**Discrete Mathematics**

**(All Programs)**

**Made By: Avishkaar Pawar**

**Roll No : AD-1224**

**Ques1.** /\**Write a Program to create a SET A and determine the cardinality of SET*

*for an input array of elements (repetition allowed) and perform the following*

*operations on the SET:*

*a) ismember (a, A): check whether an element belongs to set or not and return value*

*as true/false.*

*b) powerset(A): list all the elements of power set of A*\*/

#include <iostream>

#include <math.h>

using namespace std;

bool ismember(int size,int A[]){

    int a;

    cout << "\nEnter the element to be searched: ";

    cin >> a;

    for(int i=0;i<size;i++)

    {

        if(A[i]==a)

            return true;

    }

    return false;

}

void PowerSet(int arr[], int size){

    int b[]={0,0,0,0};

    int r=pow(2,size);

    cout<<"\nPower Set\n";

    cout<<"{ ";

    for(int i=0; i<r; i++){

        int n=0;

        for(int l=i;l>0;l=l/2){

            b[n]=l%2;

            n++;

        }

        cout<<"{ ";

        for(int j=0;j<size;j++){

            if(b[j]==1){

                cout<<arr[j]<<"  ";

            }

        }

        if(i==0){

            cout<<" } , ";

        }else{

            cout<<"\b} , ";

        }

    }

    cout<<"\b } ";

    cout<<endl;

}

int main(){

    bool x;

    int size;

    char ch = 'y';

    while(ch == 'y')

    {

        cout << "\nEnter the size of set: ";

        cin >> size;

        int A[size];

        cout << "\nEnter the elements: ";

        for(int i=0;i<size;i++)

        {

            cin >> A[i];

        }

        int count = 1;

        for(int i=1;i<size;i++){

            int j=0;

            for(j=0;j<1;j++){

                if(A[i]==A[j]){

                    break;

                }

            }

            if(i==j){

                count++;

            }

        }

        cout<<"Cardinatlity of Given set : "<<count;

        x=ismember(size,A);

        if(x==true)

            cout << "\n\tValue is present!!!";

        else

            cout << "\n\tValue is not present!!!";

        cout << "\n\nThe possible subset pairs\n" << endl;

        PowerSet(A,size);

        cout << "\nDo you want to continue? (Y/N): ";

        cin >> ch;

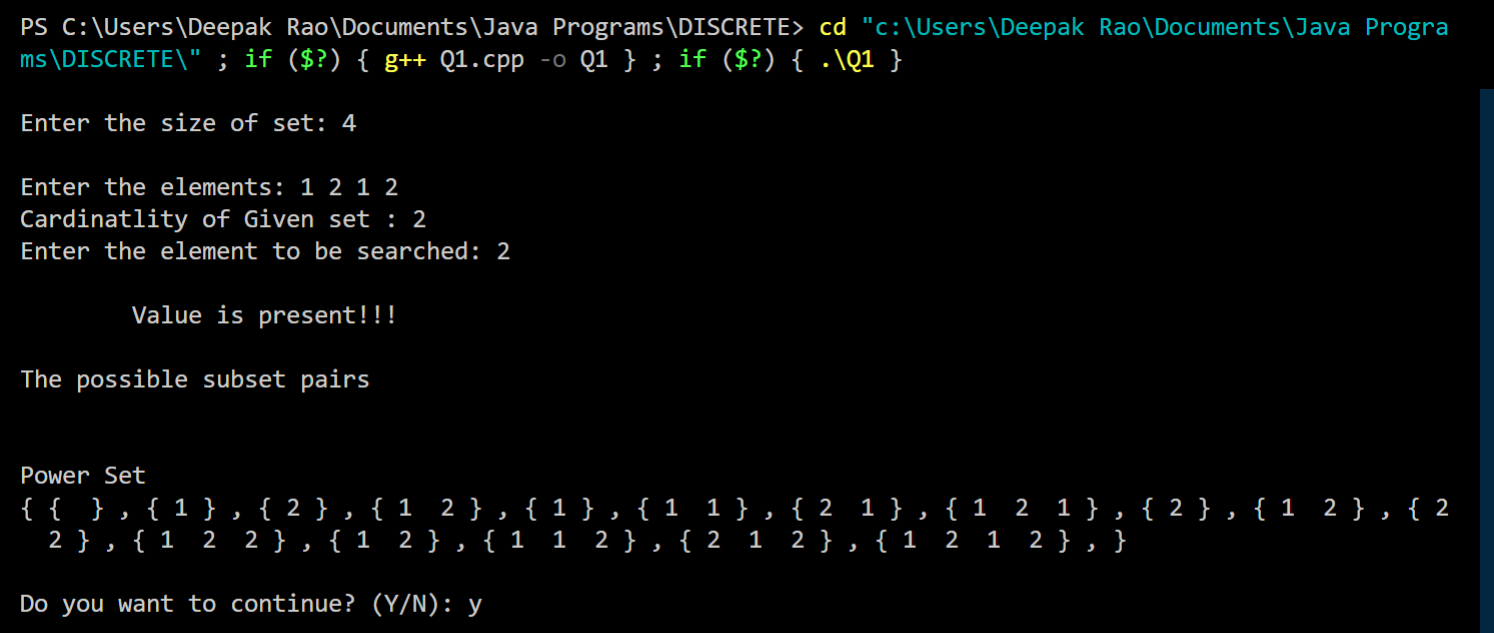
    }

    cout << "\n@@@@EXITING@@@";

    return 0;

}

**OUTPUT :-**

****

**Ques2.** /\**Create a class SET and take two sets as input from user to perform following SET*

*Operations:*

*a) Subset: Check whether one set is a subset of other or not.*

*b) Union and Intersection of two Sets.*

*c) Complement: Assume Universal Set as per the input elements from the user.*

*d) Set Difference and Symmetric Difference between two SETS*

*e) Cartesian Product of Sets.*\*/

#include <iostream>

using namespace std;

class SET

{

private:

    int i, j;

public:

    void Subset(int *\**arrA, int sizeA, int *\**arrB, int sizeB)

    {

        int c = 0;

        for (i = 0; i < sizeA; i++)

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

                if (arrA[i] == arrB[j])

                    c++;

        if (c != sizeA)

            cout << "SET A is not a subset of SET B" << endl;

        else

            cout << "SET A is a subset of SET B" << endl;

        int c1 = 0;

        for (i = 0; i < sizeB; i++)

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

                if (arrB[i] == arrA[j])

                    c1++;

        if (c != sizeB)

            cout << "SET B is not a subset of SET A" << endl;

        else

            cout << "SET B is a subset of SET A" << endl;

        cout << "-------------------------------------------------" << endl;

    }

    void UnionInter(int *\**setA, int sizeA, int *\**setB, int sizeB)

    {

        int uSize = sizeA + sizeB;

        int uSet[uSize];

        int unionSet[uSize];

        int iSet[uSize];

        int x = 0, y = 0;

        for (i = 0; i < sizeA; i++)

        {

            uSet[x] = setA[i];

            x++;

        }

        for (i = 0; i < sizeB; i++)

        {

            uSet[x] = setB[i];

            x++;

        }

        for (i = 0; i < x; i++)

        {

            for (j = i + 1; j < x; j++)

            {

                if (uSet[i] == uSet[j])

                {

                    iSet[y] = uSet[i];

                    y++;

                    for (int k = j; k < x - 1; k++)

                        uSet[k] = uSet[k + 1];

                    x--;

                }

                else

                    continue;

            }

        }

        cout << "Union of two sets is : {";

        for (i = 0; i < x; i++)

            cout << uSet[i] << " ";

        cout << "}";

        cout << endl;

        if (y != 0)

        {

            cout << "Intersection of two sets is : {";

            for (i = 0; i < y; i++)

                cout << iSet[i] << " ";

            cout << "}";

        }

        else

            cout << "No intersection found";

        cout << endl;

        cout << "-------------------------------------------------" << endl;

    }

    void Complement(int *\**setA, int sizeA, int *\**setB, int sizeB)

    {

        int sizeU;

        cout << "Enter the no. of elements of universal set : ";

        cin >> sizeU;

        cout << "Enter the elemnts of universal set : ";

        int U[sizeU];

        for (i = 0; i < sizeU; i++)

            cin >> U[i];

        int AC[sizeU], p = 0, c = 0;

        for (i = 0; i < sizeU; i++)

        {

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

            {

                if (U[i] == setA[j])

                    c++;

                else

                    continue;

            }

            if (c == 0)

            {

                AC[p] = U[i];

                p++;

            }

            c = 0;

        }

        cout << endl;

        cout << "Complement of SET A is : {";

        for (i = 0; i < p; i++)

            cout << AC[i] << " ";

        cout << "}" << endl;

        int BC[sizeU], q = 0, ctr = 0;

        for (i = 0; i < sizeU; i++)

        {

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

            {

                if (U[i] == setB[j])

                    ctr++;

                else

                    continue;

            }

            if (ctr == 0)

            {

                BC[q] = U[i];

                q++;

            }

            ctr = 0;

        }

        cout << "Complement of SET B is : {";

        for (i = 0; i < q; i++)

            cout << BC[i] << " ";

        cout << "}" << endl;

        cout << "-------------------------------------------------" << endl;

    }

    void setNSymDiff(int *\**setA, int sizeA, int *\**setB, int sizeB)

    {

        int ABDif[100], q = 0, ctr = 0;

        for (i = 0; i < sizeA; i++)

        {

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

            {

                if (setA[i] == setB[j])

                    ctr++;

                else

                    continue;

            }

            if (ctr == 0)

            {

                ABDif[q] = setA[i];

                q++;

            }

            ctr = 0;

        }

        cout << "Set difference A-B is : {";

        for (i = 0; i < q; i++)

            cout << ABDif[i] << " ";

        cout << "}" << endl;

        int BADif[100], p = 0, c = 0;

        for (i = 0; i < sizeB; i++)

        {

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

            {

                if (setB[i] == setA[j])

                    c++;

                else

                    continue;

            }

            if (c == 0)

            {

                BADif[p] = setB[i];

                p++;

            }

            c = 0;

        }

        cout << "Set difference B-A is : {";

        for (i = 0; i < p; i++)

            cout << BADif[i] << " ";

        cout << "}" << endl;

        int uSize = q + p;

        int symDif[uSize];

        int x = 0, y = 0;

        for (i = 0; i < q; i++)

        {

            symDif[x] = ABDif[i];

            x++;

        }

        for (i = 0; i < p; i++)

        {

            symDif[x] = BADif[i];

            x++;

        }

        cout << "Symmetric difference b/w two sets is : {";

        for (i = 0; i < x; i++)

            cout << symDif[i] << " ";

        cout << "}";

        cout << endl;

        cout << "-------------------------------------------------" << endl;

    }

    void cartesianPro(int *\**setA, int sizeA, int *\**setB, int sizeB)

    {

        int sizeAB, sizeBA, x = 0, y = 0;

        sizeAB = sizeA \* sizeB;

        sizeBA = sizeAB;

        int AB[sizeAB \* 2], BA[sizeBA \* 2];

        for (i = 0; i < sizeA; i++)

        {

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

            {

                AB[x++] = setA[i];

                AB[x++] = setB[j];

            }

        }

        for (i = 0; i < sizeB; i++)

        {

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

            {

                BA[y++] = setB[i];

                BA[y++] = setA[j];

            }

        }

        cout << "A X B = { ";

        for (i = 0; i < x; i++)

        {

            if (i % 2 == 0)

                cout << "(";

            cout << AB[i] << " ";

            if (i % 2 != 0)

                cout << ")";

        }

        cout << " }" << endl;

        cout << "B X A = { ";

        for (i = 0; i < y; i++)

        {

            if (i % 2 == 0)

                cout << "(";

            cout << BA[i] << " ";

            if (i % 2 != 0)

                cout << ")";

        }

        cout << " }" << endl;

        cout << "-------------------------------------------------" << endl;

    }

};

int main()

{

    cout << endl;

    int i, sizeA, sizeB;

    cout << "Enter the no. of elements in SET A : ";

    cin >> sizeA;

    int arrA[sizeA];

    cout << "Enter the elements : ";

    for (i = 0; i < sizeA; i++)

        cin >> arrA[i];

    cout << "Enter the no. of elements in SET B : ";

    cin >> sizeB;

    int arrB[sizeB];

    cout << "Enter the elements : ";

    for (i = 0; i < sizeB; i++)

        cin >> arrB[i];

    cout << "-------------------------------------------------" << endl;

    SET ob;

    cout << "\tSUBSET\n"

         << endl;

    ob.Subset(arrA, sizeA, arrB, sizeB);

    cout << "\tUNION and INTERSECTION\n"

         << endl;

    ob.UnionInter(arrA, sizeA, arrB, sizeB);

    cout << "\tCOMPLEMENT\n"

         << endl;

    ob.Complement(arrA, sizeA, arrB, sizeB);

    cout << "\tSET and SYMMETRIC DIFFERENCE\n"

         << endl;

    ob.setNSymDiff(arrA, sizeA, arrB, sizeB);

    cout << "\tCARTESIAN PRODUCT\n"

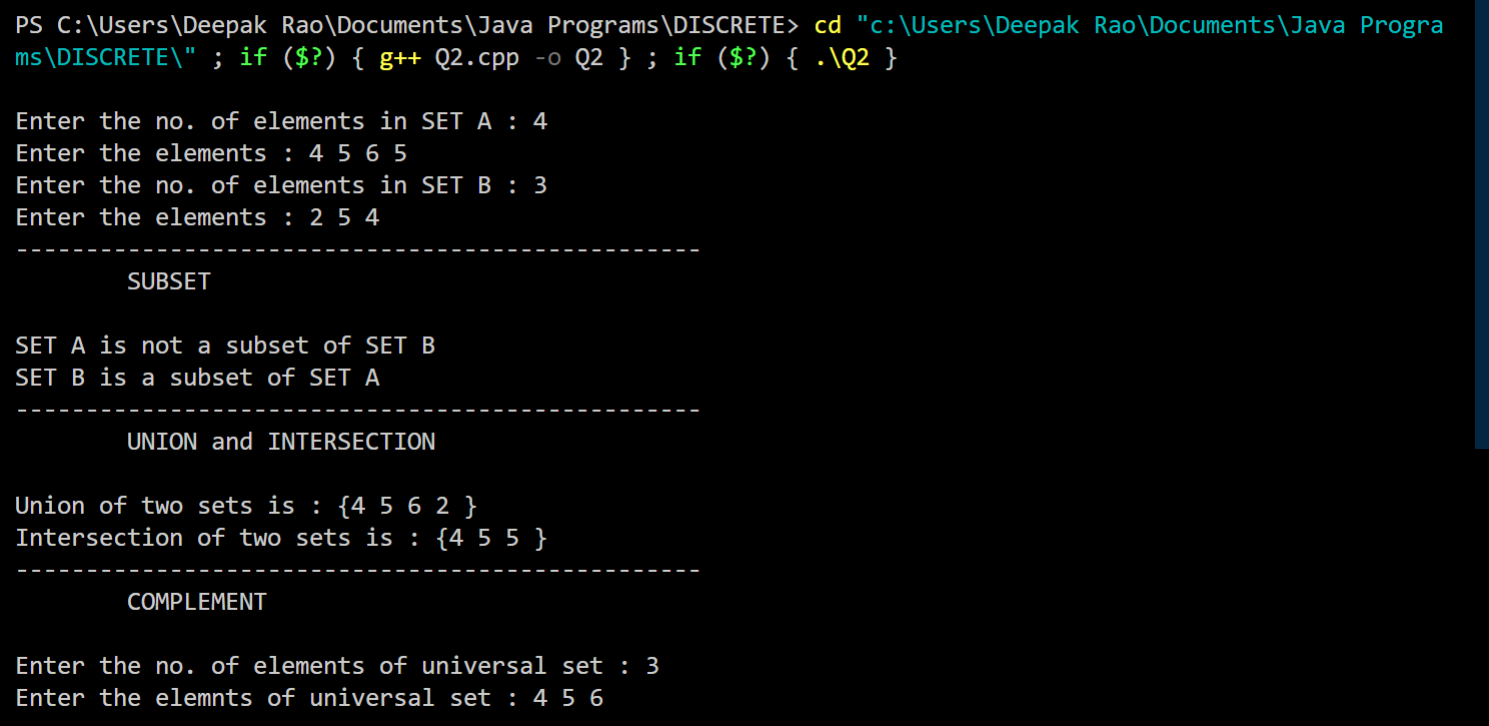
         << endl;

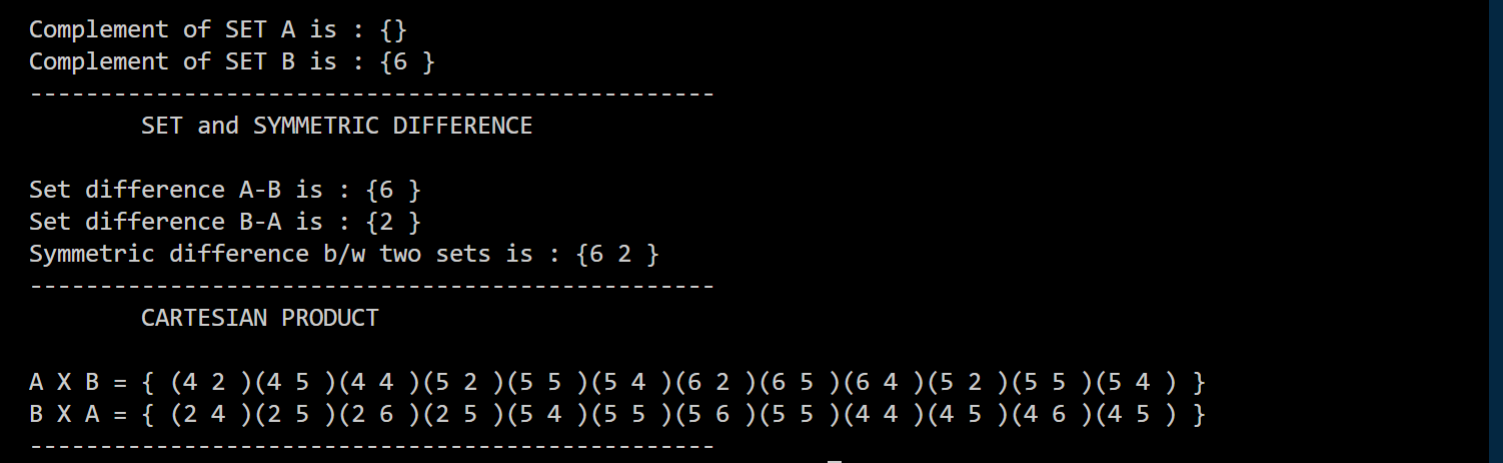
    ob.cartesianPro(arrA, sizeA, arrB, sizeB);

    return 0;

}

**OUTPUT:-**





**Ques3.** /\**Create a class RELATION, use Matrix notation to represent a relation.*

*Include functions to check if a relation is reflexive, Symmetric, Anti-symmetric*

*and Transitive. Write a Program to use this class.*\*/

#include<iostream>

#include<stdio.h>

#include<conio.h>

using namespace std;

class RELATION

{

    private:

    int i,j,k,x,y,z,ctr,iA,iB,nA,nR,\*A,\*R,\*\*RM,\*\*T;

    public:

    void empty();

    int inputSet();

    void inputRelation();

    void printSet();

    void printRelation();

    void Matrix();

    int reflexive();

    int symmetric();

    bool antiSymmetric();

    bool transitive();

};

void RELATION::empty()

{

    cout << "Set A is empty\n";

    printSet();

    cout << "Set A has no member.";

    cout << "\nHence, relation R is empty.\n";

    nR = 0;

    printRelation();

    cout << "Therefore, no matrix notation.";

    cout << "\nRelation R is NOT REFLEXIVE.";

    symmetric();

    antiSymmetric();

    transitive();

}

int RELATION::inputSet()

{

    cout << "Enter the size of SET A : ";

    cin >> nA;

    A = new int[nA];

    if(nA == 0)

        return 1;

    cout << "Enter the elements : ";

    for(i=0; i<nA; i++)

        cin >> A[i];

}

void RELATION::inputRelation()

{

    cout << "Enter the no of relations (R on A) : ";

    cin >> nR;

    R = new int[nR \* 2];

    cout << "Enter the relations in pair :\n";

    for(i=0; i<nR\*2; i++)

        cin >> R[i];

}

void RELATION::printSet()

{

    cout << "A = {";

    for(i=0; i<nA; i++)

        cout << A[i] << " ";

    cout << "}\n";

}

void RELATION::printRelation()

{

    cout << "R = {";

    for(i=0; i<nR\*2; i++)

    {

        if(i%2 == 0)

            cout << "(";

        cout << R[i] << " ";

        if(i%2 != 0)

            cout << ")";

    }

    cout << "}\n";

}

void RELATION::Matrix()

{

    cout << "\nMATRIX NOTATION\n\n";

    RM = new int \*[nA];

    for(i=0; i<nA; i++)

        RM[i]=new int[nA];

    for(i=0; i<nA; i++)

    {

        for(j=0; j<nA; j++)

        {

            RM[i][j]=0;

        }

    }

    for(i=0; i<nR\*2; i+=2)

    {

        for(j=0; j<nA; j++)

        {

            if(R[i] == A[j])

            {

                iA=j;

                break;

            }

        }

        for(k=0; k<nA; k++)

        {

            if(R[i+1] == A[k])

            {

                iB=k;

                break;

            }

        }

        RM[iA][iB]=1;

    }

    cout << "    ";

    for(int x=0; x<nA; x++)

        cout << " " << A[x] << " ";

    cout << endl << endl;

    for(i=0; i<nA; i++)

    {

        cout << A[i] << "  | ";

        for(j=0; j<nA; j++)

        {

            cout << RM[i][j] << "  ";

        }

        cout << "|";

        cout << endl;

    }

}

int RELATION::reflexive()

{

    x=0;

    for(i=0; i<nA; i++)

    {

        if(RM[i][i] == 1)

            x++;

    }

    if(x == nA)

    {

        cout << "\nRelation R is REFLEXIVE.";

        return x = 0;

    }

    else

    {

        cout << "\nRelation R is NOT REFLEXIVE.";

        return x = 1;

    }

}

int RELATION::symmetric()

{

    ctr = 0;

    for(i=0; i<nA; i++)

    {

        for(j=0; j<nA; j++)

        {

            if(RM[i][j] == RM[j][i])

                continue;

            else

            {

                ctr++;

                break;

            }

        }

    }

    if(ctr != 0)

        cout << "\nRelation R is NOT SYMMETRIC.";

    else

        cout << "\nRelation R is SYMMETRIC.";

    return ctr;

}

bool RELATION::antiSymmetric()

{

    bool flag = true;

    for(i=0; i<nR\*2; i+=2)

    {

        for(j=0; j<nR\*2; j+=2)

        {

            if((R[i] == R[j+1]) && (R[i+1] == R[j]))

                if(R[i] == R[i+1])

                {

                    continue;

                }

                else

                {

                    flag = false;

                }

        }

    }

    if(flag != true)

        cout << "\nRelation R is NOT ANTI-SYMMETRIC.";

    else

        cout << "\nRelation R is ANTI-SYMMETRIC.";

    return flag;

}

bool RELATION::transitive()

{

    bool flag = true;

    for(i=0; i<nR\*2; i+=2)

    {

        for(j=0; j<nR\*2; j+=2)

        {

            if(R[i+1] == R[j])

                for(k=0; k<nR\*2; k+=2)

                {

                    if((R[k] == R[i]) && (R[k+1] == R[j+1]))

                    {

                        flag = true;

                        break;

                    }

                    else

                        flag = false;

                }

        }

    }

    if(flag != true)

        cout << "\nRelation R is NOT TRANSITIVE.";

    else

        cout << "\nRelation R is TRANSITIVE.";

    return flag;

}

int main()

{

    int p = 0;

    RELATION ob;

    p = ob.inputSet();

    if(p == 1)

        ob.empty();

    else

    {

        ob.printSet();

        ob.inputRelation();

        ob.printRelation();

        ob.Matrix();

        ob.reflexive();

        ob.symmetric();

        ob.antiSymmetric();

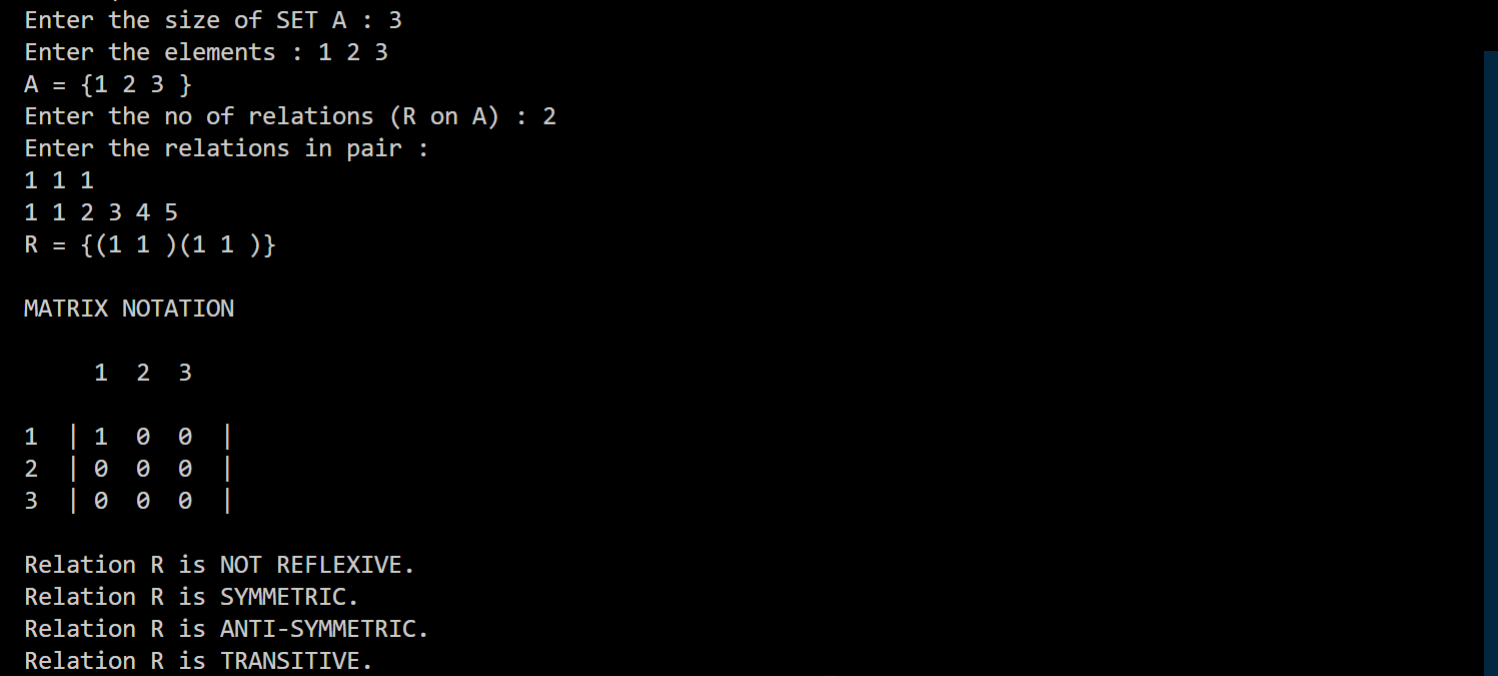
        ob.transitive();

    }

    return 0;

}

**OUTPUT:-**

****

**Ques4.** /\**Use the functions defined in Ques 3 to find check whether the given relation is:*

*a) Equivalent, or*

*b) Partial Order relation, or*

*c) None*\*/

#include<iostream>

#include "Q3.cpp"

using namespace std;

class checkRELATION : public RELATION

{

    public:

    int equivalent(int, int, bool);

    int partialOrder(int, bool, bool);

    void neither(int, int);

};

int checkRELATION::equivalent(int r, int s, bool t)

{

    if((r == 0) && (s == 0) && (t == true))

        cout << "\nRelation R is EQUIVALENT relation";

    else

        return 0;

    return 1;

}

int checkRELATION::partialOrder(int r, bool a, bool t)

{

    if((r == 0) && (a == true) && (t == true))

        cout << "\nRelation R is PARTIAL ORDER relation";

    else

        return 0;

    return 1;

}

void checkRELATION::neither(int e, int po)

{

    if((e != 1) && (po != 1))

        cout << "\nRelation R is NEITHER equivalent NOR partial order relation";

}

int main()

{

    int p=0,r,s,e,po;

    bool a,t;

    checkRELATION ob1;

    p = ob1.inputSet();

    if(p == 1)

    {

        ob1.empty();

    }

    else

    {

        ob1.printSet();

        ob1.inputRelation();

        ob1.printRelation();

        ob1.Matrix();

        r = ob1.reflexive();

        s = ob1.symmetric();

        a = ob1.antiSymmetric();

        t = ob1.transitive();

    }

    e = ob1.equivalent(r, s, t);

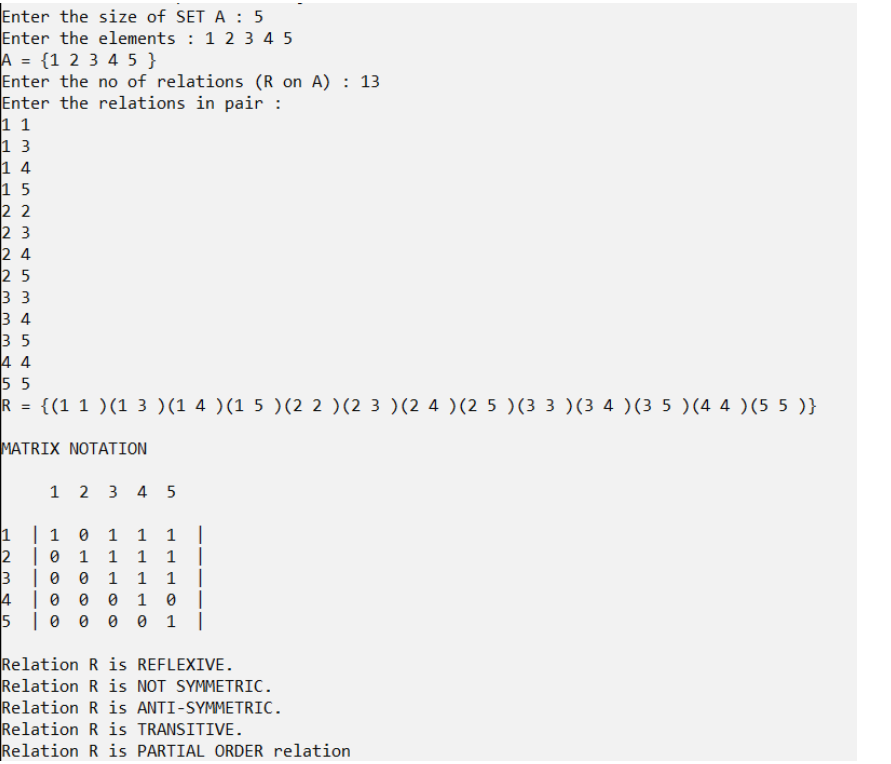
    po = ob1.partialOrder(r, a, t);

    ob1.neither(e, po);

    return 0;

}

**OUTPUT:-**

****

**Ques5.** /\**Write a Program to generate the Fibonacci Series using recursion*\*/

#include<iostream>

using namespace std;

int fibonacci(int num)

{

    if((num == 1) || (num == 0))

        return(num);

    else

        return(fibonacci(num-1) + fibonacci(num-2));

}

int main()

{

    int num,i=0;

    cout << "\nEnter the limit: ";

    cin >> num;

    cout << "\nFibonacci Series: ";

    while(i<num)

    {

        cout << " " << fibonacci(i);

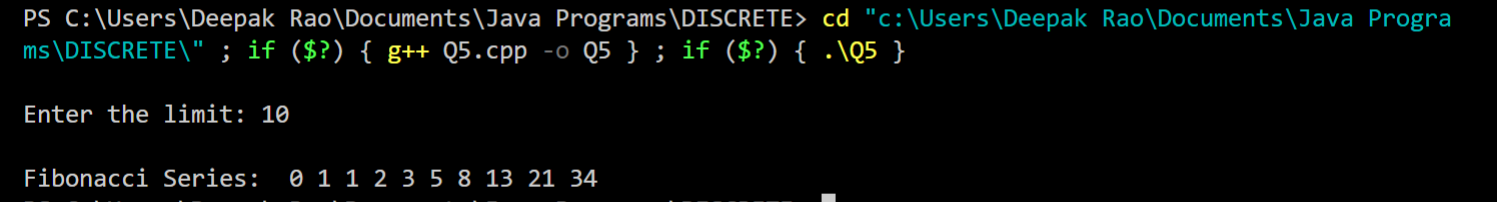
        i++;

    }

    return 0;

}

**OUTPUT:-**

****

**Ques6.** /\**Write a Program to implement Tower of Hanoi using recursion*\*/

#include<iostream>

using namespace std;

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

    if(n == 1)

    {

        cout << "Move disk 1 from rod " << from\_rod << " to rod " << to\_rod << endl;

        return;

    }

    towerOfHanoi(n-1, from\_rod, aux\_rod, to\_rod);

    cout << "Move disk " << n << " from rod " << from\_rod << " to rod " << to\_rod << endl;

    towerOfHanoi(n-1, aux\_rod, to\_rod, from\_rod);

}

int main()

{

    int n;

    cout << "\nEnter the number of disks: ";

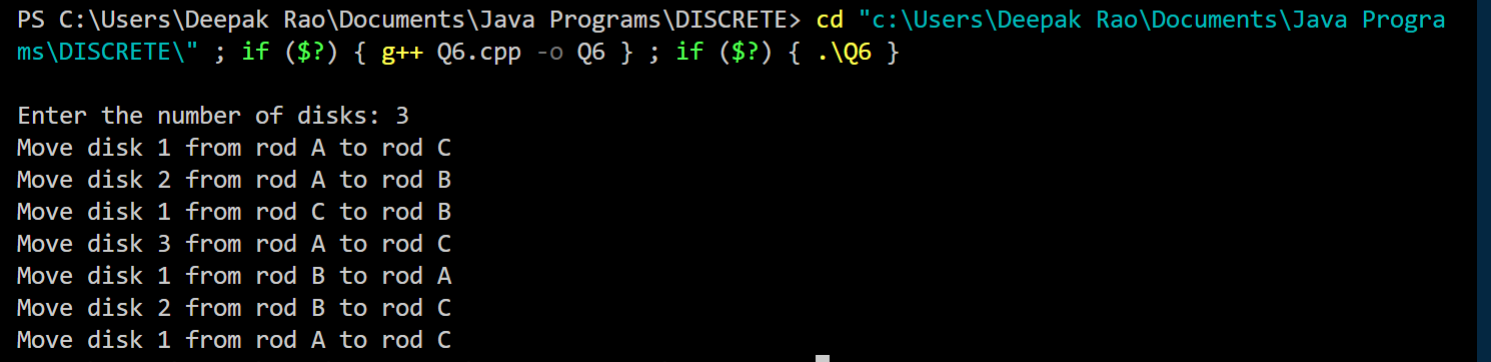
    cin >> n;

    towerOfHanoi(n, 'A', 'C', 'B');

    return 0;

}

**OUTPUT:-**



**Ques7.** /\**Write a Program to implement binary search using recursion*\*/

#include <iostream>

using namespace std;

int BinarySearch(int arr[], int num, int beg, int end)

{

    int mid;

    if (beg > end)

    {

        cout << "\nNumber is not found";

        return 0;

    }

    else

    {

        mid = (beg + end) / 2;

        if(arr[mid] == num)

        {

            cout << "\nNumber is found at " << mid+1 << " position.\n";

            return 0;

        }

        else if (num > arr[mid])

        {

            BinarySearch (arr, num, mid+1, end);

        }

        else if (num < arr[mid])

        {

        BinarySearch (arr, num, beg , mid-1);

        }

    }

}

int main()

{

    int arr[100], num, i, n, beg, end;

    cout <<"\nEnter the size of an array (Max 100): ";

    cin >> n;

    cout <<"\nEnter the sorted values: ";

    for(i=0; i<n; i++)

    {

        cin >> arr[i];

    }

    cout <<"\nEnter a value to be search: ";

    cin >> num;

    beg = 0;

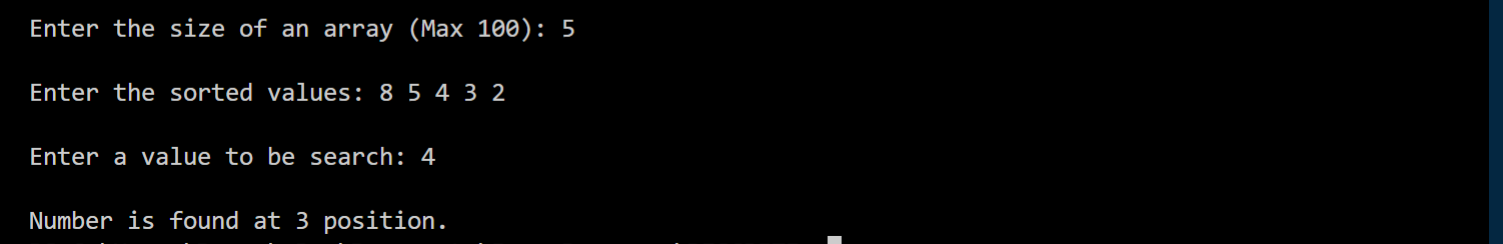
    end = n-1;

    BinarySearch (arr, num, beg, end);

    return 0;

}

**OUTPUT:-**

****

**Ques8.** /\**Write a Program to implement Bubble Sort. Find the number of comparisons during*

*each pass and display the intermediate result. Use the observed values to plot a*

*graph to analyse the complexity of algorithm.*\*/

#include<iostream>

#include<stdio.h>

#include<conio.h>

#include<cstdlib>

using namespace std;

int i,j,k;

void bubbleSort(int*\**, int);

int main()

{

    int size,ele;

    cout << "\nEnter the size of array: ";

    cin >> size;

    int array[size];

    cout << "\nWORST CASE:";

    cout << "\n-------------\n";

    for(i=0; i<size; i++)

        array[i] = size - i;

    bubbleSort(array, size);

    cout << "\n\nBEST CASE:";

    cout << "\n-------------\n";

    for(i=0; i<size; i++)

        array[i] = i+1;

    bubbleSort(array, size);

    cout << "\n\nAVERAGE CASE:";

    cout << "\n-------------\n";

    for(i=0; i<size; i++)

    {

        ele = ((int)rand()%10);

        if(ele == 0)

            continue;

        else

            array[i] = ele;

    }

    bubbleSort(array, size);

    return 0;

}

void bubbleSort(int *\**array, int size)

{

    int temp = 0;

    int ctr = 0;

    int totalCom = 0;

    cout << "Array: ";

    for(i=0; i<size; i++)

        cout << array[i] << " ";

    cout << endl << endl;

    for(i=0; i<size-1; i++)

    {

        ctr = 0;

        for(j=0; j<size-i-1; j++)

        {

            if(array[j+1] < array[j])

            {

                temp = array[j];

                array[j] = array[j+1];

                array[j+1] = temp;

            }

            ctr++;

            totalCom++;

        }

        cout << "After pass " << i+1 << ": ";

        for(k=0; k<size; k++)

            cout << array[k] << " ";

        cout << "\nComparisions made in pass " << i+1 << ": " << ctr;

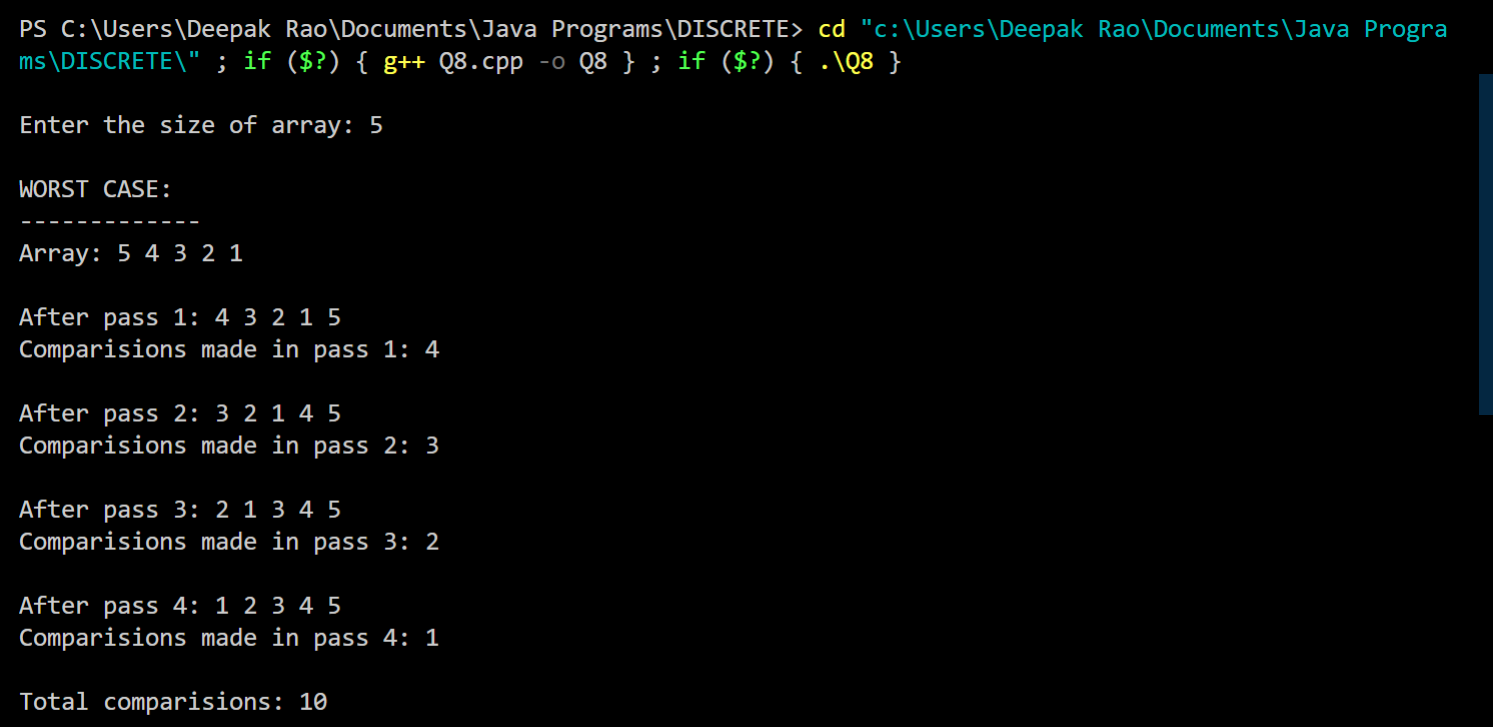
        cout << endl << endl;

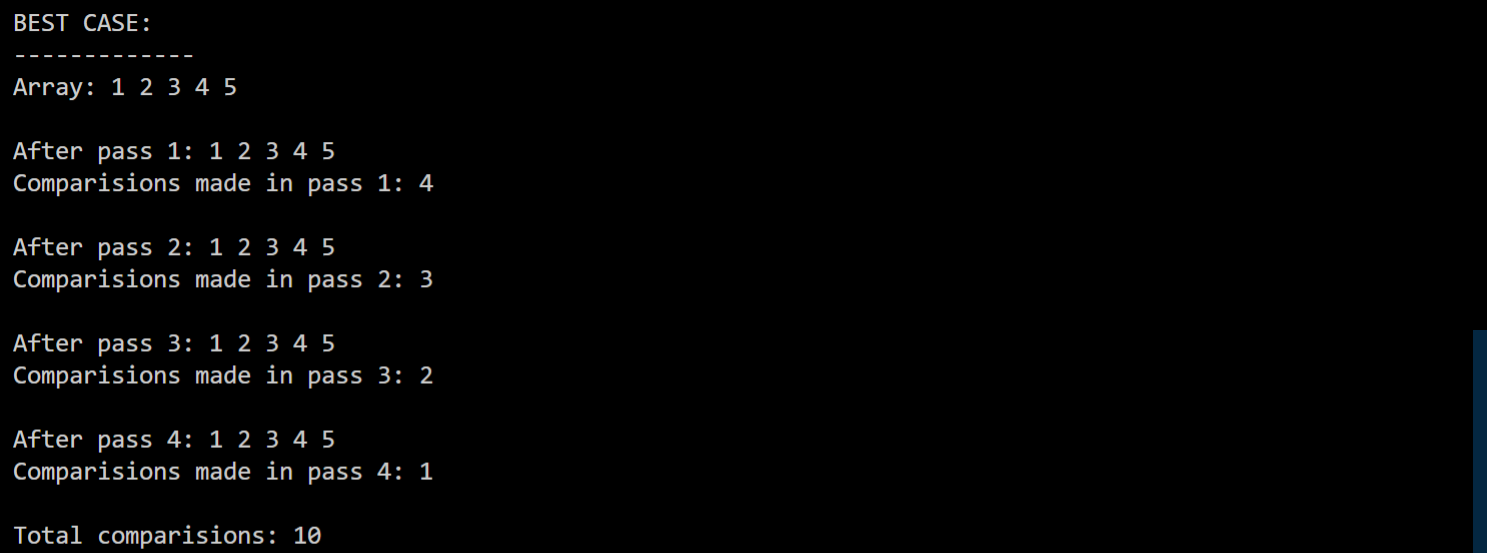
    }

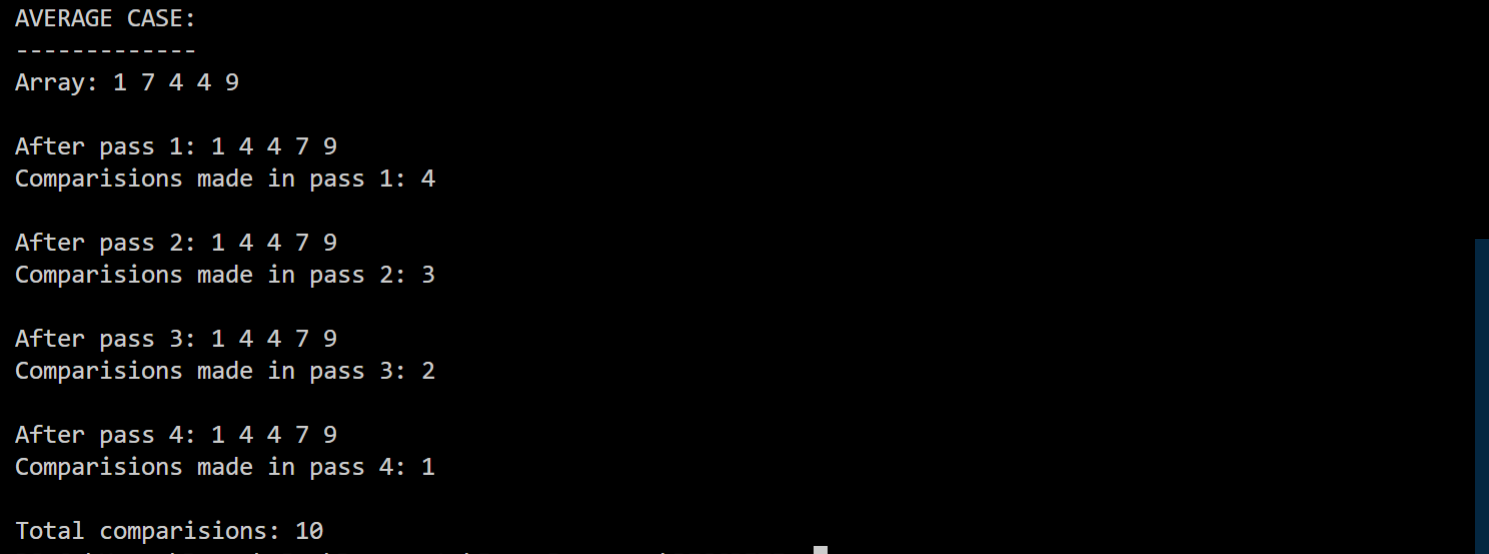
    cout << "Total comparisions: "  << totalCom;

}

**OUTPUT:-**







**Ques9.** /\**Write a Program to implement Insertion Sort. Find the number of comparisons during*

*each pass and display the intermediate result. Use the observed values to plot a graph*

*to analyse the complexity of algorithm*\*/

#include<iostream>

#include<stdio.h>

#include<conio.h>

#include<cstdlib>

using namespace std;

int i,j,k;

void insertionSort(int*\**, int);

int main()

{

    int size,ele;

    cout << "\nEnter the size of array: ";

    cin >> size;

    int array[size];

    cout << "\nWORST CASE:";

    cout << "\n-------------\n";

    for(i=0; i<size; i++)

        array[i] = size - i;

    insertionSort(array, size);

    cout << "\n\nBEST CASE:";

    cout << "\n-------------\n";

    for(i=0; i<size; i++)

        array[i] = i+1;

    insertionSort(array, size);

    cout << "\n\nAVERAGE CASE:";

    cout << "\n-------------\n";

    for(i=0; i<size; i++)

    {

        ele = ((int)rand()%10);

        if(ele == 0)

            continue;

        else

            array[i] = ele;

    }

    insertionSort(array, size);

    return 0;

}

void insertionSort(int*\** array, int size)

{

    int temp=0;

    int ctr = 0;

    int totalCom = 0;

    cout << "Array: ";

    for(i=0; i<size; i++)

        cout << array[i] << " ";

    cout << endl << endl;

    for(i=1; i<size; i++)

    {

        temp = array[i];

        ctr = 0;

        for(j=i-1; j>=0; j--)

        {

            ctr++;

            totalCom++;

            if(array[j] > temp)

            {

                array[j+1] = array[j];

            }

            else

                break;

        }

        array[j+1] = temp;

        cout << "After pass " << i << ": ";

        for(k=0; k<size; k++)

            cout << array[k] << " ";

                cout << "\nComparisions made in pass " << i << ": " << ctr;

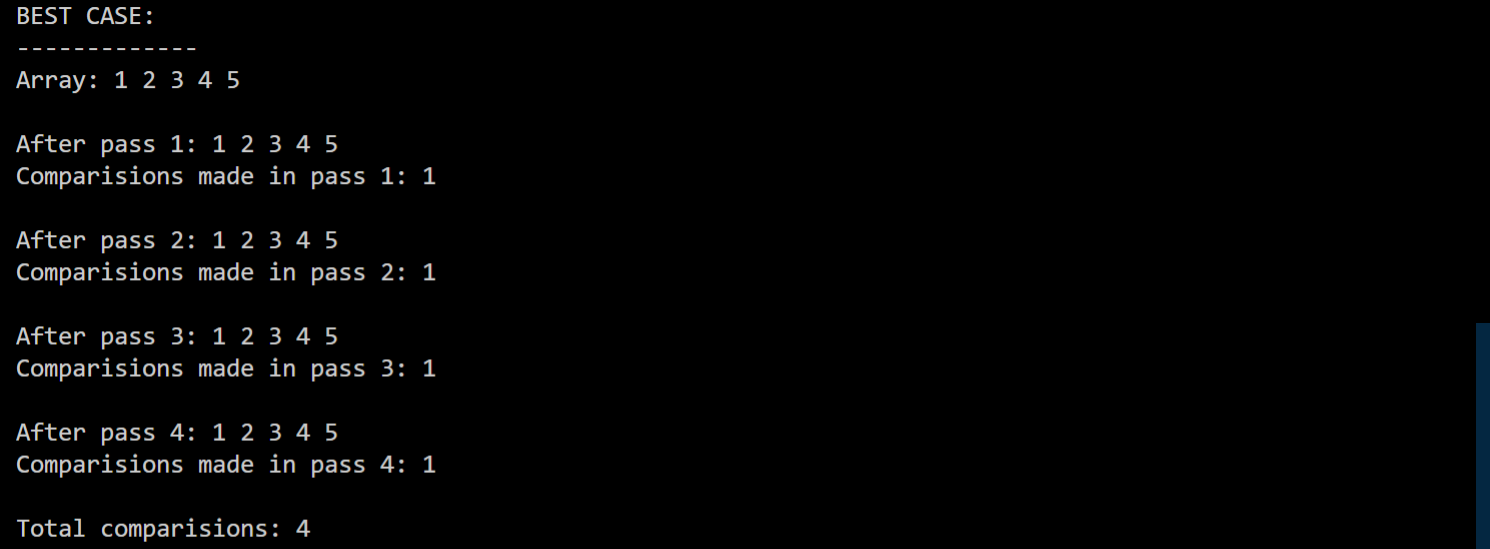
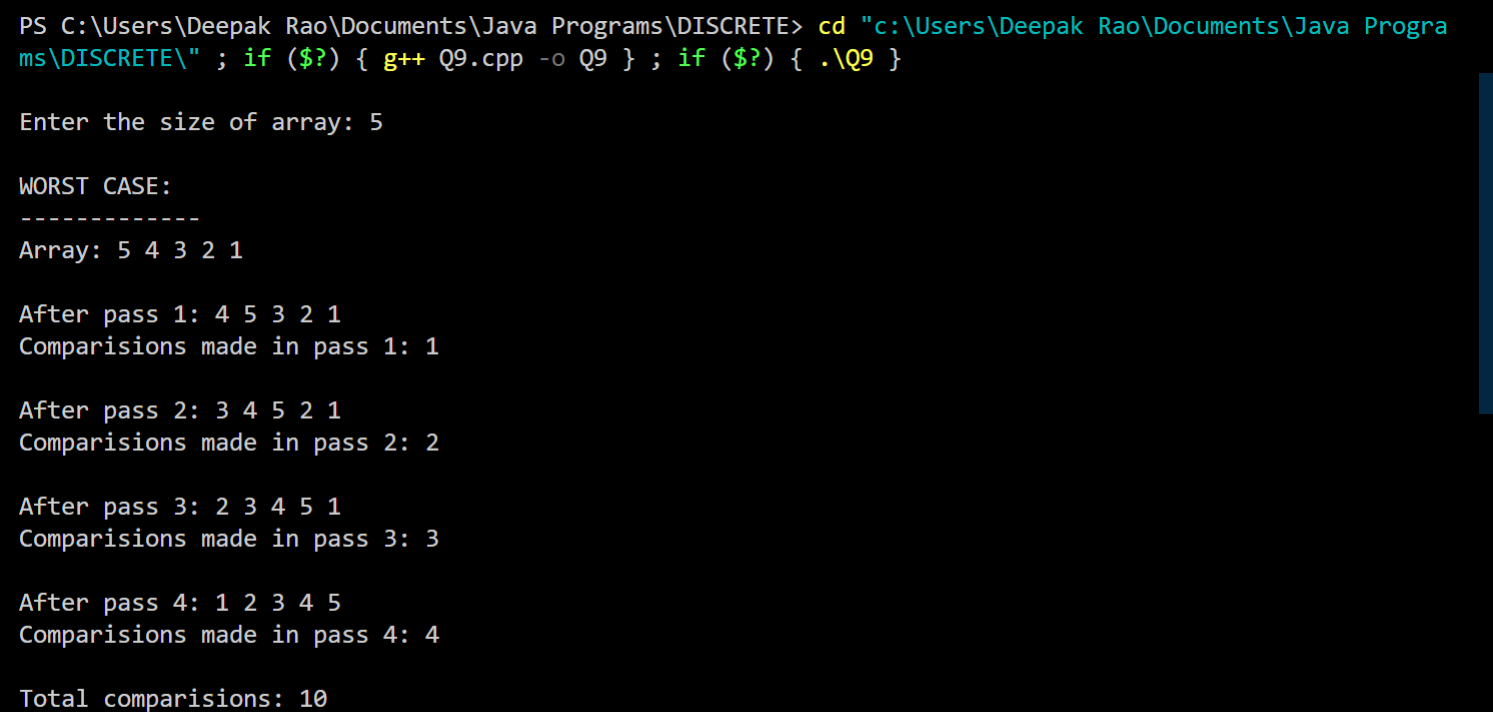
        cout << endl << endl;

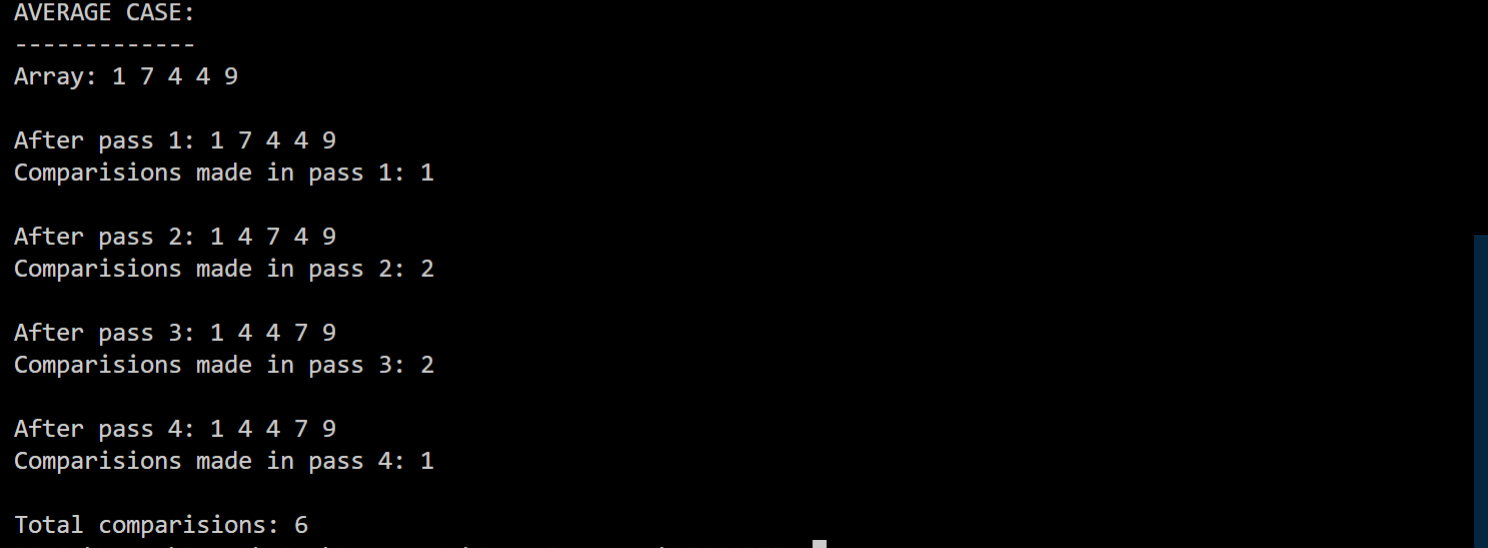
    }

    cout << "Total comparisions: "  << totalCom;

}

**OUTPUT:-**

****



**Ques10.** /\**Write a Program that generates all the permutations of a given set of digits, with*

*or without repetition. (For example, if the given set is {1,2}, the permutations are*

*12 and 21). (One method is given in Liu)*\*/

#include<iostream>

#include<stdio.h>

#include<conio.h>

#define MAX\_DIM 100

using namespace std;

void withRepetition(int*\**, int);

void withoutRepetition(int*\**, int);

void printWithRepetition(int*\**, int, int*\**, int, int);

void printWithoutRepetition(int*\**, int, int, int);

void swap(int *&*, int *&*);

int main()

{

    int size;

    char ch;

    cout << "Enter the size of set: ";

    cin >> size;

    int array[MAX\_DIM];

    cout << "Enter the elements: ";

    for(int i=0; i<size; i++)

        cin >> array[i];

    cout << "\nIs repetition allowed (Y/N): ";

    cin >> ch;

    switch(ch)

    {

        case 'Y':

            withRepetition(array, size);

            break;

        case 'N':

            withoutRepetition(array, size);

            break;

        default:

            cout << "\nWrong Choice";

    }

    return 0;

}

void withRepetition(int*\** array, int size)

{

    int data[MAX\_DIM] = {0};

    printWithRepetition(array, size, data, size-1, 0);

    cout << endl;

}

void printWithRepetition(int*\** array, int size, int *\**data, int last, int index)

{

    for(int i=0; i<size; i++)

    {

        data[index] = array[i];

        if(index == last)

        {

            cout << "{";

            for(int j=0; j<index+1; j++)

                cout << data[j] << " ";

            cout << "}";

        }

        else

        {

            printWithRepetition(array, size, data, last, index+1);

        }

    }

}

void withoutRepetition(int*\** array, int size)

{

    printWithoutRepetition(array, size, 0, size-1);

    cout << endl;

}

void printWithoutRepetition(int*\** array, int size, int start, int end)

{

    if(start == end)

    {

        cout << "{";

        for(int i=0; i<size; i++)

            cout << array[i] << " ";

        cout << "}";

    }

    else

    {

        for(int i=start; i<end+1; i++)

        {

            swap(array[start], array[i]);

            printWithoutRepetition(array, size, start+1, end);

            swap(array[start], array[i]);

        }

    }

}

void swap(int *&*a, int *&*b)

{

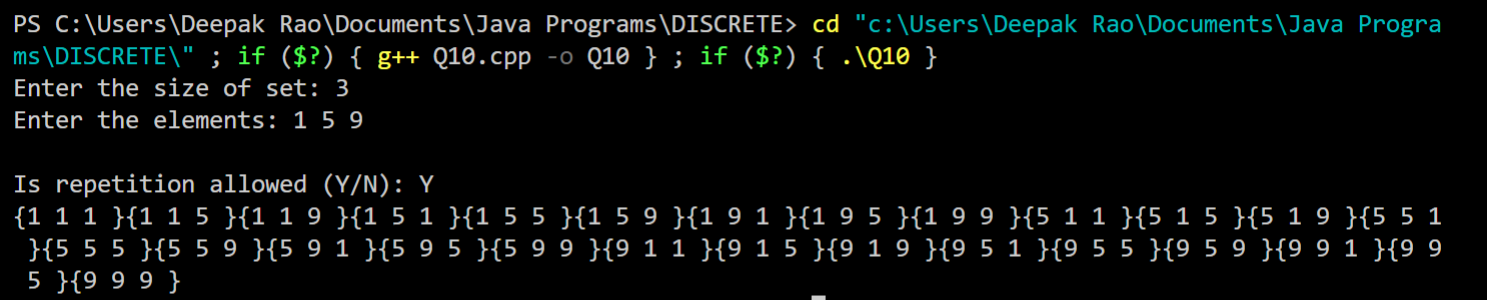
    int t = b;

    b = a;

    a = t;

}

**OUTPUT:-**

****

**Ques11.** /\**Write a Program to calculate Permutation and Combination for an input value n and r*

*using recursive formula of nCr and nPr*\*/

#include<iostream>

using namespace std;

int nCr(int, int);

int nPr(int, int);

int nPr(int n, int r)

{

    if(r == 0)

        return 1;

    if(r > n)

        return 0;

    return nPr(n-1, r) + r \* nPr(n-1, r-1);

}

int nCr(int n, int r)

{

    if(r ==0 || r ==n)

        return 1;

    return nCr(n-1, r) + nCr(n-1, r-1);

}

int main()

{

    int n,r;

    cout << "\nEnter the value of n: ";

    cin >> n;

    cout << "\nEnter the value of r: ";

    cin >> r;

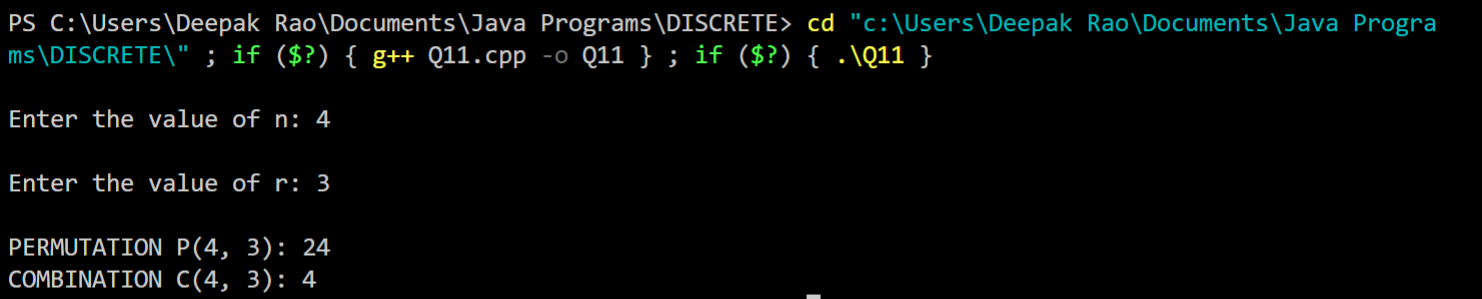
    cout << "\nPERMUTATION " << "P(" << n << ", " << r << "): " << nPr(n, r);

    cout << "\nCOMBINATION " << "C(" << n << ", " << r << "): " << nCr(n, r);

    return 0;

}

**OUTPUT:-**

****

**Ques12.** /\**For any number n, write a program to list all the solutions of the equation x1 + x2*

*+ x3 + …+ xn = C, where C is a constant (C<=10) and x1, x2,x3,…,xn are nonnegative*

*integers using brute force strategy.* \*/

#include<iostream>

using namespace std;

void bruteForce(int*\**, int, int*\**, int, int, int, int*&*);

int main()

{

    int n, C, counter = 0, size = 11;

    int arr[size], data[100] = {0};

    cout << "\nFinding solutions to x1 + x2 + ... + xn = C\n";

    cout << "Enter the value of n: ";

    cin >> n;

    for (int i=0; i <= 10; i++)

        arr[i] = i;

    cout << "Enter the sum constant (C <= 10): ";

    cin >> C;

    cout << "Possible Non-negative Integral solutions [ ";

    for(int i=0; i<n; i++)

        cout << "x" << i+1 << " ";

    cout << " ] :" << endl;

    bruteForce(arr, size, data, n-1, 0, C, counter);

    cout << "\nFound " << counter << " Solutions\n";

    return 0;

}

void bruteForce(int*\** arr, int size, int*\** data, int last, int index, int C, int *&*counter)

{

    for(int i=0; i<size; i++)

    {

        data[index] = arr[i];

        if(index == last)

        {

            int sum = 0;

            for(int j=0; j<index+1; j++)

                sum += data[j];

            if(sum == C)

            {

                cout << "[ ";

                for(int j=0; j<index+1; j++)

                    cout << data[j] << " ";

                cout << "] ";

                counter++;

            }

        }

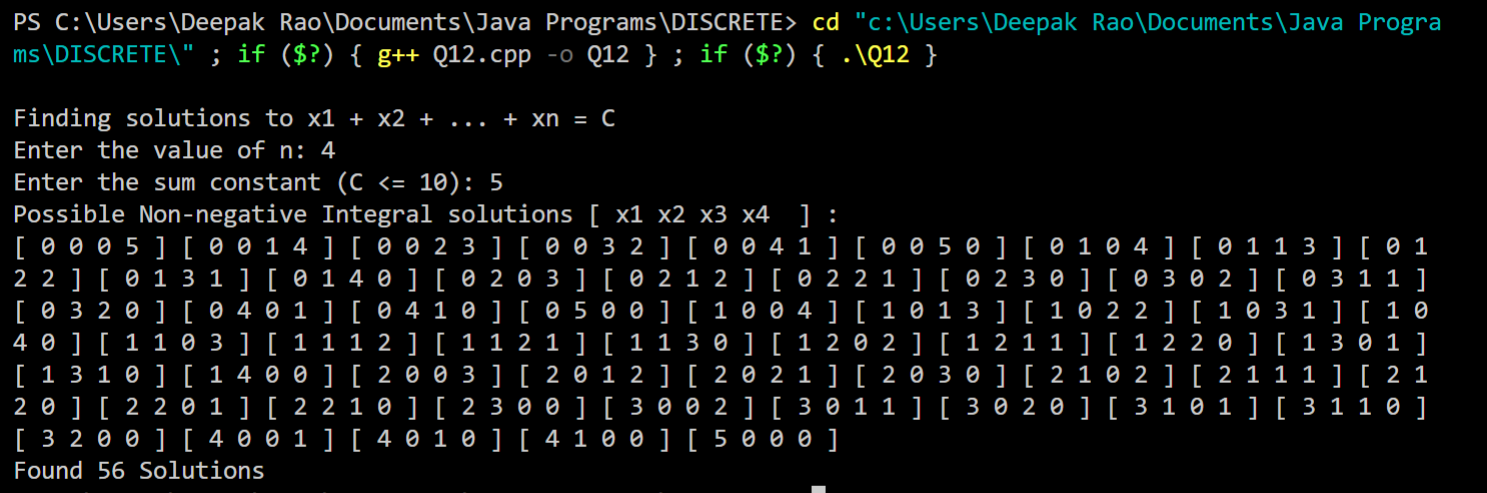
        else

            bruteForce(arr, size, data, last, index+1, C, counter);

    }

}

**OUTPUT:-**

****

**Ques13.** /\**Write a Program to accept the truth values of variables x and y, and print the*

*truth table of the following logical operations:*

*a) Conjunction f) Exclusive NOR*

*b) Disjunction g) Negation*

*c) Exclusive OR h) NAND*

*d) Conditional i) NOR*

*e) Bi-conditional*\*/

#include<iostream>

#include<stdio.h>

#include<conio.h>

using namespace std;

int main()

{

    int n;

    char x,y;

    cout << "Enter the no. of trials: ";

    cin >> n;

    bool value[n][2];

    for(int i=0; i<n; i++)

    {

        cout << "Enter the truth value for x" << i+1 << " y" << i+1 << ": ";

        cin >> x >> y;

        value[i][0] = (x == 't' || x == 'T');

        value[i][1] = (y == 't' || y == 'T');

    }

    cout << endl;

    cout << "x\ty\tAND\tOR\tXOR\tx->y\tx<->y\tXNOR\tNOT\tNAND\tNOR";

    cout << "\n-------------------------------------------------------"

         << "-------------------------------------------------------\n";

    for(int i=0; i<n; i++)

    {

        int x = value[i][0], y = value[i][1];

        cout << (x ? "T" : "F") << "\t" << (y ? "T" : "F") << "\t"

            << ((x && y) ? "T" : "F") << "\t"

            << ((x || y) ? "T" : "F") << "\t"

            << (((x || y) && !(x && y)) ? "T" : "F") << "\t"

            << ((!x || y) ? "T" : "F") << "\t"

            << (((!x || y) && (!y || x)) ? "T" : "F") << "\t"

            << ((!((x || y) && !(x && y))) ? "T" : "F") << "\t"

            << ((!x) ? "T" : "F") << " " << ((!y) ? "T" : "F") << "\t"

            << (!(x && y) ? "T" : "F") << "\t"

            << (!(x || y) ? "T" : "F") << "\n";

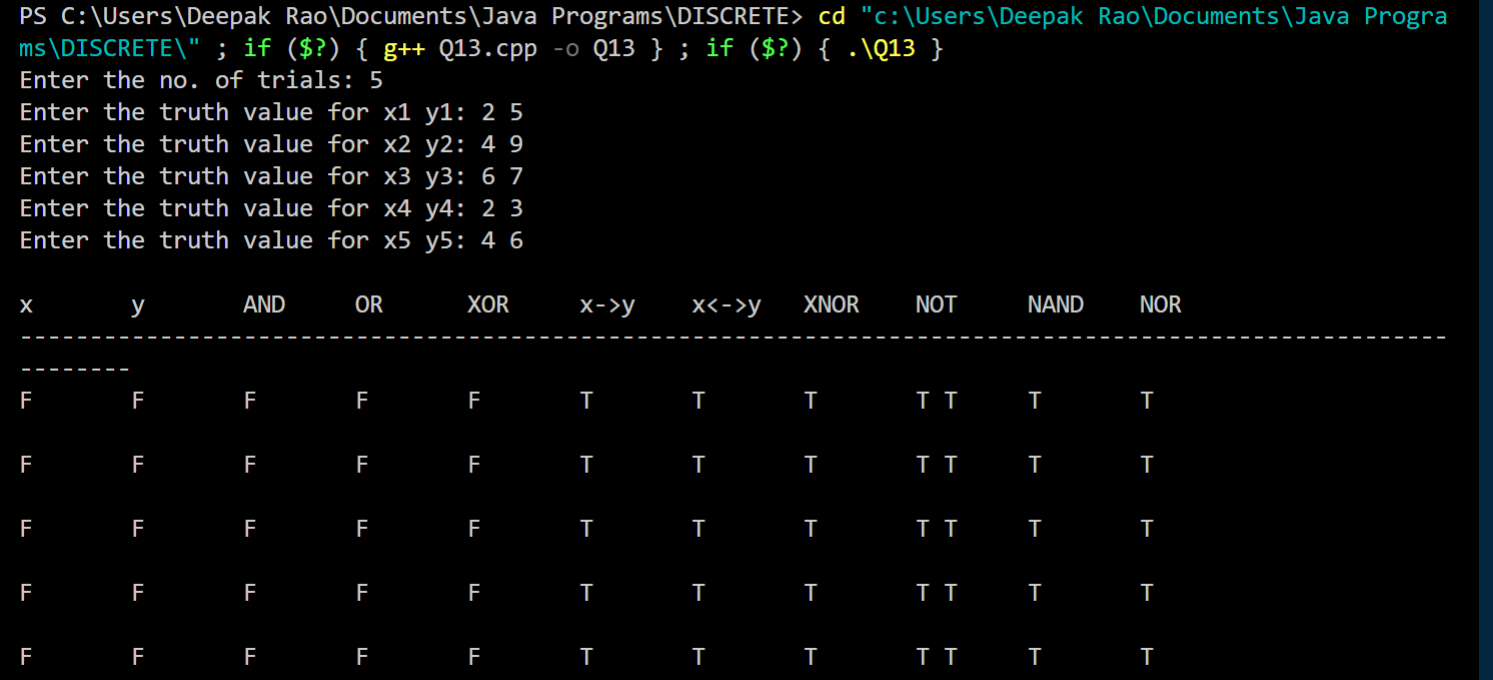
        cout << endl;

    }

    return 0;

}

**OUTPUT:-**

****

**Ques14.** /\**Write a program to accept an input n from the user and graphically represent the*

*values of T(n) where n varies from 0 to n for the recurrence relations. For e.g.*

*T(n) = T(n-1) + n, T(0) = 1, T(n) = T(n-1) + n^2, T(0) =1, T(n) = 2\*T(n)/2 + n,*

*T(1)=1*\*/

#include<iostream>

using namespace std;

int firstRecurrence(int n)

{

    if(n == 0)

        return 1;

    return firstRecurrence(n-1) + n;

}

int secondRecurrence(int n)

{

    if(n == 0)

        return 1;

    return secondRecurrence(n-1) + n\*n;

}

int thirdRecurrence(int n)

{

    if(n == 1)

        return 1;

    return 2 \* thirdRecurrence(n/2) + n;

}

int main()

{

    int n,ch;

    cout << "\nChoose recurrence relation to evaluate:\n"

         << "(1) T(n) = T(n - 1) + n and T(0) = 1\n"

         << "(2) T(n) = T(n - 1) + n^2 and T(0) = 1\n"

         << "(3) T(n) = 2 \* T(n / 2) + n and T(1) = 1\n";

    cout << "Enter the choice: ";

    cin >> ch;

    switch(ch)

    {

        case 1:

            cout << "\nEnter the value of n: ";

            cin >> n;

            cout << "\nValues for T(n) = T(n - 1) + n:\n";

            for(int i=0; i<=n; i++)

            {

                if(i == 0)

                    cout << "T(0) = " << firstRecurrence(i) << endl;

                else

                    cout << "T(" << i << ") = T(" << (i-1) << ") + "

                         << i << " = "

                         << firstRecurrence(i) << endl;

            }

        break;

        case 2:

            cout << "\nEnter the value of n: ";

            cin >> n;

            cout << "\nValues for T(n) = T(n - 1) + n^2:\n";

            for(int i=0; i<=n; i++)

            {

                if(i == 0)

                    cout << "T(0) = " << secondRecurrence(i) << endl;

                else

                    cout << "T(" << i << ") = T(" << (i-1) << ") + "

                         << i\*i << " = "

                         << secondRecurrence(i) << endl;

            }

        break;

        case 3:

            cout << "\nEnter the value of n: ";

            cin >> n;

            cout << "\nValues for T(n) = 2 \* T(n / 2) + n:\n";

            for(int i=1; i<=n; i++)

            {

                if(i == 1)

                    cout << "T(1) = " << thirdRecurrence(i) << endl;

                else

                    cout << "T(" << i << ") = 2 \* T(" << i << " / 2) + "

                         << i << " = " << "2 \* T(" << i/2 << ") + "

                         << i << " = "

                         << thirdRecurrence(i) << endl;

            }

        break;

        default:

            cout << "\nWrong choice!!!";

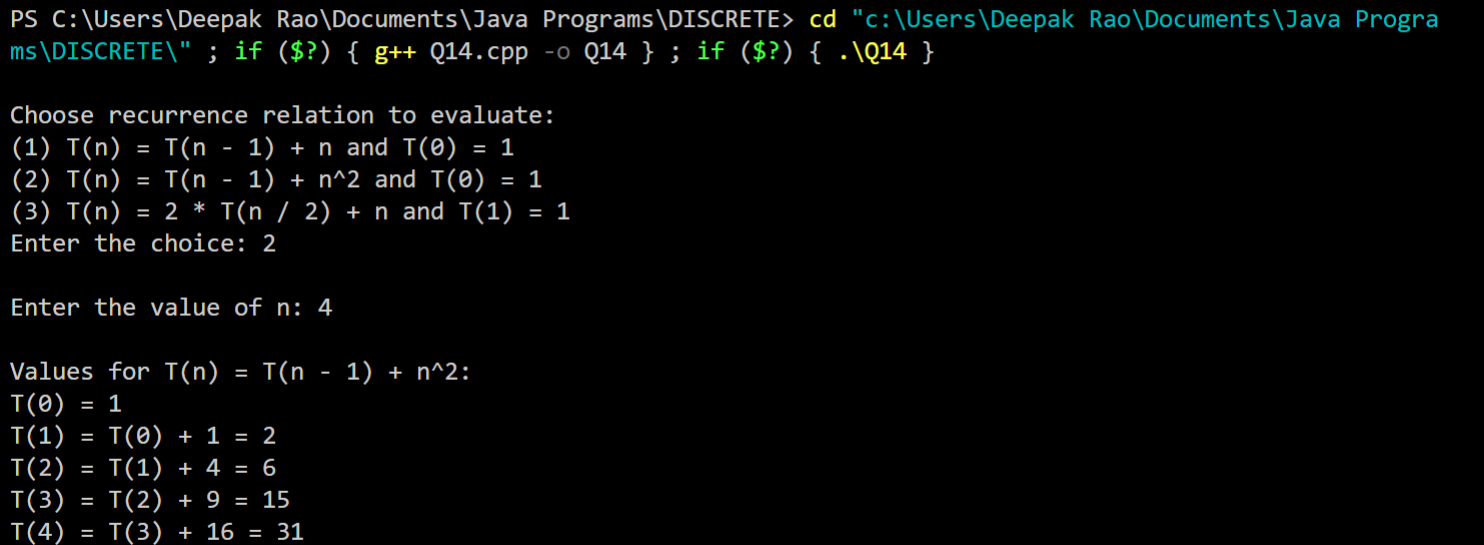
            break;

    }

    return 0;

}

**OUTPUT:-**

****

**Ques15.** /\**Write a Program to store a function (polynomial/exponential), and then evaluate*

*the polynomial. (For example store f(x) = 4n3 + 2n + 9 in an array and for a given*

*value*

*of n, say n = 5, evaluate (i.e. compute the value of f(5))*\*/

#include<iostream>

#include<stdio.h>

#include<conio.h>

#include<cmath>

using namespace std;

int i;

class FUNCTION

{

    private:

    int n;

    double \*coefficient;

    double \*exponential;

    public:

    void input();

    void display();

    double evaluate(double);

};

void FUNCTION::input()

{

    int n;

    cout << "\nEnter the number of terms: ";

    cin >> this->n;

    coefficient = new double[n];

    exponential = new double[n];

    for(i=0; i<this->n; i++)

    {

        cout << "Enter coefficient and exponential of term " << i+1 << ": ";

        cin >> coefficient[i] >> exponential[i];

    }

}

void FUNCTION::display()

{

    for(i=0; i<this->n; i++)

    {

        if(coefficient[i] >= 0)

            cout << " + ";

        else

            cout << " - ";

        cout << abs (coefficient[i]);

        if(exponential[i] != 0)

            cout << "(x^" << exponential[i] << ")";

    }

}

double FUNCTION::evaluate(double x)

{

    double result = 0.0;

    for(i=0; i<this->n; i++)

    {

        result += coefficient[i] \* (pow(x, exponential[i]));

    }

    return result;

}

int main()

{

    double x;

    FUNCTION ob;

    ob.input();

    cout << "Function is f(x) = ";

    ob.display();

    cout << "\nEnter the value of x: ";

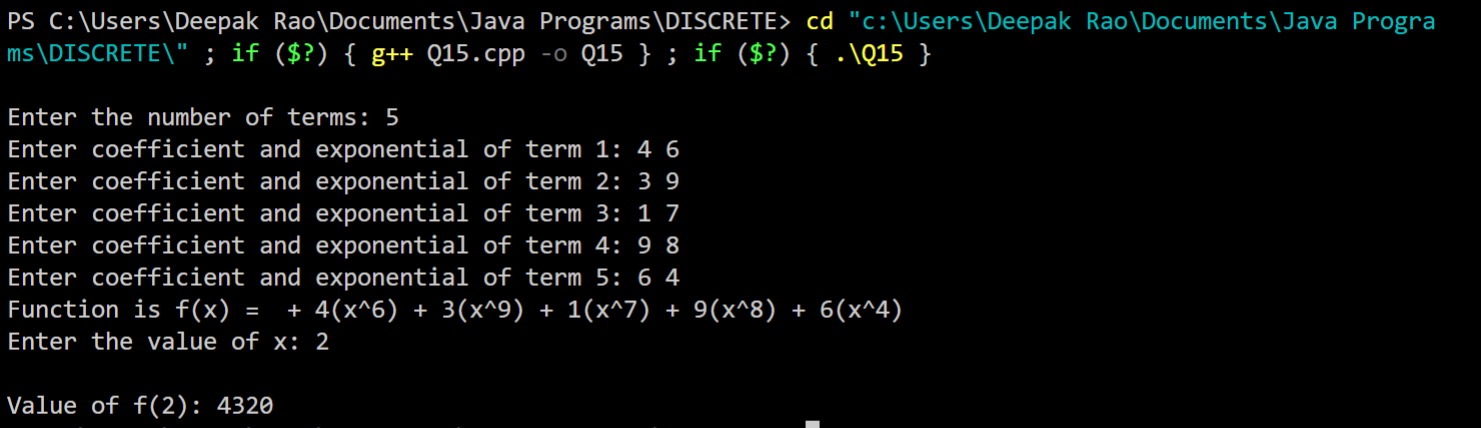
    cin >> x;

    cout << "\nValue of f(" << x << "): " << ob.evaluate(x) << endl;

    return 0;

}

**OUTPUT:-**

****

**Ques16.** /\**Write a Program to represent Graphs using the Adjacency Matrices and check if it*

*is a complete graph.* \*/

#include<iostream>

using namespace std;

int main()

{

    int n, c=0, x, p;

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

    cin >> n;

    int matrix[n][n];

    for(int i=0; i<n; i++)

        for(int j=0; j<n; j++)

             matrix[i][j] = 0;

    for(int i=0; i<n; i++)

    {

        cout << "\nEnter the no. of vertices adjacent to vertex " << i+1 << ": ";

        cin >> x;

        for(int j=0; j<x; j++)

        {

            cout << "Enter the vertex adjacent to vertex " << i+1 << ": ";

            cin >> p;

            for(int a=0; a<n; a++)

                if(a+1 == p)

                {

                    matrix[i][a] = 1;

                    break;

                }

        }

    }

    cout << "\nADJACENCY MATRIX\n";

    for(int i=0; i<n; i++)

    {

        int sum = 0;

        for(int j=0; j<n; j++)

        {

            cout << matrix[i][j] << " ";

            if(matrix[i][i] == 0)

                sum += matrix[i][j];

        }

        cout << endl;

        if(sum == (n-1))

            c++;

    }

    if(c == n)

        cout << "\nGraph is COMPLETE!!!";

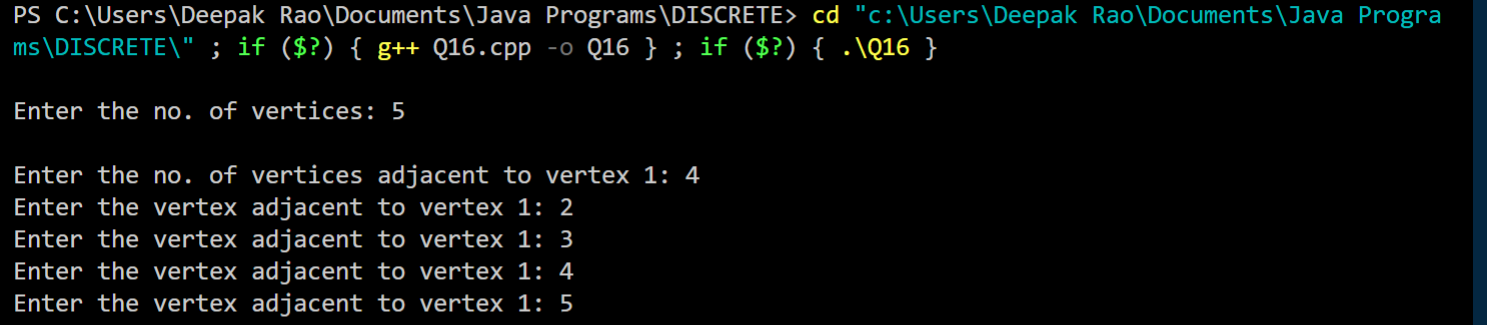
    else

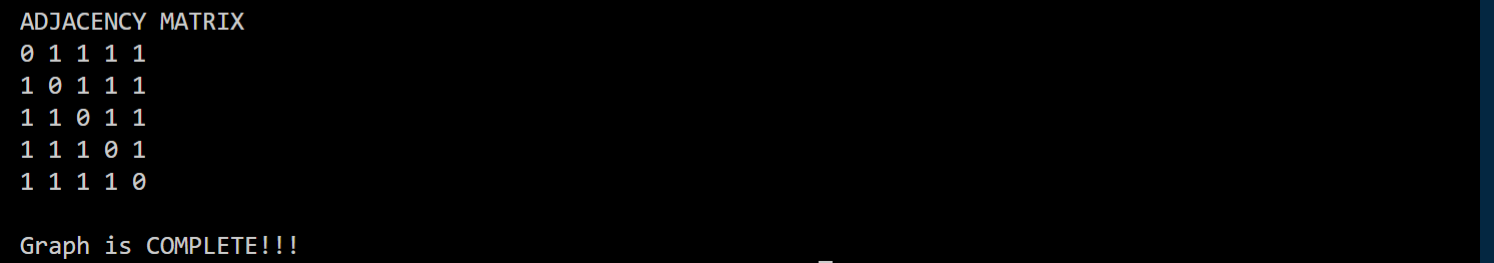
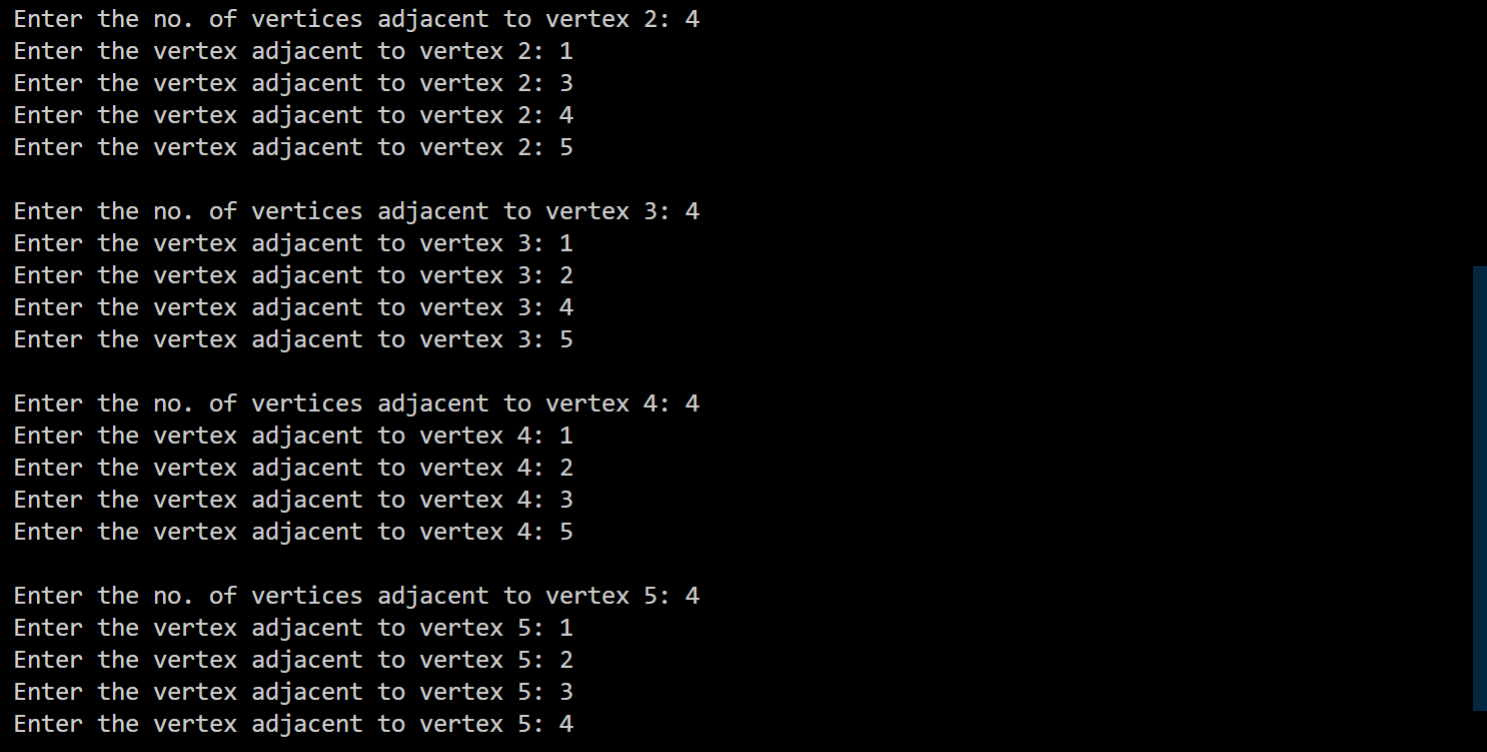
        cout << "\nGraph is NOT COMPLETE!!!";

    return 0;

}

**OUTPUT:-**

****

****

**Ques17.** /\**Write a Program to accept a directed graph G and compute the in-degree and*

*out-degree of each vertex*\*/

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

    int v, nin, nout, inver, outver;

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

    cin >> v;

    int matrix[v][v];

    for(int i=0; i<v; i++)

        for(int j=0; j<v; j++)

             matrix[i][j] = 0;

    for(int i=0; i<v; i++)

    {

        cout << "Enter the no. of edges incoming to vertex " << i+1 << ": ";

        cin >> nin;

        for(int x=0; x<nin; x++)

        {

            cout << "Enter the vertex from which incoming edge to vertex " << i+1 << " is emerging from: ";

            cin >> inver;

            matrix[i][inver -1] = -1;

        }

        cout << "Enter the no. of edges outgoing from vertex " << i+1 << ": ";

        cin >> nout;

        for(int y=0; y<nout; y++)

        {

            cout << "Enter the vertex to which outgoing edge from vertex " << i+1 << " is ending at: ";

            cin >> outver;

            matrix[i][outver -1] = 1;

        }

    }

    for(int i=0; i<v; i++)

    {

        int indegree=0, outdegree=0;

        for(int j=0; j<v; j++)

        {

            if(matrix[i][j] == 1)

                outdegree += matrix[i][j];

            if(matrix[i][j] == -1)

                indegree += matrix[i][j];

        }

        cout << "\n\nIn-degree of vertex " << i+1 << " is " << abs(indegree)

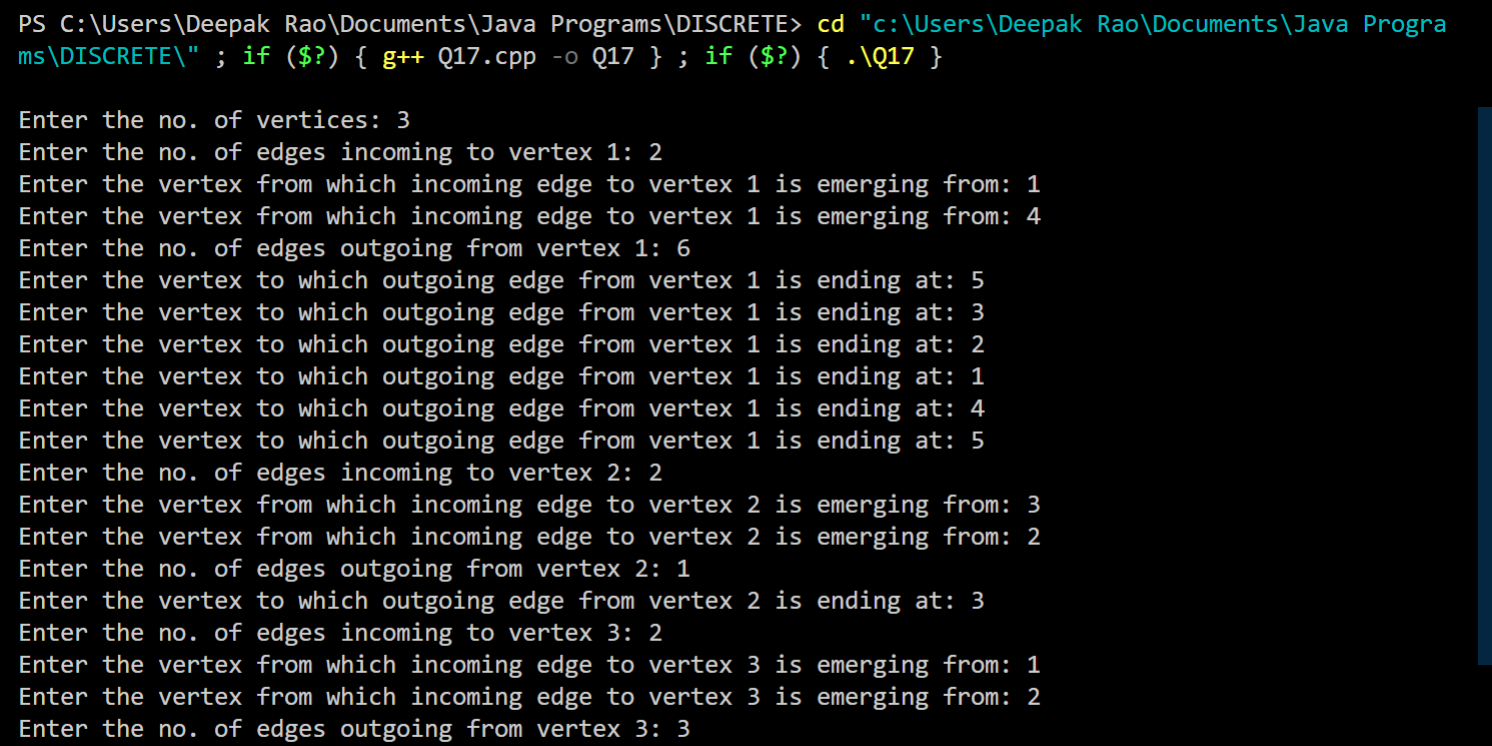
             << "\tOut-degree of vertex " << i+1 << " is " << outdegree;

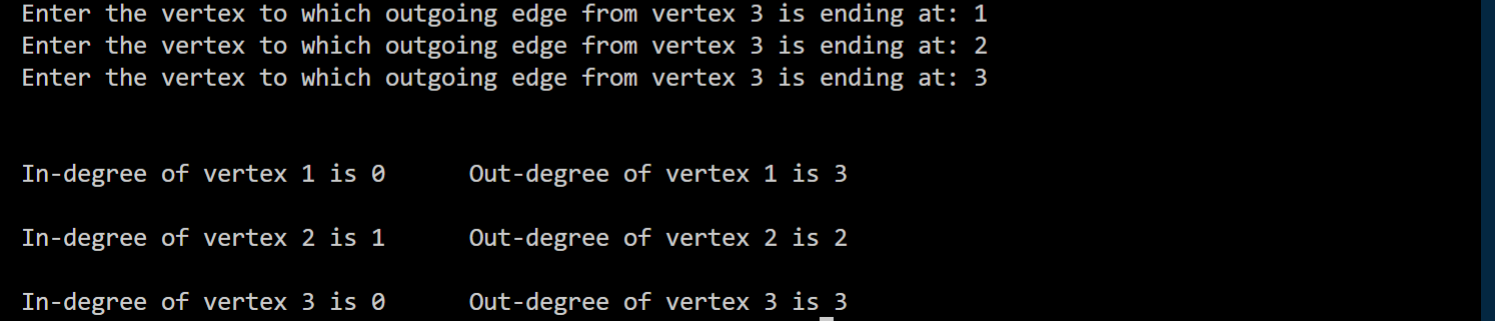
    }

    return 0;

}

**OUTPUT:-**

****

****

**Ques18.** /\**Given a graph G, Write a Program to find the number of paths of length n between*

*the source and destination entered by the user*\*/

#include<iostream>

using namespace std;

int countPaths(int graph[][100], int n, int src, int dest, int len)

{

    int count[n][n][len + 1];

    for(int e=0; e<=len; e++)

    {

        for(int i=0; i<n; i++)

        {

            for(int j=0; j<n; j++)

            {

                count[i][j][e] = 0;

                if(e == 0 && i == j)

                    count[i][j][e] = 1;

                if(e == 1 && graph[i][j])

                    count[i][j][e] = 1;

                if(e > 1)

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

                        if(graph[i][a])

                            count[i][j][e] += count[a][j][e - 1];

            }

        }

    }

    return count[src][dest][len];

}

int main()

{

    int v;

    cout << "\nEnter the nunber of vertices: ";

    cin >> v;

    int matrix[100][100];

    cout << "Enter the adjacency matrix:\n";

    for(int i=0; i<v; i++)

        for(int j=0; j<v; j++)

            cin >> matrix[i][j];

    int src, dest;

    cout << "Enter the source node: ";

    cin >> src;

    cout << "Enter the destinastion node: ";

    cin >> dest;

    int len;

    cout << "Enter the path lemgth: ";

    cin >> len;

    cout << "Total paths from node " << src

        << " to node " << dest << " having "

        << len << " edges: "

        << countPaths(matrix, v, src-1, dest-1, len);

    return 0;

}

**OUTPUT:-**

****

**Ques19.** /\**Given an adjacency matrix of a graph, write a program to check whether a given*

*set of vertices {v1,v2,v3.....,vk} forms an Euler path / Euler Circuit*

*(for circuit assume vk=v1)*\*/

#include<iostream>

using namespace std;

int main()

{

    int n;

    cout << "\nEnter the number of vertices: ";

    cin >> n;

    int matrix[n][n];

    for(int i=0; i<n; i++)

        for(int j=0; j<n; j++)

            matrix[i][j] = 0;

    cout << "Enter the adjacency matrix:\n";

    for(int i=0; i<n; i++)

        for(int j=0; j<n; j++)

            cin >> matrix[i][j];

    int degree, order = 0;

    for(int i=0; i<n; i++)

    {

        degree = 0;

        for(int j=0; j<n; j++)

            degree += matrix[i][j];

        if(degree % 2 != 0)

            order++;

    }

    if(order == 0)

        cout << "Graph has an Eulerian Circuit!" << endl;

    else if(order == 2)

        cout << "Graph has an Eulerian Path!" << endl;

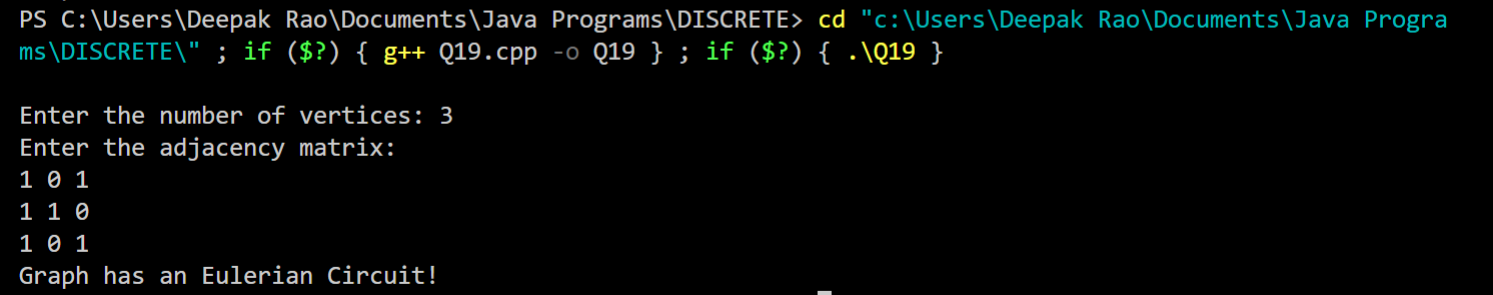
    else

        cout << "Graph is Not Eulerian!" << endl;

    return 0;

}

**OUTPUT:-**

****

**Ques20.** /\**Given a full m-ary tree with i internal vertices, Write a Program to find the*

*number of leaf nodes*\*/

#include<iostream>

using namespace std;

int calcNodes(int m, int I)

{

    int result = 0;

    result = I \* (m - 1) + 1;

    return result;

}

int main()

{

    int m,I,N;

    cout << "\nEnter the maximum no. of children in full m-ary tree: ";

    cin >> m;

    cout << "Enter the number of internal vertices: ";

    cin >> I;

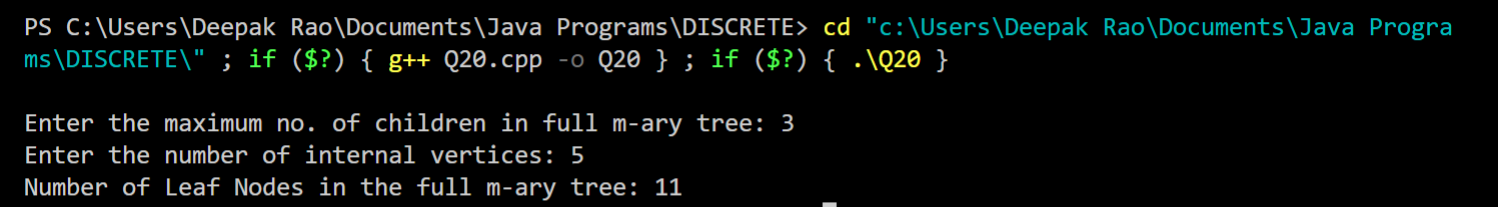
    N = calcNodes(m, I);

    cout << "Number of Leaf Nodes in the full m-ary tree: " << N;

    return 0;

}

**OUTPUT:-**

****

**Finished**