

DISCRETE MATHEMATICS FILE



Roll no-2019UCO1580

Name -HARSHIT GUPTA

Branch-COE

Section- 1

Course-CECSC01

INDEX

S. No	Name	Page Start	Page End
1.	Wap to perform operations on sets	2	6
2.	Wap to perform operations on multisets	6	12
3.	Wap to check whether given expression in tautology, contradiction or contingency.	12	19
4.	Wap to check equivalence of given relation	19	26
5.	Wap to generate next lexicographic permutation	26	28
6.	Wap to generate next lexicographic combination	29	31
7.	WAP for checking given poset is a lattice or not	32	38
8.	WAP for specifying principle of inclusion and exclusion.	38	41
9.	WAP for solving the input recurrence relation given by user.	41	46
10.	WAP for checking whether the given input is a group or not.	47	53
11.	WAP for checking whether the given input is a ring or not.	53	61
12.	WAP for finding path and cycles in any given graph	61	73

PROGRAM 1:

WAP for addition, union, intersection, difference, symmetric difference etc. of two sets.

The following code prompts the user to enter two sets 1 and 2 and then their corresponding elements . The user is then asked to select one of the four operations addition, intersection, union and symmetric difference. The stl containers set is used for the set inputs.

- **CODE IN C++**

```
#include<bits/stdc++.h>
#define tr(v,it) for(typeof(v.begin()) it=v.begin();it!=v.end();it++)
#define FOR(i,n) for(int i=0;i!=n;i++)
using namespace std;
void add(set<int> a, set<int> b)
{
    set<int> set3;
    cout<<"\nAddition is:";
    tr(a,it1)
    {
        tr(b,it2)
        {
            set3.insert(*it1+*it2);
        }
    }
    tr(set3,it)
    cout<<*it<<" ";
}
void subtract(set<int> a,set<int> b)
{
    cout<<"\nThe difference of sets is:";
    tr(a,it1)
    { if(b.find(*it1)!=b.end())
        cout<<*it1<<" ";
    else
        continue;
    }
}
void Union(set<int> a, set<int> b)
```

```

{
    cout<<"\nThe union is:";
    tr(a,it)
        cout<<*it<<" ";
    tr(b,it)
        {
            if(a.find(*it)==a.end())
                cout<<*it<<" ";
            else
                continue;
        }
}

void intersect(set<int> a, set<int> b)
{   cout<<"\nIntersection of sets is:";
    int flag=0;
    tr(a,it)
        {
            if(b.find(*it)!=b.end())
                {
                    cout<<*it<<" ";
                    flag=1;}
        }
    if(flag==0)
        cout<<"\nNULL SET.";
}

void symmetric(set<int> a,set<int> b)
{   cout<<"\nSymmetric dfference of sets is:";
    tr(a,it)
        {
            if(b.find(*it)!=b.end())
                continue;
            else
                cout<<*it<<" ";
        }
    tr(b,it)
        {
            if(a.find(*it)!=a.end())

```



```

cin>>ch;
if(ch==-1)
    break;
switch(ch)
{
    case 1:{ add(set1,set2);
        break;
    }
    case 2:{cout<<"\n Press 1 for SET1-SET2\n Press 2 for SET2-SET1";
        x:cout<<"\nEnter choice:";
        int choice;
        cin>>choice;
        if(choice==1) subtract(set1,set2);
        else if (choice==2) subtract(set2,set1);
        else
        { cout<<"\nWrong choice.\nEnter again.";
            goto x;
        }
        break;
    }
    case 3:{
        Union(set1,set2);
        break;
    }
    case 4:{
        intersect(set1,set2);
        break;
    }
    case 5:{ symmetric(set1,set2);
        break;
    }
    default:cout<<"\nWrong choice.\nEnter again." ;
        break;

}

}

```

```

return 0;
}

```

- **OUTPUT 1**

```

C:\Users\harsh\Documents\operationsonset.exe
=====
OPERATIONS ON SETS.
=====
Enter size of SET1:4
Enter elements:1
2
3
4
Enter size of SET2:5
Enter elements:2
3
7
8
9
SET 1:1,2,3,4,
SET 2:2,3,7,8,9,
Operations...
ADDITION-1 , DIFFERENCE-2 , UNION-3 , INTERSECTION-4 ,SYMMETRIC DIFFERENCE-5
Enter the choice (-1 for exit):1
Addition is:3,4,5,6,7,8,9,10,11,12,13,
Enter the choice (-1 for exit):2
Press 1 for SET1-SET2
Press 2 for SET2-SET1

```

- **OUTPUT 2**

```

8
9
SET 1:1,2,3,4,
SET 2:2,3,7,8,9,
Operations...
ADDITION-1 , DIFFERENCE-2 , UNION-3 , INTERSECTION-4 ,SYMMETRIC DIFFERENCE-5
Enter the choice (-1 for exit):1
Addition is:3,4,5,6,7,8,9,10,11,12,13,
Enter the choice (-1 for exit):2
Press 1 for SET1-SET2
Press 2 for SET2-SET1
Enter choice:1
The difference of sets is:1,4,
Enter the choice (-1 for exit):3
The union is:1,2,3,4,7,8,9,
Enter the choice (-1 for exit):4
Intersection of sets is:2,3,
Enter the choice (-1 for exit):5
Symmetric dfference of sets is:1,4,7,8,9,
Enter the choice (-1 for exit):-1
-----
Process exited after 127.5 seconds with return value 0
Press any key to continue . . .

```

PROGRAM 2:

WAP for all operations on given multisets.

The following code prompts the user to enter two multisets which are taken as input in the form of vector of strings and then the user is prompted to enter one of the four operations addition, subtraction, union and intersection.

- **CODE IN C++**

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

```

#define tr(v,it) for(typeof(v.begin()) it=v.begin();it!=v.end();it++)
#define FOR(i,n) for(int i=0;i!=n;i++)
using namespace std;
void add(map <string,int> a, map<string,int> b)
{   cout<<"\nThe addition of multisets is:";
    tr(a,it1) // to print the common and non-common values in a that are not in b
    {
        map <string,int>::iterator it2 = b.find(it1->first);
        if(it2!=b.end())
            {FOR(i,it1->second+it2->second)
                cout<<(it1->first)<<" ";
            }
        else
        {
            FOR(i,it1->second)
                cout<<it1->first<<" ";
        }
    }
    tr(b,it1) // to find the values in b that are not in a
    {
        if(a.find(it1->first)==a.end())
        {
            FOR(i,it1->second)
                cout<<it1->first<<" ";
        }
    }
}

void subtract(map <string,int> a, map<string,int> b)
{   int flag=0;
    cout<<"\nThe difference of sets is:";
    tr(a,it1) // finding if the element of a present in b or not.
    { if(b.find(it1->first)!=b.end())
        {flag=1;
            FOR(i,it1->second)
                cout<<it1->first<<" ";
        }
    }
    else
    {
        map<string,int>::iterator it2 = b.find(it1->first);
        int diff= (it1->second)-(it2->second);
    }
}

```



```

    if(diff>0)
    {   flag=1;
        FOR(i,diff)
            cout<<(it1->first)<<" ";}
    else
        continue;
}
}
if(flag==0)
    cout<<"NULL SET";
}
void Union(map<string,int> a, map<string,int> b)
{
    cout<<"\nThe union is:";
    tr(a,it1)
    {
        if(b.find(it1->first)==b.end())
        {
            FOR(i,it1->second)
                cout<<it1->first<<" ";//printing elements of a which are not common with b
        }
        else
        {
            map<string,int>::iterator it2=b.find(it1->first);
            if(it1->second>=it2->second)
            {
                FOR(i,it1->second)
                    cout<<it1->first<<" ";// printing common elements
            }
            else
            {
                FOR(i,it2->second)
                    cout<<it2->first<<" ";
            }
        }
    }
}
tr(b,it2)
{
    if(a.find(it2->first)==a.end())
    {

```



```

cout<<"\nEnter elements:";
FOR(i,n1)
{  cin>>ele;
   set1.push_back(ele);
   m1[ele]+=1;
}
cout<<"\nEnter size of MULTISSET2:";
cin>>n2;
cout<<"\nEnter elements:";
FOR(i,n2)
{  cin>>ele;
   set2.push_back(ele);
   m2[ele]+=1;
}
cout<<"\nMULTISSET 1:";
tr(set1,it)
    cout<<*it<<" ";
cout<<"\nMULTISSET 2:";
tr(set2,it)
    cout<<*it<<" ";
cout<<"\nOperations...";
cout<<"\nADDITION-1 , DIFFERENCE-2 , UNION-3 , INTERSECTION-4 ";
while(true)
{cout<<"\nEnter the choice (-1 for exit):";
cin>>ch;
if(ch== -1)
    break;
switch(ch)
{
    case 1:{ add(m1,m2);
            break;
          }
    case 2:{cout<<"\n Press 1 for SET1-SET2\n Press 2 for SET2-SET1";
            x:cout<<"\nEnter choice:";
            int choice;
            cin>>choice;
            if(choice==1) subtract(m1,m2);
            else if (choice==2) subtract(m2,m1);
            else
            { cout<<"\nWrong choice.\nEnter again.";

```

```

        goto x;
    }
    break;
}
case 3:{
    Union(m1,m2);
    break;
}
case 4:{
    intersect(m1,m2);
    break;
}
default:cout<<"\nWrong choice.\nEnter again." ;
    break;

}

}
return 0;
}

```

• OUTPUT 1



```

C:\Users\harsh\Documents\multiset.exe
=====
OPERATIONS ON MULTI-SETS.
=====
Enter size of MULTISSET1:5
Enter elements:a
b
c
d
e
Enter size of MULTISSET2:7
Enter elements:a
a
b
c
e
f
d
MULTISSET 1:a,b,c,d,e,
MULTISSET 2:a,a,b,c,e,f,d,
Operations...
ADDITION-1 , DIFFERENCE-2 , UNION-3 , INTERSECTION-4
Enter the choice (-1 for exit):1
The addition of multisets is:a a b b c c d d e e f
Enter the choice (-1 for exit):2

```

• OUTPUT 2

```
C:\Users\harsh\Documents\multiset.exe
The addition of multisets is:a a b b c c d d e e f
Enter the choice (-1 for exit):2

Press 1 for SET1-SET2
Press 2 for SET2-SET1
Enter choice:2

The difference of sets is:a f
Enter the choice (-1 for exit):2

Press 1 for SET1-SET2
Press 2 for SET2-SET1
Enter choice:1

The difference of sets is:
Enter the choice (-1 for exit):3

The union is:a a b c d e f
Enter the choice (-1 for exit):4

Intersection of sets is:a b c d e
Enter the choice (-1 for exit):-1

-----
Process exited after 34.94 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 3

WAP for checking that given expression is a tautology, contradiction or contingency.

The following code takes input expression from the user in the form of string and evaluates the postfix expression from the given infix expression. Then that infix expression is passed to each of the 2^n combinations of the n distinct variables and hence evaluated for each one of the combination. If the result is true throughout then the expression is a tautology, if it false throughout then it is a contradiction and if the output changes in between the evaluation of expression, the expression is a contingency.

- **CODE IN C++**

```
#include<iostream>

#include<stdio.h>

#include<stack>

#include<string>

#include<vector>

#include<map>

using namespace std;

char negat(char op1)

{

    if(op1=='0')

        return '1';

    else return '0';
```

```

}

char conjunc(char op1,char op2)
{
    if(op1=='1' && op2=='1')
        return '1';
    else return '0';
}

char disjunc(char op1,char op2)
{
    if(op1=='0' && op2=='0')
        return '0';
    else return '1';
}

char impli(char op1, char op2){
    if(op1=='1' && op2=='0')
        return '0';
    else return '1';
}

char eval(vector<char> post,map<char,char> m)
{
    stack<char> s;
    for(int i=0;i<post.size();i++)
    {
        if(isalpha(post[i]))
            s.push(post[i]);
        else
        {
            char op=post[i];
            if(post[i]=='-')
            {

```

```

    char op1=s.top();
    s.pop();
    if(op1!='0' && op1!='1')
        op1=m[op1];
    char res=negat(op1);
    s.push(res);
}
else
{
    char res;
    char op2=s.top();
    s.pop();
    if(op2!='0' && op2!='1')
        op2=m[op2];
    char op1=s.top();
    s.pop();
    if(op1!='0' && op1!='1')
        op1=m[op1];
    if(op=='^')
        res=conjunc(op1,op2);
    else if(op=='+')
        res=disjunc(op1,op2);
    else if(op=='>')
        res=impli(op1,op2);
    s.push(res);
}
}
}
return s.top();
}

```

```

int precedence(char op)
{
    if(op=='-')    //negation
        return 4;
    if(op=='^')    //and
        return 3;
    if(op=='+')    //or
        return 2;
    if(op=='>')    //implication
        return 1;
    return 0;
}

```

```

int main()
{
    cout<<"\n\t\t EVALUATING WHETHER GIVEN EXPRESSION IS A CONTRADICTION, TAUTOLOGY OR CONTINGENCY.\n";
    cout<<"ENTER EXPRESSION : USE( -:NEGATION, ^:AND, +:OR, >:IMPLICATION )\n";
    string exp;
    getline(cin,exp);
    stack<char> op;
    vector <char> post;
    int len=exp.length();

    //INFIX TO POSTFIX
    for(int i=0;i<len;i++)
    {
        if(isalpha(exp[i]))
            post.push_back(exp[i]);
        else
        {

```



```

if(op.empty())
    op.push(exp[i]);
else
{
    if(exp[i]=='('){
        op.push('(');
        continue;
    }
    if(exp[i]==')')
    {
        while(op.top()!='('){
            post.push_back(op.top());
            op.pop();
        }
        op.pop();
        continue;
    }
    while (!op.empty() &&(precedence(op.top()) > precedence(exp[i])) )
    {
        post.push_back(op.top());
        op.pop();
    }
    op.push(exp[i]);
}
}

while(!op.empty())
{
    post.push_back(op.top());
    op.pop();
}

```

```

}

int count=0;
map<char,char> m;
for(int i=0;i<post.size();i++) //counting different elements in exp.
{
    if(isalpha(post[i]))
    {
        if(m.count(post[i])==0)
        {
            m[post[i]]='0';
            count++;
        }
    }
}

for(map<char,char>::iterator it=m.begin();it!=m.end();it++) //print table heading
    cout<<it->first<<" ";

cout<<"Result"<<endl;

int shift=count-1;
bool tauto=true;
bool contradiction=true;
for(int i=0;i<(1<<count);i++) //binary (if count is 3 --> 1<<count=1000 --> 8
{
    int Map=1<<shift; //100
    for(map<char,char>::iterator it=m.begin();it!=m.end();it++)
    {
        if((i&Map)==0)
        {
            it->second='0';

```

```

    }
    else
    {
        it->second='1';
    }
    Map>>=1;
    cout<<it->second<<" ";
}
char ans=eval(post,m);
cout<<ans<<endl;
if(ans=='0')
    tauto=false;
if(ans=='1')
    contradiction=false;
}
if(tauto)
    cout<<"Tautology";
else if(contradiction)
    cout<<"Contradiction";
else
    cout<<"Contingency";
return 0;
}

```

- **OUTPUT 1**

```
C:\Users\harsh\Documents\ContraTautContin.exe

EVALUATING WHETHER GIVEN EXPRESSION IS A CONTRADICTION, TAUTOLOGY OR CONTINGENCY.
ENTER EXPRESSION : USE( -:NEGATION, ^:AND, +:OR, >:IMPLICATION )
A+B^C
A B C Result
0 0 0 0
0 0 1 0
0 1 0 0
0 1 1 1
1 0 0 1
1 0 1 1
1 1 0 1
1 1 1 1
Contingency
-----
Process exited after 7.602 seconds with return value 0
Press any key to continue . . .
```

• OUTPUT 2

```
C:\Users\harsh\Documents\ContraTautContin.exe

EVALUATING WHETHER GIVEN EXPRESSION IS A CONTRADICTION, TAUTOLOGY OR CONTINGENCY.
ENTER EXPRESSION : USE( -:NEGATION, ^:AND, +:OR, >:IMPLICATION )
~(A+B>C)
A B C Result
0 0 0 0
0 0 1 0
0 1 0 1
0 1 1 0
1 0 0 1
1 0 1 0
1 1 0 1
1 1 1 0
Contingency
-----
Process exited after 10.79 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 4

WAP for checking that given relation is equivalence or not.

In the following code, number of elements along with their names in the relation were input from the user. Then, the element pairs were input from the user. An element pair (i,j) represented that there was relation between i and j. This information was input in a relation matrix such that it had 1 for the cell (i,j). Now to check for symmetricity of the relation, the upper triangular matrix and the lower triangular matrix was compared as for symmetricity of a relation, $A=A^T$. Then to check for reflexivity, the trace of the relation matrix was calculated and checked if it was equal to the number of distinct elements in the pairs that were input. If yes, then it was symmetric. For transitivity, the square of the relation matrix was calculated. Since the [i][j] cell of the squared matrix denoted the number of 2 length paths

from I to J, so we just had to check if a cell [I][J] had a non- zero entry in squared matrix then does it have 1 in the actual relation matrix or not. If yes, for all non-zero entries of A^2 then the relation was transitive also.

CODE IN C++

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

#define tr(v,it) for(typeof(v.begin()) it=v.begin();it!=v.end();it++)

using namespace std;

bool TR( int A,int B, int Relate[][100],int dCount,int distinct[]);

bool transitive(int Square[100][100], int Relate[100][100], int n)
{
    for(int i=0;i<n;i+=1)
    {
        for(int j=0;j<n;j+=1)
        {
            if(Square[i][j]>=1 && Relate[i][j]==0)
                return false;
        }
    }
    return true;
}

bool symmetric(int Relate[100][100], int n)
{
    for(int i=0;i<n;i+=1)
    {
        for(int j=i+1;j<n;j+=1)
        {
            if(Relate[i][j]!=Relate[j][i])
            {
                return false;
            }
        }
    }
}
```

```

    }
    return true;
}

int main()
{
    int n;

    cout<<"\nTO CHECK FOR EQUIVALENCE OF THE GIVEN BINARY RELATION.";
    cout<<"\nEnter the number of elements :";
    cin>>n;

    string ele;
    vector <string> V;
    map <int, int> m;

    int distinct[n]; //to send as a parameter.

    cout<<"\nEnter the name of elements:\n";
    for(int i=0;i<n;i++)
    {
        cout<<"Element "<<i+1<<":";
        cin>>ele;
        V.push_back(ele);
        m[i]=0; // making map with index
    }

    cout<<"\nYour elements are:\n";
    tr(V,it)
    {cout<<it-V.begin()<<":"<<*it<<endl;
    }

    cout<<"\nEnter the element pairs:";

    int Relate[100][100];
    int copy[100][100];
    for(int q=0;q<n;q++)
    {
        for(int w=0;w<n;w++)
            Relate[q][w]=copy[q][w]=0;
    }
}

```

```

}
string v1,v2;
int count=0;
int i,j;
while(true)
{
    if(count==n*n)
        {cout<<"Max element pairs reached.";
        break;}
    s:cout<<"\nName of E1:";
    cin>>v1;
    cout<<"Name of E2:";
    cin>>v2;
    i=find(V.begin(),V.end(),v1)-V.begin();
    j=find(V.begin(),V.end(),v2)-V.begin();
    if (i==n || j==n)
        {cout<<"\nNo such vertex pair exists.\nEnter again.";
        goto s;}
    count+=1;
    m[i]+=1;
    m[j]+=1;
    Relate[i][j]=1;
    copy[i][j]=1;
    cout<<"\nEnter 1 for more pairs or -1 for exit:";
    int ch1;
    cin>>ch1;
    if(ch1==-1) break;
}
cout<<"\nThe Relation matrix is:\n";
for(int f=0;f<n;f++)
{
    for(int g=0;g<n;g++)

```

```

        {cout<<Relate[f][g]<<" ";}
    cout<<endl;
}
int dCount=0;
tr(m,it)
{ if(it->second!=0)
    { distinct[i]=it->first;
      dCount+=1;}
}
int flag=0;
cout<<"\nCHECKING FOR EQUIVALENCE OF THE RELATION...\n";
if(TR(n,n,Relate,dCount,distinct)==false)
{flag=1;
cout<<"\nRelation is not Reflexive.";}
if(symmetric(Relate,n) == false)
{ flag+=1;
cout<<"\nRelation is not Symmetric.";}
}
int elem;
for (int i=0;i<n;i+=1) // calculating the square of the matrix...
{   for(int j=0;j<n;j+=1)
    { elem=0;
      for(int k=0;k<n;k+=1)
      { elem+=copy[i][k]*copy[k][j];
      }
      Relate[i][j]=elem;
    }
}
if(transitive(Relate,copy,n)==false)
{ flag+=1;

```



```

        cout<<"\nRelation is not Transitive.";}

if(flag==0)

    cout<<"\nYOUR RELATION IS AN EQUIVALENCE RELATION!";

return 0;

}

bool TR ( int A, int B, int Relate[][100],int dCount,int distinct[])

{

    int trace=0;

    for(int i=0;i<dCount;i+=1)

    {

        if(Relate[i][i]!=1)

            return false;

    }

    return true;

}

```

• OUPUT 1

```

C:\Users\harsh\Documents\equivalence.exe
TO CHECK FOR EQUIVALENCE OF THE GIVEN BINARY RELATION.
Enter the number of elements :4

Enter the name of elements:
Element 1:a
Element 2:b
Element 3:c
Element 4:d

Your elements are:
0:a
1:b
2:c
3:d

Enter the element pairs:
Name of E1:a
Name of E2:b

Enter 1 for more pairs or -1 for exit:1

Name of E1:b
Name of E2:c

Enter 1 for more pairs or -1 for exit:1

Name of E1:c
Name of E2:d

```

• OUTPUT 2

```
C:\Users\harsh\Documents\equivalence.exe

Name of E1:c
Name of E2:d

Enter 1 for more pairs or -1 for exit:1

Name of E1:a
Name of E2:d

Enter 1 for more pairs or -1 for exit:1

Name of E1:d
Name of E2:a

Enter 1 for more pairs or -1 for exit:1

Name of E1:b
Name of E2:a

Enter 1 for more pairs or -1 for exit:1

Name of E1:-1
Name of E2:1

No such vertex pair exists.
Enter again.
Name of E1:a
Name of E2:d

Enter 1 for more pairs or -1 for exit:-1
```

- **OUTPUT 3**

```
C:\Users\harsh\Documents\equivalence.exe

Name of E1:-1
Name of E2:1

No such vertex pair exists.
Enter again.
Name of E1:a
Name of E2:d

Enter 1 for more pairs or -1 for exit:-1

The Relation matrix is:
0 1 0 1
1 0 1 0
0 0 0 1
1 0 0 0

CHECKING FOR EQUIVALENCE OF THE RELATION...

Relation is not Reflexive.
Relation is not Symmetric.
Relation is not Transitive.
-----
Process exited after 28.93 seconds with return value 0
Press any key to continue . . .
```

- **OUTPUT 4**

```
C:\Users\harsh\Documents\equivalence.exe

TO CHECK FOR EQUIVALENCE OF THE GIVEN BINARY RELATION.
Enter the number of elements :3

Enter the name of elements:
Element 1:1
Element 2:2
Element 3:3

Your elements are:
0:1
1:2
2:3

Enter the element pairs:
Name of E1:1
Name of E2:1

Enter 1 for more pairs or -1 for exit:1

Name of E1:2
Name of E2:2

Enter 1 for more pairs or -1 for exit:1

Name of E1:3
Name of E2:3

Enter 1 for more pairs or -1 for exit:1
```

• OUTPUT 5

```
C:\Users\harsh\Documents\equivalence.exe

Name of E1:3
Name of E2:3

Enter 1 for more pairs or -1 for exit:1

Name of E1:1
Name of E2:3

Enter 1 for more pairs or -1 for exit:1

Name of E1:3
Name of E2:1

Enter 1 for more pairs or -1 for exit:-1

The Relation matrix is:
1 0 1
0 1 0
1 0 1

CHECKING FOR EQUIVALENCE OF THE RELATION...

YOUR RELATION IS AN EQUIVALENCE RELATION!
-----
Process exited after 17.86 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 5

WAP for generating next permutation in lexicographic order.

A string was input from the user and the next lexicographic permutation was calculated according to the following points-

- The largest index in the string was found such that $a[\text{index}]$ is greater than atleast one element ahead of it. This was done by traversing the string from right to left and then finding the index of the element whose next element on the left is smaller than it.
- Then, the smallest element on the right of the index was found such that that element's index is greater than the index. Then it was replaced with $a[\text{index}]$
- Finally, the string was sorted from index to the end of the string to obtain the next lexicographic permutation.
- **CODE IN C++**

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

```

using namespace std;

void permute(string s)
{
    int m,index;

    int min=INT_MAX;

    for(int i=s.size()-1;i>=1;i-=1)
    {
        if(s[i-1]<s[i])
        {
            m=i-1;
            break;
        }
    }

    for(int i=m+1;i<s.size();i+=1)
    {
        if(s[i]>s[m])
        {
            if(s[i]<min)
            {
                min=s[i];
                index=i;} // finding the smallest element amongst a(m+1) to a(n) that is larger than a(m)
        }
    }

    char temp;

    temp=s[m]; //swapping the element with the smallest element greater than it
    s[m]=s[index];
    s[index]=temp;

    sort(s.begin()+m+1,s.end()); //arranging the elements ahead of a(m)

    cout<<endl<<s;
}

int main()
{
    cout<<"\nGENERATING THE LEXICOGRAPHICALLY NEXT PERMUTATION...";

    string s;

```

```

cout<<"\nEnter the string:";

cin>>s;

cout<<"\nThe next permutation of the string is:";

permute(s);

return 0;

}

```

- **OUTPUT 1**

```

C:\Users\harsh\Documents\permutation.exe
GENERATING THE LEXICOGRAPHICALLY NEXT PERMUTATION...
Enter the string:harshit

The next permutation of the string is:
harshti
-----
Process exited after 3.845 seconds with return value 0
Press any key to continue . . .

```

- **OUTPUT 2**

```

C:\Users\harsh\Documents\permutation.exe
GENERATING THE LEXICOGRAPHICALLY NEXT PERMUTATION...
Enter the string:124653

The next permutation of the string is:
125346
-----
Process exited after 5.036 seconds with return value 0
Press any key to continue . . .

```

PROGRAM 6

WAP for generating next combination in lexicographic order.

- CODE IN C++

A string was input from the user and the next lexicographic combination was calculated according to the following points-

- The largest index in the string was found such that $a[\text{index}]$ is not equal to its maximal value which is $n - k + \text{index}$. This was done by traversing the string from right to left and then finding the index of the element whose value was not equal to its maximal value. If no such index was found, the subset was the maximal subset.
- The string was then updated by incrementing the value at the index and then the subsequent values were equal to the value at the last index+1.
- Finally, the string was printed to the end of the string to obtain the next lexicographic combination.

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

#define For(i,n) for(int i=0;i<n;i+=1)

using namespace std;

void combo(string set, int n,int k)
{
    int j=0;
    int m=-1;
    for(int i=k-1;i>=0;i-=1)
    {
        if((set[i]-'0')!=(n-j))
        {
            m=i;
            break;
        }
        else j++;
    }
    if(m== -1)
    {
        cout<<"\nMaximal subset.";
        return;}
    set[m]=(set[m]+1);
```

```

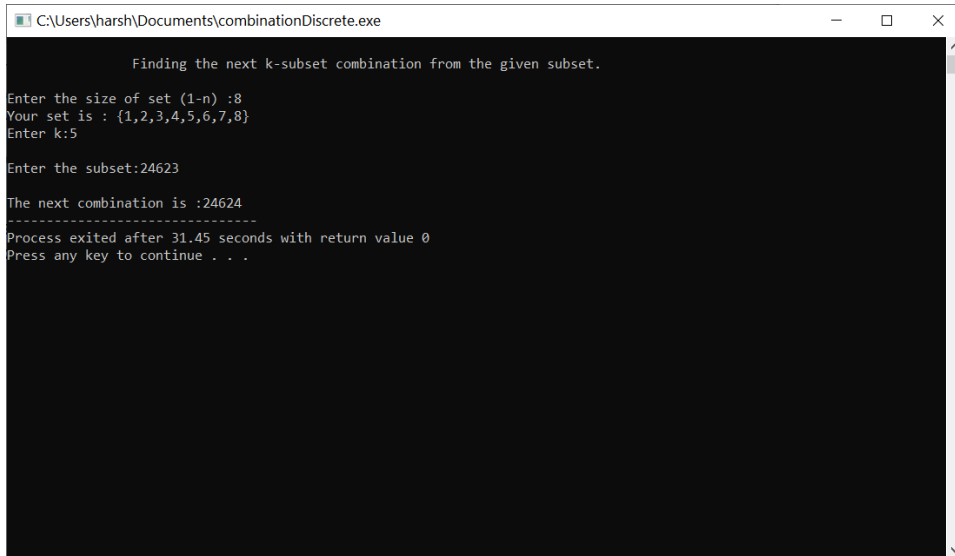
for(int i=m+1;i<k;i+=1)
    set[i]=(set[i-1]+1);
for(int i=0;i<k;i++)
{ cout<<set[i];
}
}

int main()
{ cout<<"\n\t\tFinding the next k-subset combination from the given subset.\n";
  cout<<"\nEnter the size of set (1-n) :";
  int n;
  cin>>n;
  cout<<"Your set is : {";
  For(i,n)
  { if(i!=n-1) cout<<i+1<<" ";
    else cout<<i+1<<"}";
  }
  int k;
  x:cout<<"\nEnter k:";
  cin>>k;
  if(k>n)
  { cout<<"\nInvalid k\nEnter again.";
    goto x;}
  string set;
  y:cout<<"\nEnter the subset:";
  cin>>set;
  For(i,k)
  {if(set[i]-'0'>n)
    { cout<<"\nInvalid subset\nEnter within bounds";
      goto y;}
  }
}

```

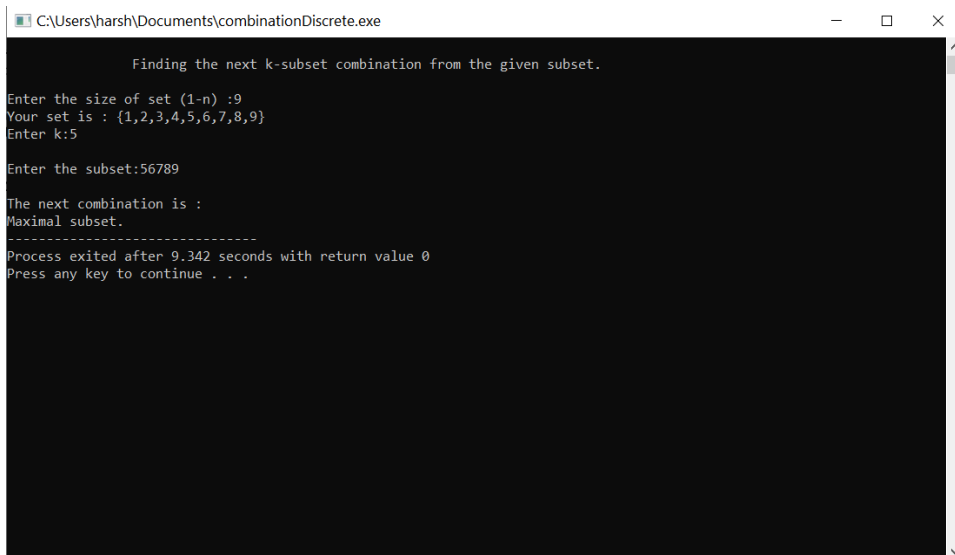
```
cout<<"\n\nThe next combination is :";  
  
combo(set,n,k);  
  
return 0;}
```

- **OUTPUT 1**



```
C:\Users\harsh\Documents\combinationDiscrete.exe  
  
Finding the next k-subset combination from the given subset.  
Enter the size of set (1-n) :8  
Your set is : {1,2,3,4,5,6,7,8}  
Enter k:5  
  
Enter the subset:24623  
  
The next combination is :24624  
-----  
Process exited after 31.45 seconds with return value 0  
Press any key to continue . . .
```

- **OUTPUT 2**



```
C:\Users\harsh\Documents\combinationDiscrete.exe  
  
Finding the next k-subset combination from the given subset.  
Enter the size of set (1-n) :9  
Your set is : {1,2,3,4,5,6,7,8,9}  
Enter k:5  
  
Enter the subset:56789  
  
The next combination is :  
Maximal subset.  
-----  
Process exited after 9.342 seconds with return value 0  
Press any key to continue . . .
```


PROGRAM 7

WAP for checking given poset is a lattice or not.

In the following code it was evaluated whether a given poset was a lattice or not. User was prompted to enter the total number of vertices and the total number of edges in the Hasse diagram of poset. Then, the pairs of vertices were input so that a hasse diagram could be formed. Depth first search was used to find out the upward connected components of a node and the downward connected components were found out iteratively. Then GLBs and LUBs were found for each element pair and if they existed then the poset was a lattice else it was not a lattice.

- **CODE IN C++**

```
#include <iostream>

#include <algorithm>

#include <vector>

#include<set>

#define tr(v,it) for(typeof(v.begin()) it = v.begin();it!=v.end();it++)

#define REP(i,n) for(int i=0; i<n; i++)

using namespace std;

vector<int> adj[100];

bool visitedForMain[100];

bool visited[100];

vector<int> Up[100];

vector<int> Down[100];

//set insertion

void insert(vector<int>&v, int data)

{

    //cout<<"Inserting "<<data<<endl;

    if(v.empty())

    {
```

```

    v.push_back(data);
    return;
}
if(data > v.back())
{
    v.push_back(data);
    return;
}
if(data < v.front())
{
    v.insert(v.begin(),data);
    return;
}
int i=0;
tr(v,it)
{
    if(*it==data)
        return;
    if(*it < data)
    {
        i++;
    }
    else
    {
        v.insert(v.begin()+i,data);
        return;
    }
}
}

```

```

//set intersection
int intersection(vector<int> a, vector<int> b)
{
    vector<int>::iterator it1=a.begin();
    vector<int>::iterator it2=b.begin();
    vector<int> c;
    while(it1!=a.end() && it2!=b.end())
    {
        if(*it1==*it2)
        {
            insert(c,*it1);
            it1++;
            it2++;
        }
        else
        {
            if(*it1>*it2)
            {
                it2++;
            }
            else
            {
                it1++;
            }
        }
    }
    return c.size();
}

```

```

void dfs(int a, int node){

```

```

    Up[a].push_back(node);

```

```

    for(int v=0; v< adj[node].size(); v++){
        dfs(a, adj[node][v]);
    }

}

int main()
{
    cout<<"\n\t\t\t\tCHECKING WHETHER GIVEN POSET IS LATTICE OR NOT\n";
    int v,e;
    cout<<"Enter no of vertices:";
    cin>>v;

    cout<<"Enter number of edges:";
    cin>>e;
    cout<<"Enter in pairs the connected component\n";
    int a,b;
    set<int> elements;

    REP(i,e){
        cout<<"Pair"<<i+1<<": ";
        cin>>a>>b;
        adj[a].push_back(b);
        elements.insert(a);
        elements.insert(b);
    }

    bool lub = true;
    bool glb = true;

```

```

//initialising Up[100]
tr(elements,it){
    dfs(*it,*it);
}

//FOR LUB
tr(elements,it){

    tr(elements,it1){
        int ub = intersection(Up[*it], Up[*it1]);

        if(!lub){
            lub= false;
            break;
        }

    }
}

if(lub) cout<<"\nLowest Upper Bound exists"<<endl;
else cout<<"\nLowest Upper Bound doesn't exist"<<endl;

//to check upLower

//FOR GLB

//to initialise Down[100]
tr(elements,it){
    for(vector<int>::iterator it1 = Up[*it].begin(); it1 != Up[*it].end();it1++){
        Down[*it1].push_back(*it);
    }
}

```

```

//FOR GLB
tr(elements,it){
    tr(elements,it1){
        int lb = intersection(Down[*it], Down[*it1]);
        if(!lb){
            glb= false;
            break;
        }

    }
}

if(glb) cout<<"\nGreatest Lower Bound Exists"<<endl;
else cout<<"\nGreatest Lower Bound doesn't exist"<<endl;
//to check GreLower
if(lub&&glb) cout<<"\nIT IS A LATTICE";
else cout<<"\nIT IS NOT A LATTICE";

return 0;
}

```

- **OUTPUT 1**

```

C:\Users\harsh\Documents\Latticeadi.exe
CHECKING WHETHER GIVEN POSET IS LATTICE OR NOT
Enter no of vertices:5
Enter number of edges:6
Enter in pairs the connected component
Pair1:1 2
Pair2:1 3
Pair3:2 4
Pair4:3 4
Pair5:2 5
Pair6:3 5
Lowest Upper Bound doesn't exist
Greatest Lower Bound Exists
IT IS NOT A LATTICE
-----
Process exited after 28.27 seconds with return value 0
Press any key to continue . . .

```

PROGRAM 8

WAP for verifying the inclusion exclusion principle on given sets

The inclusion exclusion principle for sets is a principle which defines the how to compute the Union of more than one sets. The formula for the inclusion exclusion principle for n sets is-

$$A_1 \cup A_2 \cup A_3 \cup \dots \cup A_n = A_1 + A_2 + A_3 + \dots + A_n - A_1^{A_2} - A_2^{A_3} - \dots + A_1^{A_2^{A_3}} + A_2^{A_3^{A_4}} + \dots + (-1)^{n-1} A_1^{A_2^{A_3} \dots A_n}$$

In the following program, the formula has been verified by taking the help of three sets. The elements of the set are input by the user and the cardinality of the union and the actual set union is found out.

The functions intersect and Union return the intersection and the union respectively for the two sets passed into it as parameters.

- **CODE IN C++**

```

#include<bits/stdc++.h>
#define tr(v,it) for(typeof(v.begin()) it = v.begin();it!=v.end();it++)
#define FOR(i,n) for(int i=0;i<n;i+=1)
using namespace std;
set <string> intersect (set <string> a, set<string> b)
{
    set<string> ITRS;
    tr(a,it)
    {
        if(b.find(*it)!=b.end())
            ITRS.insert(*it);
    }
    return ITRS;
}
set <string> Union (set<string> a,set<string> b)

```

```

{ set<string> U(a.begin(),a.end());
  tr(b,it)
  {
    if(a.find(*it)==a.end())
      U.insert(*it);
  }
  return U;
}

int main()
{
  cout<<"\n\t\t\t\t\tPRINCIPLE OF INCLUSION AND EXCLUSION FOR THREE SETS...";
  cout<<"\nEnter the elements of sets...";
  set <string> s1,s2,s3;
  string ele;
  int i=1;
  cout<<"\nSET 1(-1 for exit)\n";
  while(true)
  {  cout<<"Element "<<i<<":";
      cin>>ele;
      if(ele=="-1") break;
      s1.insert(ele);
      i+=1;
  }
  i=1;
  cout<<"\nSET 2(-1 for exit)\n";
  while(true)
  {  cout<<"Element "<<i<<":";
      cin>>ele;
      if(ele=="-1") break;
      s2.insert(ele);
      i+=1;
  }
  i=1;
  cout<<"\nSET 3(-1 for exit)\n";
  while(true)
  {  cout<<"Element "<<i<<":";
      cin>>ele;
      if(ele=="-1") break;
      s3.insert(ele);

```



```

    i+=1;
}
cout<<"\n\nThe sets are -\n";
cout<<"SET1 :";
tr(s1,it) cout<<*it<<" ";
cout<<"\n\nSET2 :";
tr(s2,it) cout<<*it<<" ";
cout<<"\n\nSET3 :";
tr(s3,it) cout<<*it<<" ";
cout<<"\n\nThe principle of inclusion exclusion states that the union of three sets is given by the formula -\n";
cout<<" $S1 \cup S2 \cup S3 = S1 + S2 + S3 - S1 \cap S2 - S1 \cap S3 - S2 \cap S3 + S1 \cap S2 \cap S3$ \n\t\t\t\t\t where U denoted the union and ^
denotes the intersection.";
cout<<"\n\nThe formula is true for the cardinalities of the sets S1,S2,S3 also.";
int cardinality= s1.size()+s2.size()+s3.size()- intersect(s1,s2).size()-intersect(s2,s3).size()-intersect(s1,s3).size() +
intersect(intersect(s1,s2),s3).size();
cout<<"\n\nThe resulting cardinality of the set (S1 U S2 U S3) is: "<<cardinality;
cout<<"\n\nThe result of your union is :";
set<string> s4= Union(Union(s1,s2),s3);
tr(s4,it)
{
    cout<<*it<<" ";
}
return 0;
}

```

• OUTPUT 1

```

C:\Users\harsh\Documents\Inclusionexclusion.exe
PRINCIPLE OF INCLUSION AND EXCLUSION FOR THREE SETS...
Enter the elements of sets...
SET 1(-1 for exit)
Element 1:a
Element 2:b
Element 3:c
Element 4:d
Element 5:-1

SET 2(-1 for exit)
Element 1:b
Element 2:c
Element 3:-1

SET 3(-1 for exit)
Element 1:f
Element 2:g
Element 3:h
Element 4:q
Element 5:-1

The sets are -
SET1 :a b c d
SET2 :b c
SET3 :f g h q
The principle of inclusion exclusion states that the union of three sets is given by the formula -
 $S1 \cup S2 \cup S3 = S1 + S2 + S3 - S1 \cap S2 - S1 \cap S3 - S2 \cap S3 + S1 \cap S2 \cap S3$ 
where U denoted the union and ^ denotes the intersection.
The formula is true for the cardinalities of the sets S1,S2,S3 also.

```

• OUTPUT 2

```

C:\Users\harsh\Documents\Inclusionexclusion.exe
SET 2(-1 for exit)
Element 1:b
Element 2:c
Element 3:-1

SET 3(-1 for exit)
Element 1:f
Element 2:g
Element 3:h
Element 4:q
Element 5:-1

The sets are -
SET1 :a b c d
SET2 :b c
SET3 :f g h q
The principle of inclusion exclusion states that the union of three sets is given by the formula -

$$S1 \cup S2 \cup S3 = S1 + S2 + S3 - S1 \cap S2 - S2 \cap S3 - S1 \cap S3 + S1 \cap S2 \cap S3$$

where U denoted the union and ^ denotes the intersection.
The formula is true for the cardinalities of the sets S1,S2,S3 also.
The resulting cardinality of the set (S1 U S2 U S3) is: 8
The result of your union is :a b c d f g h q
-----
Process exited after 15.12 seconds with return value 0
Press any key to continue . . .

```

PROGRAM 9

WAP to solve the given recurrence relation by the user

The following program solves the homogeneous equation of a second order linear recurrence relation. The coefficients of the r th, $(r-1)$ th and $(r-2)$ th terms are taken as input from the user, a characteristic equation is formed from the recurrence relation. Now, the characteristic equation is analysed for real, imaginary and equal roots.

If it has equal roots the solution is of the form $\rightarrow a(r) = (C1r + C2) (\text{root})^r$

If it has real or imaginary roots, the solution is of the form $\rightarrow a(r) = C1(\text{root1})^r + C2(\text{root2})^r$

If the characteristic equation has real or equal roots, the recurrence is solved fully by taking two consecutive values from the user. These consecutive values are input in the solution along with their indices to find out the value of $C1$ and $C2$.

- CODE IN C++**

```

#include<bits/stdc++.h>

#include<cmath>

#define tr(v,it) for(typeof(v.begin()) it= v.begin();it!=v.end();it++)

using namespace std;

float discr;

int main()
{
    int k;

    cout<<"\nProgram to solve the linear recurrence relation of order 2.";

    float arr[3];

    for(int i=2;i>=0;i--)
    {
        if(i==2)

```

```

{
    cout<<"Enter the coefficient of the (r)th term:";
    cin>>arr[2];}
else
{
    cout<<"Enter the coefficient of the (r- "<<2-i<<" )th term:";
    cin>>arr[i];
}
}

```

```
vector<float> Coef(arr,arr+3);
```

```
cout<<"\nThe recurrence relation is:";
```

```
int i=2;
```

```
tr(Coef,it)
```

```

{
    if(i!=0)
        cout<<"("<<*it<<"*A(r- "<<i<<" ) + ";
    else
        cout<<"("<<*it<<"*A(r)";
    i-=1;
}

```

```
cout<<"\n The characteristic equation is:\n";
```

```
int j=0;
```

```
tr(Coef,it)
```

```

{
    if(j==0)
        cout<<*it<<" +";
    else
    {   if(j!=1)
        {if(j!=2)
            cout<<"("<<*it<<"*M^ "<<j<<" ) +";

```

```

else
    cout<<"("<<*it<<"*M^"<<j<<" )";
}

else
    cout<<"("<<*it<<"*M"<<" ) +";

}

j+=1;
}

//arr[0] is my c arr[2] is my a
vector< pair < float,float> > roots;

discr=arr[1]*arr[1]-4*arr[0]*arr[2];
if(discr>0)
{
    roots.push_back(make_pair((-arr[1]+sqrt(discr))/(2*arr[2])),0));
    roots.push_back(make_pair((-arr[1]-sqrt(discr))/(2*arr[2])),0));

}

else if(discr<0)
{
    roots.push_back(make_pair((-arr[1]/(2*arr[2])),-(sqrt(-discr)/(2*arr[2]))));
    roots.push_back(make_pair((-arr[1]/(2*arr[2])),sqrt(-discr)/(2*arr[2])));

}

else
{
    roots.push_back(make_pair((-arr[1]/(2*arr[2])),0));
    roots.push_back(make_pair((-arr[1]/(2*arr[2])),0));
}

```

```

}

cout<<"\n\nThe roots are:";

if(discrim==0)
{
    cout<<"\n\nThe roots are repeated-"<<roots[0].first<<","<<roots[0].first;
    cout<<"\n\nThe homogeneous solution is: \n A(r)=(C1*r + C2)*("<<roots[0].first<<")^r";

}
else
{
    if(discrim>0)
    {
        cout<<"\n\nRoot 1 is- "<<roots[0].first;
        cout<<"\n\nRoot 2 is- "<<roots[1].first;
        cout<<"\n\nThe homogeneous solution is: C1*("<<roots[0].first<<")^r + C2*("<<roots[1].first<<")^r";

    }
    else
    {
        cout<<"\n\nRoot 1 is- "<<roots[0].first<< + ("<<roots[0].second<<")*i ";
        cout<<"\n\nRoot 2 is- "<<roots[1].first<< + ("<<roots[1].second<<")*i ";
        cout<<"\n\nThe homogeneous solution is: C1*("<<roots[0].first<< + ("<<roots[0].second<<")*i"<<")^r +
C2*("<<roots[1].first<< + ("<<roots[1].second<<")*i)"<<")^r";

    }
}

// to completely solve recurrence

if(discrim>=0)
{
    cout<<"\n\nEnter two consecutive values to completely solve this recurrence relation:";

```

```

u:cout<<"\nEnter the index of one - ";
pair<int, int > indices;
cin>>indices.first;
cout<<"Enter the index of two - ";
cin>>indices.second;
if(indices.second-indices.first!=1)
    {cout<<"\nInvalid values.\nEnter again.";
    goto u;}
else
{
    float a1,a2;
    cout<<"\nEnter value of A"<<indices.first<<" :";
    cin>>a1;
    cout<<"Enter value of A"<<indices.second<<" :";
    cin>>a2;
    if(discrim>0)
    {
        float c1,c2;
        c1=(a2-(roots[1].first)*a1)/(pow(roots[0].first,indices.second)-pow(roots[0].first,indices.first)*roots[1].first);
        c2=(a2-(roots[0].first)*a1)/(pow(roots[1].first,indices.second)-pow(roots[1].first,indices.first)*roots[0].first);
        cout<<"\nThe homogeneous solution is: A(r)="<<c1<<"*("<<roots[0].first<<")^r +
"<<c2<<"*("<<roots[1].first<<")^r";
    }
    else if(discrim==0)
    {
        float c1,c2;
        c1=(roots[0].first*a1-a2)/(pow(roots[0].first,indices.second));
        c2=((roots[0].first*a1*(1-indices.first))+a2*indices.first)/(pow(roots[0].first,indices.second));
        cout<<"\nThe homogeneous solution is: \n A(r)="<<c1<<"*r + "<<c2<<"*("<<roots[0].first<<")^r";
    }
}
}

```

```

}

return 0;

}

```

• OUTPUT 1

```

C:\Users\harsh\Documents\recurrence solve.exe
Program to solve the linear recurrence relation of order 2.
Enter the degree of the linear recurrence relation:Enter the coefficient of the (r)th term:1
Enter the coefficient of the (r-1)th term:-3
Enter the coefficient of the (r-2)th term:2

The recurrence relation is:(2*A(r-2)) + (-3*A(r-1)) + (1*A(r))
The characteristic equation is:
2 + (-3*M) + (1*M^2)
The roots are:
Root 1 is- 2
Root 2 is- 1
The homogeneous solution is: C1*(2)^r + C2*(1)^r
Enter two consecutive values to completely solve this recurrence relation:
Enter the index of one - 4
Enter the index of two - 6

Invalid values.
Enter again.
Enter the index of one - 2
Enter the index of two - 3

Enter value of A2 :4
Enter value of A3 :6

The homogeneous solution is: A(r)=0.5*(2)^r + 2*(1)^r
-----
Process exited after 19.96 seconds with return value 0
Press any key to continue . . .

```

• OUTPUT 2

```

C:\Users\harsh\Documents\recurrence solve.exe
Program to solve the linear recurrence relation of order 2.
Enter the degree of the linear recurrence relation:Enter the coefficient of the (r)th term:1
Enter the coefficient of the (r-1)th term:-8
Enter the coefficient of the (r-2)th term:16

The recurrence relation is:(16*A(r-2)) + (-8*A(r-1)) + (1*A(r))
The characteristic equation is:
16 + (-8*M) + (1*M^2)
The roots are:
The roots are repeated-4,4
The homogeneous solution is:
A(r)=(C1*r + C2)*(4)^r
Enter two consecutive values to completely solve this recurrence relation:
Enter the index of one - 3
Enter the index of two - 4

Enter value of A3 :4
Enter value of A4 :8

The homogeneous solution is:
A(r)=(0.03125*r + -0.03125)*(4)^r
-----
Process exited after 16.24 seconds with return value 0
Press any key to continue . . .

```

PROGRAM 10

WAP for checking whether the given input is a group or not.

The following code checks whether a predefined set or a user defined set with an operator op is a group or not. The user is asked to choose from 8 set choices and 6 operator choices and then it is found out if the set S and the operator op form a group or not.

- **CODE IN C++**

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

```
using namespace std;
```

```
int dp[7][6];
```

```
int identities[6];
```

```
void func()
```

```
{
```

```
for(int i=0;i<7;i++)
```

```
{
```

```
for(int j=6;j<6;j++)
```

```
dp[i][j]=0;
```

```
}
```

```
dp[2][0]=1;
```

```
dp[3][0]=1;
```

```
dp[5][0]=1;
```

```
dp[7][0]=1;
```

```
identities[0]=0;
```

```
identities[1]=1;
```

```
identities[2]=-1;
```

```
identities[3]=0;
```

```
identities[4]=-1;
```

```
identities[5]=1;
```

```
}
```



```

bool check(int preset, int op)
{
    if(op!=1){
        //cout<<"Not a Group";
        return false;
    }
    if(preset==3 | preset==4 | preset==6 | preset==7){
        //cout<<"A group";
        return true;
    }
    return false;
}

bool check(vector<float> set, int op)
{
    if(op==1)
    {
        if(find(set.begin(),set.end(),0)==set.end())
        {
            cout<<"Identity Element '0' not present";

            return false;
        }
        for(int i=0;i<set.size();i++)
        {
            for(int j=i+1;j<set.size();j++)
            {
                if(find(set.begin(),set.end(),(set[i]+set[j]))==set.end())
                {
                    cout<<"Result of "<<set[i]<<" and "<<set[j]<<"doesn't exist.";

```

```

return false;
}
}
}

for(int i=0;i<set.size();i++)
{
if(find(set.begin(),set.end(),(-1*(set[i])))==set.end())
{
cout<<"Inverse of"<<set[i]<<" doesn't exist";
return false;
}
}

cout<<"A group";
return true;
}

if(op==2)
{

if(find(set.begin(),set.end(),1)==set.end())
{
cout<<"Identity element '1' doesn't exist";
return false;
}

for(int i=0;i<set.size();i++)
{
for(int j=i+1;j<set.size();j++)
{
if(find(set.begin(),set.end(),(set[i]*set[j]))==set.end())
{
cout<<"Result of "<<set[i]<<" and "<<set[j]<<" doesn't exist";

```

```

return false;
}
}
}

for(int i=0;i<set.size();i++)
{
if(find(set.begin(),set.end(),(1/set[i]))==set.end())
{
cout<<"The inverse of "<<set[i]<<" doesn't exist";
return false;
}
}
}

//cout<<"The operator is not associative\nNot a group";

return false;
}

int main()
{
vector<float> set;

cout<<"\n1. Natural Numbers\n2. Whole Numbers\n3. Integers\n4. EvenNumbers\n5. Odd numbers\n6. Real
numbers\n7. Matrix\n8.CUSTOM SET(input by user)";

int preset;

cout<<endl;

cout<<endl;

cout<<"*****"<<endl<<endl;

cout<<"1. +(addition)\n2. *(multiplication)\n3. -(subtraction)\n4. /(division)\n5. %(modulous)\n6. ^(power)";

int op;

cout<<endl<<endl;

```

```

while(1)
{
cout<<"Enter the choice of set->";
cin>>preset;
if(preset==8)
{
float data;
cout<<"Enter the elements in the set (end by -1) -> ";
while(data!=-1){

cin>>data;
set.push_back(data);
}
}
cout<<"Enter the choice of operator->> ";
cin>>op;

if(preset==8)
{
bool val=check(set,op);
if(val)
{
cout<<"\nA GROUP";
}
else
cout<<"\nNOT A GROUP";
}
else
{
bool val=check(preset,op);

```

```

if(val)

cout<<"A GROUP";

else

cout<<"NOT A GROUP";

}

cout<<endl;


int choice;

cout<<"1. Test Another Pair\n2. Exit\nEnter choice";

cin>>choice;

if(choice!=1)

break;

}

return 0;

}

```

• OUTPUT 1

```

C:\Users\harsh\Documents\groupdiscrete.exe
1. Natural Numbers
2. Whole Numbers
3. Integers
4. EvenNumbers
5. Odd numbers
6. Real numbers
7. Matrix
8. CUSTOM SET(input by user)

*****

1. +(addition)
2. *(multiplication)
3. -(subtraction)
4. /(division)
5. %(modulous)
6. ^(power)

Enter the choice of set->3
Enter the choice of operator->> 1
A GROUP
1. Test Another Pair
2. Exit
Enter choice1
Enter the choice of set->8
Enter the elements in the set (end by -1) -> 2
-1
Enter the choice of operator->> 1
Identity Element '0' not present

```

• OUTPUT 2

```
C:\Users\harsh\Documents\groupdiscrete.exe
1. Natural Numbers
2. Whole Numbers
3. Integers
4. EvenNumbers
5. Odd numbers
6. Real numbers
7. Matrix
8.CUSTOM SET(input by user)

*****

1. +(addition)
2. *(multiplication)
3. -(subtraction)
4. /(division)
5. %(modulous)
6. ^(power)

Enter the choice of set->8
Enter the elements in the set (end by -1) -> 9
2
1
7
-1
Enter the choice of operator-> 2
Result of 9 and 2 doesn't exist
NOT A GROUP
1. Test Another Pair
2. Exit
```

PROGRAM 11

WAP for checking whether the given input is a ring or not.

A set S , and the pair of operators $o1$ and $o2$ form a ring if - $(S,o1)$ form an abelian group, $(S,o2)$ form a semigroup and operation $o2$ is distributive over the operation 1. The following code is made for predefined as well as user defined sets. The user is then asked to input two operators $o1$ and $o2$. Then it is checked if $S,o1$ form an abelian group and whether $S,o2$ form a semigroup or not. It is then checked whether $o2$ is distributive over $o1$. With a little effort, it was noticed that only $o1$ as addition and $o2$ as multiplication are the suitable candidates for the set S to be a ring.

- CODE IN C++

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

```
using namespace std;
```

```
int dp[7][6];
```

```
int identities[6];
```

```
void func()
```

```
{
```

```
for(int i=0;i<7;i++)
```

```
{
```

```
for(int j=6;j<6;j++)
```

```

dp[i][j]=0;
}
dp[2][0]=1;
dp[3][0]=1;
dp[5][0]=1;
dp[7][0]=1;
identities[0]=0;
identities[1]=1;
identities[2]=-1;
identities[3]=0;
identities[4]=-1;
identities[5]=1;
}

```

```

bool check(int preset, int op)
{
    if(op!=1){
        //cout<<<"Not a Group";
        return false;
    }
    if(preset==3 || preset==4 || preset==6 || preset==7){
        //cout<<<"A group";
        return true;
    }
    return false;
}

bool check(vector<float> set, int op)
{
    if(op==1)
    {

```

```

if(find(set.begin(),set.end(),0)==set.end())
{
cout<<"Identity Element '0' not present for operator 1\n";

return false;
}

for(int i=0;i<set.size();i++)
{
for(int j=i+1;j<set.size();j++)
{
if(find(set.begin(),set.end(),(set[i]+set[j]))==set.end())
{
cout<<"Result of "<<set[i]<<" and "<<set[j]<<" doesn't exist for operator 1.\n";
return false;
}
}
}

for(int i=0;i<set.size();i++)
{
if(find(set.begin(),set.end(),(-1*(set[i])))==set.end())
{
cout<<"Inverse of"<<set[i]<<" doesn't exist for operator 1\n";
return false;
}
}

cout<<"A group\n";
return true;
}

if(op==2)
{

```



```

if(find(set.begin(),set.end(),1)==set.end())
{
cout<<"Identity element '1' doesn't exist for operator 1\n";
return false;
}

for(int i=0;i<set.size();i++)
{
for(int j=i+1;j<set.size();j++)
{
if(find(set.begin(),set.end(),(set[i]*set[j]))==set.end())
{
cout<<"Result of "<<set[i]<<" and "<<set[j]<<" doesn't exist for operator 1\n";
return false;
}
}
}

for(int i=0;i<set.size();i++)
{
if(find(set.begin(),set.end(),(1/set[i]))==set.end())
{
cout<<"The inverse of "<<set[i]<<" doesn't exist for operator 1\n";
return false;
}
}
}

//cout<<"The operator is not associative\nNot a group";

return false;
}

```

```

//1. N 2. W 3. Z 4.EN 5.ON 6.Real 7.CUSTOM
//1. + 2. * 3. - 4. / 5. % 6. ^

bool semigroupPre(int PDset, int op2)
{ // only checking for predefined sets.
    if(op2==3 || op2==4 || op2==5 || op2==6)
        return false; // not associative
    else
    { // + and * are associative.
        if(op2==1)
        {
            if(PDset==5) return false; // odd numbers under addition are not closed.
            else return true;
        }
        else
        {
            return true;
        }
    }
}

bool semigroupUser(vector<float> UDset, int op2)
{
    if(op2==3 || op2==4 || op2==5 || op2==6)
        return false; // not associative
    else
    {
        if(op2==1)
        {
            for(int i=0;i<UDset.size()-1;i+=1)
            {
                for(int j=i+1;j<UDset.size();j+=1)

```

```

{
    if(find(UDset.begin(),UDset.end(),UDset[i]+UDset[j])==UDset.end())
        { cout<<"The result of"<<UDset[i]<<" and "<<UDset[j]<<" doesn't exist for operator 2.";
          cout<<"\nHence set is not closed.";
          return false;
        } // set is not closed under addition as result is not found.
    }
}

return true;
}

if(op2==2)
{
    for(int i=0;i<UDset.size()-1;i+=1)
    {
        for(int j=i+1;j<UDset.size();j+=1)
        {
            if(find(UDset.begin(),UDset.end(),UDset[i]*UDset[j])==UDset.end())
                { cout<<"The result of"<<UDset[i]<<" and "<<UDset[j]<<" doesn't exist for operator 2.";
                  cout<<"\nHence set is not closed.";
                  return false;
                } // set is not closed under multiplication as result is not found.
            }
        }
    }

return true;
}

}

bool abelian(int op1)
{
    if(op1==1 || op1==2) return true;
    else return false;
}

```

```

}

bool distribute(int op1,int op2)
{
    if (op1==op2) return false; //same operators don't distribute over themselves.
    else
    {if(op2==2 && op1==1) return true;
    else return false;
    }
}

int main()
{ int setno,op1,op2;

    vector<float> UDset;

    float ele;

    cout<<"\nPROGRAM TO FIND IF THE GIVEN SET AND THE OPERATIONS FORM A RING OR NOT.";
    cout<<"\nCHOOSE SET->\n ";

    cout<<"\n1. Natural Numbers\n2. Whole Numbers\n3. Integers\n4. EvenNumbers\n5. Odd numbers\n6. Real
numbers\n7.CUSTOM SET(input by user)";

    cin>>setno;

    if(setno==7)
    { cout<<"Enter the elements in the set (end by -1) -> ";
        while(true)
        {cin>>ele;
            if(ele==-1) break;
            UDset.push_back(ele);
        }
    }

    cout<<"*****"<<endl<<endl;
    cout<<"1. +(addition)\n2. *(multiplication)\n3. -(subtraction)\n4. /(division)\n5. %(modulous)\n6. ^(power)";
    cout<<"Enter the two operators->> \nOPERATOR 1:";
    cin>>op1;

```

```

cout<<"\nOPERATOR 2:";
cin>>op2;
if(setno<7)
{
    if(check(setno,op1) && abelian(op1) && semigroupPre(setno,op2) && distribute(op1,op2))
        cout<<"\nThe selected set and the operators form a RING.";
    else
        cout<<"\nThe selected set and the operators don't form a RING.";
}
else
{
    if(check(UDset,op1) && abelian(op1) && semigroupUser(UDset,op2) && distribute(op1,op2))
        cout<<"\nThe input set and the operators form a RING.";
    else
        cout<<"\nThe input set and the operators don't form a RING.";
}
return 0;
}

```

• OUTPUT 1

```

C:\Users\harsh\Documents\RINGS.exe
PROGRAM TO FIND IF THE GIVEN SET AND THE OPERATIONS FORM A RING OR NOT.
CHOOSE SET->
1. Natural Numbers
2. Whole Numbers
3. Integers
4. EvenNumbers
5. Odd numbers
6. Real numbers
7.CUSTOM SET(input by user)3
*****
1. +(addition)
2. *(multiplication)
3. -(subtraction)
4. /(division)
5. %(modulous)
6. ^(power)Enter the two operators->>
OPERATOR 1:1
OPERATOR 2:2
The selected set and the operators form a RING.
-----
Process exited after 4.255 seconds with return value 0
Press any key to continue . . .

```

• OUTPUT 2

```
C:\Users\harsh\Documents\RINGS.exe

PROGRAM TO FIND IF THE GIVEN SET AND THE OPERATIONS FORM A RING OR NOT.
CHOOSE SET->
1. Natural Numbers
2. Whole Numbers
3. Integers
4. EvenNumbers
5. Odd numbers
6. Real numbers
7.CUSTOM SET(input by user)6
*****

1. +(addition)
2. *(multiplication)
3. -(subtraction)
4. /(division)
5. %(modulous)
6. ^(power)Enter the two operators->
OPERATOR 1:1
OPERATOR 2:2

The selected set and the operators form a RING.
-----
Process exited after 7.548 seconds with return value 0
Press any key to continue . . .
```

• OUTPUT 3

```
C:\Users\harsh\Documents\RINGS.exe

1. Natural Numbers
2. Whole Numbers
3. Integers
4. EvenNumbers
5. Odd numbers
6. Real numbers
7.CUSTOM SET(input by user)7
Enter the elements in the set (end by -1) -> 2
0
-2
-1
*****

1. +(addition)
2. *(multiplication)
3. -(subtraction)
4. /(division)
5. %(modulous)
6. ^(power)Enter the two operators->
OPERATOR 1:1
OPERATOR 2:2
A group
The result of 2 and -2 doesn't exist for operator 2.
Hence set is not closed.
The input set and the operators don't form a RING.
-----
Process exited after 11.32 seconds with return value 0
Press any key to continue . . .
```

PROGRAM 12

WAP for finding path and cycles in any given graph.

The following code accepts the number of total vertices from the user and the information whether there exists an edge between two vertices. This information is stored as an adjacency matrix in the memory and the path and cycles are found from it. To find the path from a vertex to another, Breadth First Search was used. To find the number of cycles, the matrix was multiplied by itself n times and the resultant matrices' i.e. $A^2, A^3 \dots$ cells denote the number of 2, 3, ... length path from the i th vertex to the j th vertex. For finding the cycles, the trace of this matrix was taken and divided by $2*n$ to eliminate repetitions in the graph. Note this method correctly shows the cycles of odd length but tends to include extra "paths" for even lengths.

• CODE IN C++

```

#include<iostream>

#include<vector>

#include<queue>

#include<algorithm>

#define tr(v,it) for(typeof(v.begin()) it=v.begin();it!=v.end();it++)

using namespace std;

void copied(int (&ADMnew)[100][100], int (&ADM)[100][100], int len);

int TR( int A,int B, int ADM[][100]);

bool path(int i,int j, int ADM[100][100], int n) //BREADTH-FIRST-SEARCH
{
    queue <int> Q;

    Q.push(i);

    bool visited[n]={false};

    visited[i]=true;

    while(visited[j]!=true && !Q.empty())
    {
        int current= Q.front();

        for(int m=0;m<n;m+=1)
        {
            if(ADM[current][m]==1)
            {
                Q.push(m);

                visited[m]=true;
            }
        }

        Q.pop();
    }

    return (visited[j]);
}

int main()

{
    ios::sync_with_stdio(false);

    //making graphs->

```

```

cout<<"\nGRAPHS...\nImplementing paths and cycles...";

cout<<"\nEnter the no. of vertices:";

int n;

cin>>n;

string ele;

vector <string> V;

cout<<"\nEnter the name of node:\n";

for(int i=0;i<n;i++)
{   cout<<"Vertex "<<i+1<<":";

    cin>>ele;

    V.push_back(ele);

}

cout<<"\nYour vertices are:\n";

tr(V,it)

{cout<<it-V.begin()<<":"<<*it<<endl;

}

char type;

y:cout<<"\nEnter D for directed graph and U for undirected graph:";

cin>>type;

if( type!='D' && type!='U' )

    {cout<<"Invalid.\nEnter again.";

    goto y;}

cout<<"\nEnter the relation between the vertices:";

int ADM[100][100];

int copy[100][100];

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

{   for(int w=0;w<n;w++)

        ADM[q][w]=copy[q][w]=0;

```



```

}
string v1,v2;
int count=0;
int i,j;
while(true)
{
    if((type=='U' && count==n*(n-1)/2) || (type=='D' && count==(n)*(n-1)))
        {cout<<"Max edges are reached.";
        break;}
    s:cout<<"\nName of V1:";
    cin>>v1;
    cout<<"Name of V2:";
    cin>>v2;
    count+=1;
    i=find(V.begin(),V.end(),v1)-V.begin();
    j=find(V.begin(),V.end(),v2)-V.begin();
    if (i==n || j==n)
        {cout<<"\nNo such vertex pair exists.\nEnter again.";
        goto s;}
    ADM[i][j]=1;
    copy[i][j]=1;
    if(type=='U')
        {ADM[j][i]=1;
        copy[j][i]=1;}
    cout<<"\nEnter 1 for more edges or -1 for exit:";
    int ch1;
    cin>>ch1;
    if(ch1==-1) break;
}
cout<<"\nThe adjacency matrix is:\n";
for(int f=0;f<n;f++)

```

```

{   for(int g=0;g<n;g++)
        {cout<<ADM[f][g]<<" ";}
    cout<<endl;
}

cout<<"\nFINDING PATHS AND CYCLES-\n  1. TO FIND PATH BETWEEN TWO VERTICES\n  2. TO FIND NO. OF
CYCLES\n";

p:cout<<"Enter choice:";

int ch2;

cin>>ch2;

if(ch2!=1 && ch2!=2)
    {cout<<"\nInvalid choice.Enter again.";
    goto p;
    }

if(ch2==1)
{
    v:cout<<"\nEnter the start vertex V1:";
    cin>>v1;
    cout<<"\nEnter the end vertex V2:";
    cin>>v2;
    i=find(V.begin(),V.end(),v1)-V.begin();
    j=find(V.begin(),V.end(),v2)-V.begin();
    if (i==n || j==n)
        {cout<<"\nNo such vertex pair exists.\nEnter again.";
        goto v;}
    if(path(i,j,ADM,n)==true)
        cout<<"\nPath exists.";
    else
        cout<<"\nPath not found.";
}

if(ch2==2)

```

```

{
    if(n==1 || n==2)
        cout<<"\nNumber of vertices-"<<n<<"\nCycles not possible.\n";
    else
        {cout<<"\nFINDING OUT CYCLES:";
        int elem;
        for (int i=0;i<n;i+=1) // calculating the square of the matrix...
            {   for(int j=0;j<n;j+=1)
                { elem=0;
                  for(int k=0;k<n;k+=1)
                      { elem+=copy[i][k]*copy[k][j];
                      }
                  ADM[i][j]=elem;
                }
            }
        for(int m=3;m<=n;m++)
            {   int ADMnew[100][100]={0};
            for (int i=0;i<n;i+=1)
                {   for(int j=0;j<n;j+=1)
                    {   int elem=0;
                        for(int k=0;k<n;k+=1)
                            { ADMnew[i][j]+=ADM[i][k]*copy[k][j];
                            }
                    }
                }
            }
        cout<<"\nNo. of cycles of length "<<m<<" are:";
        int trace=TR(n,n,ADMnew);
        if(type=='U')   cout<<(trace/(2*m));
        else   cout<<trace;
        copied(ADMnew,ADM,n);

```

```

    }
}
}

return 0;
}

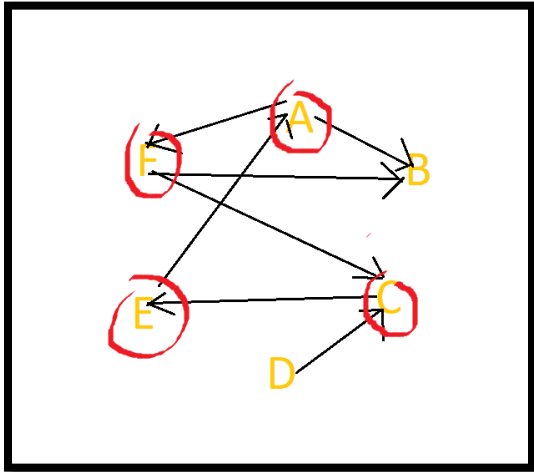
void copied(int (&ADMnew)[100][100], int (&ADM)[100][100], int len)
{
    for(int i=0;i<len;i+=1)
    {
        for(int j=0;j<len;j+=1)
            ADM[i][j]=ADMnew[i][j];
    }
}

int TR ( int A, int B, int ADM[][100])
{
    int trace=0;
    for(int i=0;i<A;i+=1)
    {
        for(int j=0;j<B;j+=1)
            {if(i==j) trace+=ADM[i][j];}
    }

    return trace;
}

```

- **IMAGE 1- THE FOLLOWING GRAPH WAS REFERENCED WHILE FINDING OUT THE CYCLES IN IT.**



- **OUTPUT 1**

```

C:\Users\harsh\Documents\discretGraph.exe
GRAPHS...
Implementing paths and cycles...
Enter the no. of vertices:6

Enter the name of node:
Vertex 1:a
Vertex 2:b
Vertex 3:c
Vertex 4:d
Vertex 5:e
Vertex 6:f

Your vertices are:
0:a
1:b
2:c
3:d
4:e
5:f

Enter D for directed graph and U for undirected graph:D

Enter the relation between the vertices:
Name of V1:a
Name of V2:b

Enter 1 for more edges or -1 for exit:1
Name of V1:c
  
```

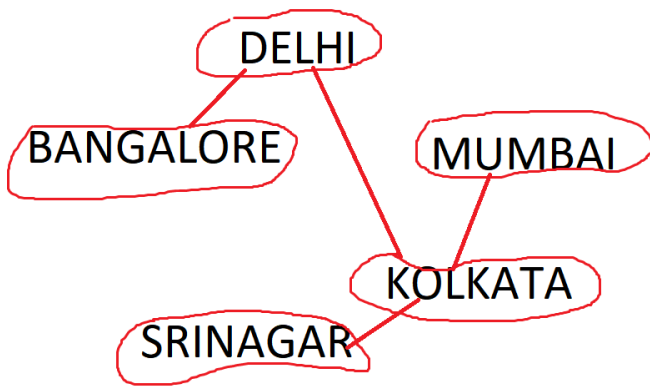
- **OUTPUT 2**

```
C:\Users\harsh\Documents\discretGraph.exe
Name of V1:c
Name of V2:e
Enter 1 for more edges or -1 for exit:1
Name of V1:e
Name of V2:a
Enter 1 for more edges or -1 for exit:1
Name of V1:d
Name of V2:c
Enter 1 for more edges or -1 for exit:1
Name of V1:a
Name of V2:f
Enter 1 for more edges or -1 for exit:1
Name of V1:f
Name of V2:b
Enter 1 for more edges or -1 for exit:1
Name of V1:f
Name of V2:c
Enter 1 for more edges or -1 for exit:1
```

- **OUTPUT 3**

```
C:\Users\harsh\Documents\discretGraph.exe
Name of V1:e
Name of V2:b
Enter 1 for more edges or -1 for exit:-1
The adjacency matrix is:
0 1 0 0 0 1
0 0 0 0 0 0
0 0 0 0 1 0
0 0 1 0 0 0
1 1 0 0 0 0
0 1 1 0 0 0
FINDING PATHS AND CYCLES-
1. TO FIND PATH BETWEEN TWO VERTICES
2. TO FIND NO. OF CYCLES
Enter choice:2
FINDING OUT CYCLES:
No. of cycles of length 3 are:0
No. of cycles of length 4 are:4
No. of cycles of length 5 are:0
No. of cycles of length 6 are:0
-----
Process exited after 44.78 seconds with return value 0
Press any key to continue . . .
```

- **IMAGE 2- THE FOLLOWING GRAPH WAS USED AS REFERENCE TO FIND THE PATH BETWEEN VERTICES**



```

C:\Users\harsh\Documents\discretGraph.exe
Enter 1 for more edges or -1 for exit:1
Name of V1:kolkata
Name of V2:delhi
Enter 1 for more edges or -1 for exit:1
Name of V1:srinagar
Name of V2:kolkata
Enter 1 for more edges or -1 for exit:1
Name of V1:srinagar
Name of V2:delhi
Enter 1 for more edges or -1 for exit:-1

The adjacency matrix is:
0 0 1 1 1
0 0 1 0 0
1 1 0 1 0
1 0 1 0 0
1 0 0 0 0

FINDING PATHS AND CYCLES-
1. TO FIND PATH BETWEEN TWO VERTICES
2. TO FIND NO. OF CYCLES
Enter choice:1
  
```

• OUTPUT 3

```
C:\Users\harsh\Documents\discretGraph.exe

The adjacency matrix is:
0 0 1 1 1
0 0 1 0 0
1 1 0 1 0
1 0 1 0 0
1 0 0 0 0

FINDING PATHS AND CYCLES-
1. TO FIND PATH BETWEEN TWO VERTICES
2. TO FIND NO. OF CYCLES
Enter choice:1

Enter the start vertex V1:delhi

Enter the end vertex V2:kolkata

Path exists.
-----
Process exited after 93.44 seconds with return value 0
Press any key to continue . . .
```