Practical File for

**Discrete Structures**

Paper Code - 32341202

Submitted By – Satyam Verma

College Roll No. – 21/10882

B.Sc. (Hons) Computer Science

Department of Computer Science

**KESHAV MAHAVIDYALAYA**

University of Delhi

Index

|  |  |  |
| --- | --- | --- |
| S.No. | Topic | Page No. |
| 1 | Practical #01 | 02 - 05 |
| 2 | Practical #02 | 06 - 12 |
| 3 | Practical #03 | 13 - 18 |
| 4 | Practical #04 | 19 – 24 |
| 5 | Practical #05 | 25 - 26 |
| 6 | Practical #06 | 27 - 28 |
| 7 | Practical #07 | 29 - 32 |
| 8 | Practical #08 | 33 - 37 |
| 9 | Practical #09 | 38 - 41 |
| 10 | Practical #10 | 42 - 44 |
| 11 | Practical #11 | 45 - 47 |
| 12 | Practical #12 | 48 - 49 |
| 13 | Practical #13 | 50 - 51 |
| 14 | Practical #14 | 52 - 53 |
| 15 | Practical #15 | 54 - 56 |
| 16 | Practical #16 | 57 - 59 |
| 17 | Practical #17 | 60 - 62 |
| 18 | Practical #18 | 63 - 66 |
| 19 | Practical #19 | 67 - 69 |
| 20 | Practical #20 | 70 - 71 |

Practical #01

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.

Code :

#include <iostream>

#include <math.h>

using namespace std;

void arraytoSet(int \*arr, int \*n)

{

if (\*n == 0 || \*n == 1)

return;

for (int i = 0; i < \*n; i++)

{

for (int j = i + 1; j < \*n; j++)

{

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

{

for (int k = j + 1; k < \*n; k++)

{

arr[j] = arr[k];

j++;

}

\*n = \*n - 1;

j--;

}

}

}

}

void print(int \*arr, int n)

{

cout << "{";

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

{

if (i == 0)

cout << arr[i];

else

cout << "," << arr[i];

}

cout << "} ";

}

bool ismember(int a, int \*arr, int \*n)

{

for (int i = 0; i < \*n; i++)

{

if (a == arr[i])

{

return true;

break;

}

}

return false;

}

/\* Helper recursive function for powerset function

set = initial set

n = initial set

set\_i = counter for initial set

power\_set = power set in which subsets will be stored

powerset\_i = current index for power set

\*/

void helper\_powerset(int \*set, int n, int set\_i, int \*power\_set, int powerset\_i)

{

//Check for terminating recursion

if (set\_i >= n)

{

print(power\_set, powerset\_i);

return;

}

//Exclude

helper\_powerset(set, n, set\_i + 1, power\_set, powerset\_i);

power\_set[powerset\_i] = set[set\_i];

//Include

helper\_powerset(set, n, set\_i + 1, power\_set, powerset\_i + 1);

}

void powerset(int \*arr, int n)

{

int power\_set[(int)pow(2, n)];

cout << "{ ";

helper\_powerset(arr, n, 0, power\_set, 0);

cout << "}\n\n" << endl;

}

int main()

{

int \*A;

int n;

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

cin >> n;

cout << "Enter the array elements : ";

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

{

cin >> A[i];

}

arraytoSet(A, &n);

cout << endl

<< "Given set A : ";

print(A, n);

cout << "\n\nCardinality of set A : " << n << endl;

cout << endl

<< "Let a = 2, then" << endl;

int a = 2;

cout << "ismember(" << a << ", A) : ";

if (ismember(a, A, &n) == 1)

cout << "true" << endl;

else

cout << "false" << endl;

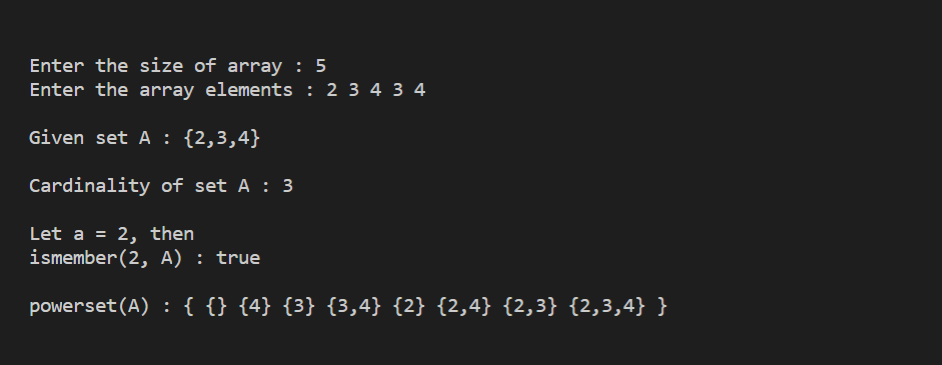
cout << "\npowerset(A) : ";

powerset(A, n);

return 0;

}

Output :



Practical #02

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.

Code:

#include <iostream>

using namespace std;

class Set

{

public:

    int elements[100];

    int size = 0;

    Set();

    void input();

    void print();

    bool subset\_of(Set);

    Set union\_with(Set);

    Set intersection\_with(Set);

    Set complement(Set);

    Set difference(Set);

    Set difference\_sym(Set);

    void cartesian\_prod(Set);

private:

    void addElement(int);

    bool has(int i);

};

Set::Set()

{

}

void Set::input()

{

    int itrCount;

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

    cin >> itrCount;

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

    for (int i = 0; i < itrCount; i++)

    {

        int e;

        cin >> e;

        this->addElement(e);

    }

    cout << endl;

}

bool Set::has(int n)

{

    for (int i = 0; i < size; i++)

    {

        if (n == elements[i])

        {

            return true;

            break;

        }

    }

    return false;

}

void Set::addElement(int n)

{

    if (!has(n))

    {

        elements[size] = n;

        ++size;

    }

}

bool Set::subset\_of(Set set)

{

    int count = 0;

    for (int i = 0; i < this->size; i++)

    {

        for (int j = 0; j < set.size; j++)

        {

            if (this->elements[i] == set.elements[j])

            {

                count++;

                break;

            }

        }

    }

    if (count == size)

        return true;

    else

        return false;

}

Set Set::union\_with(Set set)

{

    Set temp;

    for (int i = 0; i < size; i++)

    {

        temp.addElement(elements[i]);

    }

    for (int i = 0; i < set.size; i++)

    {

        temp.addElement(set.elements[i]);

    }

    return temp;

}

Set Set::intersection\_with(Set set)

{

    Set temp;

    for (int i = 0; i < size; i++)

    {

        if (set.has(elements[i]))

            temp.addElement(elements[i]);

    }

    return temp;

}

Set Set::complement(Set uni\_set)

{

    Set temp;

    for (int i = 0; i < uni\_set.size; i++)

    {

        if (!has(uni\_set.elements[i]))

            temp.addElement(uni\_set.elements[i]);

    }

    return temp;

}

Set Set::difference(Set set)

{

    Set temp;

    for (int i = 0; i < size; i++)

    {

        if (!set.has(elements[i]))

            temp.addElement(elements[i]);

    }

    return temp;

}

Set Set::difference\_sym(Set set)

{

    Set unionSet = union\_with(set);

    Set intrSet = intersection\_with(set);

    return unionSet.difference(intrSet);

}

void Set::cartesian\_prod(Set set)

{

    cout << endl

         << "{ ";

    for (int i = 0; i < size; i++)

    {

        for (int j = 0; j < set.size; j++)

        {

            cout << "{" << elements[i] << "," << set.elements[j] << "} ";

        }

    }

    cout << "}" << endl

         << endl

         << endl;

}

void Set::print()

{

    cout << "{";

    for (int i = 0; i < size; i++)

    {

        if (i == 0)

            cout << elements[i];

        else

            cout << "," << elements[i];

    }

    cout << "} ";

}

int main()

{

    Set setA = Set();

    setA.input();

    cout << "--> Set A : ";

    setA.print();

    Set setB = Set();

    setB.input();

    cout << "--> Set B : ";

    setB.print();

    if (setA.subset\_of(setB))

        cout << "\n\n--> Set A is a subset of Set B.\n";

    else

        cout << "\n\n--> Set A is not a subset of Set B.\n";

    cout << endl

         << "--> Set A union Set B : ";

    setA.union\_with(setB).print();

    cout << "\n\n--> Set A intersection Set B : ";

    setA.intersection\_with(setB).print();

    cout << "\n\n--> Let Universal Set be ";

    Set universalSet = setA.union\_with(setB);

    universalSet.print();

    cout << endl

         << "--> Complement of Set A (A') : ";

    setA.complement(universalSet).print();

    cout << endl

         << "--> Complement of Set B (B') : ";

    setB.complement(universalSet).print();

    cout << "\n\n--> Set A difference Set B : ";

    setA.difference(setB).print();

    cout << "\n\n--> Set A symmetric difference Set B : ";

    setA.difference\_sym(setB).print();

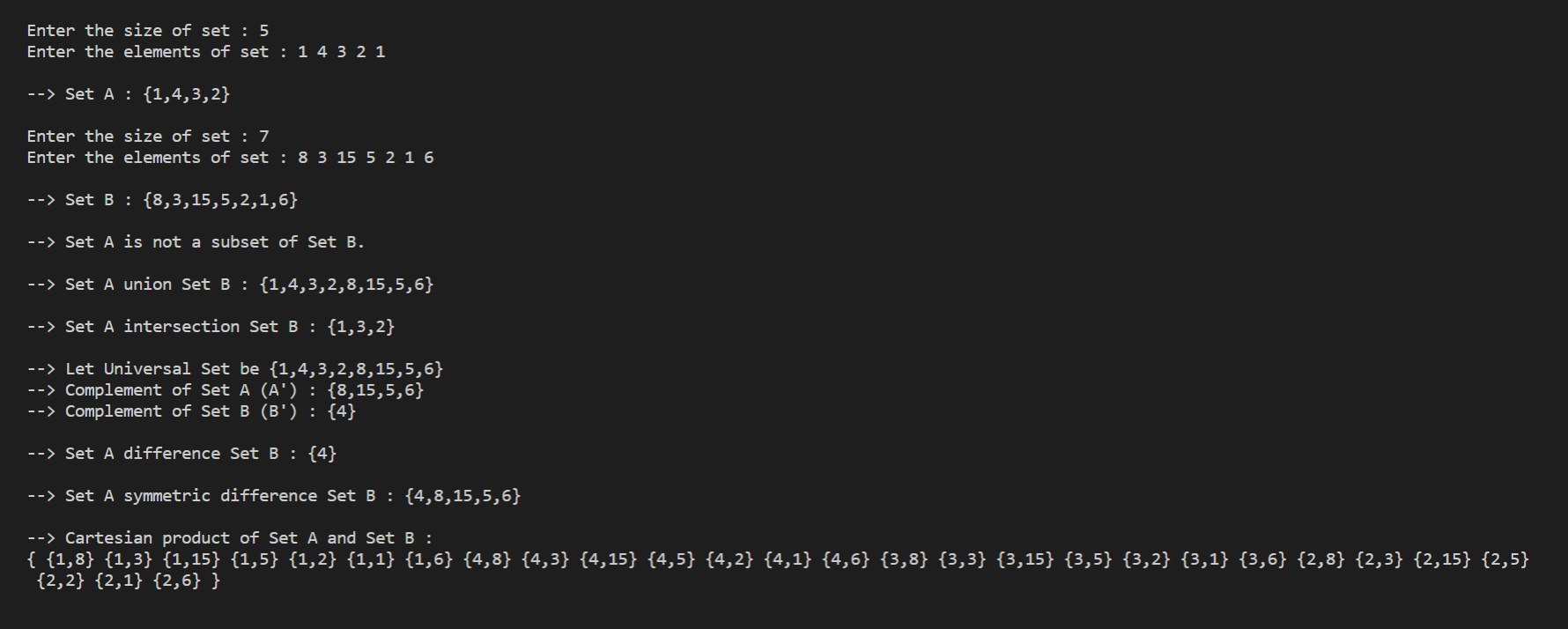
    cout << "\n\n--> Cartesian product of Set A and Set B : ";

    setA.cartesian\_prod(setB);

    return 0;

}

Output :



Practical #03

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.

Code :

#include <iostream>

using namespace std;

class Relation

{

public:

    int matrix[100][100] = {0};

    int size, matrix\_size;

    void input();

    void print\_matrix();

    void isReflexive();

    void isSymmetric();

    void isAntisymmetric();

    void isTransitive();

};

void Relation::input()

{

    int max = 0;

    cout << endl

         << endl

         << "Enter the length of relation : ";

    cin >> size;

    cout << "Enter the elements of relation : " << endl;

    for (int i = 0; i < size; i++)

    {

        cout << "Element " << i + 1 << " : ";

        int a, b;

        cin >> a >> b;

        matrix[a - 1][b - 1] = 1;

        if (a > max)

            max = a;

        else if (b > max)

            max = b;

    }

    matrix\_size = max;

}

void Relation::print\_matrix()

{

    cout << "\n--> Matrix Representation : \n";

    for (int i = 0; i < matrix\_size; i++)

    {

        cout << endl

             << "    ";

        for (int j = 0; j < matrix\_size; j++)

        {

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

        }

    }

}

void Relation::isReflexive()

{

    int flag = 0;

    for (int i = 0; i < matrix\_size; i++)

    {

        if (matrix[i][i] == 1)

            flag = 1;

        else

        {

            flag = 0;

            break;

        }

    }

    if (flag == 0)

        cout << "--> Given relation is not reflexive\n";

    else

        cout << "--> Given relation is reflexive\n";

}

void Relation::isSymmetric()

{

    int flag = 0;

    for (int i = 0; i < matrix\_size; i++)

        for (int j = 0; j < matrix\_size; j++)

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

            {

                flag = 1;

            }

            else

            {

                flag = 0;

                break;

            }

    if (flag == 0)

        cout << "--> Given relation is not symmetric\n";

    else

        cout << "--> Given relation is symmetric\n";

}

void Relation::isAntisymmetric()

{

    int flag = 0;

    for (int i = 0; i < matrix\_size; i++)

        for (int j = 0; j < matrix\_size; j++)

            if ((matrix[i][j] || matrix[j][i] == 0) && i != j)

            {

                flag = 1;

            }

            else

            {

                flag = 0;

                break;

            }

    if (flag == 0)

        cout << "--> Given relation is antisymmetric\n";

    else

        cout << "--> Given relation is not antisymmetric\n";

}

void Relation::isTransitive()

{

    int flag = 1;

    for (int i = 0; i < matrix\_size; i++)

    {

        for (int j = 0; j < matrix\_size; j++)

        {

            if (matrix[i][j] == 1)

            {

                for (int k = 0; k < size; k++)

                {

                    if (matrix[j][k] == 1 && matrix[i][k] != 1)

                    {

                        flag = 0;

                    }

                }

            }

        }

    }

    if (flag == 0)

        cout << "--> Given relation is not transitive\n";

    else

        cout << "--> Given relation is transitive\n";

}

int main()

{

    Relation rel = Relation();

    rel.input();

    rel.print\_matrix();

    cout << endl

         << endl;

    rel.isReflexive();

    rel.isSymmetric();

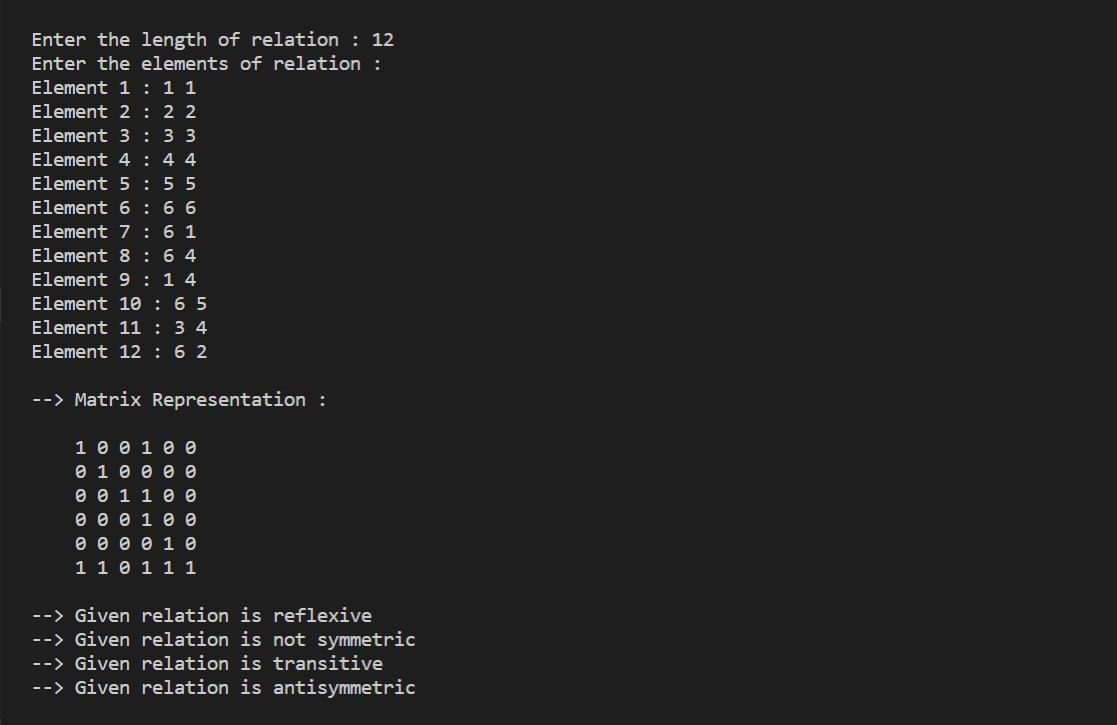
    rel.isTransitive();

    rel.isAntisymmetric();

    return 0;

}

Output :



Practical #04

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

Code :

#include <iostream>

using namespace std;

class Relation

{

public:

    int matrix[100][100] = {0};

    int size, matrix\_size;

    void input();

    void print\_matrix();

    bool isReflexive();

    bool isSymmetric();

    bool isAntisymmetric();

    bool isTransitive();

};

void Relation::input()

{

    int max = 0;

    cout << endl

         << endl

         << "Enter the length of relation : ";

    cin >> size;

    cout << "Enter the elements of relation : " << endl;

    for (int i = 0; i < size; i++)

    {

        cout << "Element " << i + 1 << " : ";

        int a, b;

        cin >> a >> b;

        matrix[a - 1][b - 1] = 1;

        if (a > max)

            max = a;

        else if (b > max)

            max = b;

    }

    matrix\_size = max;

}

void Relation::print\_matrix()

{

    cout << "\n--> Matrix Representation : \n";

    for (int i = 0; i < matrix\_size; i++)

    {

        cout << endl

             << "    ";

        for (int j = 0; j < matrix\_size; j++)

        {

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

        }

    }

}

bool Relation::isReflexive()

{

    int flag = 0;

    for (int i = 0; i < matrix\_size; i++)

    {

        if (matrix[i][i] == 1)

            flag = 1;

        else

        {

            flag = 0;

            break;

        }

    }

    return flag;

}

bool Relation::isSymmetric()

{

    int flag = 0;

    for (int i = 0; i < matrix\_size; i++)

        for (int j = 0; j < matrix\_size; j++)

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

            {

                flag = 1;

            }

            else

            {

                flag = 0;

                break;

            }

    return flag;

}

bool Relation::isAntisymmetric()

{

    int flag = 0;

    for (int i = 0; i < matrix\_size; i++)

        for (int j = 0; j < matrix\_size; j++)

            if ((matrix[i][j] || matrix[j][i] == 0) && i != j)

            {

                flag = 1;

            }

            else

            {

                flag = 0;

                break;

            }

    return !flag;

}

bool Relation::isTransitive()

{

    int flag = 1;

    for (int i = 0; i < matrix\_size; i++)

    {

        for (int j = 0; j < matrix\_size; j++)

        {

            if (matrix[i][j] == 1)

            {

                for (int k = 0; k < size; k++)

                {

                    if (matrix[j][k] == 1 && matrix[i][k] != 1)

                    {

                        flag = 0;

                    }

                }

            }

        }

    }

    return flag;

}

int main()

{

    Relation rel = Relation();

    rel.input();

    rel.print\_matrix();

    if (rel.isReflexive() && rel.isSymmetric() && rel.isTransitive())

        cout << "\n\n--> The given relation is equivalence relation.\n";

    else if (rel.isReflexive() && rel.isAntisymmetric() && rel.isTransitive())

        cout << "\n\n--> The given relation is a partially ordered relation(poset).\n";

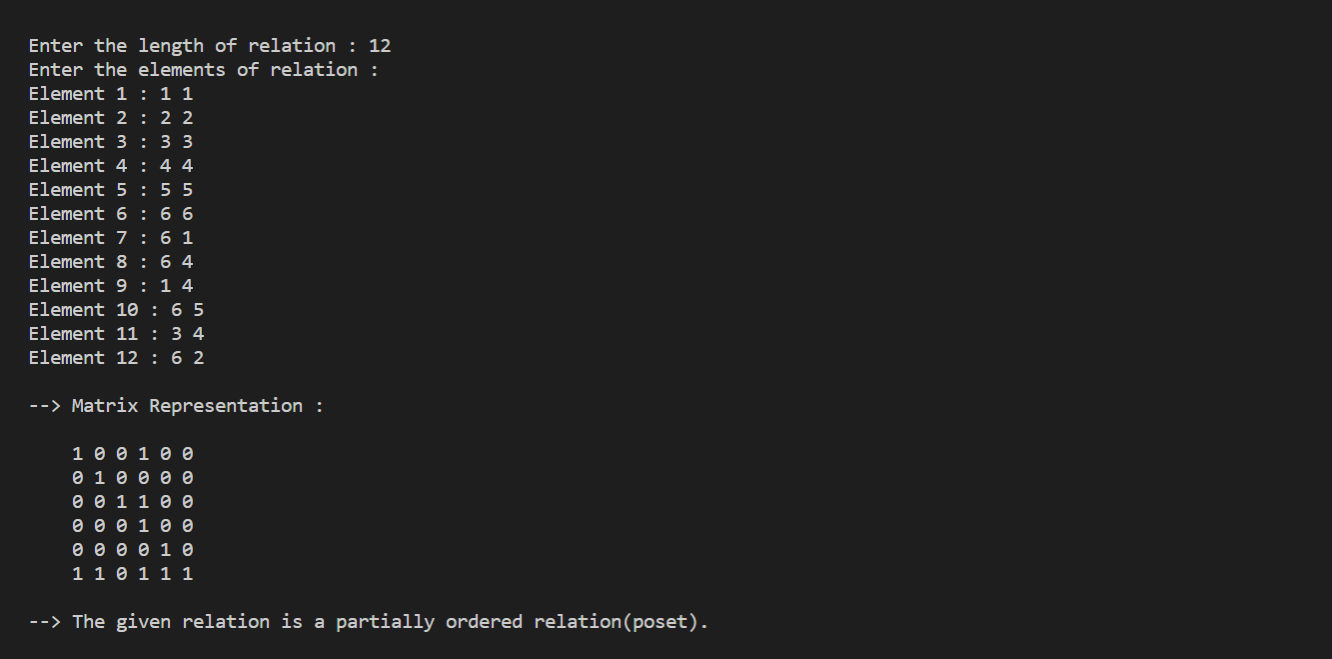
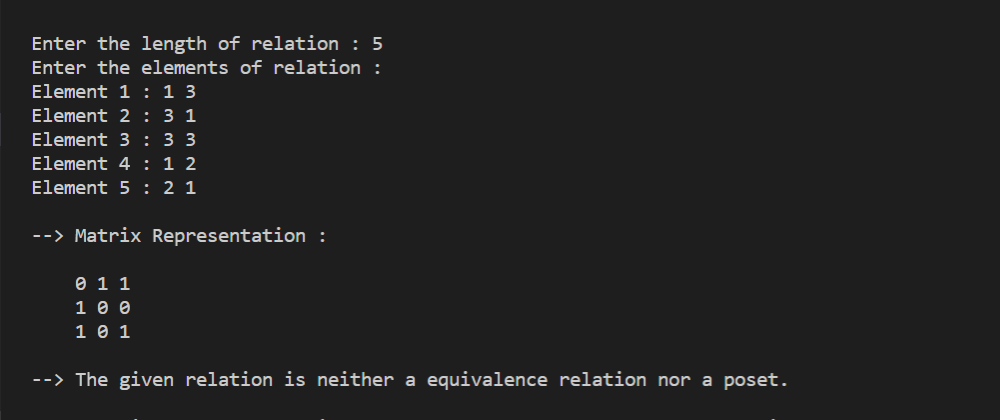
    else

        cout << "\n\n--> The given relation is neither a equivalence relation nor a poset.\n";

    return 0;

}

Output :



Practical #05

Write a Program to generate the Fibonacci Series using recursion.

Code :

#include <iostream>

using namespace std;

int fib\_at(int n)

{

    if (n == 0 || n == 1)

    {

        return n;

    }

    return fib\_at(n - 2) + fib\_at(n - 1);

}

int main()

{

    int n;

    cout << endl

         << endl

         << "Enter the size of Fibonacci series : ";

    cin >> n;

    cout << endl

         << "--> Fibonacci series (0-" << n << ") : ";

    for (int i = 0; i < n; i++)

    {

        cout << fib\_at(i) << " ";

    }

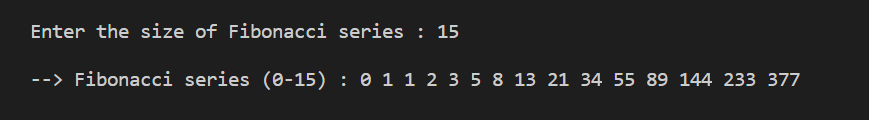
    cout << endl

         << endl;

    return 0;

}

Output :



Practical #06

Write a Program to implement Tower of Hanoi using recursion.

Code :

#include <iostream>

using namespace std;

// total moves

int TOH(int n)

{

    if (n == 1)

        return 1;

    return (2 \* TOH(n - 1) + 1);

}

void TOH\_showmoves(int n, char source\_t, char dest\_t, char spare\_t)

{

    //Base Case

    if (n == 1)

    {

        cout << endl

             << " - Move disk 1 of tower " << source\_t << " --> tower " << dest\_t;

        return;

    }

    //Step 1

    TOH\_showmoves(n - 1, source\_t, spare\_t, dest\_t);

    //Step 2

    cout << endl

         << " - Move disk " << n << " of tower " << source\_t << " --> tower " << dest\_t;

    //Step 3

    TOH\_showmoves(n - 1, spare\_t, dest\_t, source\_t);

}

int main()

{

    int n;

    cout << endl

         << endl

         << "Enter number of disks : ";

    cin >> n;

    cout << endl

         << "Let the towers are A, B and C and we are required to move the disks from A to C." << endl;

    TOH\_showmoves(n, 'A', 'C', 'B');

    cout << endl

         << endl

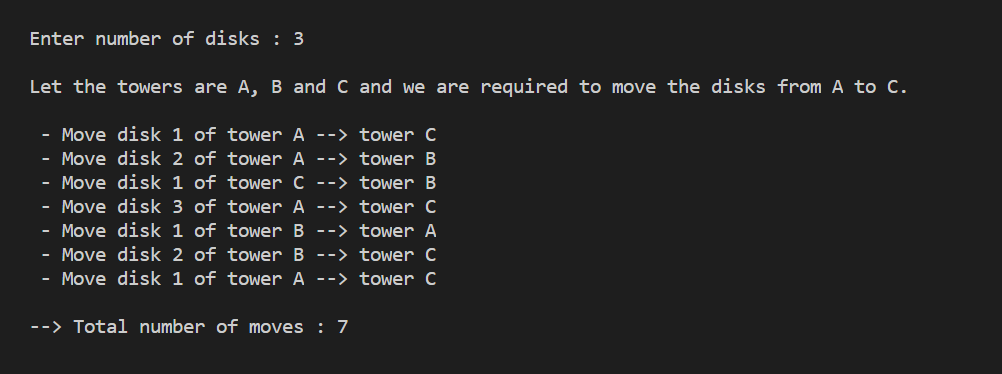
         << "--> Total number of moves : " << TOH(n) << endl

         << endl;

    return 0;

}

Output :



Practical #07

Write a Program to implement binary search using recursion.

Code :

#include <iostream>

using namespace std;

void sort(int arr[], int v)

{

    for (int i = 0; i < (v - 1); i++)

    {

        for (int j = 0; j < (v - i - 1); j++)

        {

            if (arr[j] > arr[j + 1])

            {

                float temp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = temp;

            }

        }

    }

}

int binarySearch(int arr[], int start, int end, int e)

{

    if (end >= 1)

    {

        int mid = (int)start + (end - 1) / 2;

        if (arr[mid] == e)

            return mid;

        if (arr[mid] > e)

            return binarySearch(arr, start, mid - 1, e);

        return binarySearch(arr, mid + 1, end, e);

    }

    return -1;

}

void print(int arr[], int n)

{

    cout << endl

         << " - [ ";

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

    cout << "]";

}

int main()

{

    int \*arr;

    int n, size;

    cout << endl

         << endl

         << "Enter the size of array : ";

    cin >> size;

    cout << endl

         << "Enter the elements of array : ";

    for (int i = 0; i < size; i++)

    {

        cin >> arr[i];

    }

    cout << endl

         << " - Sorting the array..." << endl;

    sort(arr, size);

    cout << " - Done!";

    print(arr, size);

    cout << endl

         << endl

         << "Enter the integer to be searched : ";

    cin >> n;

    cout << endl

         << " - Searching using Binary Search..." << endl;

    int result = binarySearch(arr, 0, size - 1, n);

    cout << " - Done!" << endl;

    if (result == -1)

        cout << endl

             << "--> Element is not present in the array.";

    else

        cout << endl

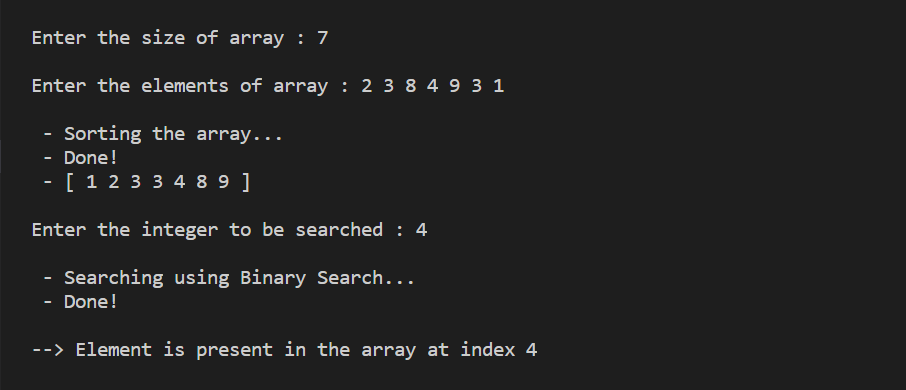
             << "--> Element is present in the array at index " << result << endl

             << endl;

    return 0;

}

Output :



Practical #08

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.

Code :

#include <iostream>

using namespace std;

void display(int ar[], int v)

{

    cout << " - [ ";

    for (int i = 0; i < v; i++)

    {

        cout << ar[i] << " ";

    }

    cout << "]";

}

int swap(int \*ar, int a, int b)

{

    int temp = ar[a];

    ar[a] = ar[b];

    ar[b] = temp;

    return \*ar;

}

int bubbleSort(int \*ar, int v)

{

    int totalCount = 0, count;

    for (int i = 0; i < (v - 1); i++)

    {

        count = 0;

        for (int j = 0; j < (v - i - 1); j++)

        {

            if (ar[j] > ar[j + 1])

            {

                ++count;

                swap(ar, j, j + 1);

            }

            else

                ++count;

        }

        totalCount += count;

        display(ar, v);

        cout << " -> " << (count) << " comparisons" << endl;

    }

    return totalCount;

}

int main()

{

    int \*arr;

    int size, comparisons;

    cout << endl

         << endl

         << "Enter the size of array : ";

    cin >> size;

    cout << endl

         << "Enter the elements of the array : ";

    for (int i = 0; i < size; i++)

    {

        cin >> arr[i];

    }

    cout << endl;

    display(arr, size);

    cout << endl

         << endl

         << " - Sorting the array using Bubble Sort Algorithm...." << endl

         << endl;

    comparisons = bubbleSort(arr, size);

    cout << endl

         << "--> Sorted. Here is the resulting array";

    display(arr, size);

    cout << endl

         << endl

         << "--> Total no. of comparisons : " << comparisons;

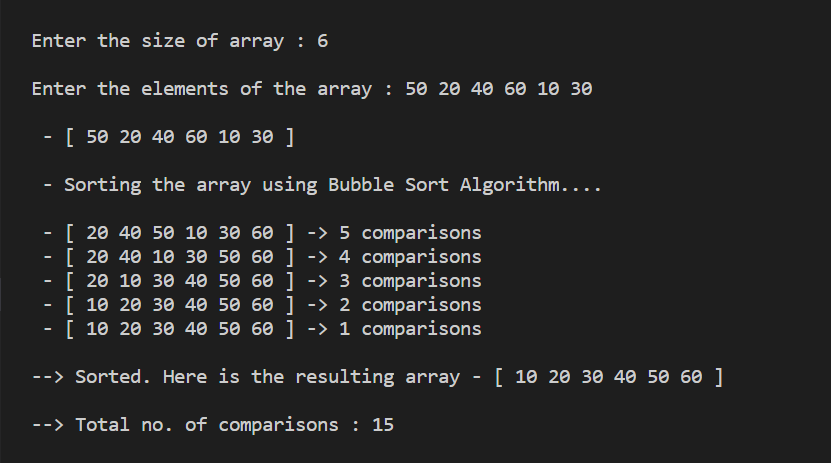
    cout << endl

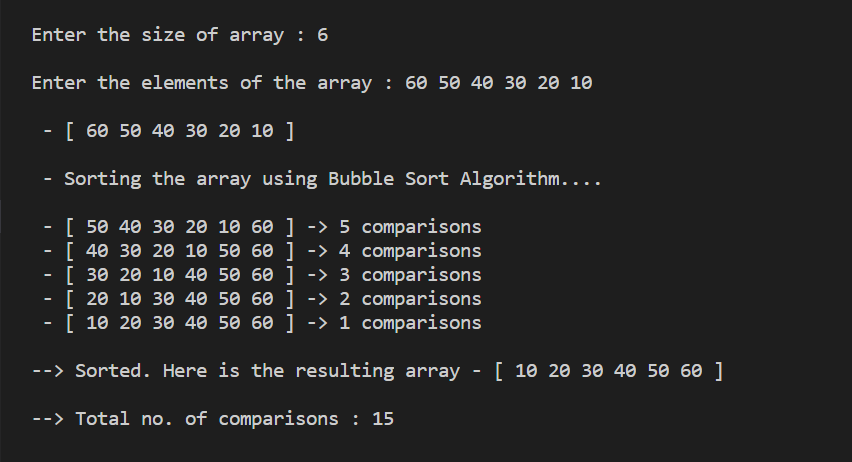
         << endl;

    return 0;

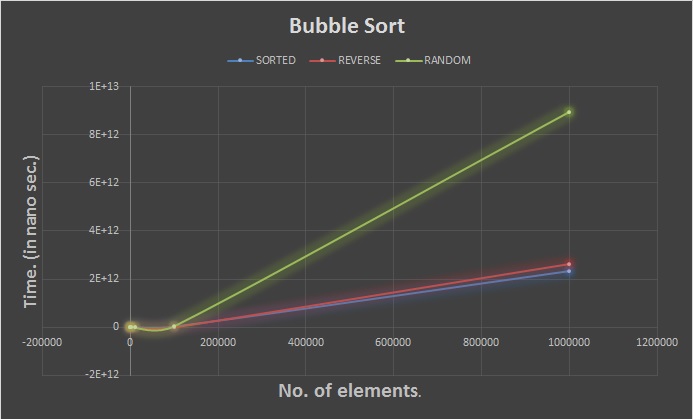
}

Output :

Average Case:

Worst Case:

Graph :



Practical #09

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.

Code :

#include <iostream>

using namespace std;

void display(int ar[], int v)

{

    cout << " - [ ";

    for (int i = 0; i < v; i++)

    {

        cout << ar[i] << " ";

    }

    cout << "]";

}

int insertionSort(int \*ar, int v)

{

    int i, j, key, totalCount = 0, count = 0;

    for (i = 1; i < v; i++)

    {

        key = ar[i];

        j = i - 1;

        count = 0;

        while (j >= 0 && key < ar[j])

        {

            ++count;

            ar[j + 1] = ar[j];

            j = j - 1;

        }

        totalCount += count;

        display(ar, v);

        cout << " -> " << (count) << " comparisons" << endl;

        ar[j + 1] = key;

    }

    return totalCount;

}

int main()

{

    int \*arr;

    int size, comparisons;

    cout << endl

         << endl

         << "Enter the size of array : ";

    cin >> size;

    cout << endl

         << "Enter the elements of the array : ";

    for (int i = 0; i < size; i++)

    {

        cin >> arr[i];

    }

    cout << endl;

    display(arr, size);

    cout << endl

         << endl

         << " - Sorting the array using Insertion Sort Algorithm...." << endl

         << endl;

    comparisons = insertionSort(arr, size);

    cout << endl

         << "--> Sorted. Here is the resulting array";

    display(arr, size);

    cout << endl

         << endl

         << "--> Total no. of comparisons : " << comparisons;

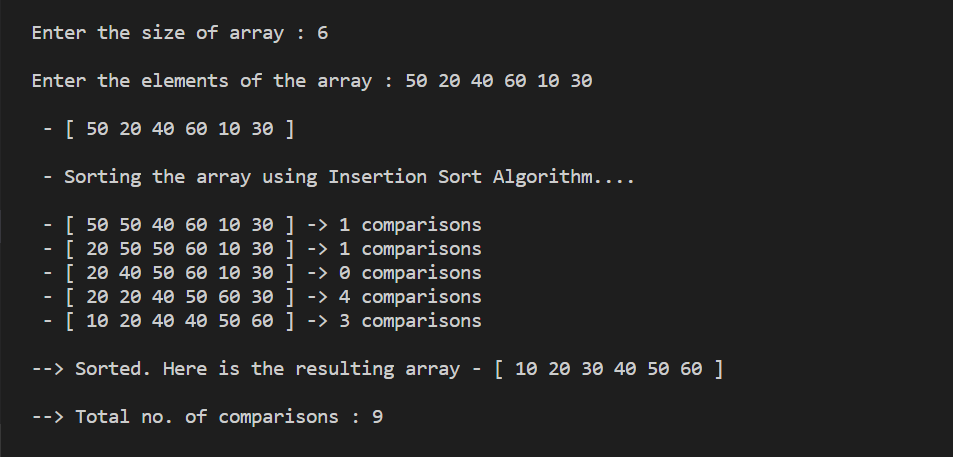
    cout << endl

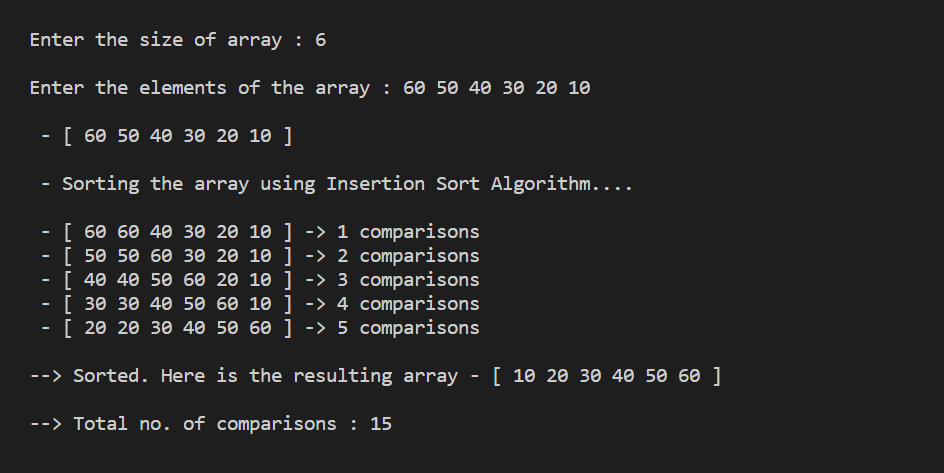
         << endl;

    return 0;

}

Output : Average Case:



Worst Case:

Graph:



Practical #10

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).

Code :

#include <iostream>

using namespace std;

void printArray(int \*ar, int v)

{

    cout << " {";

    for (int i = 0; i < v; i++)

    {

        if (i == 0)

            cout << ar[i];

        else

            cout << "," << ar[i];

        ;

    }

    cout << "}";

}

void swap(int \*a, int \*b)

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

void permutation(int \*arr, int n, int curr\_i, int \*permut\_arr, int permut\_i)

{

    if (curr\_i >= n)

    {

        printArray(permut\_arr, permut\_i);

    }

    for (int i = curr\_i; i < n; i++)

    {

        swap(arr[curr\_i], arr[i]);

        permut\_arr[permut\_i] = arr[curr\_i];

        permutation(arr, n, curr\_i + 1, permut\_arr, permut\_i + 1);

        swap(arr[curr\_i], arr[i]);

    }

}

int main()

{

    int \*arr, pArr[100];

    int size;

    cout << endl

         << endl

         << "Enter the no. of digits : ";

    cin >> size;

    cout << endl

         << "Enter the digits : ";

    for (int i = 0; i < size; i++)

    {

        cin >> arr[i];

    }

    cout << endl

         << "--> Given set of digits :";

    printArray(arr, size);

    cout << endl

         << endl

         << "--> All possible permutations of digits :";

    permutation(arr, size, 0, pArr, 0);

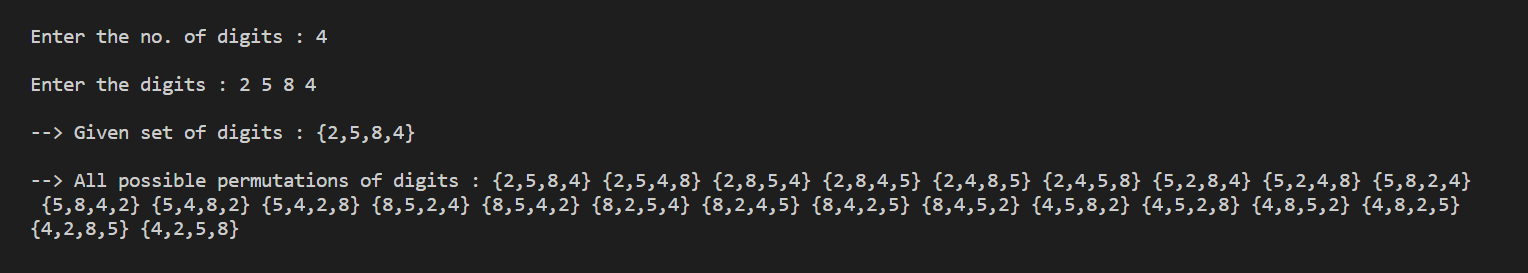
    cout << endl

         << endl;

    return 0;

}

Output :



Practical #11

Write a Program to calculate Permutation and Combination for an input value n and r using recursive formula of n Cr and n Pr .

Code :

#include <iostream>

using namespace std;

int nPr(int n, int r)

{

    if (r > n)

        return -1;

    else if (r == 0)

        return 1;

    else

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

}

int nCr(int n, int r)

{

    if (r > n)

        return -1;

    else if (n == 0 || r == 0 || n == r)

        return 1;

    else

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

}

int main()

{

    int n, r, p, c;

    cout << endl

         << endl

         << "Enter the value of n : ";

    cin >> n;

    cout << "Enter the value of r : ";

    cin >> r;

    p = nPr(n, r);

    c = nCr(n, r);

    cout << endl

         << "--> Given : n=" << n << ", r=" << r << endl;

    if (p == -1 || c == -1)

        cout

            << endl

            << "--> Invalid input!" << endl

            << endl;

    else

    {

        cout << endl

             << "--> Permutations (nPr) : " << p << endl;

        cout << endl

             << "--> Combinations (nCr) : " << c << endl

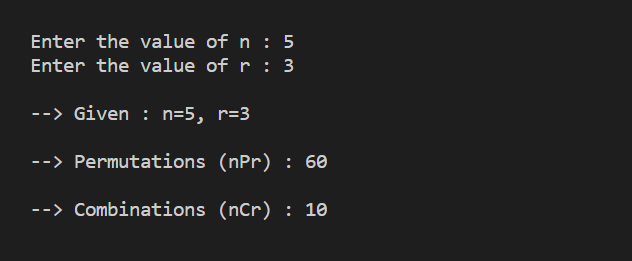
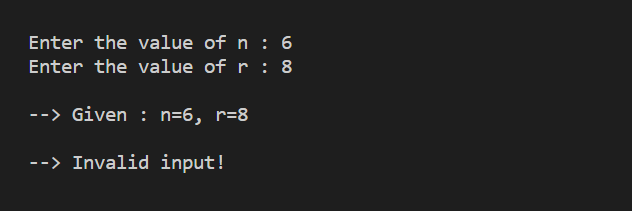
             << endl;

    }

    return 0;

}

Output :



Practical #12

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.

Code :

#include <iostream>

using namespace std;

int nCr(int n, int r)

{

    if (r > n)

        return -1;

    else if (n == 0 || r == 0 || n == r)

        return 1;

    else

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

}

void printEq(int n, int c)

{

    cout << "\nGiven equation : ";

    for (int i = 0; i < n; i++)

    {

        if (i == 0)

            cout << "x" << i + 1;

        else if (i == n - 1)

            cout << " + x" << i + 1 << " = " << c;

        else

            cout << " + x" << i + 1;

    }

    cout << endl;

}

int main()

{

    int n, c;

    cout << "\nEnter the no of variables (n) : ";

    cin >> n;

    cout << "\nEnter the value of total sum i.e. c <=10 : ";

    cin >> c;

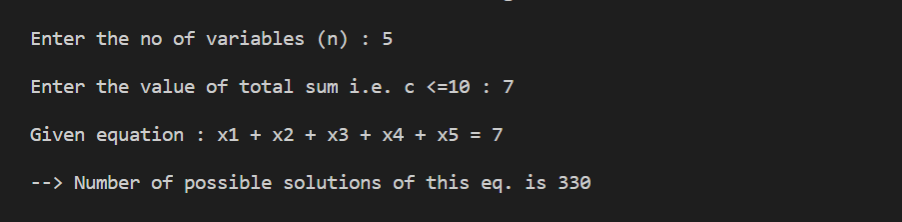
    printEq(n, c);

    cout << "\n--> Number of possible solutions of this eq. is " << nCr(n + c - 1, c) << "\n";

    return 0;

}

Output :



Practical #13

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

Code :

#include <iostream>

using namespace std;

int main()

{

    int r;

    int x[4], y[4], con[4], dis[4], xOR[4], cond[4], bicond[4], nand[4], nor[4], nx[4], ny[4];

    cout << "\nEnter the number of propositions i.e. rows : ";

    cin >> r;

    cout << "\nEnter the truth values of x and y (T=1,F=0) : \n";

    for (int i = 0; i < r; i++)

    {

        cout << "--> " << i + 1 << ". ";

        cin >> x[i] >> y[i];

        con[i] = x[i] & y[i];

        dis[i] = x[i] | y[i];

        xOR[i] = (x[i] & (!y[i])) | (y[i] & (!x[i]));

        cond[i] = (!x[i]) | y[i];

        bicond[i] = (x[i] & y[i]) | ((!x[i]) & (!y[i]));

        nand[i] = !con[i];

        nor[i] = !dis[i];

        nx[i] = !x[i];

        ny[i] = !y[i];

    }

    cout << "\n    (x,y) | Conjunction | Disjunction | XOR | Conditional | Biconditional | NAND | NOR | Negation(x) | Negation(y)\n";

    for (int i = 0; i < r; i++)

    {

        cout << "\n--> (" << x[i] << "," << y[i] << ") |      "

             << con[i] << "      |      "

             << dis[i] << "      |  "

             << xOR[i] << "  |      "

             << cond[i] << "      |       "

             << bicond[i] << "       |  "

             << nand[i] << "   |  "

             << nor[i] << "  |      "

             << nx[i] << "      |      " << ny[i];

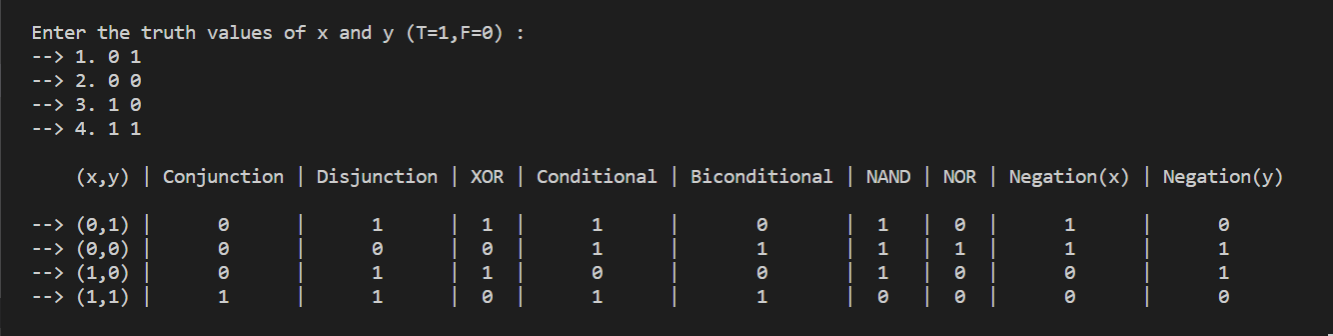
    }

    cout << endl;

    return 0;

}

Output :



Practical #14

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.

Code :

#include <iostream>

#include <math.h>

using namespace std;

int Ta(int n)

{

    if (n == 0)

        return 1;

    else

        return (n + Ta(n - 1));

}

int Tb(int n)

{

    if (n == 0)

        return 1;

    else

        return (pow(n, 2) + Tb(n - 1));

}

int Tc(int n)

{

    if (n == 0)

        return 1;

    else

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

}

int main()

{

    int n;

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

    cin >> n;

    cout << "\n--> for n=" << n << " : \n";

    cout << "\n--> 1. T(n) = T(n-1) + n, T(0) = 1 ==> " << Ta(n);

    cout << "\n--> 1. T(n) = T(n-1) + n^2, T(0) = 1 ==> " << Tb(n);

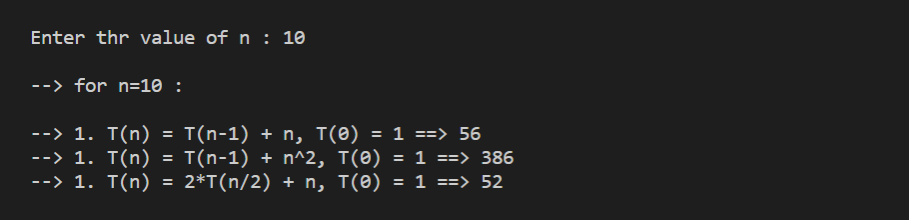
    cout << "\n--> 1. T(n) = 2\*T(n/2) + n, T(0) = 1 ==> " << Tc(n);

    cout << endl;

    return 0;

}

Output :



Practical #15

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)).

Code :

#include <iostream>

#include <math.h>

using namespace std;

void printPoly(int \*poly, int n)

{

    cout << endl

         << "Given polynomial, f(x) = ";

    for (int i = n; i >= 0; i--)

    {

        if (!poly[i] == 0)

        {

            if (i == n)

                cout << poly[i] << "x^" << i;

            else

            {

                if (poly[i] >= 0)

                    cout << " + ";

                else

                    cout << " ";

                if (!i == 0)

                    cout << poly[i] << "x^" << i;

                else

                    cout << poly[i];

            }

        }

    }

}

float evaluatePoly(int \*poly, int n, int x)

{

    float total;

    for (int i = n; i >= 0; i--)

    {

        total += poly[i] \* pow(x, i);

    }

    return total;

}

int main()

{

    int \*polynomial, deg, x;

    cout << endl

         << "Enter the degree of polynomial : ";

    cin >> deg;

    polynomial[deg] = {0};

    for (int i = deg; i >= 0; i--)

    {

        if (i == 0)

            cout << "Enter the constant term : ";

        else

            cout << "Enter the coefficient of x^" << i << " : ";

        cin >> polynomial[i];

    }

    printPoly(polynomial, deg);

    cout << endl

         << endl

         << "Enter the value of x for which the polynomial is to be evaluated : ";

    cin >> x;

    cout << endl

         << "--> f(" << x << ") : " << evaluatePoly(polynomial, deg, x);

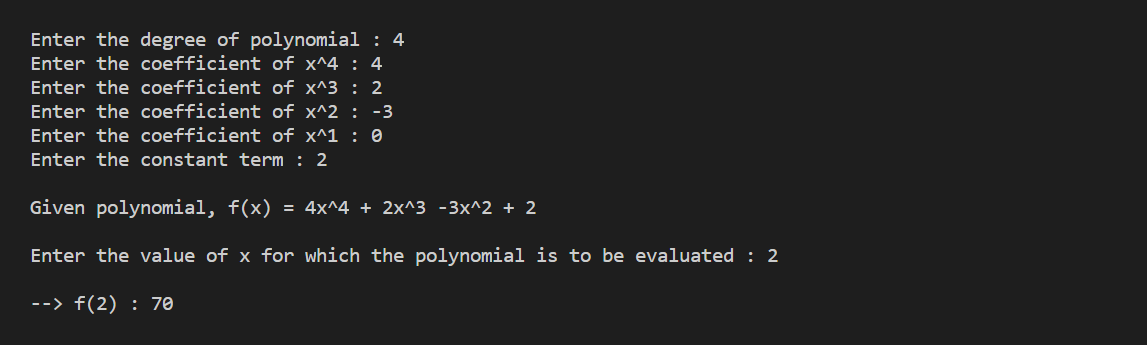
    cout << endl

         << endl;

    return 0;

}

Output :



Practical #16

Write a Program to represent Graphs using the Adjacency Matrices and check if it is a complete graph.

Code :

#include <iostream>

using namespace std;

#define N 50

int main()

{

    int adjMatrix[N][N] = {0};

    int v, e, x; //v= no of vertices , e= no of adjacent vertices, x= adjacent vertex

    int count = 0;

    cout << endl

         << endl

         << "Enter the total number of vertices : ";

    cin >> v;

    for (int i = 1; i <= v; i++)

    {

        cout << "Enter the no. of vertices adjacent to " << i << " : ";

        cin >> e;

        cout << "Enter the adjacent vertices : ";

        for (int j = 1; j <= e; j++)

        {

            cin >> x;

            adjMatrix[i - 1][x - 1] = 1;

            count++;

        }

    }

    for (int i = 0; i < v; i++)

    {

        cout << endl

             << "\t";

        for (int j = 0; j < v; j++)

        {

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

        }

    }

    if (count == ((v \* v) - v))

    {

        cout << endl

             << endl

             << "-> This is a Complete Graph" << endl;

    }

    else

    {

        cout << endl

             << endl

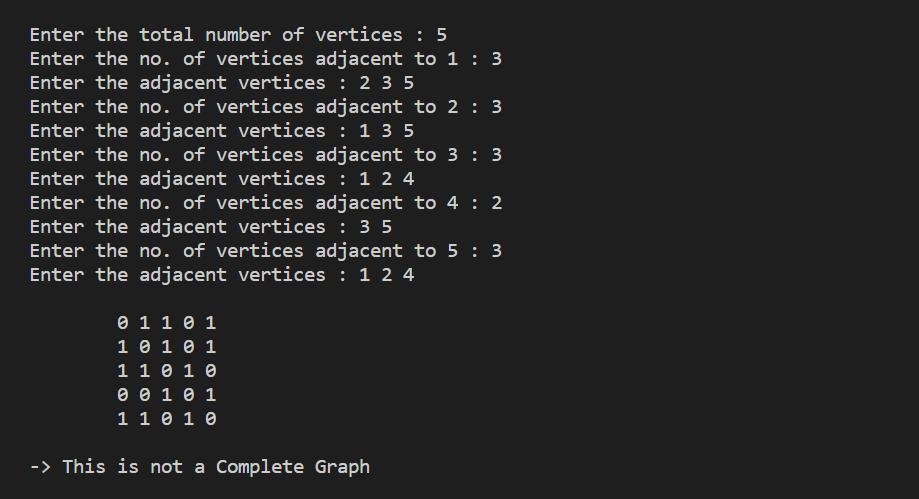
             << "-> This is not a Complete Graph" << endl;

    }

    return 0;

}

Output :



Practical #17

Write a Program to accept a directed graph G and compute the in-degree and out-degree of each vertex.

Code :

#include <iostream>

using namespace std;

#define N 25

int main()

{

    int matrix[N][N], deg[N][2] = {0};

    int v;

    char ch;

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

    cin >> v;

    for (int i = 0; i < v; i++)

    {

        cout << endl;

        for (int j = 0; j < v; j++)

        {

            cout << "=> Is there a edge " << i + 1 << "--> " << j + 1 << " ? (y/n) : ";

            cin >> ch;

            if (ch == 'Y' || ch == 'y')

            {

                matrix[i][j] = 1;

                ++deg[j][0];

                ++deg[i][1];

            }

            else

                matrix[i][j] = 0;

        }

    }

    cout << "\n--> Given Adjacency Matrix : \n";

    for (int i = 0; i < v; i++)

    {

        cout << "\n\t";

        for (int j = 0; j < v; j++)

        {

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

        }

    }

    cout << "\n\n--> Vertex | In-degree | Out-degree";

    for (int i = 0; i < v; i++)

    {

        cout << "\n-->   " << i + 1 << "    |     " << deg[i][0] << "     |     " << deg[i][1];

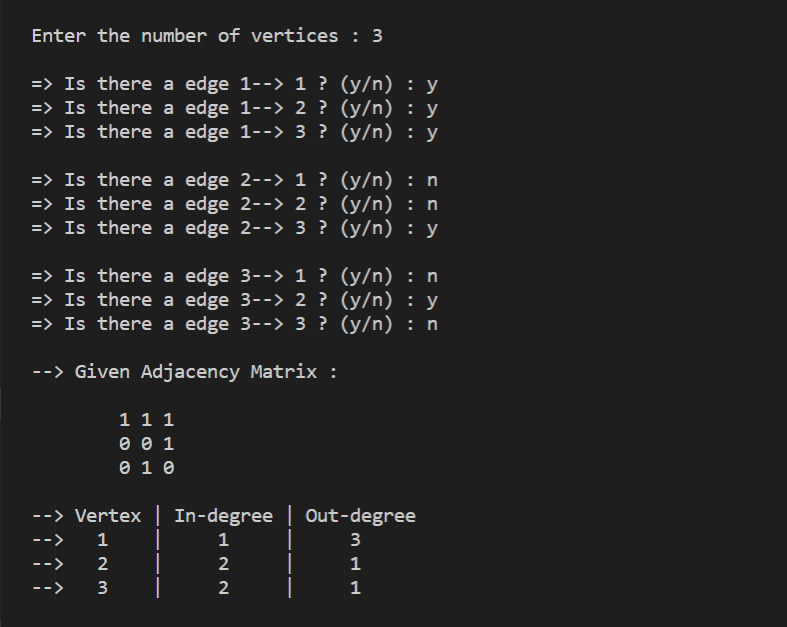
    }

    cout << endl;

    return 0;

}

Output :



Practical #18

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.

Code :

#include <iostream>

using namespace std;

#define N 10

// Function to multiply two matrices

void multiply(int a[][N], int b[][N], int res[][N], int v)

{

    int mul[N][N];

    for (int i = 0; i < v; i++)

    {

        for (int j = 0; j < v; j++)

        {

            mul[i][j] = 0;

            for (int k = 0; k < v; k++)

                mul[i][j] += a[i][k] \* b[k][j];

        }

    }

    // Storing the multiplication result in res[][]

    for (int i = 0; i < v; i++)

        for (int j = 0; j < v; j++)

            res[i][j] = mul[i][j];

}

// Function to compute G raised to the power n

void power(int G[N][N], int res[N][N], int n, int v)

{

    // Base condition

    if (n == 1)

    {

        for (int i = 0; i < v; i++)

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

                res[i][j] = G[i][j];

        return;

    }

    // Recursion call for first half

    power(G, res, n / 2, v);

    // Multiply two halves

    multiply(G, G, res, v);

    // If n is odd

    if (n % 2 != 0)

        multiply(res, G, res, v);

}

int main()

{

    int n, source, dest, res[N][N];

    int G[N][N];

    int v;

    char ch;

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

    cin >> v;

    for (int i = 0; i < v; i++)

    {

        cout << endl;

        for (int j = 0; j < v; j++)

        {

            cout << "=> Is there a edge " << i + 1 << "--> " << j + 1 << " ? (y/n) : ";

            cin >> ch;

            if (ch == 'Y' || ch == 'y')

            {

                G[i][j] = 1;

            }

            else

                G[i][j] = 0;

        }

    }

    cout << "\n--> Given Adjacency Matrix : \n";

    for (int i = 0; i < v; i++)

    {

        cout << "\n\t";

        for (int j = 0; j < v; j++)

        {

            cout << G[i][j] << " ";

        }

    }

    cout << "\n\nEnter the path length : ";

    cin >> n;

    cout << "Enter the source : ";

    cin >> source;

    cout << "Enter the destination : ";

    cin >> dest;

    power(G, res, n, v);

    cout << "\n--> Number of paths of length " << n

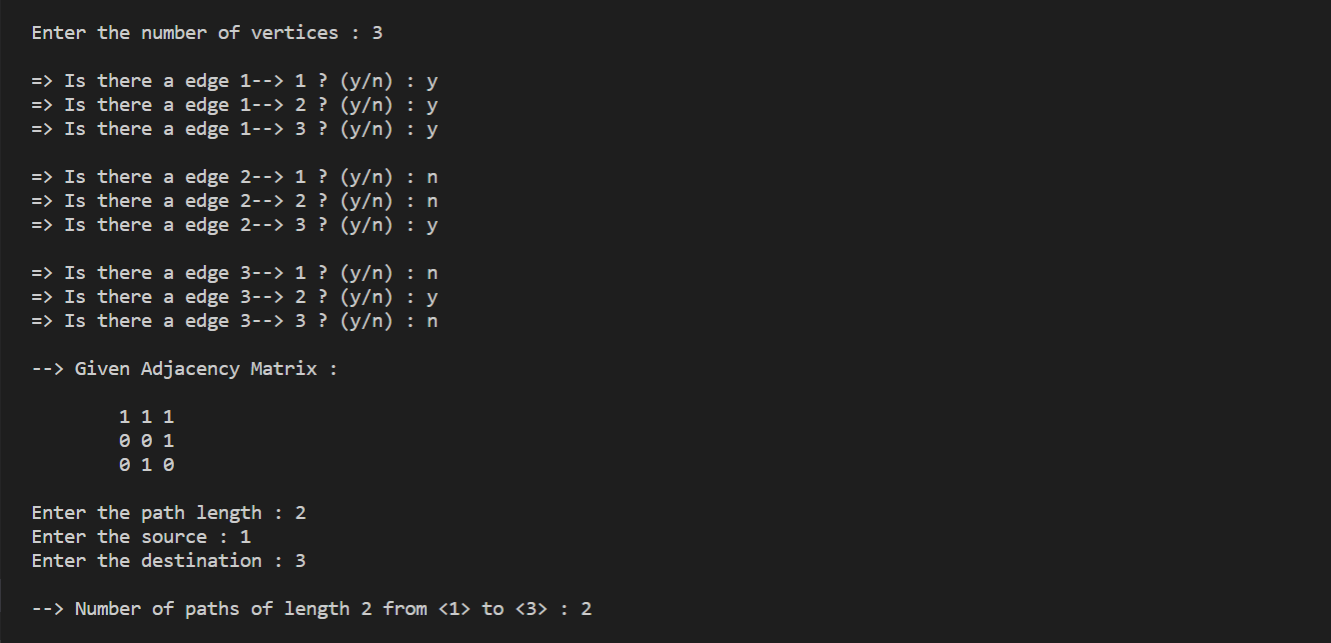
         << " from <" << source << "> to <" << dest << "> : "

         << res[source - 1][dest - 1] << "\n";

    return 0;

}

Output :



Practical #19

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).

Code :

#include <iostream>

using namespace std;

#define N 25

int main()

{

    int \*\*matrix, \*deg, \*vertices;

    int v, n, sum = 0, flag = 1, count = 0;

    cout << "\nEnter the dimension of matrix : ";

    cin >> v;

    matrix = new int \*[v];

    deg = new int[v];

    for (int i = 0; i < v; i++)

    {

        matrix[i] = new int[v];

        cout << "Enter the elements in row " << i + 1 << " : ";

        for (int j = 0; j < v; j++)

        {

            cin >> matrix[i][j];

        }

    }

    cout << "\n--> Adjacency Matrix \n ";

    for (int m = 0; m < v; m++)

    {

        cout << "\n    ";

        for (int n = 0; n < v; n++)

            cout << matrix[m][n] << " ";

    }

    for (int i = 0; i < v; i++)

    {

        sum = 0;

        for (int j = 0; j < v; j++)

        {

            sum += matrix[i][j];

        }

        deg[i] = sum;

    }

    cout << "\n\nEnter the no of vertices followed by the vertices to be checked : ";

    cin >> n;

    vertices = new int[n];

    for (int i = 0; i < n; i++)

    {

        cin >> vertices[i];

    }

    for (int i = 0; i < n; i++)

    {

        cout << "\n--> Degree of Vertex " << vertices[i] << " : " << deg[vertices[i] - 1];

    }

    for (int i = 0; i < n; i++)

    {

        if ((deg[vertices[i] - 1] % 2) != 0)

        {

            flag = 0;

            count++;

        }

    }

    if (flag == 1)

        cout << "\n\n--> There is an euler circuit.";

    else

        cout << "\n\n--> There is no euler circuit.";

    if (count == 2 || flag == 1)

        cout << "\n\n--> There is an euler path.";

    else

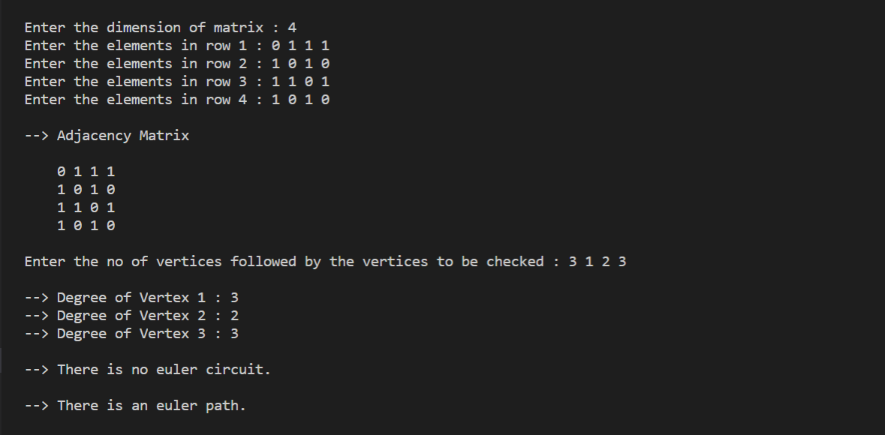
        cout << "\n\n--> There is no euler path.";

    cout << endl;

    return 0;

}

Output :



Practical #20

Given a full m-ary tree with i internal vertices, Write a Program to find the number of leaf nodes.

Code :

#include <iostream>

using namespace std;

int calcLeafNodes(int m, int i)

{

    return i \* (m - 1) + 1;

}

int main()

{

    int m, i;

    cout << endl

         << endl

         << "Enter the degree of tree : ";

    cin >> m;

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

    cin >> i;

    cout << endl

         << "--> Number of leaf nodes : " << calcLeafNodes(m, i) << endl

         << endl;

    return 0;

}

Output :

