UIT1412 Programming & Data Structures Lab II

Done By : M Badri Narayanan

IT A Second Year

Register Number : 185002018

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No. 1

EXP NO : 1
DATE : 12/12/19

M Badri Narayanan
Register No : 185002018

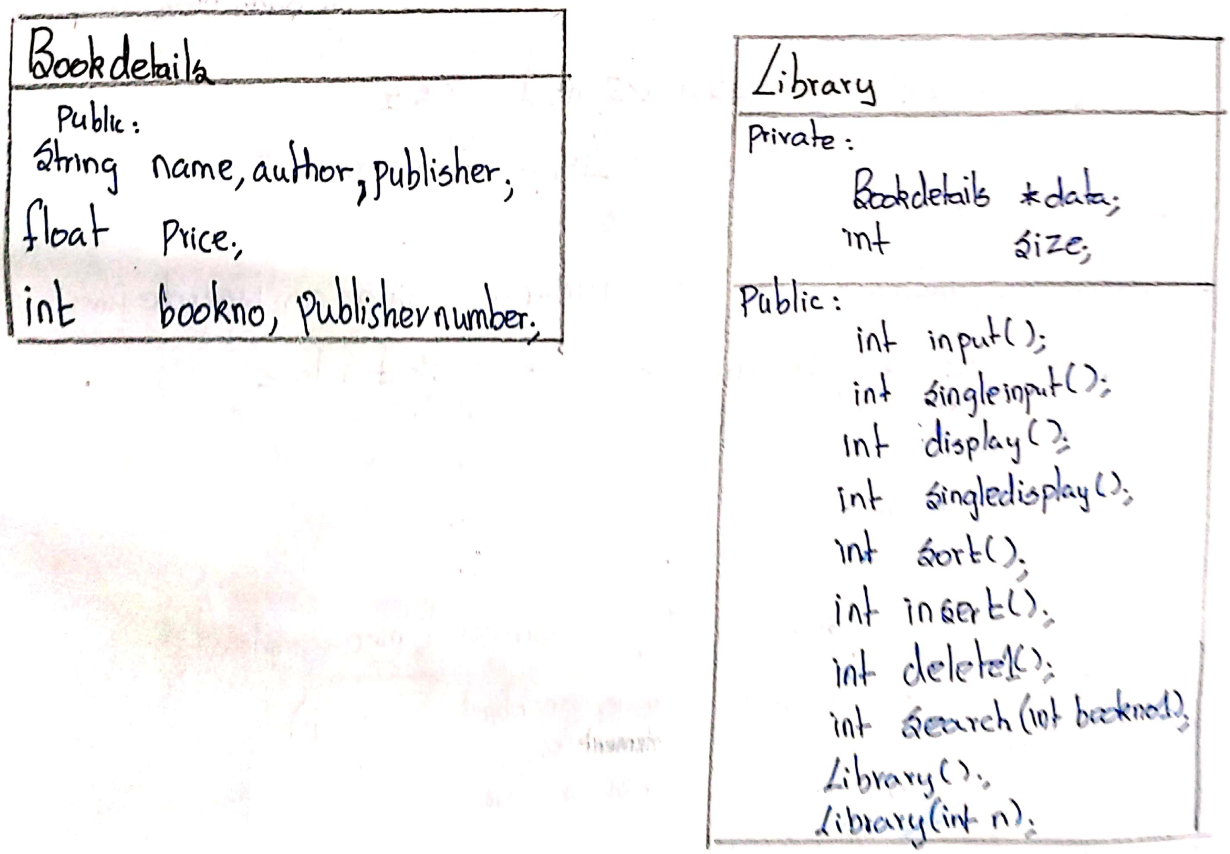
User Defined Class
- Library Management

Aim: To implement a C++ program using an userdefined class & perform operations on that class

Time Complexities

SNO	Function	Operation	Time Complexity
1	Singleinput()	Creates an object of Bookdetails class	O(1)
2	Single display()	Displays an object of Bookdetails class	O(1)
3	input()	Creates an array of Library class objects	O(n)
4	display()	Displays an array of Library class objects	O(n)
5	sort()	Performs insertion sort on an array of Library class object	$O(n^2)$

Class Diagram



M Badri Narayanan
Register No: 185002018

SNO	Function	Operation	Time Complexity
6	insert()	Inserts a data item into the array	O(n)
7	delete1()	Deletes a data item from the array	O(n)
8	search(int bookno1) Parameter: Bookno to be searched	Searches for the bookno given in parameter in the array	O(n)
9	Library()	Constructor Assigns size to 15 & all other values to NULL	O(n)
10	Library(int n) Parameter: Value of size	Parameterized Constructor Assigns size to n & all other values to NULL	O(n)

n: Size of the Library Object array

book.h

```
#ifndef book_h
#define book_h
#include<iostream>
#include<stdlib.h>
#include<string>
#include<string.h>
using namespace std;

class BookDetails
{
    public :
    string  name;
    string  author;
    float   price;
    int     bookno;
    string  publisher;
    int     publishernumber;
};
class Library
{
    private :
            BookDetails *data;
            int     size;
    public  :
            int     input();
            BookDetails singleinput();
            int     display();
            int     singledisplay(BookDetails data);
            int     sort();
            int     insert();
            int     delete1();
            int     search(int bookno1);
            Library();
            Library(int n);
};
#endif
```

book.cpp

```
#include"book.h"
BookDetails Library ::singleinput()
{
    BookDetails data;
    cout<<" Enter Book Number : ";
    cin>>data.bookno;
    cout<<" Enter Book Name : ";
    cin>>data.name;
    cout<<" Enter Author Name : ";
    cin>>data.author;
    cout<<" Enter Publisher Name : ";
    cin>>data.publisher;
    cout<<" Enter Publisher Number : ";
    cin>>data.publishernumber;
    cout<<" Enter Price : ";
    cin>>data.price;
    cout<<endl;
    return data;
}
int Library :: singledisplay(BookDetails data)
{
    cout<<" Book Number : ";
    cout<<data.bookno<<endl;
    cout<<" Book Name : ";
    cout<<data.name<<endl;
    cout<<" Author Name : ";
    cout<<data.author<<endl;
    cout<<" Publisher Name : ";
    cout<<data.publisher<<endl;
    cout<<" Publisher Number : ";
    cout<<data.publishernumber<<endl;
    cout<<" Price : ";
    cout<<data.price;
    cout<<endl<<endl;
    return 1;
}
```

```
int Library::input()
{
    cout<<"\n\n Input The Book Details \n\n";
    cout<<" Enter Number Of Books : ";
    cin>>size;
    cout<<"\n\n";
    for(int i=0;i<size;i++)
    {
        data[i]= singleinput();
    }
    return 1;
}
int Library :: display()
{
    cout<<"\n\n Displaying The Books \n\n";
    for(int i =0;i<size;i++)
    {
            singledisplay(data[i]);
    }
    return 1;
}
Library::Library()
{
    size=15;
    data=new BookDetails[size];
    for(int i=0;i<size;i++)
    {
        data[i].bookno=0;
        data[i].name = "\0";
        data[i].author = "\0";
        data[i].publishernumber=0;
        data[i].price=0;
        data[i].publisher = "\0";
    }
}
Library :: Library(int n)
{
    size=n;
    data=new BookDetails[size];
    for(int i=0;i<size;i++)
    {
```

```
            data[i].bookno=0;
            data[i].name = "\0";
            data[i].author = "\0";
            data[i].publishernumber=0;
            data[i].price=0;
            data[i].publisher = "\0";
        }
}
int Library::sort()
{
        int      n,i,j;
        n=size;
        for(i=1;i<n;i++)
        {
            BookDetails t=data[i];
            for(j=i-1;((j>=0)&&(t.bookno<data[j].bookno));j--)
            {
                data[j+1]=data[j];
            }
            data[j+1]=t;
        }
        cout<<"\n\n Sorted In Ascending Order \n\n";
        return 1;
}
int Library ::search(int bookno1)
{
        int      flag = -1;
            for(int i=0;((i<size)&&(flag==-1));i++)
            {
                if(data[i].bookno==bookno1)flag=i+1;
            }
            if(flag==-1)cout<<"\n\n Data Not Found \n\n";
            else cout<<"\n\n Data Found @ : "<<flag<<"\n\n";
            return flag;
}
int Library ::insert()
{
        BookDetails b;
        int      index;
            cout<<"\n\n Enter Details Of Book To Be Inserted \n\n";
            b = singleinput();
```

```
            cout<<" Enter Index Of Where The Record Should Be
                                Inserted : ";
            cin>>index;
            if(index>size)
            {
                cout<<"\n\n Book Cannot Be Inserted "<<endl;
                return -1;
            }
            else
            {
                for(int i=size;i>=index;i--)
                {
                    data[i]=data[i-1];
                }
                data[index]=b;
                size+=1;
                cout<<"\n\n Book Inserted \n\n";
                return 1;
            }
}
int Library ::delete1()
{
    int     n;
    int     flag,bookno1;
        n=size;
        cout<<"\n\n Enter BookNo To Be Deleted : ";
        cin>>bookno1;
        for(int i=0,flag=-1;i<n;i++)
        {
            if(data[i].bookno==bookno1)
            {
                for(int j=i;j<n-1;j++)
                {
                    data[j]=data[j+1];
                }
                flag=1;
                n--;
                i--;
            }
        }
        if(flag==-1)cout<<"\n\n Book Not Found \n\n";
```

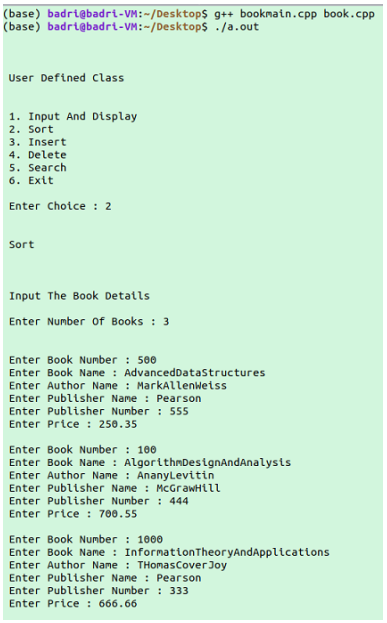
```
        else
        {
            cout<<"\n\n Book Deleted \n\n";
            size=n;
        }
        return flag;
}
```

bookmain.cpp

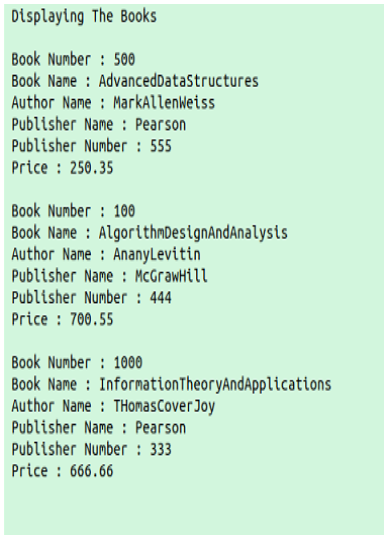
```
#include"book.h"
int main()
{
    Library a;
    int     option,choice;
        do
        {
            cout<<"\n\n\n User Defined Class \n\n\n";
            cout<<" 1. Input And Display \n";
            cout<<" 2. Sort \n";
            cout<<" 3. Insert \n";
            cout<<" 4. Delete \n";
            cout<<" 5. Search \n";
            cout<<" 6. Exit \n";
            cout<<"\n Enter Choice : ";
            cin>>option;
            switch(option)
            {
                case 1 :
                {
                    cout<<"\n\n Input and Display \n\n";
                    a.input();
                    a.display();
                    break;
                }
                case 2 :
                {
                    cout<<"\n\n Sort \n\n";
                    a.input();
                    a.display();
                    a.sort();
```

```
            a.display();
            break;
        }
        case 3 :
        {
            cout<<"\n\n Insert \n\n";
            a.input();
            a.display();
            a.insert();
            a.display();
            break;
        }
        case 4 :
        {
            cout<<"\n\n Delete \n\n";
            a.input();
            a.display();
            a.delete1();
            a.display();
            break;
        }
        case 5 :
        {
            int  bookno1,temp;
            cout<<"\n\n Search \n\n";
            a.input();
            a.display();
            cout<<"\n\n Enter Book No
                        To Be Searched : ";
            cin>>bookno1;
            temp=a.search(bookno1);
            break;
        }
        default:
        {
            cout<<"\n\n Exiting The Program.....\n\n";
            break;
        }
    }
    cout<<"\n\n Do You Want To Continue
                ( Type 0 Or 1 ) :  ";
```

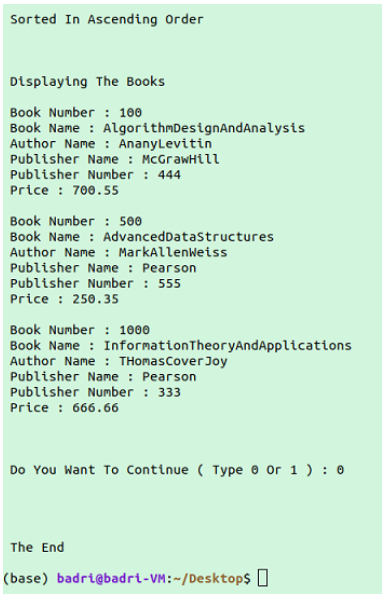
```
            cin>>choice;
            cout<<endl<<endl;
        }while(choice);
        cout<<"\n\n The End \n\n";
        return 0;
}
```



(a)



(b)



(c)

Figure 1: Output For I

Result A C++ program was implemented using a user defined class and operations were performed on it.

SSN COLLEGE OF ENGINEERING Sheet No.....
RECORD SHEET

M Badri Narayanan
Register No: 185002018

EXP NO: 2
DATE: 19/12/19

Dynamic Memory Access, Constructor, Destructor - Matrices Application

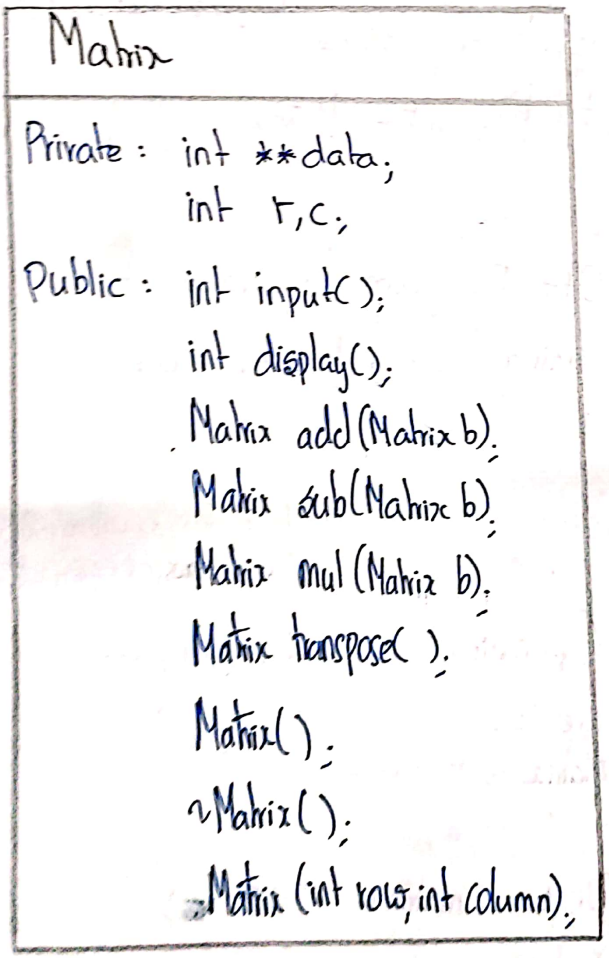
Aim : To create a C++ program to implement matrix operation using Dynamic Memory Access, Constructors, Destructors

Time Complexities

r: No of rows of calling matrix
c: No of columns of calling matrix

SNO	Function	Operation	Time Complexity
1	int input()	Creates a matrix of r rows & c columns	O(r*c)
2	int display()	Displays a matrix of r rows & c columns	O(r*c)
3	Matrix add(Matrix b) Parameter: Matrix Operand 2 Return Datatype: Matrix Addition Resultant	Returns the resultant matrix after performing addition between the calling matrix & the matrix passed as parameter	O(r*c)

Class Diagram



SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

SNO	Function	Operation	Time Complexities
4	Matrix &ub (Matrix b) Parameter: Matrix Subtraction Operand 2 ReturnDatatype: Resultant Matrix	Returns the resultant matrix after performing Subtraction between calling matrix & matrix passed as parameter	$O(r * c)$
5	Matrix mul (Matrix b) Parameter: Matrix Multiplication Operand 2 Return Datatype: Resultant Matrix	Returns the resultant matrix after performing Multiplication between calling matrix & matrix passed as parameter	$O(r_1 * c_2)$ r_1: No of rows of calling matrix c_2: No of columns of parameter matrix
6	Matrix transpose() Return Datatype: Resultant Matrix	Returns the transpose of the calling matrix	$O(c * r)$
7	Matrix ( ) Constructor	Assigns row & column to 15 & creates a 15x15 matrix with zero	$O(15^2)$
8	Matrix (int rows, int columns) Constructor Parameter: Values for r & c	Assigns r to rows & c to columns & creates a rows * column matrix with zero	$O(rows * columns)$
9	~Matrix ( )	Destructor. Deallocates the memory created for matrix	$O(r)$

matrix.h

```
matrix.h
#ifndef matrix_h
#define matrix_h
#include<iostream>
#include<stdlib.h>
using namespace std;

class Matrix
{
        int     **data;
        int     r,c;
    public :
        int     input();
        int     display();
        Matrix  add(Matrix b);
        Matrix  sub(Matrix b);
        Matrix  mul(Matrix b);
        Matrix  transpose();
        Matrix();
        ~Matrix();
        Matrix(int row,int column);
};
#endif
```

matrix.cpp

```
#include"matrix.h"

int Matrix :: input()
{
    cout<<"\n\n Entering The Matrix \n\n";
    cout<<" Enter Number Of Rows :  ";
    cin>>r;
    cout<<" Enter Number Of Columns : ";
    cin>>c;
    cout<<"\n\n";
    for(int i=0;i<r;i++)
    {
```

```
            for(int j=0;j<c;j++)
            {
                cout<<" Enter Data :  ";
                cin>>data[i][j];
            }
            cout<<"\n\n";
        }
        cout<<endl;
        return 1;
    }


    int Matrix :: display()
    {
        cout<<"\n\n Displaying The Matrix \n\n";
        for(int i=0;i<r;i++)
        {
            for(int j=0;j<c;j++)
            {
                cout<<" "<<data[i][j]<<" ";
            }
            cout<<endl;
        }
        cout<<endl<<endl;
        return 1;

    }
    Matrix Matrix :: add(Matrix b)
    {
        Matrix temp;
            if((r==b.r)&&(c==b.c))
            {
                cout<<" \n\n Matrix Addition Possible \n\n";
                for(int i=0;i<r;i++)
                {
                    for(int j=0;j<c;j++)
                    {
                        temp.data[i][j]=data[i][j]+b.data[i][j];
                    }
                }
            }
```

```
            else cout<<"\n\n Matrix Addition Not Possible \n\n ";
            temp.r=r;
            temp.c=c;
            return temp;
}
Matrix Matrix :: sub(Matrix b)
{
    Matrix  temp;
            if((r==b.r)&&(c==b.c))
            {
                cout<<" \n\n Matrix Subtraction Possible \n\n";
                for(int i=0;i<r;i++)
                {
                    for(int j=0;j<c;j++)
                    {
                        temp.data[i][j]=data[i][j]-b.data[i][j];
                    }
                }
            }
            else cout<<"\n\n Matrix Subtraction Not Possible \n\n ";
            temp.r=r;
            temp.c=c;
            return temp;
}
Matrix Matrix :: mul(Matrix b)
{
    Matrix  temp;
    int     i,j,k;
            if(c==b.r)
            {
                cout<<"\n\n Matrix Multiplication Possible \n\n";
                for(i=0;i<r;i++)
                {
                    for(j=0;j<b.c;j++)
                    {
                        for(k=0;k<b.r;k++)
                        {
                            temp.data[i][j] +=
                                data[i][k]*b.data[k][j];
                        }
                    }
```

```
            }
        }
        else cout<<"\n\n Matrix Multiplication
                        Not Possible \n\n";
        temp.r=i;
        temp.c=j;
        return temp;

}
Matrix Matrix :: transpose()
{
    Matrix      temp;
                for (int i=0;i<c;++i)
                {
                    for (int j=0;j<r;++j)
                    {
                        temp.data[i][j]=data[j][i];
                    }
                }
                temp.r=c;
                temp.c=r;
                return temp;
}
Matrix :: Matrix()
{
    int     i;
        r=15;
        c=15;
        data= new int*[r];
        for(i=0;i<r;i++)
        {
            data[i]=new int[c];
        }
        for(i=0;i<r;i++)
        {
            for(int j=0;j<c;j++)
            {
                data[i][j]=0;
            }
        }
}
```

```
Matrix :: Matrix(int row,int column)
{

    int      i;
        data= new int*[r];
        for(i=0;i<row;i++)
        {
            data[i]=new int[column];
        }
        for(i=0;i<row;i++)
        {
            for(int j=0;j<column;j++)
            {
                data[i][j]=0;
            }
        }
}

Matrix :: ~Matrix()
{
    for(int i = 0 ; i < r ; i++)
    {
        delete [] data[i];
    }
    delete[] data;
}
```

matrixmain.cpp

```
#include"matrix.h"
int main()
{

    int     option,choice;
    Matrix  a;
        cout<<"\n\n\n Matrix Operations \n\n\n";
        do
        {
            cout<<" 1. Input And Display \n";
            cout<<" 2. Addition \n";
            cout<<" 3. Subtraction \n";
            cout<<" 4. Multiplication \n";
            cout<<" 5. Transpose \n";
            cout<<" 6. Exit \n";
            cout<<" Enter Choice : ";
            cin>>option;
            switch(option)
            {
                case 1 :
                {
                    cout<<"\n\n Input and Display \n\n";
                    a.input();
                    a.display();
                    break;
                }
                case 2 :
                {
                    Matrix b,c;
                    cout<<"\n\n Addition \n\n";
                    a.input();
                    a.display();
                    b.input();
                    b.display();
                    c=a.add(b);
                    c.display();
                    break;
                }
```

```
case 3 :
{
    Matrix b,c;
    cout<<"\n\n Subtraction \n\n";
    a.input();
    a.display();
    b.input();
    b.display();
    c=a.sub(b);
    c.display();
    break;
}
case 4 :
{
    Matrix b,c;
    cout<<"\n\n Multiplication \n\n";
    a.input();
    a.display();
    b.input();
    b.display();
    c=a.mul(b);
    c.display();
    break;
}


case 5 :
{
    Matrix b;
    cout<<"\n\n Matrix Transpose \n\n";
    a.input();
    a.display();
    b=a.transpose();
    b.display();
    break;
}
default:
{
    cout<<"\n\n Exiting The Program.....\n\n";
    break;
}
```

```
        }
        cout<<"\n\n Do You Want To Continue
                        ( Type 0 Or 1 ) : ";
        cin>>choice;
        cout<<endl<<endl;
    }while(choice);
    cout<<"\n\n The End \n\n";
    return 0;
}
```

```
(base) badri@badri-VM:~/Desktop$ g++ matrixmain.cpp matrix.cpp
(base) badri@badri-VM:~/Desktop$ ./a.out


 Matrix Operations


 1. Input And Display
 2. Addition
 3. Subtraction
 4. Multiplication
 5. Transpose
 6. Exit
 Enter Choice : 2


 Addition



 Entering The Matrix

 Enter Number Of Rows :  2
 Enter Number Of Columns : 2


 Enter Data :  1
 Enter Data :  2


 Enter Data :  3
 Enter Data :  4




 Displaying The Matrix

 1  2
 3  4
```

(a)

```
Entering The Matrix

Enter Number Of Rows :  2
Enter Number Of Columns : 2


Enter Data :  5
Enter Data :  6


Enter Data :  7
Enter Data :  8




Displaying The Matrix

5  6
7  8



Matrix Addition Possible


Displaying The Matrix

6  8
10  12



Do You Want To Continue ( Type 0 Or 1 ) : 0



The End
(base) badri@badri-VM:~/Desktop$
```

(b)

Figure 2: Output For II

Result A C++ program was created and implemented to perform matrix operations using dynamic memory access, constructors & destructors.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

EXP NO: 3
DATE : 21/1/20

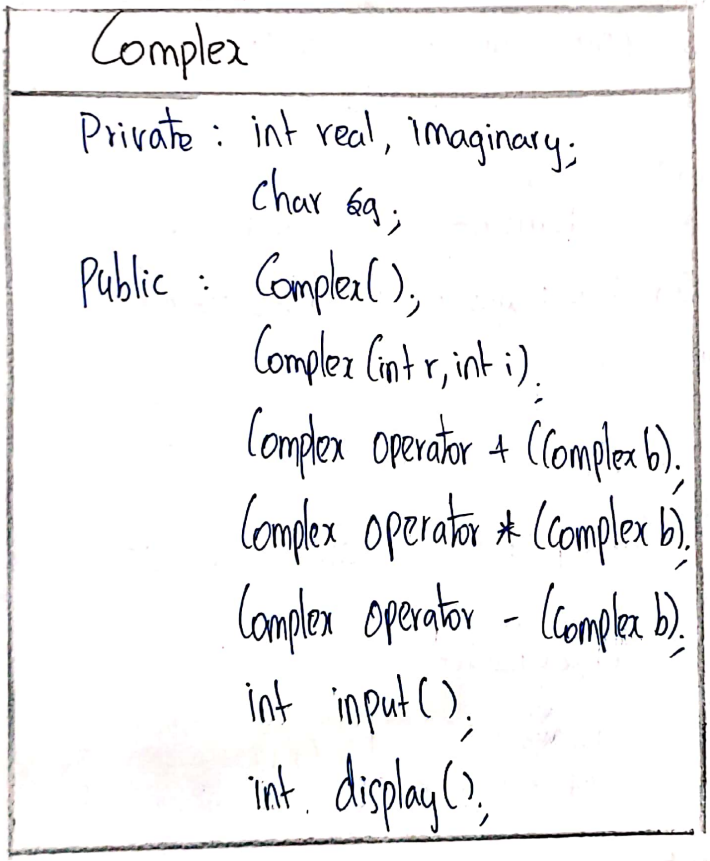
Polymorphism - Complex Numbers Application

Aim: To create & implement a C++ program to perform Complex number operations using Polymorphism

Time Complexities

SNO	Function	Operation	Time Complexity
1	int input()	Creates a Complex number	O(1)
2	int display()	Displays a Complex number	O(1)
3	Complex operator + (Complex b) Parameter : Complex Number Addition Operand 2 Return Datatype : Resultant Complex Number	Performs complex number addition between calling & the complex no passed as parameter	O(1)
4	Complex operator - (Complex b) Parameter : Complex Number Subtraction Operand 2 Return Datatype : Resultant Complex No	Performs complex number subtraction between calling & the complex no passed as parameter	O(1)

Class Diagram



SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

SNO	Function	Operation	Time Complexity
5	Complex operator * (Complex b) Parameter : Complex Number Multiplication Operand 2 Return Datatype : Resultant Complex Number	Performs Complex no Multiplication between calling complex no & Complex no passed as parameter	O(1)
6	Complex( )	Constructor Assigns data to 10 + 10 i	O(1)
7	Complex (int r, int i) Parameter : Value for real & imaginary parts of complex no	Constructor Assigns real to r & imaginary to i & sig accordingly	O(1)

complex.h

```
#ifndef complex_h
#define complex_h
#include<iostream>
#include<stdlib.h>
#include<math.h>

using namespace std;

class Complex
{
        int     real;
        int     imaginary;
        char    sg;
    public  :
        Complex();
        Complex(int r,int i);
        int input();
        int display();
        Complex operator + (Complex b);
        Complex operator - (Complex b);
        Complex operator * (Complex b);
};
#endif
```

complex.cpp

```
#include"complex.h"
Complex :: Complex()
{
    real = 10;
    imaginary = 10 ;
    if(imaginary>=0)sg='+';
    else sg='-';
}
Complex :: Complex(int r,int i)
{
    real = r;
    imaginary = i;
    if(imaginary>=0)sg='+';
    else sg='-';
}

int Complex :: input()
{
    cout<<"\n\n Input \n\n";
    cout<<" Enter Real Part : ";
    cin>>real;
    cout<<" Enter Imaginary Part : ";
    cin>>imaginary;
    if(imaginary>=0)sg='+';
    else sg='-';
    cout<<"\n\n";
    return 1;
}
int Complex :: display()
{
    cout<<"\n\n Display \n\n";
    cout<<" Complex Number : "<<real<<" "<<sg<<" "<<abs(imaginary)<
    return 1;

}
```

```
Complex Complex :: operator + (Complex b)
{
    Complex c;
    c.real = real + b.real ;
    c.imaginary = imaginary + b.imaginary ;
    if(c.imaginary>=0)c.sg='+';
    else c.sg='-';
    return c;
}
Complex Complex :: operator - (Complex b)
{
    Complex c;
    c.real = real - b.real ;
    c.imaginary = imaginary - b.imaginary ;
    if(c.imaginary>=0)c.sg='+';
    else c.sg='-';
    return c;
}
Complex Complex :: operator * (Complex b)
{
    Complex c;
    c.real = (real * b.real) - (imaginary * b.imaginary) ;
    c.imaginary = (real * b.imaginary) + (b.real * imaginary) ;
    if(c.imaginary>=0)c.sg='+';
    else c.sg='-';
    return c;
}
```

complexmain.cpp

```
#include"complex.h"
int main()
{
    Complex a;
    int     choice,option;
        cout<<"\n\n\n Complex Numbers With Operator Overloading
                                        \n\n\n";
        do
        {
            cout<<" 1. Input And Display \n";
            cout<<" 2. Addition \n";
```

```
cout<<" 3. Subtraction \n";
cout<<" 4. Multiplication \n";
cout<<" 5. Exit \n";
cout<<" Enter Choice : ";
cin>>option;
switch(option)
{
    case 1 :
    {
        cout<<"\n\n Input and Display \n\n";
        a.input();
        a.display();
        break;
    }
    case 2 :
    {
        Complex b,c;
        cout<<"\n\n Addition \n\n";
        a.input();
        a.display();
        b.input();
        b.display();
        c=a+b;
        c.display();
        break;
    }
    case 3 :
    {
        Complex b,c;
        cout<<"\n\n Subtraction \n\n";
        a.input();
        a.display();
        b.input();
        b.display();
        c=a-b;
        c.display();
        break;
    }
```

```
                case 4 :
                {
                    Complex b,c;
                    cout<<"\n\n Multiplication \n\n";
                    a.input();
                    a.display();
                    b.input();
                    c=a*b;
                    c.display();
                    break;
                }
                default:
                {
                    cout<<"\n\n Exiting The Program.....\n\n";
                    break;
                }
            }
            cout<<"\n\n Do You Want To Continue ( Type 0 Or 1 ) :
                                                    ";
            cin>>choice;
            cout<<endl<<endl;
        }while(choice);
        cout<<"\n\n The End \n\n";
        return 0;
}
```

```
(base) badri@badri-VM:~/Desktop$ g++ complexmain.cpp complex.cpp
(base) badri@badri-VM:~/Desktop$ ./a.out


Complex Numbers With Operator Overloading


1. Input And Display
2. Addition
3. Subtraction
4. Multiplication
5. Exit
Enter Choice : 2


Addition



Input

Enter Real Part : 3
Enter Imaginary Part : 4



Display

Complex Number : 3 + 4i


Input

Enter Real Part : 4
Enter Imaginary Part : -5



Display

Complex Number : 4 - 5i


Display

Complex Number : 7 - 1i


Do You Want To Continue ( Type 0 Or 1 ) : 1
```

(a)

```
1. Input And Display
2. Addition
3. Subtraction
4. Multiplication
5. Exit
Enter Choice : 3


Subtraction



Input

Enter Real Part : 2
Enter Imaginary Part : 2



Display

Complex Number : 2 + 2i


Input

Enter Real Part : 5
Enter Imaginary Part : 5



Display

Complex Number : 5 + 5i


Display

Complex Number : -3 - 3i


Do You Want To Continue ( Type 0 Or 1 ) : 0



The End
(base) badri@badri-VM:~/Desktop$
```

(b)

Figure 3: Output For III

Result A C++ program was created and implemented to perform complex number operations using polymorphism.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

EXP NO : 4
DATE : 16/11/20

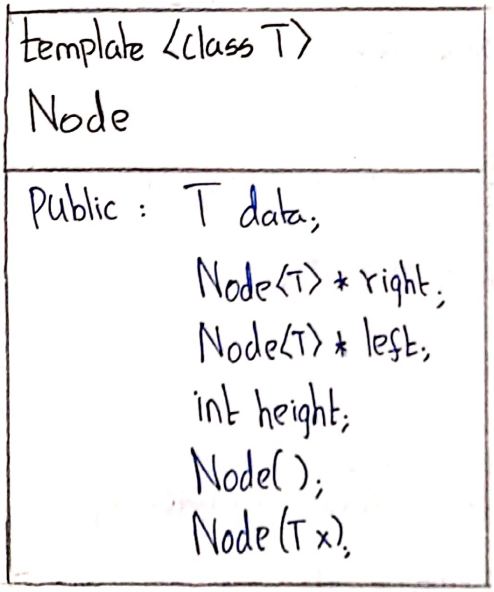
Template Class - BST Application

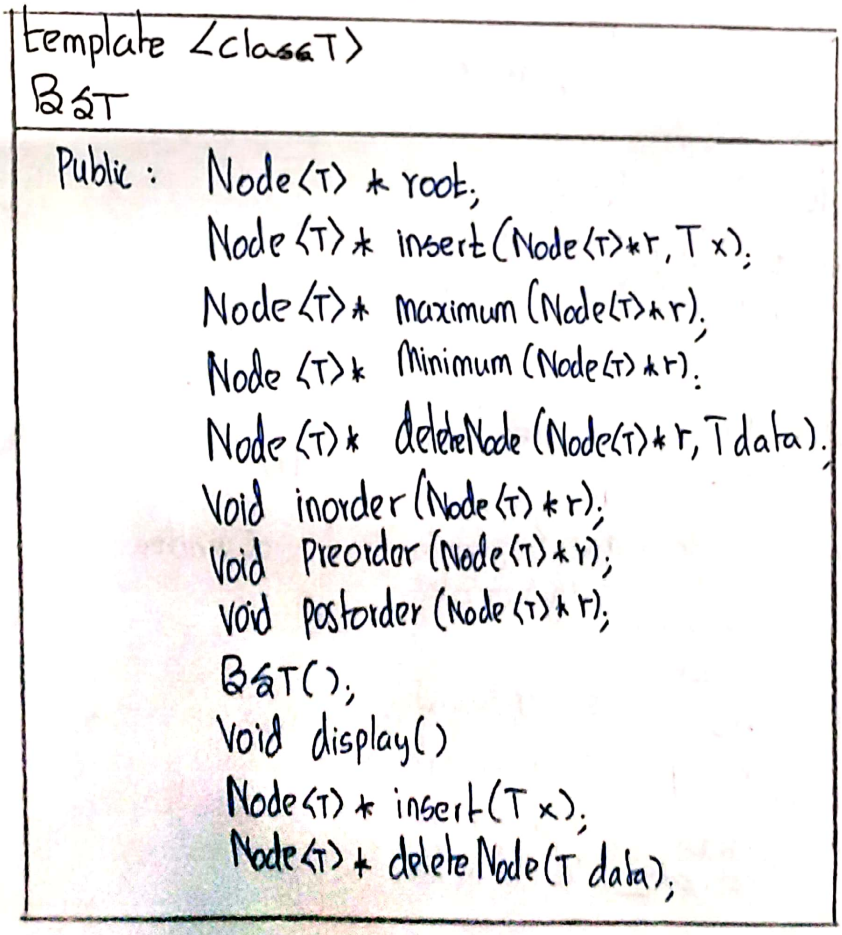
Aim: To create & implement a C++ program to perform operations using templates.

Time Complexity

SNO	Function	Operation	Time Complexity
1	Node()	Initializes a node's right, left to NULL, data, height to 0	O(1)
2	Node (T x) Parameter : Value of a node's data	Initializes a node's right, left to NULL, height to 0 & data to x	O(1)
3	Void inorder (Node<T> *r) Parameter: Node to be considered as root node	Inorder traversal starting from node given in parameter as root node	O(n) n: No of nodes in tree
4	Void preorder (Node<T> *r) Parameter: Node to be considered as root node	Preorder traversal starting from node given in parameter as root node	O(n)

Class Diagram





SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayan
185002018

SNO	Function	Operation	Time Complexity
5	Void postorder (Node<T> *r) Parameter: Node to be considered as root	Postorder traversal starting from node given in parameter as root node	O(n)
6	Node<T>* maximum (Node<T>*r) Parameter: Node to be considered as root Return datatype: Address of max node	Finds the address of the maximum node in the B&T starting from node given in Parameter as root node	O(n)
7	Node<T>* minimum (Node<T>*r) Parameter: Node to be considered as root Return datatype: Address of min node	Finds the address of the minimum node in the B&T starting from node given in Parameter as root node	O(n)
8	Node<T>* insert (Node<T>* r, T x) Parameter: Node to be considered as root Data to be inserted Return Datatype: Root of the new B&T	Returns the root of the B&T after inserting the data given in parameter & considering the node given in parameter as root node	O(h) h: Height of B&T

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
RS002018

SNO	Function	Operation	Time Complexity
9	Node(T)* deleteNode (Node(T)*r, T data) Parameter: Node to be considered as root, Data to be deleted Return Datatype: Root of new BST	Returns the address of the root of new BST after deleting data given in parameter & considering node given in parameter as root node	O(h)
10	BST( )	Constructor Initializes root to NULL	O(1)
11	Void display()	In order display starting from root node	O(n)
12	Node(T)* insert(T x) Parameter: Data to be inserted Return Datatype: Address of Root of new BST	Returns Address of root of new BST after inserting data given in the parameter & starting from root node	O(h)
13	Node(T)* deleteNode(T data) Parameter : Data to be deleted Return Datatype: Address of root of new BST	Returns root of new BST after deleting data given in parameter & starting from root node	O(h)

bst.h

```
#ifndef bst_h
#define bst_h
#include <iostream>
#include<string>

using namespace std;

template <class T>
class Node
{
    public:
    T data;
    Node<T>* right;
    Node<T>* left;
    int height;
    Node();
    Node(T x);
};

template <class T>
class BST
{
    public:
    Node<T>* root;
    Node<T>* insert(Node<T>* r, T x);
    void inorder(Node<T>* r);
    void preorder(Node<T>* r);
    void postorder(Node<T>*r);
    Node<T>* minimum(Node<T>* r);
    Node<T>* maximum(Node<T>* r);
    Node<T>* deleteNode(Node<T>* r, T data);
    BST();
    Node<T>* insert(T x);
    void display();
    Node<T>* deleteNode(T data);
    Node<T>* getRoot();
};
#endif
```

bst1.h

```
#include"bst.h"

template<class T>
Node<T> :: Node()
{
    right = NULL;
    left = NULL;
    height = 0;
}

template<class T>
Node<T> :: Node(T x)
{
    right = NULL;
    left = NULL;
    data = x;
    height = 0;
}

template<class T>
Node<T>* BST<T> :: insert(Node<T>* r, T x)
{
        if (r==NULL)
            r = new Node<T>(x);
        else
        {
            if (x>r->data)
                r->right = insert(r->right, x);
            else if (x<r->data)
                r->left = insert(r->left, x);
        }
        return r;
}
```

```
template<class T>
void BST<T> :: inorder(Node<T>* r)
{
        if (r==NULL)
            return;
        inorder(r->left);
        cout<<" "<<r->data<<"   ";
        inorder(r->right);
}
template<class T>
void BST<T> :: preorder(Node<T>* r)
{
        if (r==NULL)
            return;
        cout<<" "<<r->data<<"   ";
        preorder(r->left);
        preorder(r->right);
}
template<class T>
void BST<T> :: postorder(Node<T> *r)
{
    if(r == NULL)return;
    postorder(r->left);
    postorder(r->right);
    cout<<" "<<r->data<<"   ";
}

template<class T>
Node<T>* BST<T> :: minimum(Node<T>* r)
{
        Node<T>* temp = r;
        while (temp->left!=NULL)
            temp = temp->left;
        return temp;
}
```

```
template<class T>
Node<T>* BST<T> :: maximum(Node<T>* r)
{
        Node<T>* temp = r;
        while (temp->right!=NULL)
            temp = temp->right;
        return temp;
}
template<class T>
Node<T>* BST<T> :: deleteNode(Node<T>* r, T data)
{
        if (r == NULL) return r;
        if (data < r->data)
            r->left = deleteNode(r->left, data);
        else if (data > r->data)
            r->right = deleteNode(r->right, data);
        else
        {
            if (r->left == NULL)
            {
                Node<T>* temp = r->right;
                delete r;
                return temp;
            }
            else if (r->right == NULL)
            {
                Node<T>* temp = r->left;
                delete r;
                return temp;
            }
            Node<T>* temp = minimum(r->right);
            r->data = temp->data;
            r->right = deleteNode(r->right, temp->data);
        }
        return r;
}
```

```
template<class T>
BST<T> :: BST()
{
    root = NULL;
}
template<class T>
Node<T>* BST<T> :: insert(T x)
{
   root = insert(root, x);
   return root;
}
template<class T>
void BST<T> :: display()
{
    cout<<"\n\n InOrderDisplay \n\n";
    inorder(root);
}

template<class T>
Node<T>* BST<T> :: deleteNode(T data)
{
    return deleteNode(root, data);
}
Node<T>* BST<T> :: getRoot()
{
    return root;
}
```

display_tree1.h

```
#ifndef display_tree
#define display_tree

#include "bst1.h"


#include <iostream>
#include <vector>
#include <unordered_map>
#include <cmath>
#include <iomanip>
```

```
#include <algorithm>

#define NOVAL1 -2147483648
#define NOVAL2 2147483647

using namespace std;

int max_level(Node<int> *root, int level)
{
    if (root==NULL)return level-1;
    else
        return max(max_level(root->left, level+1),
                   max_level(root->right, level+1));
}

void preorder_bst(Node<int> *root, int level,
            unordered_map<int, vector<int>> &map,
                          int max_level)
{
    if (level>max_level)return;
    if (root == NULL)
    {
        map[level].push_back(NOVAL1);
        preorder_bst(NULL, level + 1, map, max_level);
        preorder_bst(NULL, level + 1, map, max_level);
    }
    else
    {
        map[level].push_back(root->data);
        preorder_bst(root->left, level + 1, map, max_level);
        preorder_bst(root->right, level + 1, map, max_level);
    }
}

vector<int> reorder(Node<int>* r)
{
    unordered_map<int, vector<int>> m;
    preorder_bst(r, 1, m, max_level(r, 1));
    vector<int> v;
    int start = max_level(r, 1);
    for (int i = start; i > 0; i--)
```

```
    {
        for (int j: m[i])
            v.push_back(j);
    }
    return v;

}

int maxdigits(vector<int> v)
{
    int max = 0;
    for (auto x:v)
    {
        if (x!=NOVAL1 && x!=NOVAL2)
        {
            if (abs(x)>max)
            {
                max = abs(x);
            }
        }
    }
    return floor(log10(max))+1;
}

void display_bst(Node<int>* r)
{
    cout<<"\n\n Displaying The Tree  \n\n";
    vector<int> v = reorder(r);
    int no_rows = ceil(log2(v.size()));
    int no_columns = 2*pow(2, no_rows-1)-1;
    int width = maxdigits(v);
    int** matrix = new int*[no_rows];
    for (int i=0; i<no_rows; i++)
        matrix[i] = new int[no_columns];
    for (int i=0; i<no_rows; i++)
        for (int j=0; j<no_columns; j++)
            matrix[i][j] = NOVAL2;
    vector <int> lastpos;
    vector<int> v2;
    int a, b, c;
    int k=0;
```

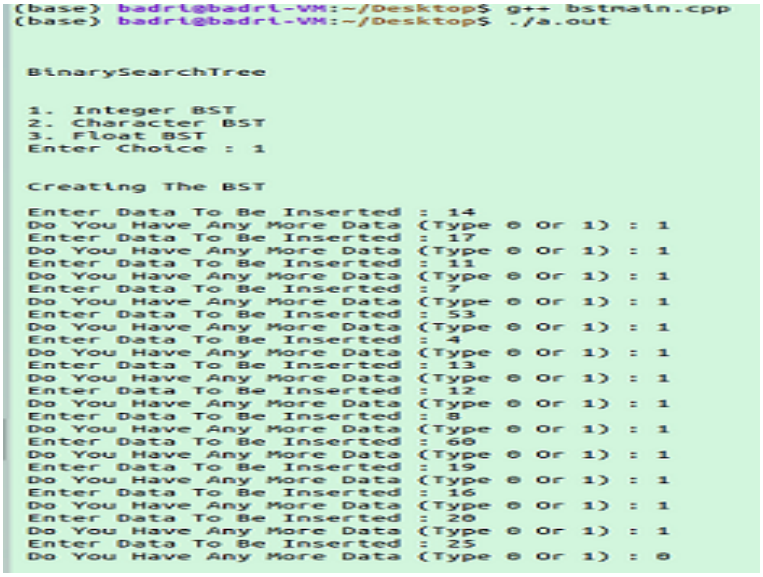
```
        int x=-1, y=1;
        while (x<no_columns)
        {
            lastpos.push_back(x);
            lastpos.push_back(y);
            x+=2;
            y+=2;
        }
        for (int i=no_rows-1; i>=0; i--)
        {
            a = 0;
            b = 1;
            while (b<lastpos.size())
            {
                c = (lastpos.at(a)+lastpos.at(b))/2;
                matrix[i][c] = v[k++];
                v2.push_back(c);
                a+=2;
                b+=2;
            }
            lastpos = v2;
            v2.clear();
        }
        for (int i=0; i<no_rows; i++)
        {
            for (int j=0; j<no_columns; j++)
            {
                if (matrix[i][j]==NOVAL1)
                    cout<<setfill(' ')<<setw(width)<<"x";
                else if(matrix[i][j]!=NOVAL2)
                    cout<<setfill(' ')<<setw(width)<<matrix[i][j];
                else
                    cout<<setfill(' ')<<setw(width)<<" ";
                cout<<" ";

            }
            cout<<"\n\n";
        }
    }
    #endif
```

bstmain.cpp

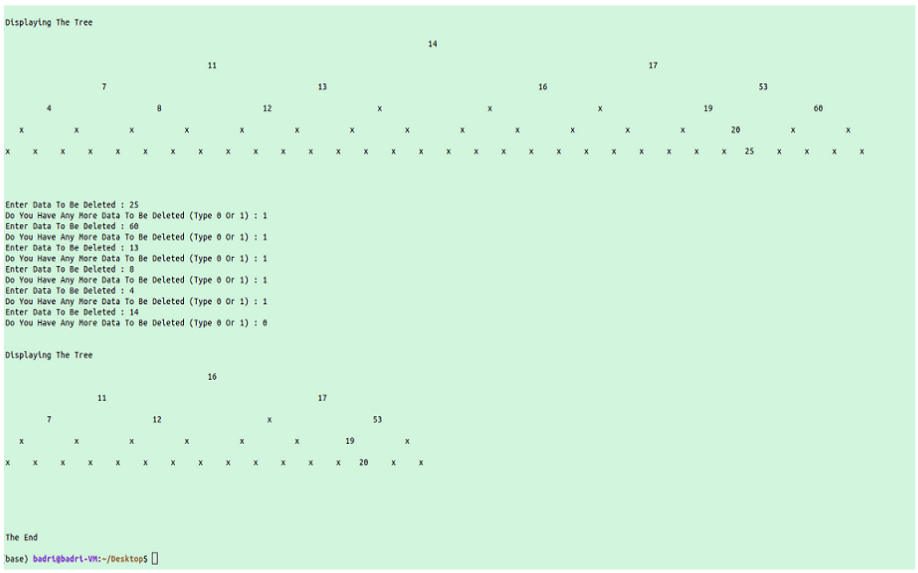
```
#include"display_tree1.h"

int main()
{
        cout<<"\n\n\n BinarySearchTree \n\n\n";
    int     choice;
        cout<<" 1. Integer BST \n";
        cout<<" 2. Character BST \n";
        cout<<" 3. Float BST \n";
        cout<<" Enter Choice : ";
        cin>>choice;
        switch(choice)
        {
            case 1 :
            {
                cout<<"\n\n Creating The BST \n\n";
                int data;
                BST<int> b;
                do
                {
                    cout<<" Enter Data To Be Inserted : ";
                    cin>>data;
                    b.insert(data);
                    cout<<" Do You Have Any More Data
                              (Type 0 Or 1) : ";
                    cin>>choice;
                }while(choice);
                display_bst(b.getRoot());
                cout<<"\n\n\n";
                do
                {
                    cout<<" Enter Data To Be Deleted : ";
                    cin>>data;
                    b.deleteNode(data);
                    cout<<" Do You Have Any More Data To Be
                            Deleted (Type 0 Or 1) : ";
                    cin>>choice;
                }while(choice);
```

```
            display_bst(b.getRoot());
            cout<<"\n\n\n";
            break;
        }
        case 2 :
        {
            cout<<"\n\n Creating The BST \n\n";
            char data;
            BST<char> b;
            do
            {
                cout<<" Enter Data To Be Inserted : ";
                cin>>data;
                b.insert(data);
                cout<<" Do You Have Any More Data
                        (Type 0 Or 1) : ";
                cin>>choice;
            }while(choice);
            b.display();
            cout<<"\n\n\n";
            do
            {
                cout<<" Enter Data To Be Deleted : ";
                cin>>data;
                b.deleteNode(data);
                cout<<" Do You Have Any More Data To Be
                    Deleted (Type 0 Or 1) : ";
                cin>>choice;
            }while(choice);
            b.display();
            cout<<"\n\n\n";
            break;
        }
        case 3 :
        {
            cout<<"\n\n Creating The BST \n\n";
            float data;
            BST<float> b;
            do
            {
                cout<<" Enter Data To Be Inserted : "
```

```
                    cin>>data;
                    b.insert(data);
                    cout<<" Do You Have Any More Data
                        (Type 0 Or 1) : ";
                    cin>>choice;
                }while(choice);
                b.display();
                cout<<"\n\n\n";
                do
                {
                    cout<<" Enter Data To Be Deleted : ";
                    cin>>data;
                    b.deleteNode(data);
                    cout<<" Do You Have Any More Data To Be
                        Deleted (Type 0 Or 1) : ";
                    cin>>choice;
                }while(choice);
                b.display();
                cout<<"\n\n\n";
                break;
            }
            default :
            {
                cout<<"\n\n Default \n\n";
                break;
            }
        }
        cout<<"\n\n The End \n\n";
        return 0;
}
```



(a)



(b)

Figure 4: Output For IV

Result A C++ program was created and implemented to perform operations on a BST using templates.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

EXP NO: 5
DATE: 23/11/20

Virtual Inheritance - 3D Objects Mensuration Application

Aim: To create & implement a C++ program to find the Volume & Surface areas of 3D objects using Virtual Inheritance

Time Complexity:

SNO	Function	Operation	Time Complexity
1	Circle() Cylinder() Cone()	Constructor Empty Constructor	O(1)
2	int getcircle() Belongs to Circle Class	Gets the radius of a circle	O(1)
3	float area() Belongs to Circle Class Return datatype: Returns the area of a circle	Calculates area of a circle	O(1)

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M. Badri Narayanan
RS2002018

SNO	Function	Operation	Time Complexity
4	float circumference() Belongs to Circle Class Returns: Circumference of a Circle	Calculates Circumference of a Circle	O(1)
5	int display() Belongs to Circle Class	Displays radius of the Circle	O(1)
6	Virtual float volume()=0 Virtual float tsa()=0 Virtual float csa()=0 Belongs to Circle Class	Virtual functions which are inherited & used to find tsa, csa, volume of cone & cylinder	
7	int getheight() Belongs to Cylinder Class	Gets the height of Cone/Cylinder	O(1)
8	float volume() float tsa() float csa() Belongs to Cylinder Class	Returns Volume of Cylinder Returns TSA of Cylinder Returns CSA of Cylinder	O(1) O(1) O(1)

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

SNO	Function	Operation	Time Complexity
9	int cylinderdisplay() Belongs to Cylinder Class	Display Height & Radius of Cylinder	O(1)
10	int getlength() Belongs to Cone Class	Get Height & Radius of Cone & calculate length of Cone	O(1)
11	int conedisplay() Belongs to Cone Class	Displays Length, Height, Radius of Cone Class	O(1)
12	float volume() float tsa() float csa() Belongs to Cone Class	Returns Volume of Cone Returns T&A of Cone Returns C&A of Cone	O(1) O(1) O(1)

virtualinheritance.h

```
#ifndef virtualinheritance_h
#define virtualinheritance_h

#include<iostream>
#include<string>
#include<cmath>

using namespace std;

class Circle
{
    public :
    float   radius;
    int     getcircle();
    float   area();
    float   circumference();
    int     display();
    virtual float volume() = 0;
    virtual float tsa() = 0;
    virtual float csa() = 0;
    Circle();
};

class Cylinder : public Circle
{
    public :
    float height;
    int   getheight();
    float volume();
    float tsa();
    float csa();
    int cylinderdisplay();
    Cylinder();
};
```

```
class Cone : public Cylinder
{
    public :
    float length;
    float volume();
    float tsa();
    float csa();
    int   getlength();
    int   conedisplay();
    Cone();
};
#endif
```

virtualinheritance.cpp

```
#include"virtualinheritance.h"

Circle :: Circle()
{

}

Cylinder :: Cylinder()
{

}

Cone :: Cone()
{

}

int Circle :: getcircle()
{
    cout<<"\n\n Enter Radius : ";
    cin>>radius;
    cout<<"\n\n";
    if(radius>=0)return 1;
    else return 0;
}
```

```
float Circle :: area()
{
    return 3.14 * radius * radius ;
}

float Circle :: circumference()
{
    return 2 * 3.14 * radius;
}
int Circle :: display()
{
    cout<<" Radius : "<<radius<<"\n\n";
    return 1;
}
int Cylinder :: getheight()
{
    Circle :: getcircle();
    cout<<"\n\n Enter Height : ";
    cin>>height;
    cout<<"\n\n";
    if(height>=0)return 1;
    else return 0;
}
float Cylinder :: tsa()
{
    return 2 * 3.14 * radius * (height + radius);
}

float Cylinder :: csa()
{
    return 2 * 3.14 * radius * height;
}

float Cylinder :: volume()
{
    return 3.14 * radius * radius * height ;
}
```

```
int Cylinder :: cylinderdisplay()
{
    Circle :: display();
    cout<<" Height : "<<height<<"\n\n";
    return 1;
}
int Cone :: getlength()
{
    float sum;
    Cylinder :: getheight();
    sum = (radius * radius ) + (height * height );
    length = sqrt(sum);
}

int Cone :: conedisplay()
{
    Cylinder :: cylinderdisplay();
    cout<<" Length : "<<length<<"\n\n";
    return 1;
}
float Cone :: tsa()
{
    return 3.14 * radius * (length + radius);

}

float Cone :: csa()
{
    return 3.14 * radius * length;
}
float Cone :: volume()
{
    return (1/3.0) * 3.14 * radius * radius * height;
}
```

virtualinhertitancemain.cpp

```
#include"virtualinheritance.h"

int main()
{

    int choice;
    do
    {
            cout<<"\n\n\n Virtual Inheritance \n\n\n";
            cout<<" 1. Circle \n";
            cout<<" 2. Cylinder \n";
            cout<<" 3. Cone \n";
            cout<<" Enter Choice : ";
            cin>>choice;
            switch(choice)
            {
                case 1 :
                {
                    Cylinder c;
                    c.getcircle();
                    c.display();
                    cout<<" Area Is : "<<c.area()<<"\n\n";
                    cout<<" Circumference Is
                        : "<<c.circumference()<<"\n\n";
                    break;
                }
                case 2 :
                {
                    Cylinder c;
                    c.getheight();
                    c.cylinderdisplay();
                    cout<<" TSA Is : "<<c.tsa()<<"\n\n";
                    cout<<" CSA Is : "<<c.csa()<<"\n\n";
                    cout<<" Volume Is : "<<c.volume()<<"\n\n";
                    break;
                }
```

```
                case 3 :
                {
                    Cone c;
                    c.getlength();
                    c.conedisplay();
                    cout<<" TSA Is : "<<c.tsa()<<"\n\n";
                    cout<<" CSA Is : "<<c.csa()<<"\n\n";
                    cout<<" Volume Is : "<<c.volume()<<"\n\n";
                    break;
                }
                default :
                {
                    cout<<"\n\n Default \n\n";
                    break;
                }
            }
            cout<<" Do You Want To Continue (Type 0 Or 1) : ";
            cin>>choice;
    }while(choice);
    cout<<"\n\n\n The End \n\n\n";
    return 0;
}
```

```
(base) badri@badri-VM:~/Desktop$ g++ virtualInheritancemain.cpp virtualInheritance.cpp
(base) badri@badri-VM:~/Desktop$ ./a.out


Virtual Inheritance


1. Circle
2. Cylinder
3. Cone
Enter Choice : 1


Enter Radius : 3


Radius : 3

Area Is : 28.26

Circumference Is : 18.84

Do You Want To Continue (Type 0 Or 1) : 1


Virtual Inheritance


1. Circle
2. Cylinder
3. Cone
Enter Choice : 3


Enter Radius : 3



Enter Height : 6


Radius : 3

Height : 6

Length : 6.7082

TSA Is : 91.4513

CSA Is : 63.1913

Volume Is : 56.52
```

(a)

```
Do You Want To Continue (Type 0 Or 1) : 1


Virtual Inheritance


1. Circle
2. Cylinder
3. Cone
Enter Choice : 2


Enter Radius : 5



Enter Height : 4


Radius : 5

Height : 4

TSA Is : 282.6

CSA Is : 125.6

Volume Is : 314

Do You Want To Continue (Type 0 Or 1) : 0


The End


(base) badri@badri-VM:~/Desktop$
```

(b)

Figure 5: Output For V

Result A C++ program was created and implemented to find the volume, tsa, csa of 3D Objects using virtual inheritance.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.

M Badri Narayanan
185002018

EXP NO: 6
DATE: 11

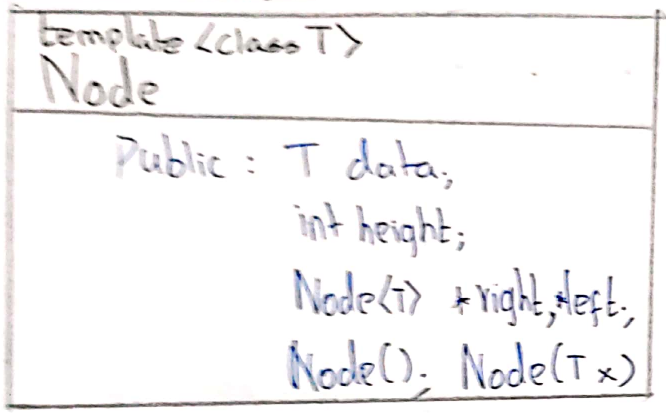
AVL Tree ADT - Implementation

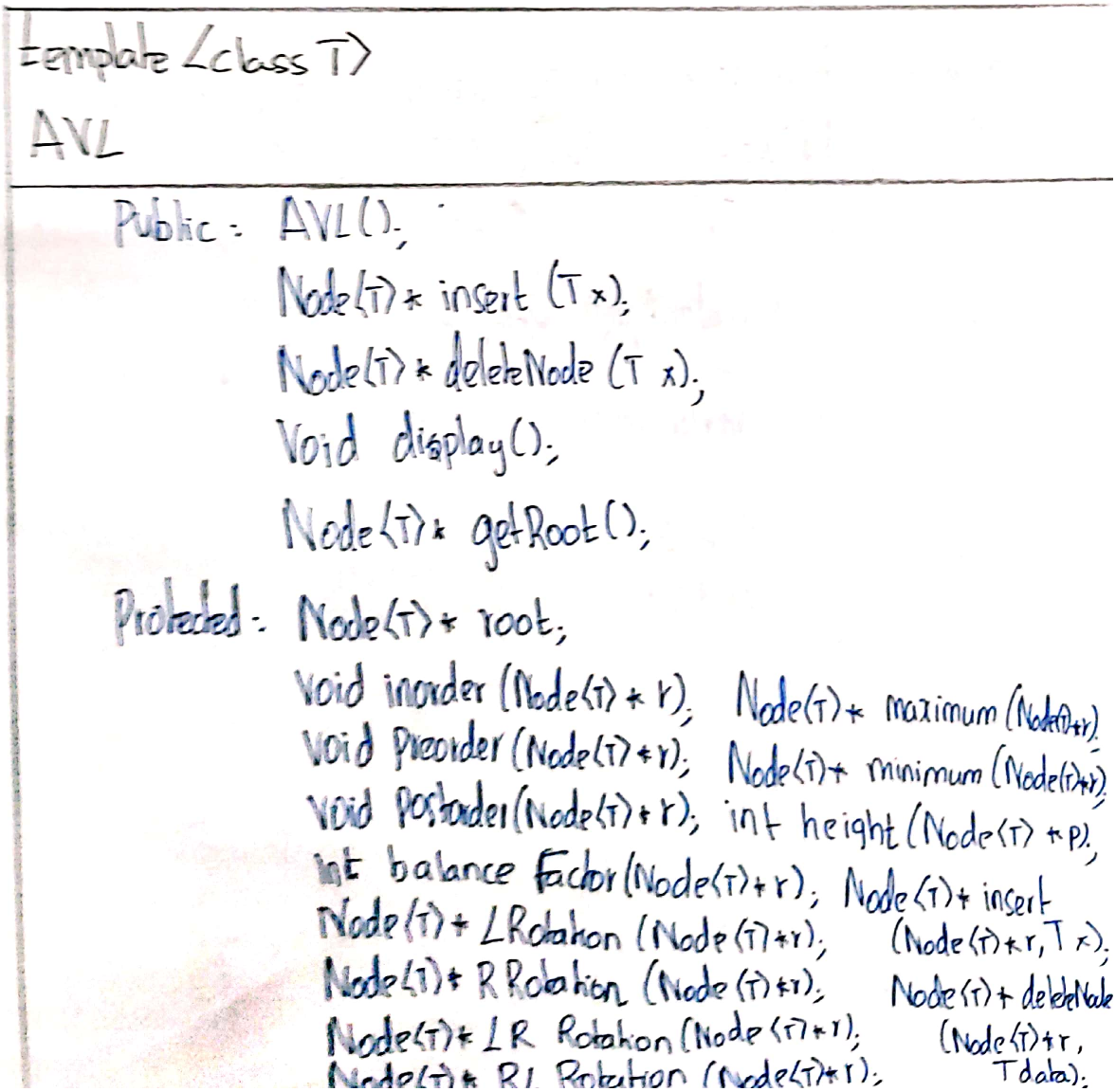
Aim: To implement AVL Tree ADT using a C++ Program.

Time Complexity

SNO	Function	Operation	Time Complexity
1	Node()	Refer BST	O(1)
2	Node(T x)	Refer BST	O(1)
3	Void inorder(Node<T> *r)	Refer BST	O(n) n: No of nodes
4	Void preorder(Node<T>*r)	Refer BST	O(n)
5	Void postorder (Node<T>*r)	Refer BST	O(n)
6	Void display()	Refer BST	O(n)
7	Node<T>* maximum(Node<T>*r)	Refer BST	O(n)
8	Node<T> * minimum (Node<T>*r)	Refer BST	O(n)

Class Diagram





SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.

M Badri Narayanan
185002018

SNO	Function	Operation	Time Complexity
9	AVL()	Initializes root to NULL	O(1)
10	int height (Node<T>*p) Parameter: Root of Tree whose height of larger subtree is to be found Return Datatype: Height of larger subtree	Returns the height of larger subtree of the node given in Parameter	O(1)
11	int balanceFactor (Node<T>*r) Parameter: Root of Tree whose balance factor is to be found Return Datatype: Balancefactor of the node given in parameter	Returns the balance factor of a subtree whose root is given in the parameter	O(1)
12	Node<T>*LRotation (Node<T>*r) Parameter: Root node on which Left Rotation is to be performed Returndatatype: Rotated Node	Performs Left Rotation to node specified in Parameter, its child & its parent	O(1)

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

SNO	Function	Operation	Time Complexity
13	Node<T> * RRotation (Node<T>*r) Parameter: Node on which rotation is to be performed. Return Datatype: Rotated Node	Perform Right Rotation on Node specified in parameter, its child & parent	O(1)
14	Node<T> * LRRotation (Node<T>*r) Parameter: Node on which rotation is to be performed Return Datatype: Rotated Node	Perform Left Right Rotation on node specified in Parameter, its child & parent	O(1)
15	Node<T>* RLRotation (Node<T>*r) Parameter: Node on which rotation is to be performed. Return Datatype: Rotated Node	Performs Right Left Rotation on node specified in Parameter, its child & parent	O(1)

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
1E5002018

SNO	Function	Operation	Time Complexity
16	Node<T>* insert(Node<T>*r, T x) Parameter: Data to be inserted, Node to be considered as root Return Datatype: Root of new AVL Tree	Inserts data given in parameter considering Node given in parameter. Then performs balancing operations	O(h) h: Height of Tree
17	Node<T>* insert(T x) Parameter: Data to be inserted Return Datatype: Root of new AVL Tree	Inserts data given in parameter, Starting from root node & then performing balancing operation	O(h)
18	Node<T>* deleteNode (Node<T>*r, T data) Parameter: Data to be deleted, Node to be considered as root node Return Datatype: Root of new AVL Tree	Deletes data given in Parameter, considering node given in parameter as root, then performing balancing operations	O(h)
19	Node<T>* deleteNode (T data) Parameter: Data to be deleted Return Datatype: Root of new AVL Tree	Deletes data given in parameter, starting from root node & then performs balancing operations	O(h)

avl.h

```
#ifndef avl_h
#define avl_h
#include <iostream>
#include<string>
using namespace std;

template <class T>
class Node
{
    public:
            T data;
            Node<T>* right;
            Node<T>* left;
            int height;
            Node();
            Node(T x);
};

template <class T>
class AVL
{
    protected:
            Node<T>* root;
            void inorder(Node<T>* r);
            void preorder(Node<T>* r);
            void postorder(Node<T>* r);
            Node<T>* minimum(Node<T>* r);
            Node<T>* maximum(Node<T>* r);
            int height(Node<T>* p);
            int balanceFactor(Node<T>* r);
            Node<T>* LRotation(Node<T>* r);
            Node<T>* RRotation(Node<T>* r);
            Node<T>* LRRotation(Node<T>* r);
            Node<T>* RLRotation(Node<T>* r);
            Node<T>* insert(Node<T>* r, T x);
            Node<T>* deleteNode(Node<T>* r, T data);
```

```
    public:
            AVL();
            Node<T>* insert(T x);
            Node<T>* deleteNode(T x);
            void display();
            Node<T>* getRoot();

};
#endif
```

avl1.h

```
#include"avl.h"

template<class T>
Node<T> :: Node()
{
        right = NULL;
         left = NULL;
         height = 0;
}
template<class T>
Node<T> :: Node(T x)
{
    right = NULL;
    left = NULL;
    data = x;
    height = 0;
}
template<class T>
AVL<T> :: AVL()
{
    root = NULL;
}
```

```
template<class T>
void AVL<T> :: inorder(Node<T>* r)
{
    if (r==NULL)return;
    inorder(r->left);
    cout<<" "<<r->data<<" ";
    inorder(r->right);
}
template<class T>
void AVL<T> :: preorder(Node<T>* r)
{
        if (r==NULL)return;
        cout<<" "<<r->data<<"  ";
        preorder(r->left);
        preorder(r->right);
}
template<class T>
void AVL<T> :: postorder(Node<T> *r)
{
    if(r == NULL)return;
    postorder(r->left);
    postorder(r->right);
    cout<<" "<<r->data<<"  ";
}
template<class T>
Node<T>* AVL<T> :: minimum(Node<T>* r)
{
        Node<T>* temp = r;
        while (temp->left!=NULL)
            temp = temp->left;
        return temp;
}
template<class T>
Node<T>* AVL<T> :: maximum(Node<T>* r)
{
        Node<T>* temp = r;
        while (temp->right!=NULL)
            temp = temp->right;
        return temp;
}
```

```
template<class T>
int AVL<T> :: height(Node<T>* p)
{
    int hl, hr;
    hl = (p && p->left) ? (p->left)->height : 0;
    hr = (p && p->right) ? (p->right)->height : 0;
    return hl > hr ? hl + 1 : hr + 1;
}
template<class T>
int AVL<T> :: balanceFactor(Node<T>* r)
{
    return height(r->left)-height(r->right);
}
template<class T>
Node<T>* AVL<T> :: LRotation(Node<T>* r)
{
    Node<T>* rc = r->right;
    r->right = rc->left;
    rc->left = r;
    r->height = height(r);
    rc->height = height(rc);
    return rc;
}
template<class T>
Node<T>* AVL<T> :: RRotation(Node<T>* r)
{
    Node<T>* lc = r->left;
    r->left = lc->right;
    lc->right = r;
    r->height = height(r);
    lc->height = height(lc);
    return lc;
}
template<class T>
Node<T>* AVL<T> :: LRRotation(Node<T>* r)
{
    r->left = LRotation(r->left);
    r = RRotation(r);
    return r;
}
```

```
template<class T>
Node<T>* AVL<T> :: RLRotation(Node<T>* r)
{
    r->right = RRotation(r->right);
    r = LRotation(r);
    return r;
}
template<class T>
Node<T>* AVL<T> :: insert(Node<T>* r, T x)
{
    if (r==NULL)r = new Node<T>(x);
    else
    {
        if (x>r->data)r->right = insert(r->right, x);
        else if (x<r->data)r->left = insert(r->left, x);
    }
    r->height = height(r);
    if (balanceFactor(r)==2 && balanceFactor(r->left)==1)
                    r = RRotation(r);

    if (balanceFactor(r)==2 && balanceFactor(r->left)==-1)
                    r = LRRotation(r);

    if (balanceFactor(r)==-2 && balanceFactor(r->right)==1)
                    r = RLRotation(r);

    if (balanceFactor(r)==-2 && balanceFactor(r->right)==-1)
                    r = LRotation(r);
    return r;
}
template<class T>
Node<T>* AVL<T> :: deleteNode(Node<T>* r, T data)
{
    if (r == NULL) return r;
    if (data < r->data)
    {
        r->left = deleteNode(r->left, data);
    }
```

```
        else if (data > r->data)
        {
            r->right = deleteNode(r->right, data);
        }
        else
        {
            if (r->left == NULL)
            {
                Node<T>* temp = r->right;
                delete r;
                return temp;
            }
            else if (r->right == NULL)
            {
                Node<T>* temp = r->left;
                delete r;
                return temp;
            }
            Node<T>* temp = minimum(r->right);
            r->data = temp->data;
            r->right = deleteNode(r->right, temp->data);
        }
        r->height = height(r);
        if (balanceFactor(r)==2 && ((balanceFactor(r->left)==1) ||
                    (balanceFactor(r->left)==0)))

                        r = RRotation(r);

    if (balanceFactor(r)==2 && balanceFactor(r->left)==-1)
                        r = LRRotation(r);

    if (balanceFactor(r)==-2 && ((balanceFactor(r->right)==1) ||
                        (balanceFactor(r>right)==0)))

                        r = RLRotation(r);

        if (balanceFactor(r)==-2 && balanceFactor(r->right)==-1)
                        r = LRotation(r);
        return r;
    }
```

```
template<class T>
Node<T>* AVL<T> :: insert(T x)
{
    root = insert(root, x);
    return root;
}
template<class T>
Node<T>* AVL<T> :: deleteNode(T x)
{
    root = deleteNode(root, x);
    return root;
}
template<class T>
void AVL<T> :: display()
{
    cout<<"\n\n InOrderDisplay \n\n";
    inorder(root);
}
template<class T>
Node<T>* AVL<T> :: getRoot()
{
    return root;
}
```

display_tree.h

```
#ifndef display_tree
#define display_tree

#include "avl1.h"

#include <iostream>
#include <vector>
#include <unordered_map>
#include <cmath>
#include <iomanip>
#include <algorithm>

#define NOVAL1 -2147483648
#define NOVAL2 2147483647
```

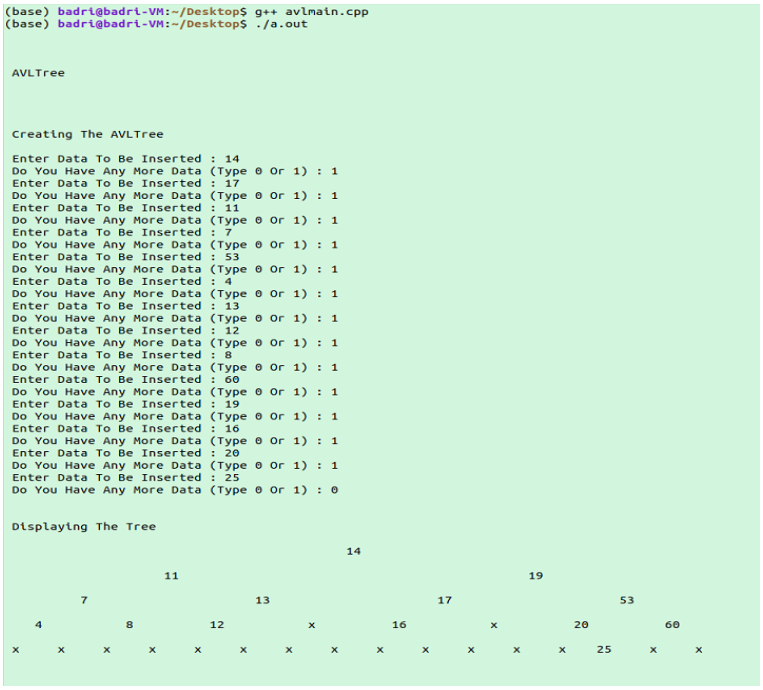
```
using namespace std;

int max_level(Node<int> *root, int level)
{
    if (root==NULL)return level-1;
    else
        return max(max_level(root->left, level+1),
                   max_level(root->right, level+1));
}

void preorder_bst(Node<int> *root, int level,
            unordered_map<int, vector<int>> &map,
                          int max_level)
{
    if (level>max_level)return;
    if (root == NULL)
    {
        map[level].push_back(NOVAL1);
        preorder_bst(NULL, level + 1, map, max_level);
        preorder_bst(NULL, level + 1, map, max_level);
    }
    else
    {
        map[level].push_back(root->data);
        preorder_bst(root->left, level + 1, map, max_level);
        preorder_bst(root->right, level + 1, map, max_level);
    }
}
vector<int> reorder(Node<int>* r)
{
    unordered_map<int, vector<int>> m;
    preorder_bst(r, 1, m, max_level(r, 1));
    vector<int> v;
    int start = max_level(r, 1);
    for (int i = start; i > 0; i--)
    {
        for (int j: m[i])
           v.push_back(j);
    }
    return v;
}
```

```
int maxdigits(vector<int> v)
{
    int max = 0;
    for (auto x:v)
    {
        if (x!=NOVAL1 && x!=NOVAL2)
        {
            if (abs(x)>max)
            {
                max = abs(x);
            }
        }
    }
    return floor(log10(max))+1;
}

void display_bst(Node<int>* r)
{
    cout<<"\n\n Displaying The Tree  \n\n";
    vector<int> v = reorder(r);
    int no_rows = ceil(log2(v.size()));
    int no_columns = 2*pow(2, no_rows-1)-1;
    int width = maxdigits(v);
    int** matrix = new int*[no_rows];
    for (int i=0; i<no_rows; i++)
        matrix[i] = new int[no_columns];
    for (int i=0; i<no_rows; i++)
        for (int j=0; j<no_columns; j++)
            matrix[i][j] = NOVAL2;
    vector <int> lastpos;
    vector<int> v2;
    int a, b, c;
    int k=0;
    int x=-1, y=1;
    while (x<no_columns)
    {
        lastpos.push_back(x);
        lastpos.push_back(y);
        x+=2;
        y+=2;
    }
```

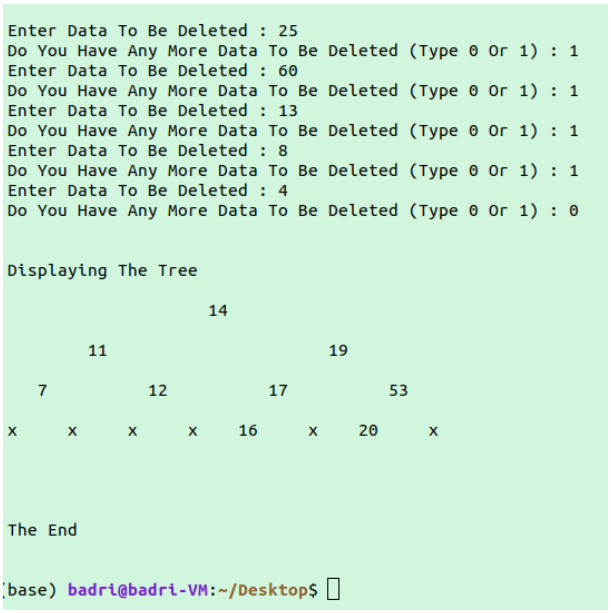
```
        for (int i=no_rows-1; i>=0; i--)
        {
            a = 0;
            b = 1;
            while (b<lastpos.size())
            {
                c = (lastpos.at(a)+lastpos.at(b))/2;
                matrix[i][c] = v[k++];
                v2.push_back(c);
                a+=2;
                b+=2;
            }
            lastpos = v2;
            v2.clear();
        }
        for (int i=0; i<no_rows; i++)
        {
            for (int j=0; j<no_columns; j++)
            {
                if (matrix[i][j]==NOVAL1)
                    cout<<setfill(' ')<<setw(width)<<"x";
                else if(matrix[i][j]!=NOVAL2)
                    cout<<setfill(' ')<<setw(width)<<matrix[i][j];
                else
                    cout<<setfill(' ')<<setw(width)<<" ";
                cout<<" ";

            }
            cout<<"\n\n";
        }
    }
    #endif
```

avlmain.cpp

```
#include"display_tree.h"
int main()
{
    cout<<"\n\n\n AVLTree \n\n\n";
    int data;
    int choice;
    AVL<int> b;
    cout<<"\n\n Creating The AVLTree \n\n";
    do
    {
        cout<<" Enter Data To Be Inserted : ";
        cin>>data;
        b.insert(data);
        cout<<" Do You Have Any More Data (Type 0 Or 1) : ";
        cin>>choice;
    }while(choice);
    display_bst(b.getRoot());
    cout<<"\n\n\n";

    do
    {
        cout<<" Enter Data To Be Deleted : ";
        cin>>data;
        b.deleteNode(data);
        cout<<" Do You Have Any More Data To Be Deleted (Type 0 Or
        cin>>choice;
    }while(choice);


    display_bst(b.getRoot());
    cout<<"\n\n\n The End \n\n\n";
    return 0;
}
```



(a)



(b)

Figure 6: Output For VI

Result A C++ program was created to implement AVL Tree ADT.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

EXP NO: 7

DATE: 11

M Badri Narayanan
185002018

Splay Tree ADT Implementation

Aim: To create a C++ program to implement Splay Tree ADT

Time Complexity

SNO	Function	Operation	Time Complexity
1	Node( ): Constructor	Assigns data to 0, right & left to NULL	O(1)
2	Node(T x) Parameter: Value of data This is a Constructor	Assigns data to x, right & left to NULL	O(1)
3	Node<T> * maximum(Node<T> *r)	Refer BST	O(n) n: No of Nodes
4	Node<T> * minimum(Node<T> *r)	Refer BST	O(n)
5	Node <T> * RightRotate(Node<T>*r)	Refer AVL	O(1)
6	Node<T>* LeftRotate (Node<T> *r)	Refer AVL	O(1)

Class Diagram

Template <class T>
Node
Public: T data; Node(T x); Node<T>* right; Node<T>* left; Node();

Template <class T>
Splay
Private: Node<T>* root; Node<T>* maximum (Node<T>* r); void inorder (Node<T>*); Node<T>* minimum (Node<T>* r); void preorder (Node<T>* r); void postorder (Node<T>* r); Node<T>* RightRotate (Node<T>* r); Node<T>* LeftRotate (Node<T>* r); Node<T>* splay (Node<T>* r, T key); Node<T>* insert (Node<T>* r, T k); Node<T>* delete_key (Node<T>* r, T key); Node<T>* search (Node<T>* r, T k);
Public: Splay(); Node<T>* getRoot(); int display(); Node<T>* delete_key (T key); Node<T>* insert (T x); Node<T>* search (T key);

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
18500 2018

SNO	Function	Operation	Time Complexity
7	Void inorder (Node<T> * r)	Refer BST	O(n)
8	Void preorder (Node<T> * r)	Refer BST	O(n)
9	Void postorder (Node<T> * r)	Refer BST	O(n)
10	Node<T>* Splay (Node<T> * r, T key) Parameter: Data to be splayed, Node to be considered as root Return Datatype: Root of Splayed Tree	Performs splaying operation considering data given in parameter as data to be splayed & node given in parameter as root node	O(log n)
11	Node<T>* insert (Node<T> * r, T k) Parameter: Data to be inserted, Node to be considered as root node Return Datatype: Root of Splayed Tree	Inserts data given in parameter, considering node given in parameter & performs splaying operation	O(log n)

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No. .....

M Badri Narayanan
1600201018

SNO	Function	Operation	Time Complexity
12	Splay()	Constructor Initializes Root to NULL	O(1)
13	Node<T>* getRoot() Same for BST, AVL	Returns Root of Tree	O(1)
14	Node<T>* Search (Node<T>* r, Tkey) Parameter: Data to be searched, Node to be considered as root node Return Datatype: Root of Splayed Tree	Searches for data given in Parameter considering node given as Parameter as root node & Performs Splaying Operation	O(log n)
15	Node<T>* delete_key (Node<T>* r, Tkey) Parameter: Data to be deleted, Node to be considered as root Return Datatype: Root of Splayed Tree	Deletes data given in parameter, considering node in Parameter as root node & Performs Splaying Operation	O(log n)

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

SNO	Function	Operation	Time Complexity
16	Node<T>* insert(T x) Parameter: Data to be inserted Return Datatype: Root of Splayed Tree	Inserts data given in parameter considering the root node & Performs Splaying Operation	O(log n)
17	Node<T>* search(T key) Parameter: Data to be searched Return Datatype: Root of Splayed Tree	Searches for data given in parameter, considering the root node & Performs Splaying Operation	O(log n)
18	Node<T>* delete_key(T key) Parameter: Data to be deleted Return Datatype: Root of Splayed Tree	Deletes data given in parameter, Considering the root node & Performs Splaying Operation	O(log n)
19	int display()	Inorder Display of Splay Tree starting from root node	O(n)

splay.h

```
#ifndef splay_h
#define splay_h

#include<iostream>
using namespace std;

template <class T>
class Node
{
    public:

    T data;
    Node<T>*right;
    Node<T>*left;

    Node()
    {
        right = NULL;
        left = NULL;
    }
    Node(T x)
    {
        right = NULL;
        left = NULL;
        data = x;
    }
};
template <class T>
class Splay
{
    Node<T> *root;
    Node<T>* minimum(Node<T>* r)
    {
        Node<T>* temp = r;
        while (temp->left!=NULL)
            temp = temp->left;
        return temp;
    }
```

```
Node<T>* maximum(Node<T>* r)
{
    Node<T>* temp = r;
    while (temp->right!=NULL)
        temp = temp->right;
    return temp;
}

void preorder(Node<T> *t)
{
    if(t == NULL)return;
    cout<<" "<<t->data<<" ";
    preorder(t->left);
    preorder(t->right);
}

void inorder(Node<T> *t)
{
    if(t == NULL)return;
    inorder(t->left);
    cout<<" "<<t->data<<" ";
    inorder(t->right);
}

void postorder(Node<T> *t)
{
    if(t == NULL)return ;
    postorder(t->left);
    postorder(t->right);
    cout<<" "<<t->data<<" ";
}

Node<T>* rightRotate(Node<T> *x)
{
    Node<T> *y = x->left;
    x->left = y->right;
    y->right = x;
    return y;
}
```

```
Node<T>* leftRotate(Node<T> *x)
{
    Node<T> *y = x->right;
    x->right = y->left;
    y->left = x;
    return y;
}
Node<T>* splay(Node<T> *r,T key)
{
    if (r == NULL || r->data == key) return r;
    if (r->data > key)
    {
        if (r->left == NULL) return r;
        if (r->left->data > key)
        {
            r->left->left = splay(r->left->left, key);
            r = rightRotate(r);
        }
        else if (r->left->data < key)
        {
            r->left->right = splay(r->left->right, key);
            if (r->left->right != NULL)
                r->left = leftRotate(r->left);
        }
        return (r->left == NULL)? r: rightRotate(r);
    }
    else
    {
        if (r->right == NULL) return r;
        if (r->right->data > key)
        {
            r->right->left = splay(r->right->left, key);
            if (r->right->left != NULL)
                r->right = rightRotate(r->right);
        }
        else if (r->right->data < key)
        {
            r->right->right = splay(r->right->right, key);
            r = leftRotate(r);
        }
        return (r->right == NULL)? r: leftRotate(r);
```

```
    }
}
Node<T>* insert(Node<T> *r,T k)
{
    if (r == NULL)return new Node<T>(k);
    r = splay(r, k);
    if (r->data == k) return r;
    Node<T> *newnode = new Node<T>(k);
    if (r->data > k)
    {
        newnode->right = r;
        newnode->left = r->left;
        r->left = NULL;
    }
    else
    {
        newnode->left = r;
        newnode->right = r->right;
        r->right = NULL;
    }
    return newnode;
}
Node<T>* search(Node<T>* r,T key)
{
    return splay(r,key);
}
Node<T>* delete_key(Node<T>* r,T key)
{
    Node<T> *temp;
    if (!r) return NULL;
    r = splay(r, key);
    if (key != r->data) return r;
    if (!r->left)
    {
        temp = r;
        r = r->right;
    }
    else
    {
        temp = r;
        r = splay(r->left , key);
```

```
            r->right = temp->right;
        }
        delete temp;
        return r;
    }

    public:

    Splay()
    {
        root = NULL;
    }
    Node<T>* insert(T x)
    {
        root = insert(root,x);
        return root;
    }
    Node<T>* search(T key)
    {
        root = search(root,key);
        return root;
    }
    Node<T>* delete_key(T key)
    {
        root = delete_key(root,key);
        return root;
    }
    int display()
    {
        cout<<"\n\n InOrder Display \n\n";
        inorder(root);
        cout<<endl;
        return 0;
    }
    Node<T>* getRoot()
    {
        return root;
    }
};
#endif
```

display_tree3.h

```
#ifndef display_tree
#define display_tree

#include "splay.h"

#include <iostream>
#include <vector>
#include <unordered_map>
#include <cmath>
#include <iomanip>
#include <algorithm>

#define NOVAL1 -2147483648
#define NOVAL2 2147483647

using namespace std;

int max_level(Node<int> *root, int level)
{
    if (root==NULL)return level-1;
    else
        return max(max_level(root->left, level+1),
                   max_level(root->right, level+1));
}

void preorder_bst(Node<int> *root, int level,
            unordered_map<int, vector<int>> &map,
                        int max_level)
{
    if (level>max_level)return;
    if (root == NULL)
    {
        map[level].push_back(NOVAL1);
        preorder_bst(NULL, level + 1, map, max_level);
        preorder_bst(NULL, level + 1, map, max_level);
    }
```

```
        else
        {
            map[level].push_back(root->data);
            preorder_bst(root->left, level + 1, map, max_level);
            preorder_bst(root->right, level + 1, map, max_level);
        }
    }
    vector<int> reorder(Node<int>* r)
    {
        unordered_map<int, vector<int>> m;
        preorder_bst(r, 1, m, max_level(r, 1));
        vector<int> v;
        int start = max_level(r, 1);
        for (int i = start; i > 0; i--)
        {
            for (int j: m[i])
               v.push_back(j);
        }
        return v;
    }
    int maxdigits(vector<int> v)
    {
        int max = 0;
        for (auto x:v)
        {
            if (x!=NOVAL1 && x!=NOVAL2)
            {
                if (abs(x)>max)
                {
                    max = abs(x);
                }
            }
        }
        return floor(log10(max))+1;
    }

    void display_bst(Node<int>* r)
    {
        cout<<"\n\n Displaying The Tree  \n\n";
        vector<int> v = reorder(r);
        int no_rows = ceil(log2(v.size()));
```

```
int no_columns = 2*pow(2, no_rows-1)-1;
int width = maxdigits(v);
int** matrix = new int*[no_rows];
for (int i=0; i<no_rows; i++)
    matrix[i] = new int[no_columns];
for (int i=0; i<no_rows; i++)
    for (int j=0; j<no_columns; j++)
        matrix[i][j] = NOVAL2;
vector <int> lastpos;
vector<int> v2;
int a, b, c;
int k=0;
int x=-1, y=1;
while (x<no_columns)
{
    lastpos.push_back(x);
    lastpos.push_back(y);
    x+=2;
    y+=2;
}
for (int i=no_rows-1; i>=0; i--)
{
    a = 0;
    b = 1;
    while (b<lastpos.size())
    {
        c = (lastpos.at(a)+lastpos.at(b))/2;
        matrix[i][c] = v[k++];
        v2.push_back(c);
        a+=2;
        b+=2;
    }
    lastpos = v2;
    v2.clear();
}
for (int i=0; i<no_rows; i++)
{
    for (int j=0; j<no_columns; j++)
    {
        if (matrix[i][j]==NOVAL1)
            cout<<setfill(' ')<<setw(width)<<"x";
```

```
                else if(matrix[i][j]!=NOVAL2)
                    cout<<setfill(' ')<<setw(width)<<matrix[i][j];
                else
                    cout<<setfill(' ')<<setw(width)<<" ";
                cout<<" ";

            }
            cout<<"\n\n";
        }
    }
#endif
```

splaymain.cpp

```
#include"display_tree.h"
int main()
{
    cout<<"\n\n\n Splay Tree \n\n\n";
    int data;
    int choice;
    Splay<int> b;
    cout<<"\n\n Creating The Splay Tree \n\n";
    do
    {
        cout<<" Enter Data To Be Inserted : ";
        cin>>data;
        b.insert(data);
        cout<<" Do You Have Any More Data (Type 0 Or 1)
                    : ";
        cin>>choice;
    }while(choice);
    cout<<"\n\n Final Splay Tree \n\n";
    display_bst(b.getRoot());
    cout<<"\n\n\n";
    do
    {
        cout<<" Enter Data To Be Searched : ";
        cin>>data;
        b.search(data);
        cout<<" Do You Have Any More Data To Be Searched
                        (Type 0 Or 1) : ";
```

```
        cin>>choice;
    }while(choice);
    cout<<"\n\n Final Splay Tree \n\n";
    display_bst(b.getRoot());
    cout<<"\n\n\n";
    do
    {
        cout<<" Enter Data To Be Deleted : ";
        cin>>data;
        b.delete_key(data);
        cout<<" Do You Have Any More Data To Be Deleted
                    (Type 0 Or 1) : ";
        cin>>choice;
    }while(choice);
    cout<<"\n\n Final Splay Tree \n\n";
    display_bst(b.getRoot());
    cout<<"\n\n\n The End \n\n\n";
    return 0;
}
```

```
(base) badri@badri-VM:~/Desktop$ g++ splaymain.cpp -o splaymain
(base) badri@badri-VM:~/Desktop$ ./splaymain


Splay Tree



Creating The Splay Tree

Enter Data To Be Inserted : 3
Do You Have Any More Data (Type 0 Or 1) : 1
Enter Data To Be Inserted : 4
Do You Have Any More Data (Type 0 Or 1) : 1
Enter Data To Be Inserted : 5
Do You Have Any More Data (Type 0 Or 1) : 0


Final Splay Tree



Displaying The Tree

     5

 4       x

3   x   x   x



Enter Data To Be Searched : 3
Do You Have Any More Data To Be Deleted (Type 0 Or 1) : 0
```

(a)

```
Final Splay Tree


Displaying The Tree

     3

  x       4

x   x   x   5



Enter Data To Be Deleted : 5
Do You Have Any More Data To Be Deleted (Type 0 Or 1) : 0

Final Splay Tree


Displaying The Tree

  4

3   x



The End

(base) badri@badri-VM:~/Desktop$
```

(b)

Figure 7: Output For VII

Result A C++ program was created to implement Splay Tree ADT.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

EXP NO: 8
DATE : 11

M Badri Narayanan
185002018

Red Black Tree ADT Implementation

Aim: To create a C++ Program to implement Red Black Tree ADT

Time Complexity

SNO	Function	Operation	Time Complexity
1	Node() Constructor	Initializes right, left to NULL & col to RED	O(1)
2	Node(T x) Constructor Parameter: Value of Data	Initializes right, left to NULL, col to RED, data to x	O(1)
3	RBT() Constructor	Initializes root to NULL	O(1)
4	~RBT() Destructor	Deallocates memory of the RBT	O(n) n: No of Nodes

Class Diagram

Template <class T>

enum Colour {RED, BLACK};

Node
Public : T data; Node<T> * right, * left; Colour col; Node(); Node(T x)

Template <class T>

RBT

private : Node<T> * root. Node<T> * makeEmpty(Node<T> * t);
Node<T> * rotateRightChild(Node<T> * t);
Node<T> * rotateLeftChild(Node<T> * t);
Node<T> * doubleRotateLeft(Node<T> * t);
Node<T> * doubleRotateRight(Node<T> * t);
Node<T> * maximum(Node<T> * r);
Node<T> * minimum(Node<T> * r);
void inorder(Node<T> * t). void preorder(Node<T> * t);
void postorder(Node<T> * t); Node<T> * insert(Node<T> * t, T x);
Node<T> * resolve(Node<T> * t); bool isLeaf(Node<T> * t).
bool isChildRed(Node<T> * t); bool isGrandparent(Node<T> * t);
Public : RBT(); void display(); ~RBT();
Node<T> * insert(T x); Node<T> * getRoot();

SSN COLLEGE OF ENGINEERING Sheet No.....
RECORD SHEET Register No : 185002018
M Badri Narayanan

SNO	Function	Operation	Time Complexity
5	Void inorder (Node<T>*r)	Refer BST	O(n)
6	Void preorder (Node<T>*r)	Refer BST	O(n)
7	Void postorder (Node<T>*r)	Refer BST	O(n)
8	bool isLeaf (Node<T>*r)	Checks if node given in parameter is a leaf node	O(1)
9	bool isChildRed (Node<T>*r)	Checks if node given in parameter has a Red Child	O(1)
10	bool isGrandparent (Node<T>*r)	Checks if node given in parameter is a grandparent	O(1)
11	Node<T>*makeEmpty (Node<T>*t)	Deallocates the memory of RBT considering node given in parameter as root node	O(n)
12	Node<T>*getRoot()	Refer SPLAY TREE ADT	O(1)

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

SNO	Function	Operation	Time Complexity
13	Node<T>* maximum (Node<T>*r)	Refer BST	O(n)
14	Node<T>* minimum (Node<T>*r)	Refer BST	O(n)
15	Node<T>* rotateLeft(child (Node<T>*r)	Refer L Rotation (Node<T>*r) of AVL Tree ADT	O(1)
16	Node<T>* rotate Right(child (Node<T>*r)	Refer R Rotation (Node<T>*r) of AVL Tree ADT	O(1)
17	Node<T>* doubleRotateLeft (Node<T>*t)	Refer RL Rotation (Node<T>*r) of AVL Tree ADT	O(1)
18	Node<T>* doubleRotate Right (Node<T>*t)	Refer LR Rotation (Node<T>*r) of AVL Tree ADT	O(1)

SSN COLLEGE OF ENGINEERING

RECORD SHEET

Sheet By: M Bhadri Narayanan

1CB002018

SNO	Function	Operation	Time Complexity
19	Node(T)* resolve(Node(T)*r)	Returns the root of RBT after performing operations to satisfy Red Black Tree property considering Node passed in parameter as root node	O(log n)
20	Node(T)* insert(Node(T)*r, T x)	Returns the root of RBT after inserting data given in parameter considering Node given in parameter as root Node after fixing RBT Property violations	O(log n)
21	Node(T)* insert(T x) Returns root of inserted RBT	Performs RBTree insert of the data given in parameter starting from root node & fixing RBT property violations	O(log n)
22	int display()	Inorder Display of RBT	O(n)

RBT.h

```
#ifndef RBT_H
#define RBT_H

#include<iostream>

using namespace std;

enum Colour {RED, BLACK};

template <class T>
class Node
{
    public:

    T data;
    Node<T>*right;
    Node<T>*left;
    Colour col;
    Node()
    {
        right = NULL;
        left = NULL;
        col = RED;
    }

    Node(T x)
    {
        right = NULL;
        left = NULL;
        data = x;
        col = RED;
    }
};
```

```
template <class T>
class RBT
{
    Node<T> *root;
    Node<T> *rotateRightChild(Node<T> *t)
    {

        if(t->right)
        {
            Node<T> *u = t->right;
            t->right = u->left;
            u->left = t;
            return u;
        }

        else
            return t;
    }
    Node<T> *rotateLeftChild(Node<T> *t)
    {
        if(t->left)
        {
            Node<T> *u = t->left;
            t->left = u->right;
            u->right = t;
            return u;
        }
        else return t;
    }
    Node<T> *doubleRotateLeft(Node<T> *t)
    {
        t->left = rotateRightChild(t->left);
        t = rotateLeftChild(t);
        return t;
    }
    Node<T> *doubleRotateRight(Node<T> *t)
    {
        t->right = rotateLeftChild(t->right);
        t = rotateRightChild(t);
        return t;
    }
```

```
Node<T>* minimum(Node<T>* r)
{
        Node<T>* temp = r;
        while (temp->left!=NULL)
            temp = temp->left;
        return temp;
}
Node<T>* maximum(Node<T>* r)
{
    Node<T>* temp = r;
    while (temp->right!=NULL)
        temp = temp->right;
    return temp;
}
void preorder(Node<T> *t)
{
    if(t == NULL)return;
    cout<<" "<<t->data<<"-"<<
                (t->col ? "BLACK":"RED")<<endl;
    preorder(t->left);
    preorder(t->right);
}
void inorder(Node<T> *t)
{
    if(t == NULL)return;
    inorder(t->left);
    cout<<" "<<t->data<<"-"<<
                (t->col ? "BLACK":"RED")<<endl;
    inorder(t->right);
}
void postorder(Node<T> *t)
{
    if(t == NULL)return ;
    postorder(t->left);
    postorder(t->right);
    cout<<" "<<t->data<<"-"<<
                (t->col ? "BLACK":"RED")<<endl;
}
```

```
bool isLeaf(Node<T> *t)
{
    if(t->right == NULL && t->left == NULL)
        return true;
    return false;
}
bool isChildRed(Node<T> *t)
{
    if(t->left && t->left->col == RED)
        return true;

    if(t->right && t->right->col == RED)
        return true;

    return false;
}
bool isGrandparent(Node<T> *t)
{
    if(t->left)
    {
        if(t->left->right || t->left->left)
            return true;
    }
    if(t->right)
    {
        if(t->right->right || t->right->left)
            return true;
    }
    return false;
}
Node<T> *makeEmpty(Node<T> *t)
{
    if(t == NULL)return NULL;
    makeEmpty(t->left);
    makeEmpty(t->right);
    delete t;
    return NULL;
}
```

```
Node<T> *resolve(Node<T> *t)
{
    if(isGrandparent(t))
    {
        if(t->right && t->left)
        {
            if(t->right->col == RED &&
                        t->left->col == RED)
            {
                if(isChildRed(t->right) ||

                            isChildRed(t->left))
                {
                    t->right->col = t->col;
                    t->left->col = t->col;
                    t->col = RED;
                }
            }

            else if(t->right->col == RED &&
                            t->left->col == BLACK)
            {
                if(t->right->right &&
                            t->right->right->col == RED)
                {
                    t = rotateRightChild(t);
                    t->col = BLACK;
                    t->left->col = RED;
                }

                else if(t->right->left &&
                            t->right->left->col == RED)
                {
                    t = doubleRotateRight(t);
                    t->col = BLACK;
                    t->left->col = RED;
                }

            }
```

```
        else if(t->right->col == BLACK &&
                        t->left->col == RED)
        {
            if(t->left->right &&
                        t->left->right->col == RED)
            {
                t = doubleRotateLeft(t);
                t->col = BLACK;
                t->right->col = RED;
            }

            else if(t->left->left &&
                        t->left->left->col == RED)
            {
                t = rotateLeftChild(t);
                t->col = BLACK;
                t->right->col = RED;
            }

        }
    }

    else if(t->right && t->right->col == RED)
    {
        if(t->right->right &&
                    t->right->right->col == RED)
        {
            t = rotateRightChild(t);
            t->col = BLACK;
            t->left->col = RED;
        }

        else if(t->right->left &&
                    t->right->left->col == RED)
        {
            t = doubleRotateRight(t);
            t->col = BLACK;
            t->left->col = RED;
        }
    }
```

```
            else if(t->left && t->left->col == RED)
            {
                if(t->left->right &&
                            t->left->right->col == RED)
                {
                    t = doubleRotateLeft(t);
                    t->col = BLACK;
                    t->right->col = RED;
                }

                else if(t->left->left &&
                                t->left->left->col == RED)
                {
                    t = rotateLeftChild(t);
                    t->col = BLACK;
                    t->right->col = RED;
                }
            }
        }
        return t;
    }
    Node<T> *insert(Node<T> *t, T x)
    {
        if(t == NULL)t = new Node<T>(x);
        else if(x < t->data) t->left = insert(t->left, x);
        else if(x > t->data)t->right = insert(t->right, x);
        t = resolve(t);
        return t;
    }

    public:
    RBT()
    {
        root = NULL;
    }
    ~RBT()
    {
        root = makeEmpty(root);
    }
```

```
    Node<T>* insert(T x)
    {
        root = insert(root, x);
        root->col = BLACK;
        return root;
    }
    int display()
    {
        cout<<"\n\n InOrder Display \n\n";
        inorder(root);
        cout<<endl;
        return 0;
    }
    Node<T>* getRoot()
    {
        return root;
    }

};
#endif
```

display_tree2.h

```
#ifndef display_tree
#define display_tree

#include "RBT.h"

#include <iostream>
#include <vector>
#include <unordered_map>
#include <cmath>
#include <iomanip>
#include <algorithm>

#define NOVAL1 -2147483648
#define NOVAL2 2147483647

using namespace std;
```

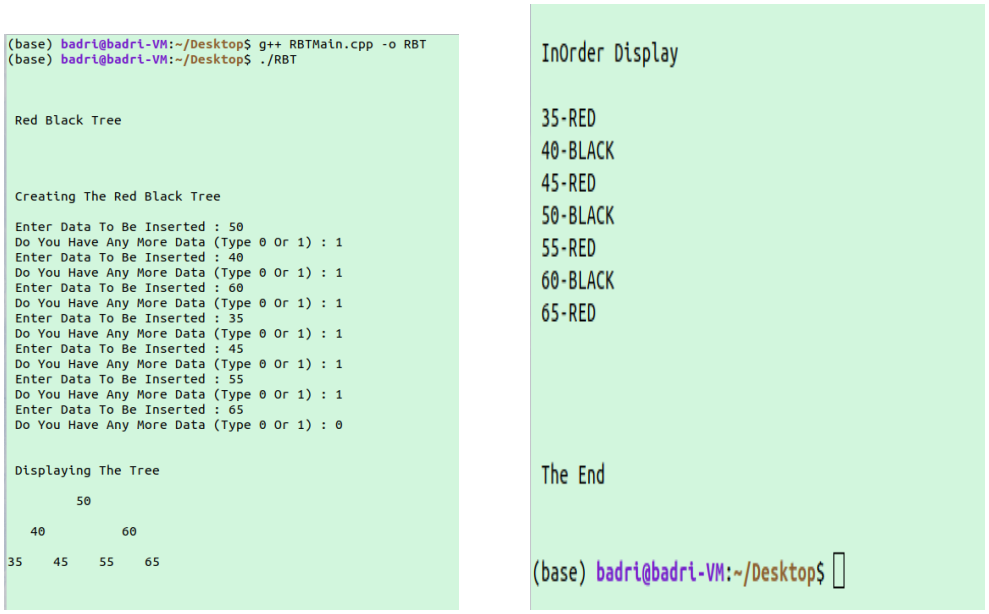
```
int max_level(Node<int> *root, int level)
{
    if (root==NULL)return level-1;
    else
        return max(max_level(root->left, level+1),
                   max_level(root->right, level+1));
}

void preorder_bst(Node<int> *root, int level,
            unordered_map<int, vector<int>> &map,
                          int max_level)
{
    if (level>max_level)return;
    if (root == NULL)
    {
        map[level].push_back(NOVAL1);
        preorder_bst(NULL, level + 1, map, max_level);
        preorder_bst(NULL, level + 1, map, max_level);
    }
    else
    {
        map[level].push_back(root->data);
        preorder_bst(root->left, level + 1, map, max_level);
        preorder_bst(root->right, level + 1, map, max_level);
    }
}
vector<int> reorder(Node<int>* r)
{
    unordered_map<int, vector<int>> m;
    preorder_bst(r, 1, m, max_level(r, 1));
    vector<int> v;
    int start = max_level(r, 1);
    for (int i = start; i > 0; i--)
    {
        for (int j: m[i])
           v.push_back(j);
    }
    return v;
}
```

```
int maxdigits(vector<int> v)
{
    int max = 0;
    for (auto x:v)
    {
        if (x!=NOVAL1 && x!=NOVAL2)
        {
            if (abs(x)>max)
            {
                max = abs(x);
            }
        }
    }
    return floor(log10(max))+1;
}

void display_bst(Node<int>* r)
{
    cout<<"\n\n Displaying The Tree  \n\n";
    vector<int> v = reorder(r);
    int no_rows = ceil(log2(v.size()));
    int no_columns = 2*pow(2, no_rows-1)-1;
    int width = maxdigits(v);
    int** matrix = new int*[no_rows];
    for (int i=0; i<no_rows; i++)
        matrix[i] = new int[no_columns];
    for (int i=0; i<no_rows; i++)
        for (int j=0; j<no_columns; j++)
            matrix[i][j] = NOVAL2;
    vector <int> lastpos;
    vector<int> v2;
    int a, b, c;
    int k=0;
    int x=-1, y=1;
    while (x<no_columns)
    {
        lastpos.push_back(x);
        lastpos.push_back(y);
        x+=2;
        y+=2;
    }
```

```
        for (int i=no_rows-1; i>=0; i--)
        {
            a = 0;
            b = 1;
            while (b<lastpos.size())
            {
                c = (lastpos.at(a)+lastpos.at(b))/2;
                matrix[i][c] = v[k++];
                v2.push_back(c);
                a+=2;
                b+=2;
            }
            lastpos = v2;
            v2.clear();
        }
        for (int i=0; i<no_rows; i++)
        {
            for (int j=0; j<no_columns; j++)
            {
                if (matrix[i][j]==NOVAL1)
                    cout<<setfill(' ')<<setw(width)<<"x";
                else if(matrix[i][j]!=NOVAL2)
                    cout<<setfill(' ')<<setw(width)<<matrix[i][j];
                else
                    cout<<setfill(' ')<<setw(width)<<" ";
                cout<<" ";

            }
            cout<<"\n\n";
        }
    }
    #endif
```

RBTMain.cpp

```
#include"display_tree2.h"
int main()
{
    cout<<"\n\n\n Red Black Tree \n\n\n";
    int data;
    int choice;
    RBT<int> b;
    cout<<"\n\n Creating The Red Tree \n\n";
    do
    {
        cout<<" Enter Data To Be Inserted : ";
        cin>>data;
        b.insert(data);
        cout<<" Do You Have Any More Data (Type 0 Or 1) : ";
        cin>>choice;
    }while(choice);
    display_bst(b.getRoot());
    b.display();
    cout<<"\n\n\n The End \n\n\n";
    return 0;
}
```



(a) (b)

Figure 8: Output For VIII

Result A C++ program was created to implement Red Black Tree ADT.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

EXP NO: 9
DATE: 11

M Badri Narayanan
185002018

Fibonacci Heap ADT Implementation

Aim: To create a C++ program to implement Fibonacci Heap ADT

Time Complexity

SNO	Function	Operation	Time Complexity
1	Node() Constructor	Initializes left, right, parent, child to NULL & order to -1	O(1)
2	Node (T x) Constructor Parameter: Value for data	Initializes left, right, parent, child to NULL, order to -1 & data to x	O(1)
2	fib_heap()	Initializes min to NULL	O(1)

Class Diagram

Template <class T>

Node
Public : T data; int order; Node(T) * left, * right, * parent, * child; Node(); Node (T x)

Template <class T>

fib_heap

Private: Node<T> * min;
Public: fib_heap(); int insert (T x);
int insert (Node<T> * temp);
int display(); int del (Node<T> * temp);
int update_min(); T extract_min();
Node<T> * link (Node<T> * x);

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

SNO	Function	Operation	Time Complexity
4	int insert (T x)	Inserts data given in parameter	(H) (1)
5	int insert (Node<T>* temp)	Inserts the tree given in parameter	(H) (1)
6	int display()	Displays the fibonacci Heap	O(n) n: Total no of nodes in Heap
7	Node<T> * link (Node<T>* x)	Merges the tree given in parameter with the existing heap	(H) (1)
8	T extract_min()	Returns the min element in Heap	O(log n)
9	int update_min()	Updates min element in Heap	(H) (1)

fibonacciheap.h

```
#ifndef fibonacciheap_h
#define fibonacciheap_h

#include<iostream>

using namespace std;

template<class T>
class Node
{
    public:
    T data;
    Node<T> *left,*right;
    Node<T>* parent,*child;
    int order;

        Node()
        {
            left = right = parent = child = NULL;
            order=-1;
        }
        Node(T x)
        {
            left = right = parent = child = NULL;
            order = 0;
            data = x;
        }

};

template<class T>
class fib_heap
{
    Node<T>* min;
    public:
```

```
fib_heap()
{
    min=NULL;
}
int insert(T x)
{
    Node<T>* temp=new Node<T>(x);
    if(min==NULL)
    {
        min=temp;
        min->left=min;
        min->right=min;
    }
    else
    {
        temp->left=min->left;
        temp->right=min;
        min->left=temp;
        (temp->left)->right=temp;
        if(x<min->data)min=temp;
    }
    return 0;
}
int insert(Node<T>* temp)
{
    if(min==NULL)
    {
        min=temp;
        min->left=min;
        min->right=min;
    }
    else
    {
        temp->left=min->left;
        temp->right=min;
        min->left=temp;
        (temp->left)->right=temp;
        if(temp->data<min->data)min=temp;
    }
    return 0;
}
```

```
int display()
{
    Node<T>* temp=min;
    if(min==NULL)return -1;
    cout<<" "<<min->data<<endl;
    temp=min->left;
    while(temp!=min&& temp!=NULL)
    {
        cout<<" "<<temp->data<<endl;
        temp=temp->left;
    }
    return 0;
}
Node<T>* link(Node<T>* x)
{
    Node<T>* c=x->child;
    if(c==NULL)
    {
        cout<<" NULL Child \n\n"<<endl;
        return NULL;
    }
    else
    {
        Node<T>* p=c->right;
        insert(c);
        while(p!=c&& p!=NULL)
        {
            insert(p);
            p=p->right;
        }
    }

}
int update_min()
{
    Node<T>* p=min;
    Node<T>* q=p;
    p=min->right;
    while(p!=NULL && p!=q )
    {
```

```
                if(p->data<min->data)
                {
                    min=p;
                }
                p=p->right;
            }
        }
        int del(Node<T>* temp)
        {
            Node<T>* x=temp;
            (temp->left)->right=x->right;
            (temp->right)->left=x->left;
            delete temp;
        }
        T extract_min()
        {
            T temp=min->data;
            Node<T>* t=min;
            cout<<" "<<temp<<"\n\n";
            if(min->child==NULL)cout<<" No Child \n\n";
            link(min);
            min->data = 99;
            update_min();
            del(t);
            return temp;
        }
};

#endif
```

fibonacciheapmain.cpp

```
#include"fibonacciheap.h"

int main()
{
    cout<<"\n\n\n Fibonacci Heap \n\n\n";
    int data;
    int choice;
    fib_heap<int> f;
    cout<<"\n\n Creating The Fibonacci Heap \n\n";
```

```
    do
    {
        cout<<" Enter Data To Be Inserted : ";
        cin>>data;
        f.insert(data);
        cout<<" Do You Have Any More Data (Type 0 Or 1) : ";
        cin>>choice;
    }while(choice);
    cout<<"\n\n Final Fibonacci Heap \n\n";
    f.display();
    cout<<"\n\n\n";
    do
    {
        cout<<" Extract Min ";
        cout<<" Minimum Data : "<<f.extract_min()<<"\n\n";
        cout<<" Do You Want To Continue (Type 0 Or 1) : ";
        cin>>choice;
    }while(choice);
    cout<<"\n\n Final Fibonacci Heap \n\n";
    f.display();
    cout<<"\n\n\n The End \n\n\n";
    return 0;
}
```

```
(base) badri@badri-VM:~/Desktop$ g++ fibonacciheapmain.cpp -o fibonacciheap
(base) badri@badri-VM:~/Desktop$ ./fibonacciheap


Fibonacci Heap



Creating The Fibonacci Heap

Enter Data To Be Inserted : 5
Do You Have Any More Data (Type 0 Or 1) : 1
Enter Data To Be Inserted : 6
Do You Have Any More Data (Type 0 Or 1) : 1
Enter Data To Be Inserted : 88
Do You Have Any More Data (Type 0 Or 1) : 1
Enter Data To Be Inserted : 99
Do You Have Any More Data (Type 0 Or 1) : 1
Enter Data To Be Inserted : 100
Do You Have Any More Data (Type 0 Or 1) : 1
Enter Data To Be Inserted : 33
Do You Have Any More Data (Type 0 Or 1) : 1
Enter Data To Be Inserted : 111
Do You Have Any More Data (Type 0 Or 1) : 0


Final Fibonacci Heap

5
111
33
100
99
88
6
```

(a)

```
Extract Min  Minimum Data :  5

No Child

NULL Child


5

Do You Want To Continue (Type 0 Or 1) : 1
Extract Min  Minimum Data :  6

No Child

NULL Child


6

Do You Want To Continue (Type 0 Or 1) : 0


Final Fibonacci Heap

33
100
99
88
111



The End

(base) badri@badri-VM:~/Desktop$
```

(b)

Figure 9: Output For IX

Result A C++ program was created to implement Fibonacci Heap ADT.

M Badri Narayanan
Register Number : 185002018

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

EXP NO: 10
DATE: 11

M Badri Narayanan
185002018

BFS & DFS Graph Traversal Algorithms Implementation

Aim: To create a C++ program to implement BFS & DFS Graph Traversal algorithms

Time Complexity

V: No of Vertices
E: No of Edges

SNO	Function	Operation	Time Complexity
1	Graph(int V) Constructor	Initializes No Of Vertices to V & allocates memory to adj of size V	O(1)
2	int addEdge(T V1, T V2)	Adds a Edge from V1 to V2	O(1)
3	int BFS(T source) Parameter: source for BFS traversal	BFS traversal of graph starting from source	O(V+E)
4	int DFS(T source) Parameter: source for DFS traversal	DFS traversal of graph starting from source	O(V+E)

Class Diagram

template <class T>

Graph

Public : Graph(int v);
int addEdge(T v1, T v2);
int BFS(T source);
int DFS(T source);

bfsanddfs.h

```
#ifndef bfsanddfs_h
#define bfsanddfs_h

#include<iostream>
#include<list>
#include<string>

using namespace std;

template<class T>
class Graph
{
    int          no_of_vertices;
    list<int>    *adj;

    public:
            Graph(int v)
            {
                no_of_vertices = v;
                adj = new list<int>[no_of_vertices];
            }
            int addEdge(T v1, T v2)
            {
                if(isalpha(v1))
                {
                    if(v1 >= 65 && v1 <= 90)
                    {
                        int v3 = (int)v1-65;
                        int v4 = (int)v2-65;
                        adj[v3].push_back(v4);
                    }
                    else if(v1 >= 97 && v1 <= 122)
                    {
                        int v3 = (int)v1-97;
                        int v4 = (int)v2-97;
                        adj[v3].push_back(v4);
                    }
                }
```

```
        else adj[v1].push_back(v2);
}
int BFS(T source)
{
        int      flag = 0;
        int      i,vertex, source1;
        list<int>   Q;
        list<int> :: iterator it;
        bool *visited = new bool[no_of_vertices];
        for(i=0;i<no_of_vertices;i++)
        {
            visited[i] = false;
        }
        if(isalpha(source))
        {
            if(source>=65 && source<=90)
            {
                flag=1;
                source1= (int)source-65;
            }
            else if(source>=97 && source<=122)
            {
                flag=2;
                source1= (int)source-97;
            }
            visited[source1] = true;
            Q.push_back(source1);
        }
        else
        {
            visited[source] = true;
            Q.push_back(source);
        }
        while(!Q.empty())
        {
            vertex = Q.front();
            Q.pop_front();
            if(flag==1)
            {
                cout<<" "<<char(vertex+65)<<" ";
            }
```

```
            else if(flag==2)
            {
                cout<<" "<<char(vertex+97)<<" ";
            }
            else cout<<" "<<vertex<<" ";
            for(it = adj[vertex].begin();
                it!=adj[vertex].end();it++)
            {
                if(!visited[*it])
                {
                    visited[*it] = true;
                    Q.push_back(*it);
                }
            }
        }
        return 0;
}
int DFS(T source)
{
        int     flag,source1;
        int     i,vertex;
        bool    *visited = new bool[no_of_vertices];
        list<int> Stack;
        list<int> :: iterator it;
        flag = 0;
        for(i=0;i<no_of_vertices;i++)
        {
            visited[i] = false;
        }
        if(isalpha(source))
        {
            if(source>=65 && source<=90)
             {
                flag = 1;
                source1= (int)source-65;
            }
            else if(source>=97 && source<=122)
            {
                flag = 2;
                source1= (int)source-97;
            }
```

```
                visited[source1] = true;
                Stack.push_back(source1);
            }
            else
            {
                visited[source] = true;
                Stack.push_back(source);
            }
             while(!Stack.empty())
            {
                vertex = Stack.back();
                Stack.pop_back();
                if(flag==1)
                {
                    cout<<" "<<char(vertex+65)<<" ";
                }
                else if(flag==2)
                {
                    cout<<" "<<char(vertex+97)<<" ";
                }
                else cout<<" "<<vertex<<" ";
                for(it = adj[vertex].begin();
                    it!=adj[vertex].end();it++)
                {
                    if(!visited[*it])
                    {
                        visited[*it] = true;
                        Stack.push_back(*it);
                    }
                }
            }
            return 0;
        }
};
#endif
```

bfsanddfs.cpp

```
#include "bfsanddfs.h"
int main()
{
    int     choice;
    do
    {
        cout<<"\n\n Creating The Graph \n\n";
        cout<<" 1. Character \n";
        cout<<" 2. Integer \n";
        cout<<" Enter Choice : ";
        cin>>choice;
        cout<<endl<<endl;
        if(choice == 1)
        {
            int     n;
            char    vertex1,vertex2,source;
            int     vertexcontinue;
            cout<<" Enter Number Of Vertices : ";
            cin>>n;
            Graph<char> G(n);
            cout<<" Enter Source Vertex : ";
            cin>>source;
            cout<<endl<<endl;
            do
            {
                cout<<"\n Enter Vertex 1 Of Edge : ";
                cin>>vertex1;
                cout<<"\n Enter Vertex 2 Of Edge : ";
                cin>>vertex2;
                G.addEdge(vertex1,vertex2);
                cout<<"\n Do You Have Any More Edges
                ( Type 0 Or 1 To Continue) : ";
                cin>>vertexcontinue;

            }while(vertexcontinue == 1);
            cout<<"\n\n BFS & DFS Graph Traversal \n\n";
            cout<<" 1. BFS \n";
            cout<<" 2. DFS \n";
```

```
        cout<<"\n\n BFS \n\n";
        G.BFS(source);
        cout<<endl<<endl;
        cout<<"\n\n DFS \n\n";
        G.DFS(source);
        cout<<endl<<endl;
    }
    if(choice == 2)
    {
        int     n;
        int     vertex1,vertex2,source;
        char    vertexcontinue;
        cout<<" Enter Number Of Vertices : ";
        cin>>n;
        Graph<int> G1(n);
        cout<<" Enter Source Vertex : ";
        cin>>source;
        cout<<endl<<endl;
        do
        {
            cout<<"\n Enter Vertex 1 Of Edge : ";
            cin>>vertex1;
            cout<<"\n Enter Vertex 2 Of Edge : ";
            cin>>vertex2;
            G1.addEdge(vertex1,vertex2);
            cout<<"\n Do You Have Any More Edges
            (Type 0 Or 1 To Continue) : ";
            cin>>vertexcontinue;
        }while(vertexcontinue == 'Y');
        cout<<"\n\n BFS & DFS Graph Traversal \n\n";
        cout<<" 1. BFS \n";
        cout<<" 2. DFS \n";
        cout<<"\n\n BFS \n\n";
        G1.BFS(source);
        cout<<endl<<endl;
        cout<<"\n\n DFS \n\n";
        G1.DFS(source);
        cout<<endl<<endl;
    }
    cout<<"\n Do You Want To Continue
    ( Type 0 Or 1) : ";
```

```
        cin>>choice;
    }while(choice == 1);
    cout<<"\n\n The End \n\n";
    return 0;
}
```

```
(base) badri@badri-VM:~/Desktop$ g++ bfsanddfsmain.cpp -o bfsanddfsmain
(base) badri@badri-VM:~/Desktop$ ./bfsanddfsmain


 Creating The Graph

 1. Character
 2. Integer
 Enter Choice : 1


 Enter Number Of Vertices : 4
 Enter Source Vertex : A



 Enter Vertex 1 Of Edge : A

 Enter Vertex 2 Of Edge : B

 Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 1

 Enter Vertex 1 Of Edge : B

 Enter Vertex 2 Of Edge : C

 Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 1

 Enter Vertex 1 Of Edge : C

 Enter Vertex 2 Of Edge : D
```

(a)

```
Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 1

Enter Vertex 1 Of Edge : D

Enter Vertex 2 Of Edge : A

Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 0


BFS & DFS Graph Traversal

1. BFS
2. DFS


BFS

A  B  C  D



DFS

A  B  C  D


Do You Want To Continue ( Type 0 Or 1) : 0


The End

(base) badri@badri-VM:~/Desktop$
```

(b)

Figure 10: Output For X

Result A C++ program was created to implement BFS & DFS Graph Traversal Algorithms.

SSN COLLEGE OF ENGINEERING
RECORD SHEET
Sheet No.....

EXP NO : 11
DATE : 1 1

M Badri Narayanan
185002018

Topological Sort - DFS Application

Aim: To create a C++ program to implement topological sort.

Time Complexity:

V : No of Vertices in Graph
E : No of Edges in Graph

SNO	Function	Operation	Time Complexity
1	Graph(int ve) Constructor	Initializes V to ve & allocates memory of size ve to adj	O(1)
2	Void addEdge (int ve, int w)	Adds a Edge from ve to w	O(1)
3	Void topologicalSortUtill (int ve, bool visited[], stack<int> & Stack)	Performs topological sort for a single vertex	O(V + E)
4	Void topologicalSort()	Performs Topological Sort	O(V+E)

Class Diagram

Graph
Private: int V; list<int> *adj; void topologicalSortUntill(int v, bool visited[], stack<int>&, stack); Public: Graph(int v); Void addEdge(int v, int w) Void topologicalSort();

topologicalsort.h

```
#ifndef topological_h
#define topologicalsort_h

#include<iostream>
#include <list>
#include <stack>
using namespace std;

class Graph
{
    int V;

    list<int> *adj;

    void topologicalSortUtil(int v, bool visited[],
                             stack<int> &Stack)
    {
        visited[v] = true;
        list<int>::iterator i;

        for (i = adj[v].begin(); i != adj[v].end(); ++i)
        {
            if (!visited[*i])topologicalSortUtil(*i,
                                        visited, Stack);
        }
        Stack.push(v);
    }
    public:
    Graph(int v)
    {
        V = v;
        adj = new list<int>[V];
    }
    void addEdge(int v, int w)
    {
        adj[v].push_back(w);
    }
```

```
    void topologicalSort()
    {

        stack<int> Stack;

        bool *visited = new bool[V];

        for (int i = 0; i < V; i++)
        {
            visited[i] = false;
        }

        for (int i = 0; i < V; i++)
        {
            if (visited[i] == false) topologicalSortUtil(i, visited

        }

        while (Stack.empty() == false)
        {
            cout <<" "<<Stack.top() << " ";
             Stack.pop();
        }
    }
};

#endif
```

topologicalsort.cpp

```
#include"topologicalsort.h"

int main()
{
    cout<<"\n\n Topological Sort \n\n";

    int n;

    cout<<" Enter Number Of Vertices : ";
    cin>>n;

    Graph G(n);


    int vertexcontinue,vertex1,vertex2;

    do
    {
        cout<<"\n Enter Vertex 1 Of Edge : ";
        cin>>vertex1;
        cout<<"\n Enter Vertex 2 Of Edge : ";
        cin>>vertex2;
        G.addEdge(vertex1,vertex2);
        cout<<"\n Do You Have Any More Edges ( Type 0 Or 1 To Cont
        cin>>vertexcontinue;

    }while(vertexcontinue == 1);

    cout << "\n\n Topological Sort Of Given Graph \n\n";

    G.topologicalSort();

    cout<<"\n\n The End \n\n";

    return 0;
}
```

```
(base) badri@badri-VM:~/Desktop$ g++ topologicalsort.cpp -o topological
(base) badri@badri-VM:~/Desktop$ ./topological


Topological Sort

Enter Number Of Vertices : 6

Enter Vertex 1 Of Edge : 5

Enter Vertex 2 Of Edge : 2

Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 1

Enter Vertex 1 Of Edge : 5

Enter Vertex 2 Of Edge : 0

Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 1

Enter Vertex 1 Of Edge : 4

Enter Vertex 2 Of Edge : 0

Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 1
```

(a)

```
Enter Vertex 1 Of Edge : 4

Enter Vertex 2 Of Edge : 1

Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 1

Enter Vertex 1 Of Edge : 2

Enter Vertex 2 Of Edge : 3

Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 1

Enter Vertex 1 Of Edge : 3

Enter Vertex 2 Of Edge : 1

Do You Have Any More Edges ( Type 0 Or 1 To Continue) : 0


Topological Sort Of Given Graph

5  4  2  3  1  0

The End

(base) badri@badri-VM:~/Desktop$
```

(b)

Figure 11: Output For XI

Result A C++ program was created to implement Topological Sort.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

EXP NO: 12a
DATE: 11

M Badri Narayanan
185002018

Single Source Shortest Path Algorithm - Dijkstra's Algorithm

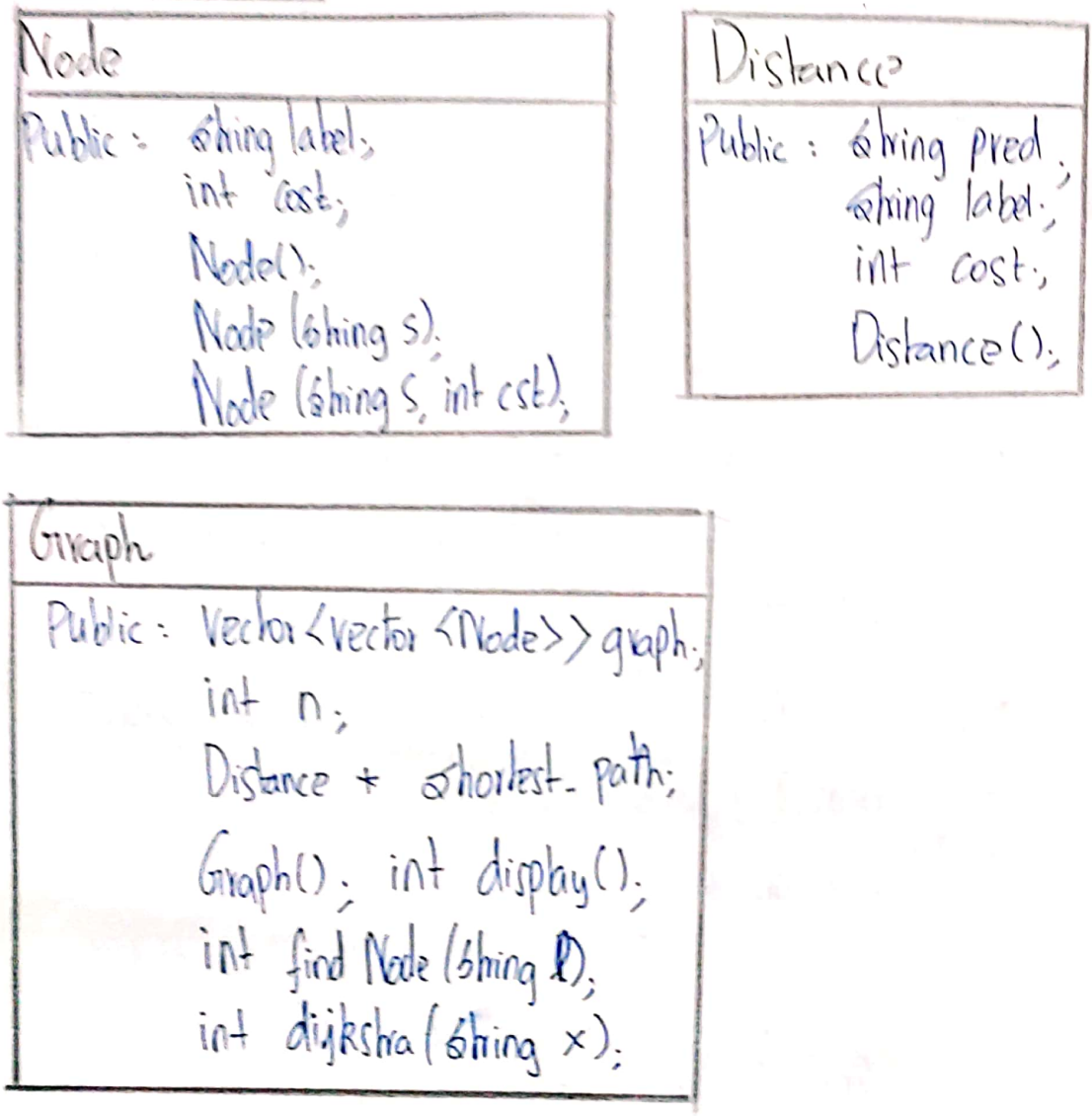
Aim: To create a C++ program to implement single source shortest path dijkstra's algorithm

Time Complexity

max = 100000000
n1: No of vertices, n2: No of Adj Vertices

SNO	Function	Operation	Time Complexity
1	Node()	Empty Constructor	O(1)
2	Node(string s) Constructor	Assigns label to s & cost to max	O(1)
3	Node(string s, int cst) Constructor	Assigns label to s & cost to cst	O(1)
4	Distance() Constructor	Assigns cost to max	O(1)
5	Graph() Constructor	Creates a graph of n vertices	O(n1 * n2)

Class Diagram



SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
1E5002

SNO	Function	Operation	Time Complexity
6	int display()	Displays Graph details	O(n1 + n2)
7	int dijkstra (string s)	Performs Dijkstra Algorithm on Graph, with parameter considered as source	$O(V^2)$ Adjacency Matrix O(ElogV) Adjacency List V: No of Vertices E: No of Edges
8	int findNode(string l)	Finds if a node passed as parameter exists in graph	O(n1)

dijkstra.h

```
#ifndef dijkstra_h
#define dijkstra_h

#include <iostream>
#include <vector>

using namespace std;

#define max 100000000

class Node
{
    public:
    string label;
    int     cost;

    Node() {}

    Node(string s)
    {
        label = s;
        cost = max;
    }
    Node(string s, int cst)
    {
        label = s;
        cost = cst;
    }
};
class Distance
{
    public:
        string pred;
        string label;
        int cost;
```

```
        Distance()
        {
            cost = max;
        }
};


class Graph
{
    public:
            vector< vector<Node> > graph;
            Distance* shortest_path;
            int n;

    Graph()
    {
        string s;
        cout <<" Enter Number Of Vertices : ";
        cin >>n;
        string t;
        for (int i = 0; i < n; i++)
        {
            vector<Node> singleList;
            cout<<"\n Details For Vertex "<<i + 1 <<"\n";
            cout <<"\n Enter Label Of Node : ";
            cin >>s;
            singleList.push_back(Node(s, 0));
            cout <<"\n Enter Number of Adjacent Nodes : ";
            int n2, cst;
            cin >>n2;
            for (int j = 0; j < n2; j++)
            {
                cout<<"\n Details For Adjacent Node "
                    <<j + 1<<"\n";
                cout <<"\n Enter Label Of Adjacent Node : ";
                cin>>s;
                cout<<"\n Enter Cost : ";
                cin>>cst;
                singleList.push_back(Node(s, cst));
            }
            graph.push_back(singleList);
```

```
    }
    shortest_path = new Distance[n];
}

int display()
{
    cout<<"\n\n Displaying Graph Details \n\n";
    for (int i = 0; i < n; i++)
    {
        int s = graph[i].size();
        for (int j = 0; j < s; j++)
        {
            cout <<" "<<graph[i][j].label<<"  "
                << graph[i][j].cost<<"\n";
        }
    }
    cout<<"\n\n";
    return 0;
}

int findNode(string l)
{
    for (int i = 0; i < n; i++)
    {
        if (graph[i][0].label == l)return i;
    }
    return -1;
}

int dijkstra(string x)
{
    cout<<"\n\n The Shortest Path Is \n\n";
    bool visited[1001] = { false };
    int y = findNode(x);
    for (int i = 0; i < n; i++)
    {
        if (i == y)
        {
            shortest_path[i].pred = x;
            shortest_path[i].label = graph[i][0].label;
            shortest_path[i].cost = 0;
```

```
        }
        else
        {
            shortest_path[i].pred = x;
            shortest_path[i].label = graph[i][0].label;
            shortest_path[i].cost = max;
        }
    }
    for (int i = 0; i < n; i++)
    {
        cout << y << endl;
        int s = graph[y].size();
        visited[y] = true;
        for (int j = 1; j < s; j++)
        {
            int c = findNode(graph[y][j].label);
            if (visited[c] == false)
                if (graph[y][j].cost + shortest_path[y].cost
                        < shortest_path[c].cost)
                {
                    shortest_path[c].cost = graph[y][j].cost
                            + shortest_path[y].cost;
                    shortest_path[c].pred = graph[y][0].label;
                }
        }
        int min = max;
        int minpos = -1;
        for (int j = 0; j < n; j++)
        {
            if (shortest_path[j].cost < min &&
                    visited[j]==false)
            {
                min = shortest_path[j].cost;
                minpos = j;
            }
        }
        if (minpos == -1) break;
        y = minpos;
    }
```

```
        for (int i = 0; i < n; i++)
        {
            cout <<" "<<shortest_path[i].label<< " - "
                    <<shortest_path[i].pred;
            if (shortest_path[i].cost == max)cout<< " - inf \n";
            else cout<<" - " <<shortest_path[i].cost<< endl;
        }
        return 0;
    }

};

#endif
```

dijkstra.cpp

```
#include"dijkstra.h"

int main()
{

    cout<<"\n\n Dijkstra Algorithm \n\n";

    Graph G;
    string source;
    G.display();
    cout<<"\n\n Enter Source Vertex : ";
    cin>>source;
    cout<<"\n\n";
    G.dijkstra(source);
    cout<<"\n\n";
    cout<<"\n\n The End \n\n";
    return 0;
}
```

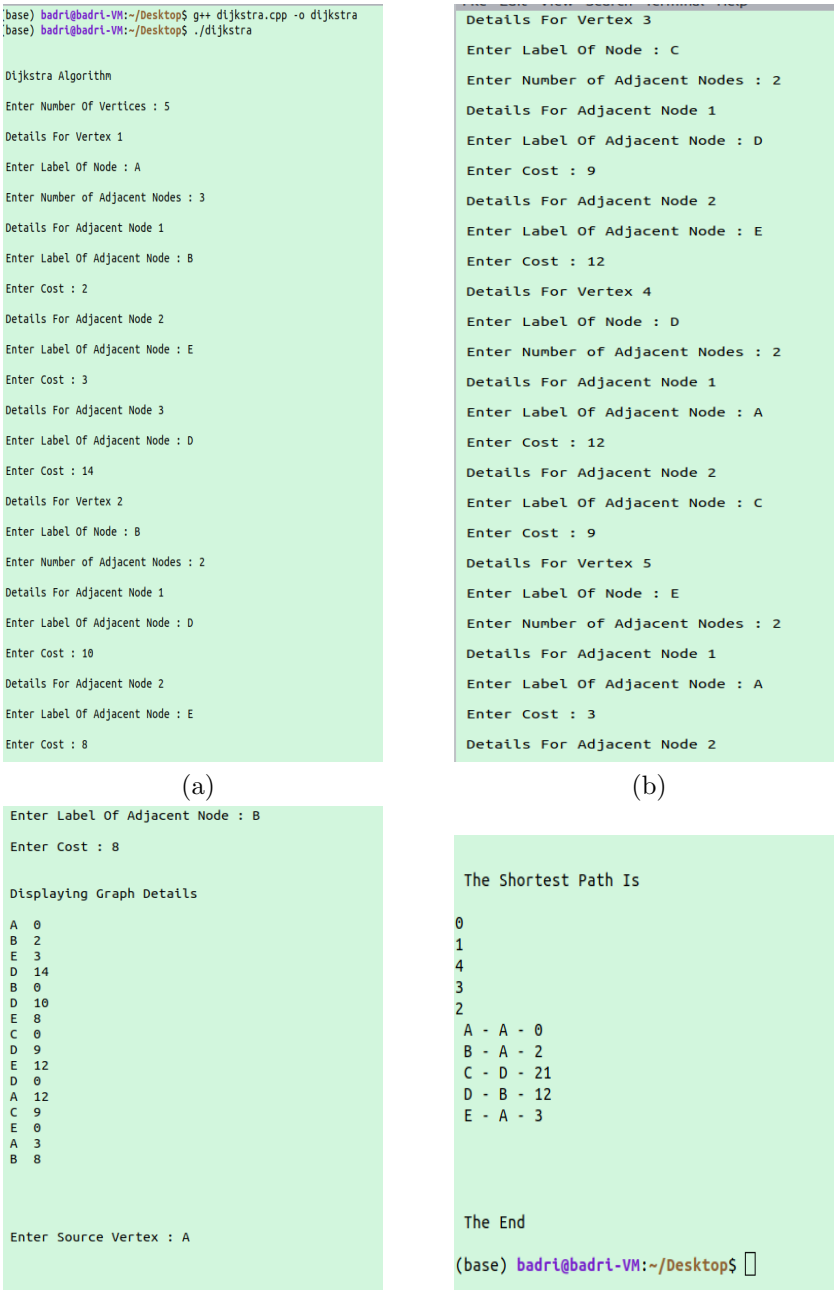


Figure 12: Output For XII Dijkstra

Result A C++ program was created to implement Single Source Shortest Path Dijkstra Algorithm.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
Register No: 185002018

EXP NO: 12b

DATE: 11

Single Source Shortest Path Algorithm - Bellman Ford Algorithm

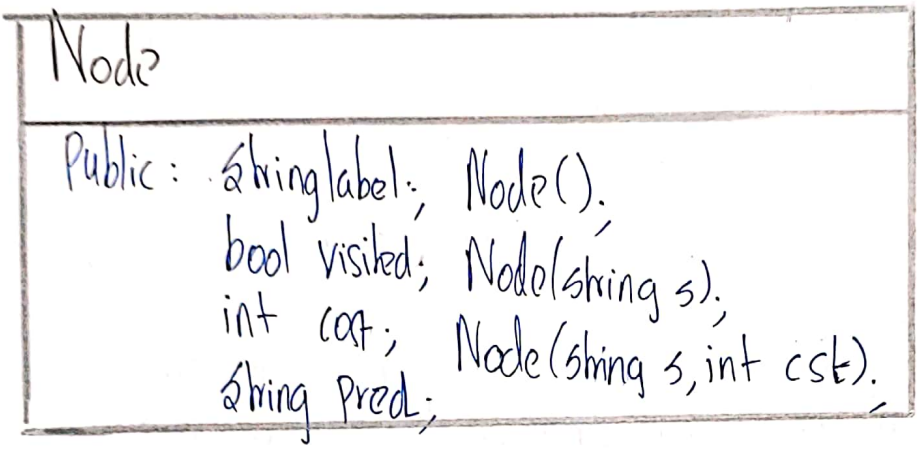
Aim: To create a C++ program to implement single source shortest path bellman ford's algorithm

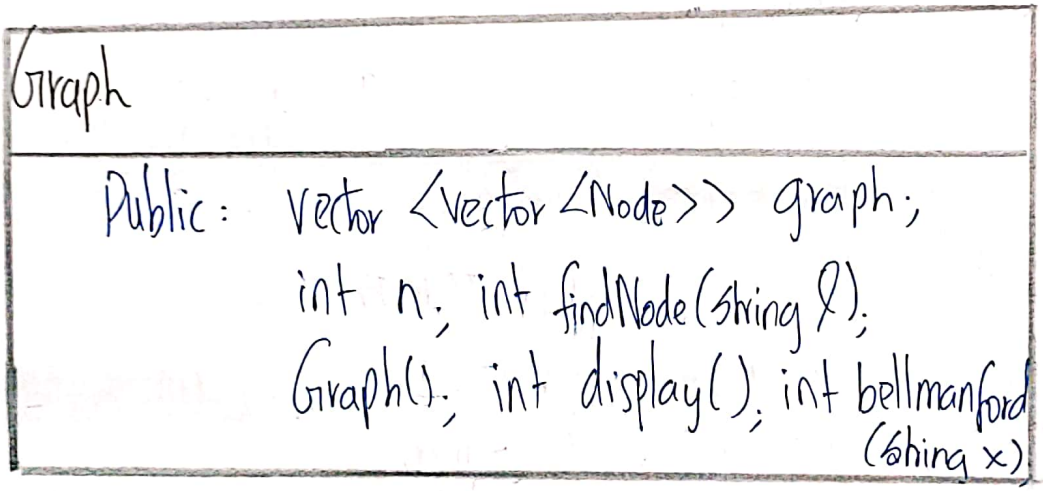
Time Complexity

max = 99 n1 : No of Nodes/Vertices
n2 : No of Adj Nodes

SNO	Function	Operation	Time Complexity
1	Node()	Empty Constructor	O(1)
2	Node(string s) Constructor	Initializes label to s, cost to max, visited to false	O(1)
3	Node(string s, int cst) Constructor	Initializes label to s, Cost to cst, Visited to false	O(1)
4	Graph() Constructor	Graph is created with n Vertices	O(n1 + n2)

Class Diagram





SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

M Badri Narayanan
185002018

SNO	Function	Operation	Time Complexity
5	int display()	Displays Graph details	O(n1+n2)
6	int bellmanford (string s)	Performs Bellman Ford Algorithm on Graph with node passed as parameter as source	O(VE) V: No of Vertices E: No of Edges
7	int findNode(string n)	Refer Dijkstra	O(n1)

bellmanford.h

```
#ifndef bellmanford_h
#define bellmanford_h

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

#define max 99

class Node
{
    public:
    string label;
    bool visited;
    int cost;
    string pred;
    Node()
    {

    }

    Node(string s)
    {
        label = s;
        visited = false;
        cost = max;
    }

    Node(string s, int cst)
    {
        label = s;
        visited = false;
        cost = cst;
    }

};
```

```
class Graph
{
    public:
            vector< vector<Node> > graph;
            int n;

    Graph()
    {
        string s;
        cout <<" Enter Number Of Vertices : ";
        cin >>n;
        string t;
        for (int i = 0; i < n; i++)
        {
            vector<Node> singleList;
            cout<<"\n Details For Vertex "<<i + 1 <<"\n";
            cout <<"\n Enter Label Of Node : ";
            cin >>s;
            singleList.push_back(Node(s, 0));
            cout <<"\n Enter Number of Adjacent Nodes : ";
            int n2, cst;
            cin >>n2;
            for (int j = 0; j < n2; j++)
            {
                cout<<"\n Details For Adjacent Node "
                    <<j + 1<<"\n";
                cout <<"\n Enter Label Of Adjacent Node : ";
                cin>>s;
                cout<<"\n Enter Cost : ";
                cin>>cst;
                singleList.push_back(Node(s, cst));
            }
            graph.push_back(singleList);
        }
    }

    int display()
    {
        cout<<"\n\n Displaying Graph Details \n\n";
        for (int i = 0; i < n; i++)
        {
```

```
            int s = graph[i].size();
            for (int j = 0; j < s; j++)
            {
                cout <<" "<<graph[i][j].label<<"   "
                    << graph[i][j].cost<<"\n";
            }
        }
        cout<<"\n\n";
        return 0;
}

int findNode(string l)
{
        for (int i = 0; i < n; i++)
        {
            if (graph[i][0].label == l)return i;
        }
        return -1;
}

int bellmanford(string x)
{
        int y = findNode(x);
        for (int i=0; i<n; i++)
        {
            if (i==y)
            {
                graph[i][0].cost = 0;

            }
            else
            {
                graph[i][0].cost = max;
            }
        }
        for(int k=0;k<n-1;k++)
        {
            for (int i=0; i<n; i++)
            {
                int s = graph[i].size();
                for (int j=1; j<s; j++)
```

```
                {
                    int c = findNode(graph[i][j].label);
                    if(graph[i][j].cost+graph[i][0].cost
                            < graph[c][0].cost)
                    {
                        graph[c][0].cost = graph[i][j].cost
                                + graph[i][0].cost;
                        graph[c][0].pred = graph[i][0].label;
                    }
                }
            }
        }
        for (int i=0; i<n; i++)
        {
            cout<<" "<<graph[i][0].label<<" - "<<
                    graph[i][0].cost<<endl;
        }
        cout<<endl<<endl;
    }

};

#endif
```

bellmanford.cpp

```
#include"bellmanford.h"

int main()
{

    cout<<"\n\n Bellman Ford Algorithm \n\n";

    Graph G;
    string source;
    G.display();
    cout<<"\n\n Enter Source Vertex : ";
    cin>>source;
    cout<<"\n\n";
    G.bellmanford(source);
    cout<<"\n\n";
```

```
    cout<<"\n\n The End \n\n";
    return 0;
}
```

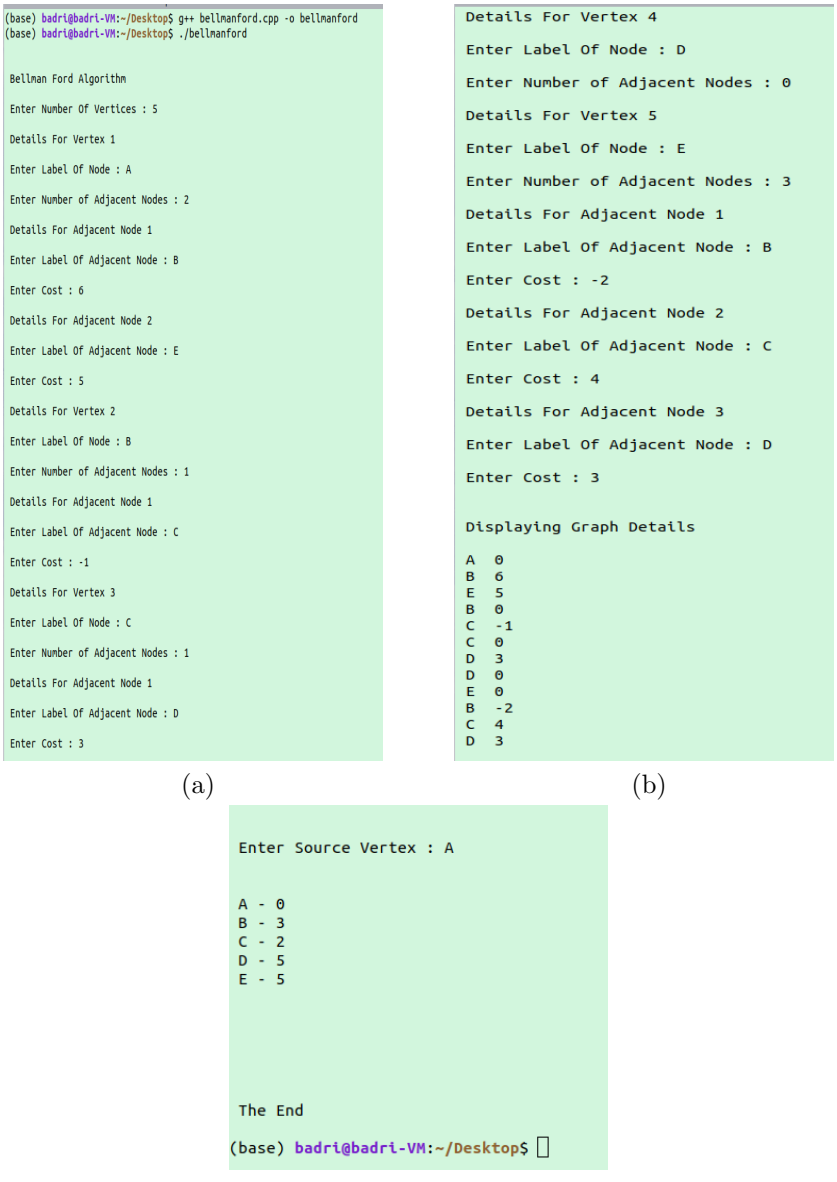


Figure 13: Output For XII Bellman Ford

Result A C++ program was implemented to find the single source shortest using Bellman Ford Algorithm.

SSN COLLEGE OF ENGINEERING
RECORD SHEET
Sheet No.....

EXP NO: 12
DATE:

M Badri Narayanan
185002018

All Pairs Shortest Path Algorithm - Floyd & Warshall Algorithm

Aim: To create a C++ program to implement all pairs shortest path algorithm - Floyd & Warshall Algorithm

Time Complexity

n: No of Vertices

SNO	Function	Operation	Time Complexity
1	Graph() Constructor	Gets the value for no of vertices & allocates mat to n×n size	O(n)
2	int weightmatrix	Creates the weight matrix	$O(n^2)$
3	int adjacencymatrix	Creates the adjacency matrix	$O(n^2)$
4	int warshall()	Finds Transitive Closure using Warshall's Algorithm	$O(n^3)$
5	int floyd()	Performs Floyd Algorithm	$O(n^3)$

Class Diagram

Graph
Public : int n; Graph(); int adjacencymatrix(); int **mat; int weightmatrix(); int floyd(); int warshall();

floydandwarshall.h

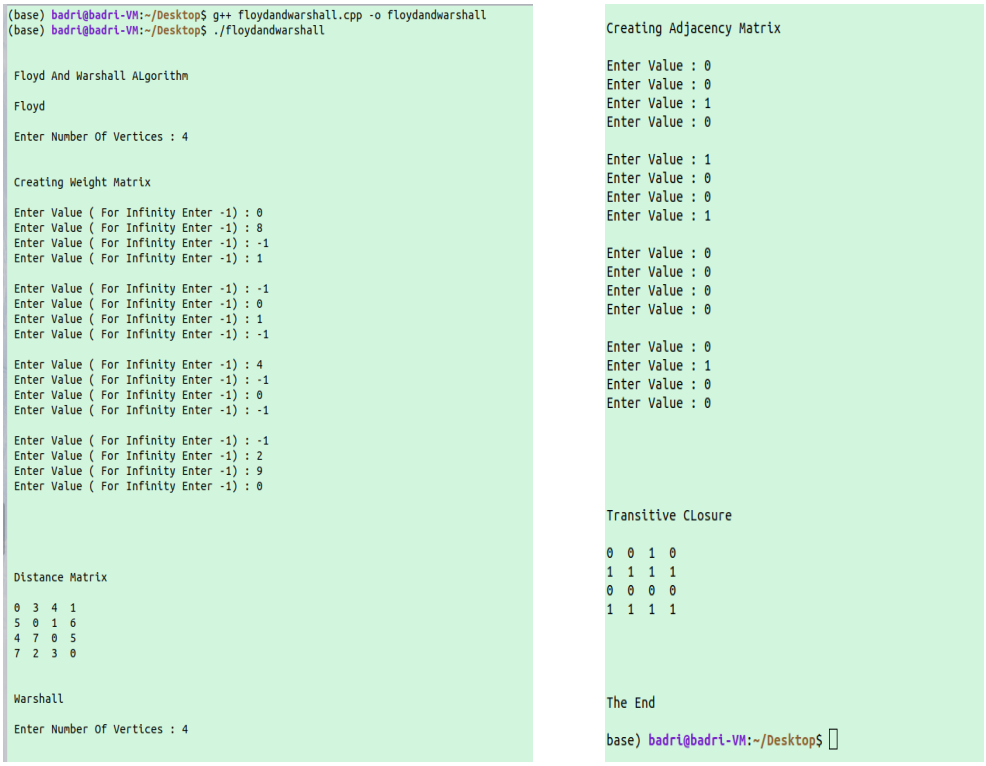
```
#ifndef floydandwarshall_h
#define floydandwarshall_h

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

#define inf 10000

class Graph
{
    public:

    int      n;
    int      **mat;

    Graph()
    {
        cout<<" Enter Number Of Vertices : ";
        cin>>n;
        mat = new int*[n];
        for(int i = 0; i < n; i++)
        {
            mat[i] = new int[n];
        }
    }
    int weightmatrix()
    {
        cout<<"\n\n Creating Weight Matrix \n\n";
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < n; j++)
            {
                cout<<" Enter Value ( For Infinity
                                    Enter -1) : ";
                cin>>mat[i][j];
                if(mat[i][j] == -1)mat[i][j] = inf;
            }
```

```
            cout<<"\n";
        }
        cout<<"\n\n";
        return 0;
    }
    int adjacencymatrix()
    {
        cout<<"\n\n Creating Adjacency Matrix \n\n";
        for(int i = 0; i < n; i++)
        {
            for(int j = 0; j < n; j++)
            {
                cout<<" Enter Value : ";
                cin>>mat[i][j];
            }
            cout<<"\n";
        }
        cout<<"\n\n";
        return 0;
    }
    int warshall()
    {
        int i,j,k;

        for(k = 0; k < n; k++)
        {
            for(j = 0; j < n; j++)
            {
                for(i = 0; i < n; i++)
                {
                    mat[i][j] = mat[i][j] | (mat[i][k] &

                    mat[k][j]);
                }
            }
        }
```

```
        cout<<"\n\n Transitive Closure \n\n";
        for(i = 0; i < n; i++)
        {
            for(j = 0; j < n; j++)
            {
                cout<<" "<<mat[i][j]<<" ";
            }
            cout<<endl;
        }
        cout<<"\n\n";
        return 0;
    }
    int floyd()
    {
        int k, j, i;
        for(k = 0; k < n; k++)
        {
            for(j = 0; j < n; j++)
            {
                for(i = 0; i < n; i++)
                {
                    mat[i][j] = min(mat[i][j], mat[i][k] +

                    mat[k][j]);
                }
            }
        }
        cout<<"\n\n Distance Matrix \n\n";
        for(i = 0; i < n; i++)
        {
            for(j = 0; j < n; j++)
            {
                cout<<" "<<mat[i][j]<<" ";
            }
            cout<<endl;
        }
         cout<<"\n\n";
        return 0;
    }
};
#endif
```

floydandwarshall.cpp

```
#include"floydandwarshall.h"
int main()
{

    cout<<"\n\n Floyd And Warshall Algorithm \n\n";
    cout<<" Floyd \n\n";
    Graph G;
    G.weightmatrix();
    G.floyd();
    cout<<" Warshall \n\n";
    Graph graph;
    graph.adjacencymatrix();
    graph.warshall();
    cout<<"\n\n The End \n\n";
    return 0;
}
```



(a) (b)

Figure 14: Output For XIII Floyd & Warshall Algorithm

Result A C++ program was created to implement All Pairs Shortest Algorithm - Floyd & Warshall Algorithms.

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No.....

EXP NO: 14

M Badri Narayanan
185002018

DATE: 11

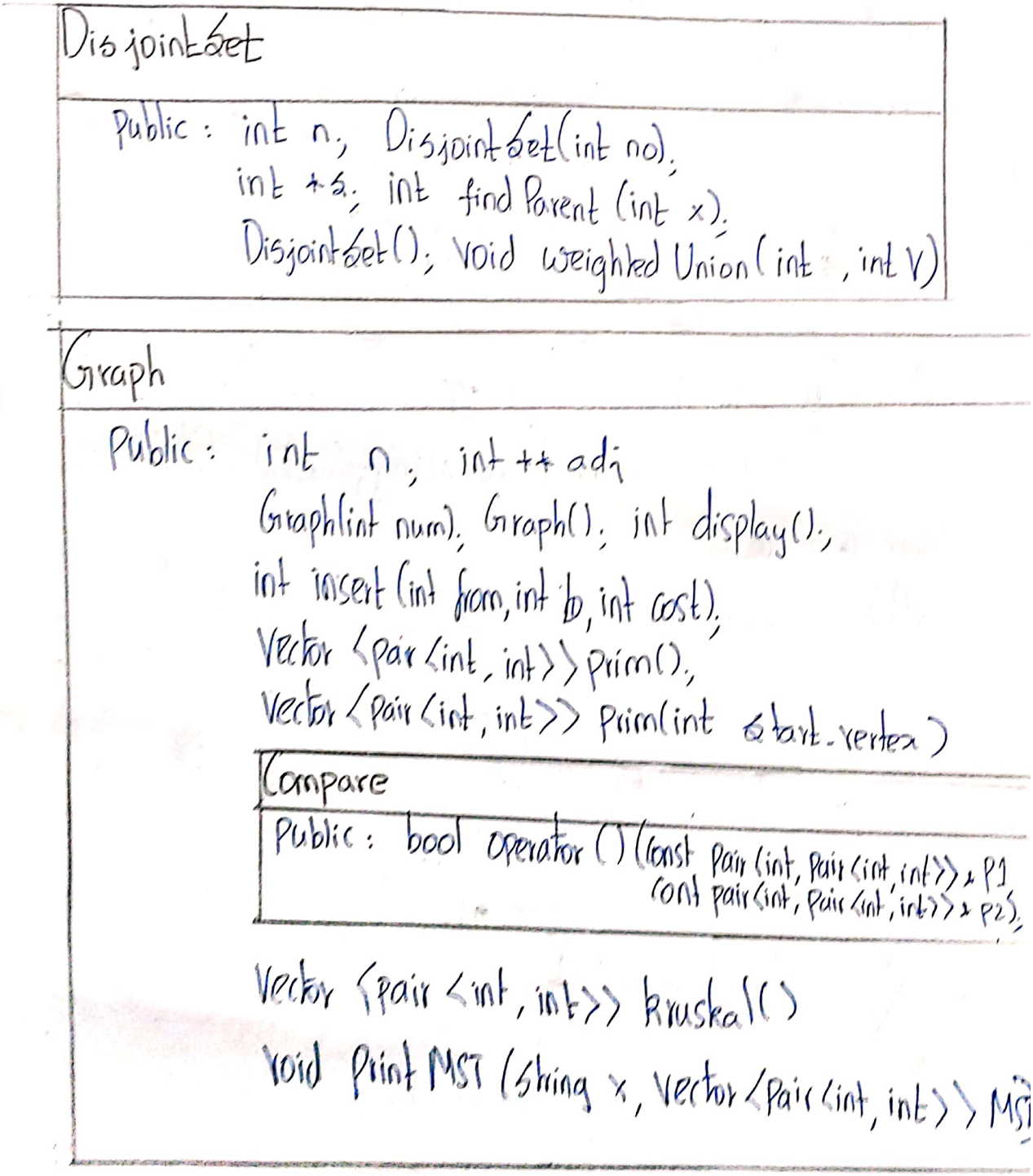
Minimum Spanning Tree Implementation - Kruskal's & Prim's Algorithm

Aim: To create a C++ program to implement Minimum Spanning Tree using Kruskal & Prim's Algorithm

Time Complexity MAX : 1000 00000 n: No. of Vertices

SNO	Function	Operation	Time Complexity
1	DisjointSet()	Empty Constructor	O(1)
2	DisjointSet(int no)	Assigns n to no+1 & allocates & a size of n & initialize to -1.	O(no+1) = O(n)
2	int findParent(int x)	Finds Parent of node specified in parameter	O(n)

Class Diagram



SSN COLLEGE OF ENGINEERING 18500218
RECORD SHEET Sheet No.....
M Badri Narayanan

SNO	Function	Time Complexity & Operation	
4	Void weighted Union(int u, int v)	Does the Union of the two nodes specified in parameter	O(n)
5	Graph( ) Constructor	Calls Graph(1000)	O(1000)
6	Graph (int num)	Assigns n to numV Allocates memory to adj of size n x n & assigns each element of adj to MAX	O(n)
7	int insert (int from, int to, int cost)	Assigns adj[from][to] to cost	O(1)
8	int display( )	Displays the graph details	O(n²)

M Badri Narayanan

SSN COLLEGE OF ENGINEERING
RECORD SHEET

Sheet No. 18500 2018

S.NO	Function	Time Complexity & Operation
9	Vector<pair<int,int>> Prim()	O(ElogV) E: No of edges V: No of Vertices — Finds the MST of given graph using Prim's Algorithm
10	Vector<pair<int,int>> Prim(int start_vertex)	O(ElogV), Finds the MST of given graph using Prim's Algorithm
11	bool operator() (const pair<int, pair<int,int>> *p1, const pair<int, pair<int,int>> *p2);	Returns 1 if p1.first > p2.first else 0 — O(1)
12	Vector<pair<int,int>> Kruskal()	Finds MST of given graph using Kruskal Algorithm — O(ElogV)
12	Void Print MST (String x, Vector<pair<int,int>> MST)	Prints the MST — O(ElogV)

primandkruskal.h

```
#ifndef primandkruskal_h
#define primandkruskal_h

#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
#include <queue>
#include <stack>
using namespace std;

#define MAX 100000000

class DisjointSet
{
    public:

    int      n;
    int     *s;

    DisjointSet(){}
    DisjointSet(int no)
    {
        n = no+1;
        s = new int[n];
        for (int i=0; i<n; i++)
        {
            s[i] = -1;
        }
    }
    int findParent(int x)
    {
        while (s[x]>=0)
        {
            x = s[x];
        }
        return x;
    }
```

```
    void weightedUnion(int u, int v)
    {
        u = findParent(u);
        v = findParent(v);
        if (s[u]<s[v])
        {
            s[u]+=s[v];
            s[v] = u;
        }
        else
        {
            s[v]+=s[u];
            s[u] = v;
        }
    }
};

class Graph
{
    public:
            int      n;
            int     **adj;

            Graph(int num)
            {
                n = num+1;
                adj = new int *[n];
                for (int i=0; i<n; i++)
                {
                    adj[i] = new int[n];
                }
                for (int i=0; i<n; i++)
                {
                    for (int j=0; j<n; j++)
                    {
                        adj[i][j] = MAX;
                    }
                }
            }
```

```
Graph()
{
    Graph(1000);
}
int insert(int from, int to, int cost)
{
    adj[from][to] = cost;
    return 0;
}
int display()
{
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j++)
        {
            if (adj[i][j]!=MAX)
            {
                cout<<" "<<i<<" - "<<j<<" -

                    "<<adj[i][j]<<endl;
            }
        }
        cout<<"\n\n";
    }
    return 0;
}
vector< pair<int, int> > prim()
{
    int near[n];
    for (int i=0; i<n; i++)
    {
        near[i] = MAX;
    }
    vector< pair<int, int> > MST;
    int min = MAX;
    int mini, minj;
    for (int i=1; i<n; i++)
    {
        for (int j=i; j<n; j++)
        {
            if (adj[i][j]<min)
```

```
            {
                min = adj[i][j];
                mini=i;
                minj=j;
            }
        }
    }
    MST.push_back(make_pair(mini, minj));
    near[mini]=near[minj] =0;
    for (int i=1; i<n; i++)
    {
        if (near[i]!=0)
        {
            near[i] = (adj[mini][i]<adj[minj][i])?
                    mini:minj;
        }
    }
    for (int z= 1; z<n-2; z++)
    {
        int k = -1;
        min = MAX;
        for (int i=1; i<n; i++)
        {
            if (near[i]!=0)
            {
                if (adj[i][near[i]]<min)
                {
                    min=adj[i][near[i]];
                    k = i;
                }
            }
        }
        MST.push_back(make_pair(k, near[k]));
        near[k] = 0;
        for (int i=1; i<n; i++)
        {
            if (near[i]!=0)
            {
                if (adj[i][k]<adj[i][near[i]])
                {
                    near[i] = k;
```

```
                }
            }
        }
    }
    return MST;
}
vector<pair<int, int>> prim(int start_vertex)
{
    int near[n];
    for (int i=0; i<n; i++)
    {
        near[i] = MAX;
    }
    vector< pair<int, int>> MST;
    int min = MAX;
    int mini = start_vertex, minj;
    for (int i=1; i<n; i++)
    {
        if (adj[mini][i]<min)
        {
            min = adj[mini][i];
            minj = i;
        }
    }
    MST.push_back(make_pair(mini, minj));
    near[mini]=near[minj] =0;
    for (int i=1; i<n; i++)
    {
        if (near[i]!=0)
        {
            near[i] = (adj[mini][i]<adj[minj][i])
                    ?mini:minj;
        }
    }
    for (int z= 1; z<n-2; z++)
    {
        int k = -1;
        min = MAX;
        for (int i=1; i<n; i++)
        {
            if (near[i]!=0)
```

```
                {
                    if (adj[i][near[i]]<min)
                    {
                        min=adj[i][near[i]];
                        k = i;
                    }
                }
            }
            MST.push_back(make_pair(k, near[k]));
            near[k] = 0;
            for (int i=1; i<n; i++)
            {
                if (near[i]!=0)
                    if (adj[i][k]<adj[i][near[i]])
                        near[i] = k;
            }
        }
        return MST;
}
class compare
{
    public:
    bool operator()(const pair<int, pair<int, int>>
            &p1, const pair<int,
                        pair<int, int>> &p2)
    {
        return p1.first>p2.first;
    }
};
vector< pair<int, int>> kruskal()
{
    priority_queue<pair<int, pair<int, int>> ,
            vector<pair<int, pair<int, int>> >,
                        compare > pq;

    vector<pair<int, int>> MST;
    for (int i=1; i<n; i++)
    {
        for (int j=i; j<n; j++)
        {
            if (adj[i][j]!=MAX)
```

```
                    {
                        pq.push(make_pair(adj[i][j],
                            make_pair(i, j)));
                    }
                }
            }
            DisjointSet d(n);
            pair<int, int> p;
            while (pq.size())
            {
                p = (pq.top()).second; pq.pop();
                if (d.findParent(p.first)!=d.findParent(
                                    p.second))
                {
                    MST.push_back(p);
                    d.weightedUnion(p.first, p.second);
                }
            }
            return MST;
        }

        void printMST(string x, vector<pair<int, int>> MST)
        {
            cout<<" ------------------------------------\n\n\n";
            cout<<" "<<x<<" 's MST : "<<endl;
            cout<<" From \t To \t Cost "<<endl;
            for (auto edge:MST)
            {
                cout<<" "<<edge.first<<" - "<<edge.second
                <<"- "<<adj[edge.first][edge.second]<<endl;
            }
            cout<<" ------------------------------------\n\n\n";
        }

};

#endif
```

primandkruskal.cpp

```
#include"primandkruskal.h"

int main()
{
    int from, to, x, n, cost, edgecount;
    int option;
    do
    {

        cout <<"\n\n Prims & Kruskal Algorithms \n\n";
        cout<<" 1. Prims \n";
        cout<<" 2. Kruskal \n";
        cout<<"\n Enter Number Of Vertices : ";
        cin >>n;
        cout << "\n Enter Number Of Edges : ";
        cin >>edgecount;
        Graph g(n);
        for (int i = 0; i < edgecount; i++)
        {
            cout<<"\n Details For Edge "<<i+1<<"\n";
            cout<<"\n Enter From : ";
            cin>>from;
            cout<<"\n Enter To : ";
            cin>>to;
            cout<<"\n Enter Cost : ";
            cin>>cost;
            g.insert(from, to, cost);
            g.insert(to, from, cost);
        }
        auto MST = g.kruskal();
        g.printMST(" Kruskal ", MST);
        MST = g.prim();
        g.printMST(" Prims ", MST);
        cout<<"\n\n Prims Algorithm With A Starting Vertex \n\n";
        cout<<"\n Enter Starting Vertex : ";
        cin>>x;
        MST = g.prim(x);
        g.printMST(" Prims ", MST);
        cout<<" Do You Want To Continue (Type 0 Or 1 ) : ";
```

```
        cin>>option;
    }while(option == 1);
    cout<<"\n\n The End \n\n";
    return 0;
}
```

```
(base) badri@badri-VM:~/Desktop$ g++ primandkruskal.cpp -o primandkruskal
(base) badri@badri-VM:~/Desktop$ ./primandkruskal


Prims & Kruskal Algorithms

1. Prims
2. Kruskal

Enter Number Of Vertices : 12

Enter Number Of Edges : 20

Details For Edge 1

Enter From : 1

Enter To : 2

Enter Cost : 3

Details For Edge 2

Enter From : 2

Enter To : 6

Enter Cost : 6

Details For Edge 3

Enter From : 6

Enter To : 10

Enter Cost : 5

Details For Edge 4

Enter From : 10

Enter To : 12

Enter Cost : 9

Details For Edge 5
```

(a)

```
Enter From : 12
Enter To : 11
Enter Cost : 8
Details For Edge 6
Enter From : 11
Enter To : 7
Enter Cost : 6
Details For Edge 7
Enter From : 7
Enter To : 3
Enter Cost : 4
Details For Edge 8
Enter From : 1
Enter To : 3
Enter Cost : 5
Details For Edge 9
Enter From : 3
Enter To : 4
Enter Cost : 2
Details For Edge 10
Enter From : 4
Enter To : 5
Enter Cost : 1
Details For Edge 11
```

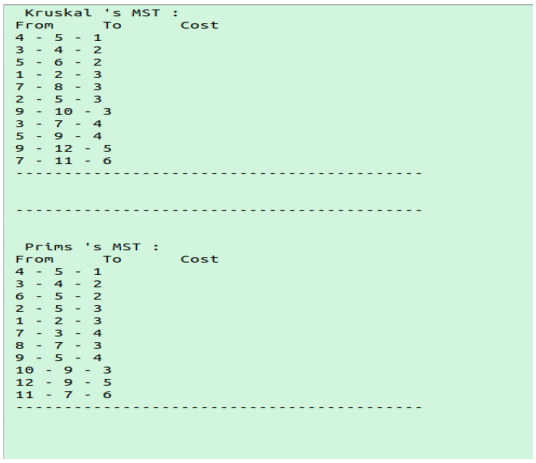
(b)

```
Enter From : 5
Enter To : 6
Enter Cost : 2
Details For Edge 12
Enter From : 7
Enter To : 8
Enter Cost : 3
Details For Edge 13
Enter From : 8
Enter To : 9
Enter Cost : 6
Details For Edge 14
Enter From : 9
Enter To : 10
Enter Cost : 3
Details For Edge 15
Enter From : 1
Enter To : 4
Enter Cost : 4
Details For Edge 16
Enter From : 2
Enter To : 5
```

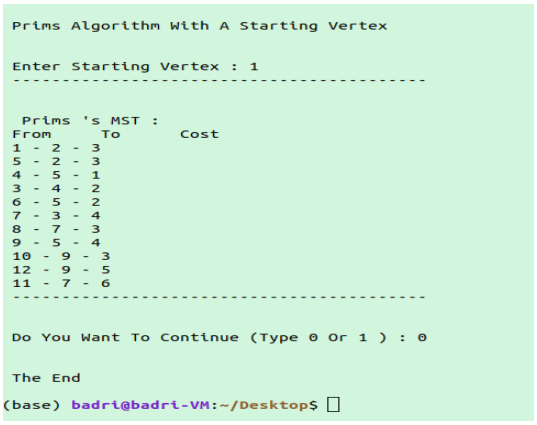
(c)

```
Enter Cost : 3
Details For Edge 17
Enter From : 4
Enter To : 8
Enter Cost : 5
Details For Edge 18
Enter From : 5
Enter To : 9
Enter Cost : 4
Details For Edge 19
Enter From : 8
Enter To : 11
Enter Cost : 7
Details For Edge 20
Enter From : 9
Enter To : 12
Enter Cost : 5
----------------------------------------
```

(d)



(e)



(f)

Figure 15: Output For XIIV Prim's & Kruskal Algorithm

Result A C++ program was created to implement Minimum Spanning Tree using Prim's & Kruskal Algorithm.