            nn->left = current;
            break;
        } else {
            current = current->right;
        }
    }
}
}
}
}
void inorder(){
    treeNode* current = headnode->left;
    while (current->lthread == true)
        current = current->left;
    while (current != headnode){
        cout << current->data<<" ";
        if(current->rthread == false)
        {
            current = current->right;
        }
        else
        {
            current = current->right;
            while (current->lthread != false)
                current = current->left;
        }
    }
}
treeNode* getRoot()
{
    return headnode->left;
}
}tree;
int main(){
    char choice='y';
    int ch;
    int data;
    while(choice == 'y'){
        cout << "\tMENU" << endl;

```

```

        cout << "\t1.insert Binay Search tree node" << endl;
        cout << "\t2.Inorder traversal" << endl;
        cout << "\tenter your choice" << endl;
        cin >> ch;
        switch(ch)
        {
            case 1:
                cout << "Enter node data: " << endl;
                cin >> data;
                tree.insert(data);
                break;
            case 2 :
                tree.inorder();
                break;
            default :
                cout << "\tINVALID CHOICE" << endl;
        }
        cout << "\tDo you wish to continue" << endl;
        cout << "\tIf yes enter y" << endl;
        cin >> choice;
    }
}

```

OUTPUT:-

```

C:\Users\USER\Downloads\ThreadedBinaryTree.exe
MENU
1.insert Binay Search tree node
2.Inorder traversal
enter your choice
1
Enter node data:
1
Do you wish to continue
If yes enter y
y
MENU
1.insert Binay Search tree node
2.Inorder traversal
enter your choice
1
Enter node data:
13
Do you wish to continue
If yes enter y
y
MENU
1.insert Binay Search tree node
2.Inorder traversal
enter your choice
1
Enter node data:
23
Do you wish to continue
If yes enter y
y
MENU
1.insert Binay Search tree node
2.Inorder traversal
enter your choice
1
Enter node data:
34
Do you wish to continue
If yes enter y
y
MENU
1.insert Binay Search tree node
2.Inorder traversal
enter your choice
1
Enter node data:
12
Do you wish to continue
If yes enter y
y
MENU
1.insert Binay Search tree node

```

SY-C Department of Computer Engineering, VIIT. 2018-19

ASSIGNMENT 3

AIM: -Represent a Graph using adjacency matrix.

OBJECTIVE: -

THEORY: - Graph is a data structure that consists of following two components:

1. A finite set of vertices also called as nodes.
2. A finite set of ordered pair of the form (u, v) called as edge.

Graphs are used to represent many real-life applications: Graphs are used to represent networks. The networks may include paths in a city or telephone network or circuit network.

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

1. Adjacency Matrix
2. Adjacency List

There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

Adjacency Matrix:

Adjacency Matrix is a 2D array of size $V \times V$ where V is the number of vertices in a graph. Let the 2D array be $adj[][]$, a slot $adj[i][j] = 1$ indicates that there is an edge from vertex i to vertex j . Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If $adj[i][j] = w$, then there is an edge from vertex i to vertex j with weight w .

The adjacency matrix for the above example graph is:

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

ALGORITHM: -

```
void addEdge(vector<int> adj[], int u, int v)
```

```
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}
```

```
// A utility function to print the adjacency list
```

```
// representation of graph
```

```
void printGraph(vector<int> adj[], int V)
```

```
{
    for (int v = 0; v < V; ++v)
    {
        cout << "\n Adjacency list of vertex "
              << v << "\n head ";
        for (auto x : adj[v])
            cout << "-> " << x;
        printf("\n");
    }
}
```

CODE:-

```
#include<iostream>
```

```
#define MAX 10
```

```
using namespace std;
```

```
class airport
```

```
{
    string city[MAX];
    int distance[10][10];
```

```
public :
```

```
    int n;
```

```
    airport();
```

```
    void read_city();
```

```
    void show_graph();
```

```
};
```

```
airport::airport()
```

```
{
```

```

    n=0;

    for(int i=0;i<MAX;i++)
    {
        for(int j=0;j<MAX;j++)
            distance[i][j]=0;
    }
}

void airport::read_city()
{
    int k;
    cout<<"\nEnter the no. of cities: " ;
    cin>>n;
    cout<<"Enter city name:\n";
    for(int k=0;k<n;k++)
    {
        cout<<k+1<<" ] ";
        cin>>city[k];
    }
    for(int i=0;i<n;i++)
    {
        for(int j=i+1 ; j<n ; j++)
        {
            cout<<"\nEnter Distance between "<<city[i]<<" to "<<city[j]<<": ";
            cin>>distance[i][j];
            distance[j][i]=distance[i][j];
        }
    }
}

```

```
}  
  
void airport::show_graph()  
{  
    cout<<"\t";  
    for(int k=0;k<n;k++)  
    {  
        cout<<city[k]<<"\t";  
    }  
    cout<<endl;  
    for(int i=0;i<n;i++)  
    {  
        cout<<city[i]<<"\t";  
        for(int j=0;j<n;j++)  
        {  
            cout<<distance[i][j]<<"\t";  
        }  
        cout<<endl;  
    }  
}  
  
int main()  
{  
    airport obj;  
    obj.read_city();  
    obj.show_graph();  
}
```

OUTPUT:-

```

C:\Users\USER\Documents>employ.exe
Enter the no. of cities: 3
Enter city name:
1) pune
2) mumbai
3) nagpur
Enter Distance between pune to mumbai: 12
Enter Distance between pune to nagpur: 16
Enter Distance between mumbai to nagpur: 17
pune      mumbai  nagpur
0         12     16
mumbai    12     0     17
nagpur    16     17     0
Process returned 0 (0x0)   execution time : 30.438 s
Press any key to continue.

```

CONCLUSION: -

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

Cons: Consumes more space $O(V^2)$. Even if the graph is sparse (contains a smaller number of edges), it consumes the same space. Adding a vertex is $O(V^2)$ time.

ASSIGNMENT 4

AIM: To implement prims algorithm for minimum spanning tree.

OBJECTIVE: For a weighted graph G, find the minimum spanning tree using prims algorithm.

THEORY:

In computer science, Prim's (also known as Jarník's) algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm operates by building this tree one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex.

The algorithm was developed in 1930 by Czech mathematician Vojtěch Jarník and later rediscovered and republished by computer scientists Robert C. Prim in 1957 and Edsger W. Dijkstra in 1959. Therefore, it is also sometimes called the Jarník's algorithm, Prim–Jarník algorithm, Prim–Dijkstra algorithm or the DJP algorithm.

ALGORITHM:

- 1) Create a set *mstSet* that keeps track of vertices already included in MST.
 - 2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
 - 3) While *mstSet* doesn't include all vertices
 - a) Pick a vertex *u* which is not there in *mstSet* and has minimum key value.
 - b) Include *u* to *mstSet*.
 - c) Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v*
- The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

PROGRAM:

```
#include <iostream>

using namespace std;

class graph
{
    int a[100][100];

    int v;

public:
```

```
void insert_edge(int n1,int n2,int wt)
```

```
{
    if(n1-1>=v||n2-1>=v)
        cout<<"Vertex request out of range\n";
    else
    {
        a[n1-1][n2-1]=wt;
        a[n2-1][n1-1]=wt;
    }
}
```

```
void display()
```

```
{
    for(int i=0;i<v;i++)
    {
        for(int j=0;j<v;j++)
        {
            cout<<a[i][j]<<"\t";
        }
        cout<<endl;
    }
}
```

```
void update_v(int n)
```

```
{
    v=n;
}
```

```
void prims(int src)
```

```

{

    int sp[v],dist[v],visited[v],parent[v],c=0;

    for(int i=0;i<v;i++)
    {
        visited[i]=0;
        dist[i]=9999;
    }

    dist[src-1]=0;
    parent[src-1]=-1;
    for(int i=0;i<v;i++)
    {
        int min=9999,min_ind;
        for(int j=0;j<v;j++)
        {
            if(!visited[j] && dist[j]<min )
            {
                min=dist[j];
                min_ind=j;
            }
        }

        int U=min_ind;
        visited[U]=1;
        sp[c]=U;
        c++;

        for(int V=0;V<v;V++)
        {

```

```

        if(!visited[V] && a[U][V] && a[U][V]<dist[V] && dist[U]!=9999)

        {

            parent[V]=U;

            dist[V]=a[U][V];

        }

    }

}

for(int i=0;i<c;i++){

    cout<<sp[i]+1<<" link from "<<parent[i]+1<<endl;

}

cout<<endl;

}

};

int main(){

    char r;

    do

    {

        graph g;

        char op;

        int v;

        cout<<"Enter number of vertices: ";

        cin>>v;

        g.update_v(v);

        do{

            int c;

            cout<<"\n=====Menu=====\\n";

```

cout<<"1] Insert edge\n2] Increase number of vertices\n3] Display matrix\n4] Find shortest path\n";

cout<<"_____ \n";

cout<<"Enter your choice: ";

cin>>c;

switch(c){

case 1: {

int n1,n2,wt;

cout<<"Enter the nodes between which there is an edge\n";

cin>>n1>>n2;

cout<<"Enter weight: ";

cin>>wt;

g.insert_edge(n1,n2,wt);

}

break;

case 2: {

int n;

cout<<"Enter the number by which you wish to increase the vertices: ";

cin>>n;

v+=n;

g.update_v(v);

} break;

case 3: {

g.display();

}

break;

case 4: {

int src,dst;

cout<<"Source: ";

cin>>src;

g.prims(src);

}

break;

default:cout<<"Error 404.....page not found\n";

}

cout<<"Do you wish to continue(y/n): ";

cin>>op;

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

cout<<"Test pass(y/n): ";

cin>>r;

}while(r=='n' || r=='N');

cout<<"*****\n";

cout<<"* Thank You! *\n";

cout<<"*****\n";

return 0;

}

OUTPUT:

```

C:\Users\admin\Documents\SD PROGRAM\A4_prims.exe
Do you wish to continue(y/n): y

=====Menu=====
1) Insert edge
2) Increase number of vertices
3) Display matrix
4) Find shortest path

Enter your choice: 3
0 4 5 0
4 0 6 0
5 6 0 0
0 0 0 0
Do you wish to continue(y/n): y

=====Menu=====
1) Insert edge
2) Increase number of vertices
3) Display matrix
4) Find shortest path

Enter your choice: 4
Source: 0
1 link from 0320409
1 link from 1
1 link from 33
1 link from 1
Do you wish to continue(y/n):

```

CONCLUSION:

Time Complexity of the above program is $O(V^2)$. If the input graph is represented using adjacency list, then the time complexity of Prim's algorithm can be reduced to $O(E \log V)$ with the help of binary heap.

ASSIGNMENT 5

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

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

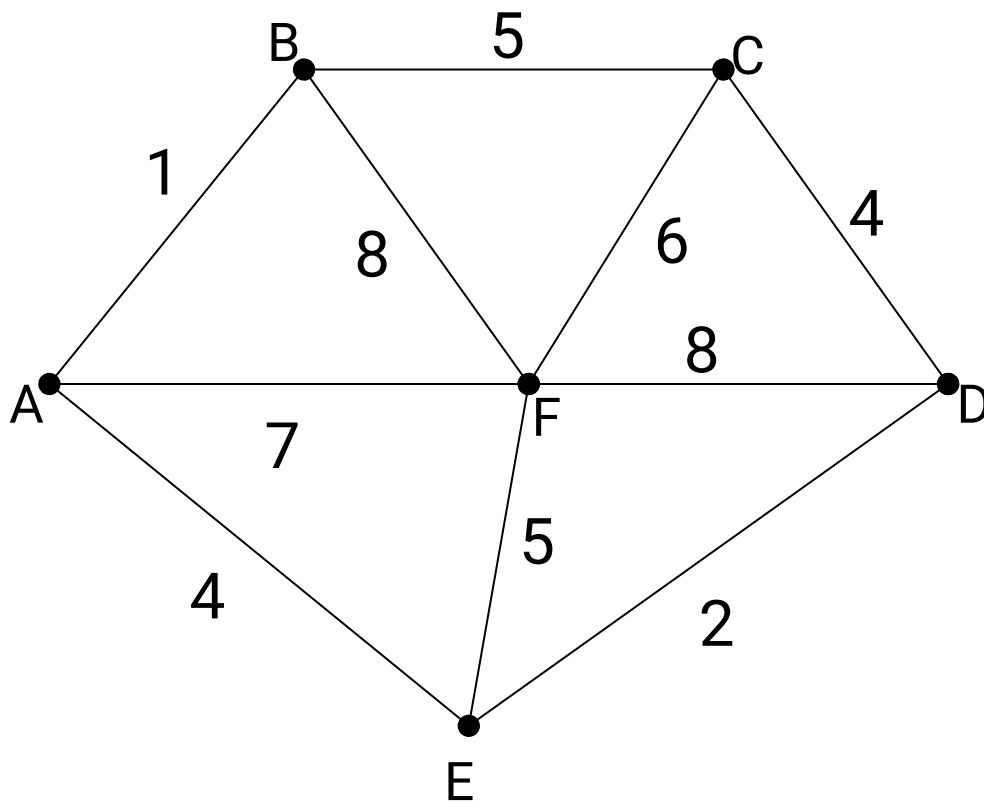
Theory: A spanning tree of the graph is a connected (if there is at least one path between every pair of vertices in a graph) subgraph in which there are no cycle. Suppose you have a connected undirected graph with a weight (or cost) associated with each edge. The cost of a spanning tree would be the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

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

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

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

Example:



The solution is

AB 1

ED 2

CD 4

AE 4

EF 5

Total weight of tree: 16

Algorithm:

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

{

1.Sort E in increasing order of weight

2.let $G=(V,E)$ and $T=(A,B), A=V, B$ is null

set and let $n = \text{count}(V)$

3.Initialize n set ,each containing a different element of v.

4.while($|B| < n-1$) do

```
begin
```

```
    e=<u,v>the shortest edge not yet considered
```

```
    U=Member(u)
```

```
    V=Member(v)
```

```
    if( Union(U,V))
```

```
        update in B and add the cost
```

```
    }}
```

```
end
```

```
5.T is the minimum spanning tree
```

```
}
```

Program code:

```
include<iostream>
```

```
using namespace std;
```

```
#define MAX 30
```

```
typedef struct edge
```

```
{
```

```
    int u,v,w;
```

```
}edge;
```

```
typedef struct edgelist
```

```
{
```

```
    edge data[MAX];
```

```
    int count;
```

```
}edgelist;
```

edgelist elist;

int G[MAX][MAX],n;

edgelist spanlist;

void kruskal();

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

void union1(int belongs[],int c1,int c2);

void sort();

void print();

int main()

{

int i,j;

cout<<"\nEnter number of city's:";

cin>>n;

cout<<"\nEnter the adjacency matrix of city ID's:\n";

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

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

cin>>G[i][j];

kruskal();

print();

}

```
void kruskal()
```

```
{
    int belongs[MAX],i,j,cno1,cno2;
    elist.count=0;

    for(i=1;i<n;i++)
        for(j=0;j<i;j++)
        {
            if(G[i][j]!=0)
            {
                elist.data[elist.count].u=i;
                elist.data[elist.count].v=j;
                elist.data[elist.count].w=G[i][j];
                elist.count++;
            }
        }

    sort();

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

    spanlist.count=0;

    for(i=0;i<elist.count;i++)
    {
        cno1=find(belongs,elist.data[i].u);
```

```
cno2=find(belongs,elist.data[i].v);
```

```
if(cno1!=cno2)
```

```
{
```

```
    spanlist.data[spanlist.count]=elist.data[i];
```

```
    spanlist.count=spanlist.count+1;
```

```
    union1(belongs,cno1,cno2);
```

```
}
```

```
}
```

```
}
```

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

```
{
```

```
    return(belongs[vertexno]);
```

```
}
```

```
void union1(int belongs[],int c1,int c2)
```

```
{
```

```
    int i;
```

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

```
        if(belongs[i]==c2)
```

```
            belongs[i]=c1;
```

```
}
```

```
void sort()
```

```
{
```

```

    int i,j;

    edge temp;

    for(i=1;i<elist.count;i++)
        for(j=0;j<elist.count-1;j++)
            if(elist.data[j].w>elist.data[j+1].w)
            {
                temp=elist.data[j];
                elist.data[j]=elist.data[j+1];
                elist.data[j+1]=temp;
            }
    }

    void print()
    {
        int i,cost=0;
        for(i=0;i<spanlist.count;i++)
        {
            cout<<"\n"<<spanlist.data[i].u<<" "<<spanlist.data[i].v<<" "<<spanlist.data[i].w;
            cost=cost+spanlist.data[i].w;
        }
        cout<<"\n\nMinimum cost of the telephone lines between the cities:"<<cost<<"\n";
    }

```

Output:

Enter number of city's:6

Enter the adjacency matrix of city ID's:

0 3 1 6 0 0

3 0 5 0 3 0

1 5 0 5 6 4

6 0 5 0 0 2

0 3 6 0 0 6

0 0 4 2 6 0

2 0 1

5 3 2

1 0 3

4 1 3

5 2 4

Minimum cost of the telephone lines between the cities:13

Conclusion:

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

ASSIGNMENT:6

AIM:

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

OBJECTIVE: To study and learn the concepts of heap data structure.

THEORY:

Heap definition- It is a Complete (Binary) Tree with each node having HEAP PROPERTY. Elements are filled level by level from left- to-right. If A is a parent node of B, then the key (the value) of node A is ordered with respect to the key of node B with the same ordering applying across the heap.

Types of heap: 1) Min heap

2) Max heap

○ **MAX HEAP definition:**

Complete (Binary) tree with the property that the **value of each node** is at least as large as the value of its children (i.e. \geq value of its children)

○ **MIN HEAP definition:**

Complete (Binary) tree with the property that the **value of each node** is at most as large as the value of its children (i.e. \leq value of its children)

ALGORITHM:

To maintain the max heap property i.e. MAXHEAPIFY

MAX-HEAPIFY (A, i, n)

1. $l \leftarrow \text{LEFT}(i)$
2. $r \leftarrow \text{RIGHT}(i)$
3. **if** $l \leq n$ and $A[l] > A[i]$
4. **then** $\text{largest} \leftarrow l$
5. **else** $\text{largest} \leftarrow i$
6. **if** $r \leq n$ and $A[r] > A[\text{largest}]$
7. **then** $\text{largest} \leftarrow r$
8. **if** $\text{largest} \neq i$

9. **then** exchange $A[i] \leftrightarrow A[\text{largest}]$

10. MAX-HEAPIFY (A, largest, n)

PROGRAM:

```
#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;

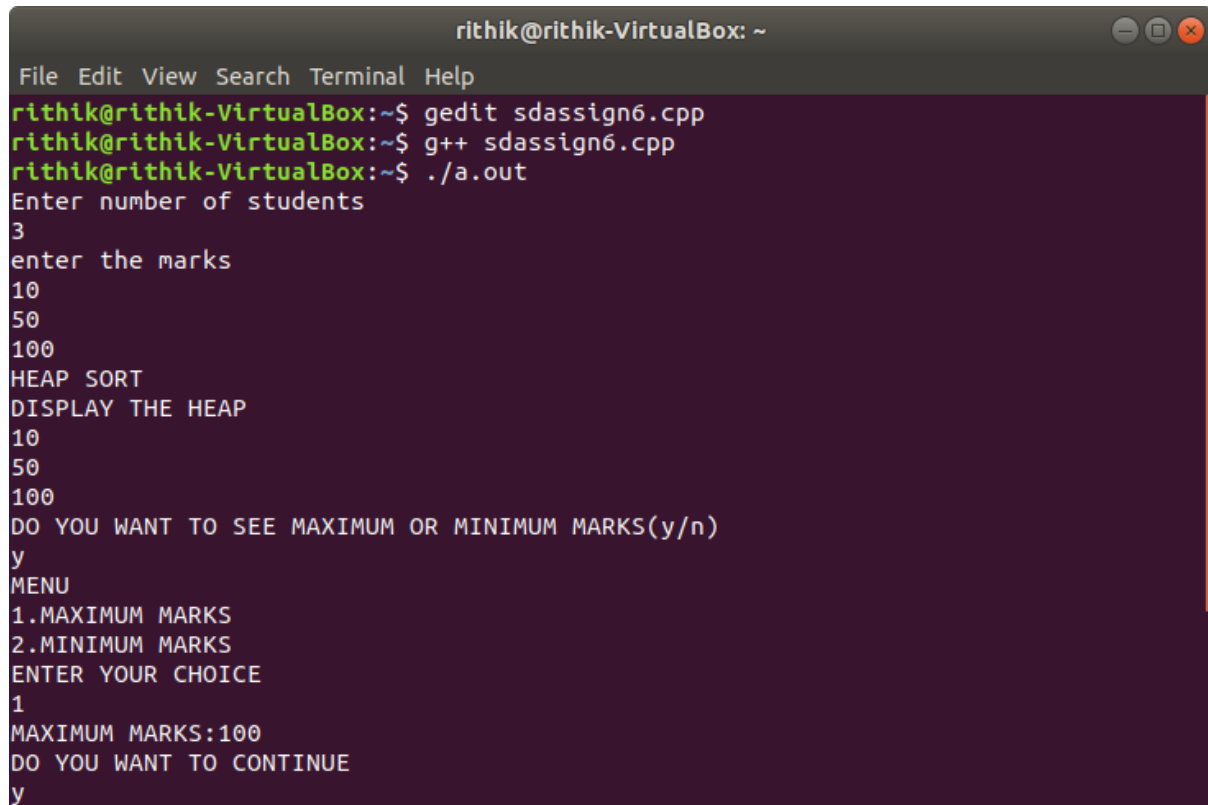
    if(largest!=i)
    {
        int t= a[i];
        a[i]=a[largest];
        a[largest]=t;
        heapify(a,n,largest);
    }
}
void heap:: printarray(int a[],int n)
{
    for(int i=0;i<n;i++)
```

```

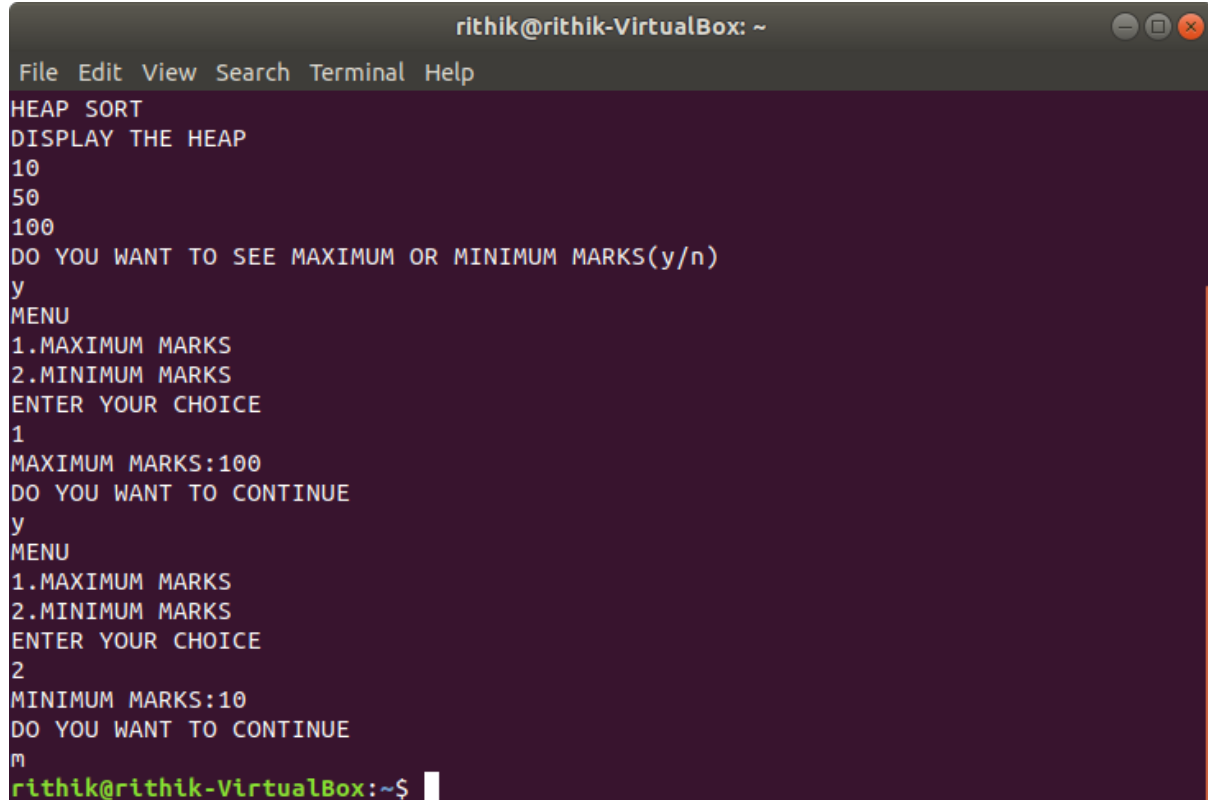
    {
        cout<<a[i]<<" ";
        cout<<"\n";
    }
}
void heap::maximum(int a[],int n)
{
    cout<<"MAXIMUM MARKS:"<<a[n-1]<<endl;
}
void heap::minimum(int a[],int n)
{
    cout<<"MINIMUM MARKS:"<<a[0]<<endl;
}
int main()
{
    heap h;
    int a[100],n;
    cout<<"Enter number of students"<<endl;
    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:

```
rithik@rithik-VirtualBox: ~
File Edit View Search Terminal Help
rithik@rithik-VirtualBox:~$ gedit sdassign6.cpp
rithik@rithik-VirtualBox:~$ g++ sdassign6.cpp
rithik@rithik-VirtualBox:~$ ./a.out
Enter number of students
3
enter the marks
10
50
100
HEAP SORT
DISPLAY THE HEAP
10
50
100
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:100
DO YOU WANT TO CONTINUE
y
```



```
rithik@rithik-VirtualBox: ~  
File Edit View Search Terminal Help  
HEAP SORT  
DISPLAY THE HEAP  
10  
50  
100  
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:100  
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  
n  
rithik@rithik-VirtualBox:~$
```

CONCLUSION: We successfully implemented heap data structure.

ASSIGNMENT 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 and learn the concepts of double hashing.

THEORY:

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:

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

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

(We repeat by increasing i when collision occurs)

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

A popular second hash function is:

$\text{hash2}(\text{key}) = \text{PRIME} - (\text{key} \% \text{PRIME})$ where PRIME is a prime smaller than the TABLE_SIZE .

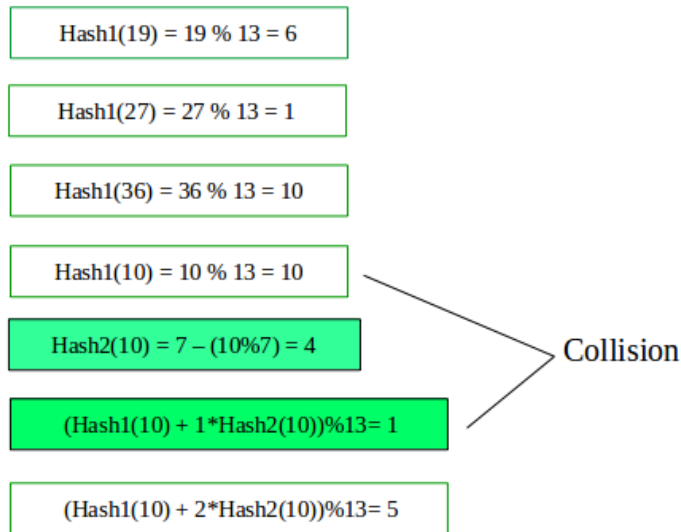
A good second Hash function is:

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

ALGORITHM:

Lets say, Hash1 (key) = key % 13

Hash2 (key) = 7 – (key % 7)



PROGRAM:

```
#include <iostream>
using namespace std;
class dr
{
    int n=10;
    int arr[100][3];
    int c;
public:
    dr()
    {
        cout<<"Table of size "<<n<<" created\n";
        for(int i=0;i<n;i++)
        {
            arr[i][0]=0;
            arr[i][1]=-1;
            arr[i][2]=-1;
        }
        c=0;
    }
    void add(int,int);
    int find_key(int);
    void display();
    void update_val(int,int);
};
void dr::add(int key,int value)
{
    int new_hash_addr1,new_hash_addr2,main_hash_addr=-1,j=0;
```

```

    if(this->find_key(key)!=-1)
    {
        cout<<"Key already exists\n";
        return;
    }
    if(c==(n-1))
    {
        cout<<"Table full, request denied\n";
    }
    new_hash_addr1=(key)%n;
    new_hash_addr1=1+(key%(n-1));
    if(arr[new_hash_addr1][1]==-1)
    {
        arr[new_hash_addr1][0]=key;
        arr[new_hash_addr1][1]=value;
    }
    else if(arr[new_hash_addr2][1]==-1)
    {
        arr[new_hash_addr2][0]=key;
        arr[new_hash_addr2][1]=value;
    }
    else
    {
        while(arr[new_hash_addr2][2]!=-1)
        {
            main_hash_addr=new_hash_addr2;
            new_hash_addr2=arr[main_hash_addr][2];
        }
        main_hash_addr=new_hash_addr2;
        for(int i=0;i<n;i++)
        {
            new_hash_addr2=(main_hash_addr+i)%n;
            if(arr[new_hash_addr2][1]==-1)
            {
                arr[new_hash_addr2][0]=key;
                arr[new_hash_addr2][1]=value;
                arr[main_hash_addr][2]=new_hash_addr2;
                c++;
                break;
            }
        }
    }
}

void dr::display()
{
    cout<<"Key\t\tValue\t\tChain\n";
    for(int i=0;i<n;i++)
    {
        cout<<arr[i][0]<<"\t\t"<<arr[i][1]<<"\t\t"<<arr[i][2]<<endl;
    }
}

int dr::find_key(int key)
{
    int search_addr=key%n,f=0;
    while(arr[search_addr][0]!=key && arr[search_addr][2]!=-1)

```

```

{
    search_addr=arr[search_addr][2];
}
if(arr[search_addr][0]==key)
{
    return arr[search_addr][1];
}
else if(arr[search_addr][2]==-1)
{
    return -1;
}
}
int main()
{
    char r;
    do
    {
        char op;
        dr table;
        int c;
        do
        {
            cout<<"-----Menu-----\n";
            cout<<"1] Insert value\n2] Display\n";
            cout<<"_____ \n";
            cout<<"Enter your choice: ";
            cin>>c;
            switch(c)
            {
                case 1: {
                    int key,val;
                    cout<<"Enter key: ";
                    cin>>key;
                    cout<<"Enter value: ";
                    cin>>val;
                    table.add(key,val);
                }
                break;
                case 2: table.display();
                break;
                default:cout<<"Invalid\n";
            }
            cout<<"\nDo you wish to go again? ";
            cin>>op;
        }while(op=='y' || op=='Y');
        cout << "Test pass?(y/n): " << endl;
        cin>>r;
    }while(r=='n' || r=='N');
    cout<<"*****\n";
    cout<<"*   Thank You!   *\n";
    cout<<"*****\n";
    return 0;
}

```


OUTPUT:

```

himanshu@himanshu-Inspiron-3558: ~/new/sd
-----Menu-----
1) Insert value
2) Display

Enter your choice: 1
Enter key: 5
Enter value: 45

Do you wish to go again? y
-----Menu-----
1) Insert value
2) Display

Enter your choice: 1
Enter key: 4
Enter value: 56

Do you wish to go again? y
-----Menu-----
1) Insert value
2) Display

Enter your choice: 1
Enter key: 1
Enter value: 49

Do you wish to go again? y
-----Menu-----
1) Insert value
2) Display

Enter your choice: 1

```

```

himanshu@himanshu-Inspiron-3558: ~/new/sd
-----Menu-----
1) Insert value
2) Display

Enter your choice: 1
Enter key: 1
Enter value: 49

Do you wish to go again? y
-----Menu-----
1) Insert value
2) Display

Enter your choice: 1
Enter key: 6
Enter value: 89

Do you wish to go again? y
-----Menu-----
1) Insert value
2) Display

Enter your choice: 1
Enter key: 7
Enter value: 46

Do you wish to go again? y
-----Menu-----
1) Insert value
2) Display

Enter your choice: 1
Enter key: 6
Enter value: 44

```

```

himanshu@himanshu-Inspiron-3558: ~/new/sd
File Edit View Search Terminal Help
Enter your choice: 1
Enter key: 6
Enter value: 89
Do you wish to go again? y
-----Menu-----
1) Insert value
2) Display
Enter your choice: 1
Enter key: 7
Enter value: 46
Do you wish to go again? y
-----Menu-----
1) Insert value
2) Display
Enter your choice: 2
Key      Value      Chain
0        -1         -1
0        -1         -1
1        49         -1
0        -1         -1
0        -1         -1
4        56         -1
5        45         -1
6        89         -1
7        46         -1
0        -1         -1
Do you wish to go again? 

```

CONCLUSION: We successfully implemented open addressing using double hashing.

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 employee. If record of student does not exist an appropriate message is displayed. If it is, then the system displays, the student details. Use sequential file to main the data.

OBJECTIVE: -

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

THEORY: -

A **sequential file** is one that contains and stores data in chronological order. The data itself may be ordered or un ordered in the file. Unlike a [random-access file](#), sequential files must be read from the beginning, up to the location of the desired data. Sequential files are often stored on [sequential access](#) devices, like a magnetic tape.

A sequential file contains records organized by the order in which they were entered. The order of the records is fixed.

Records in sequential files can be read or written only sequentially.

After you place a record into a sequential file, you cannot shorten, lengthen, or delete the record. However, you can update a record if the length does not change. New records are added at the end of the file.

If the order in which you keep records in a file is not important, sequential organization is a good choice whether there are many records or only a few. Sequential output is also useful for printing reports.

ALGORITHM: -

1.CREATE A FILE HAVING COLLECTION OF RECORDS

```
void Create ()
{
    char ch='y';
    ofstream seqfile;
    seqfile.open("stud.DAT",ios::out|ios::binary);
    do
```

```

{
    cout<<"\n Enter roll no: ";
    cin>>Records.rollno;
    seqfile.write((char*)&Records,sizeof(Records));
    cout<<"\nDo you want to add more records?";
    cin>>ch;
    }while(ch=='y');
    seqfile.close();
}

```

2.DISPLAY OF FILE

Void Display()

```

{
    ifstream seqfile;
    seqfile.open("stud.DAT",ios::in|ios::binary);
    seqfile.seekg(0,ios::beg);
    cout<<"\n The Contents of file are ..."<<endl;
    while (seqfile.read((char *)&Records,sizeof(Records)))
    {
        if (Records.rollno!=-1)
        {
            cout<<"\nRoll No: "<<Records.rollno;
        }
    }
    seqfile.close();
}

```

3.SEARCHING A RECORD

int: Search ()

```

{
    fstream seqfile;
    int id,pos,offset;
    cout<<"\n Enter the no for searching the record ";
    cin>>id;
    seqfile.open("stud.DAT",ios::in|ios::binary);
    pos=-1;
    seqfile.seekg(0,ios::beg);
    int i=0;
    while(seqfile.read((char *)&Records,sizeof(Records)))
    {
        if(id==Records.rollno)
        {
            pos=i;
            break;
        }
        i++;
    }
    seqfile.close();
    return pos;
}

```

4.DELETION OF RECORD:-

```

void deletion(){
    int id,pos;
    cout<<"For deletion"<<endl;
    fstream seqfile;
    pos=Search();
}

```

```

    seqfile.open("stud.DAT",ios::in|ios::binary|ios::out);

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

    if(pos==-1)
    {
        cout<<"\n Record is not present in the file";

        return;
    }

    int offset=pos*sizeof(Records);

    seqfile.seekp(offset);

    Records.rollno=-1;

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

    seqfile.seekg(0);

    seqfile.close();
}

```

PROGRAM CODE: -

```

#include<iostream>

#include<fstream>

#include<string.h>

using namespace std;

typedef struct data
{
    char name[10];

    int rollno;

    char div;

    char address[100];

}Rec;

```

class student

```
{

    Rec Records;

public:
    void Create();
    void Display();
    int Search();
    void deletion();
};

void student::Create()
{
    char ch='y';
    ofstream seqfile;
    seqfile.open("stud.DAT",ios::out|ios::binary);
    do
    {
        cout<<"\n Enter Name: ";
        cin>>Records.name;
        cout<<"\n Enter roll no: ";
        cin>>Records.rollno;
        cout<<"\n Enter division";
        cin>>Records.div;
        cout<<"\n Enter ADDRESS: ";
        cin>>Records.address;
        seqfile.write((char*)&Records,sizeof(Records));
        cout<<"\nDo you want to add more records?";
```

```

cin>>ch;

}while(ch=='y');

seqfile.close();
}

void student::Display()
{
    ifstream seqfile;

    seqfile.open("stud.DAT",ios::in|ios::binary);
    seqfile.seekg(0,ios::beg);

    cout<<"\n The Contents of file are ..."<<endl;
    while(seqfile.read((char *)&Records,sizeof(Records)))
    {
        if(Records.rollNo!=-1)
        {
            cout<<"\nName: "<<Records.name<<flush;
            cout<<"\nRoll No: "<<Records.rollNo;
            cout<<"\nDivision : "<<Records.div;
            cout<<"\nAddress: "<<Records.address;
            cout<<"\n";
        }
    }
    seqfile.close();
}

int student::Search()
{
    fstream seqfile;
    int id,pos,offset;

```



```

cout<<"\n Enter the roll no for searching the record ";

cin>>id;

seqfile.open("stud.DAT",ios::in|ios::binary);

pos=-1;

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

int i=0;

while(seqfile.read((char *)&Records,sizeof(Records)))
{
    if(id==Records.rollno)
    {
        pos=i;
        break;
    }
    i++;
}

seqfile.close();

return pos;
}

void student::deletion()
{
    int id,pos;

    cout<<"For deletion"<<endl;

    fstream seqfile;

    pos=Search();

    seqfile.open("stud.DAT",ios::in|ios::binary|ios::out);

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

    if(pos!=-1){

```

```
    cout<<"\n Record is not present in the file";

    return;

}

int offset=pos*sizeof(Records);

seqfile.seekp(offset);

strcpy(Records.name,"");

Records.rollno=-1;

Records.div=-1;

strcpy(Records.address,"");

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

seqfile.seekg(0);

seqfile.close();

}

int main()

{

student e;

char ans='y';

int choice,key;

int h=0;

do

{

    cout<<"1.Create"<<endl;

    cout<<"2.Display"<<endl;

    cout<<"3.Search"<<endl;

    cout<<"4.Delete"<<endl;

    cout<<"Enter your choice"<<endl;

    cin>>choice;
```

```
    switch(choice)
    {
    case 1:
        e.Create();
        break;
    case 2:
        e.Display();
        break;
    case 3:
        h=e.Search();
        if(h<0)
            cout<<"\n Student not present in file"<<endl;
        else
            cout<<"\n Student is present in file"<<endl;
        break;
    case 4:
        e.deletion();
        break;
    }
    cout<<"Do you want to continue"<<endl;
    cin>>ans;
}while (ans=='y');
return 0;
}
```

OUTPUT:-

```

C:\Users\admin\Desktop\SD2\assignment7\sassignment8.exe
1.Create
2.Display
3.Search
4.Delete
Enter your choice
1

Enter Name: RAJ

Enter roll no: 10

Enter division c

Enter ADDRESS: wedfgh

Do you want to add more records?y

Enter Name: ayush

Enter roll no: 20

Enter division a

Enter ADDRESS: werghj

Do you want to add more records?n
Do you want to continue
y
1.Create
2.Display

```

```

C:\Users\admin\Desktop\SD2\assignment7\sassignment8.exe
2.Display
3.Search
4.Delete
Enter your choice
2

The Contents of file are ...

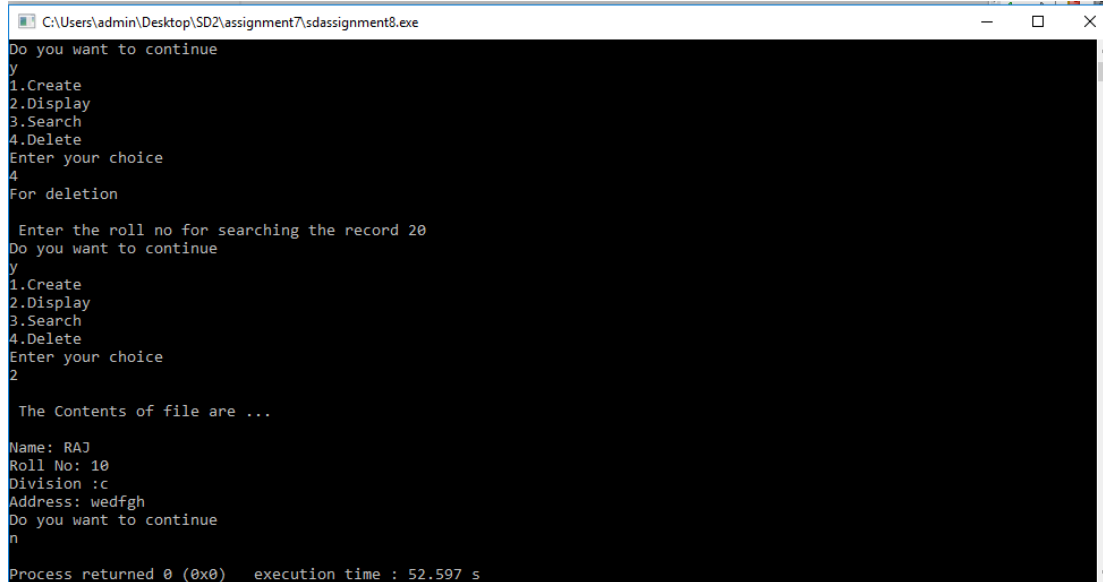
Name: RAJ
Roll No: 10
Division :c
Address: wedfgh

Name: ayush
Roll No: 20
Division :a
Address: werghj
Do you want to continue
y
1.Create
2.Display
3.Search
4.Delete
Enter your choice
3

Enter the roll no for searching the record 20

Student is present in file
Do you want to continue

```



```
C:\Users\admin\Desktop\SD2\assignment7\sdassignment8.exe
Do you want to continue
y
1.Create
2.Display
3.Search
4.Delete
Enter your choice
4
For deletion
Enter the roll no for searching the record 20
Do you want to continue
y
1.Create
2.Display
3.Search
4.Delete
Enter your choice
2
The Contents of file are ...
Name: RAJ
Roll No: 10
Division :c
Address: wedfgh
Do you want to continue
n
Process returned 0 (0x0)   execution time : 52.597 s
```

CONCLUSION: -

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

ASSIGNMENT 9

Title:

Index Sequential File

Problem Statement:

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.

Software and Hardware Requirement:

64-bit Open source Linux or its derivative.

Open Source C++ Programming tool like G++/GCC.

Theory:

Index Sequential File:

This is basically a mixture of sequential and indexed file organization 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 organization gives the index a tree structure. Obviously, this type of file organization will require a direct access device, such as a hard disk.

Indexed sequential file organization is very useful where records are often retrieved randomly and are also processed in (sequential) key order. Banks may use this organization 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.

PROGRAM:

```
#include <iostream>

#include<fstream>

#include<string>

using namespace std;

typedef struct seq_file
{
    int id;

    char name[20],desg[20];

    long int sal;
} record;

typedef struct ind_file
{
    int id;
} index;

class file
{
    record data;

    index info;

public:

    void get_data()
```

```
{  
    cout<<"Enter id: ";  
    cin>>data.id;  
    cout<<"Enter name: ";  
    cin>>data.name;  
    cout<<"Enter designation: ";  
    cin>>data.desg;  
    cout<<"Enter salary: ";  
    cin>>data.sal;  
    info.id=data.id;  
}  
  
void add()  
{  
    fstream out1;  
    fstream out2;  
    out1.open("pos.txt",ios::app);  
    out2.open("rec.txt",ios::app);  
    get_data();  
    out2.write((char*)&data,sizeof(data));  
    out1.write((char*)&info,sizeof(info));  
    out1.close();  
    out2.close();  
}  
  
void search_rec(int id)  
{  
    int pos=0,loc=-1;  
    fstream out1;
```



```

    fstream out2;

    out1.open("pos.txt");
    out2.open("rec.txt");
    loc=sizeof(info)*pos;
    out2.seekg(loc,ios::beg);
    for(pos=0;out2.read((char*)&info,sizeof(info));pos++)
    {
        loc=sizeof(info)*pos;
        out2.seekg(loc,ios::beg);
        out2.read((char*)&info,sizeof(info));
        if(info.id==id)
        {
            break;
        }
    }
    if(loc==-1)
    {
        cout<<"Record not found\n";
    }
    else
    {
        pos--;
        pos=sizeof(data)*pos;
        out1.seekg(pos,ios::beg);
        out1.read((char*)&data,sizeof(data));
        cout<<"Record found\n";
        cout<<data.id<<"\t"<<data.name<<"\t"<<data.desg<<"\t"<<data.sal<<endl;
    }
}

```

```

    }

    out1.close();

    out2.close();

}

};

int main()
{
    char r;

    do
    {
        char op;
        file f;
        do
        {
            int c;

            cout<<"\n=====Menu=====\\n";

            cout<<"1] Add record\\n2] Search record\\n3] Delete record\\n";

            cout<<"_____\\n";

            cout<<"Enter your choice: ";

            cin>>c;

            switch(c)
            {
                case 1: {
                    f.add();
                }

                break;

                case 2: {

```

```

        int id;

        cout<<"Enter id to search: ";

        cin>>id;

        f.search_rec(id);

    }

    break;

case 3: {

    }

    break;

case 4: {

    }

    break;

default:cout<<"Error 404.....page not found\n";

}

cout<<"Do you wish to continue(y/n): ";

cin>>op;

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

cout<<"Test pass(y/n): ";

cin>>r;

}while(r=='n' || r=='N');

cout<<"*****\n";

cout<<"*   Thank You!   *\n";

cout<<"*****\n";

return 0;

}

```

OUTPUT:

```
Select C:\Users\admin\Documents\SD PROGRAM\A9_emsq.exe

=====Menu=====
1] Add record
2] Search record
3] Delete record

Enter your choice: 1
Enter id: 12345
Enter name: vedika
Enter designation: ceo
Enter salary: 250000
Do you wish to continue(y/n): y

=====Menu=====
1] Add record
2] Search record
3] Delete record

Enter your choice: 1
Enter id: 2354
Enter name: pooja
Enter designation: vp
Enter salary: 260000
Do you wish to continue(y/n): y

=====Menu=====
1] Add record
2] Search record
3] Delete record
```

Conclusion:

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