**ATMA RAM SNATAN DHARMA COLLEGE**

**DISCRETE STRUCTURE**

**PRACTICAL FILE**

**Submitted by :**

**Arunesh Singh**

**18016**

**BSc.(H.) CS**

**Submitted to :**

**Dr Shalini Gupta**

**Department of Computer Science**

**1. 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;

bool ismember(int set[] , int n, int key){

    for(int i=0 ; i<n; i++){

        if(key == i){

            return true;

        }

    }

    return false;

}

void powerSet(int set[], int n){

    int pow\_size = pow(2,n);

    cout<<"Power Set : "<<endl;

    for(int i=0 ; i<pow\_size; i++){

        cout<<"(";

        for(int j=0 ; j<n; j++){

            if( i & ( 1<<j )){

                cout<<set[j]<<" ";

            }

        }

        cout<<")";

    }

}

int main(){

    int n;

    cout<<"No of elements : ";

    cin>>n;

    int set[n];

    cout<<"Enter elements : ";

    for(int i=0 ; i<n; i++){

        cin>>set[i];

    }

    int key;

    cout<<"Enter an element to check : ";

    cin>>key;

    if(ismember(set,n,key))

        cout<<key<<" is a member of the set.\n";

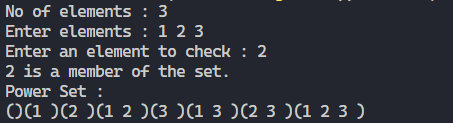
    else

        cout << key << " is not a member of the set.\n";

    powerSet(set,n);

}

**Output:**



**2. 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 "bits/stdc++.h"

using namespace std;

class SET{

    public:

    int size;

    vector<int> set;

    SET(int n) {

        size = n;

        cout<<"\nEnter the elements : ";

        for(int i=0 ; i<size ; i++){

            int t;

            cin>>t;

            set.push\_back(t);

        }

    }

};

int isSubset(vector<int> A, vector<int> B)

{

    sort(A.begin(), A.end());

    for (auto element : B)

    {

        if (binary\_search(A.begin(), A.end(), element) == false)

        {

            return 0;

        }

    }

    return 1;

}

vector<int> unionSet(vector<int> A, vector<int> B)

{

    sort(A.begin(), A.end()) ;

    for (auto element : B )

    {

        if(binary\_search(A.begin(),A.end(),element) == false)

            A.push\_back(element);

    }

    return A;

}

vector<int> intersectionSet(vector<int> A, vector<int> B)

{

    vector<int> iSet;

    sort(A.begin(), A.end());

    for(auto element : B)

    {

        if(binary\_search(A.begin(),A.end(),element) == true)

            iSet.push\_back(element);

    }

    return iSet;

}

vector<int> complement(vector<int> A, vector<int> B)

{

    vector <int> cset;

    sort(B.begin(), B.end());

    for (auto element : A)

    {

        if (binary\_search(B.begin(), B.end(), element) == false)

            cset.push\_back(element);

    }

    return cset;

}

vector<int> SetDiff(vector<int> A, vector<int> B)

{

    vector<int> iset(intersectionSet(A,B));

    vector<int> set\_diff(complement(A,iset));

    return set\_diff;

}

vector<int> SymmDiff(vector<int> A, vector<int> B)

{

    vector <int> uset(unionSet(A,B));

    vector <int> iset(intersectionSet(A,B));

    return SetDiff(uset,iset);

}

void cartesianP(vector<int> A , vector<int> B){

    vector<pair<int,int>> cp;

    for(auto one : A){

        for(auto two : B){

            cp.push\_back(make\_pair(one,two));

        }

    }

    for(auto ele : cp){

        cout<<ele.first<<",";

        cout<<ele.second<<" ; ";

    }cout<<endl;

}

void menu(SET A, SET B){

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

    cout << "0.Exit" << endl;

    cout << "1.Subset - Check whether one set is a subset of other or not" << endl;

    cout << "2.Union  of the two sets" << endl;

    cout << "3.Intersection of the two sets" << endl;

    cout << "4.Complement" << endl;

    cout << "5.Set difference" << endl;

    cout << "6.Symmetric Difference" << endl;

    cout << "7.Cartesian product" << endl;

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

    cout << "Please enter your choice:";

    int t;

    cin>>t;

    vector <int> arr;

    switch (t)

    {

    case 0:

        return;

    case 1:

        if(isSubset(A.set,B.set) == 1){

            cout<<"\nB is subset of A"<<endl;

        }

        else{

            cout<<"\nB is not subset of A"<<endl;

        }

        break;

    case 2:

        arr = (unionSet(A.set,B.set));

        cout << "\nUnion Set of Set A and Set B : ";

        for (auto element : arr)

        {

            cout << element << " ";

        }

        cout << endl;

        break;

    case 3:

        arr = (intersectionSet(A.set, B.set));

        cout << "\nInersection Set of Set A and Set B : ";

        for (auto element : arr)

        {

            cout << element << " ";

        }

        cout << endl;

        break;

    case 4:

        if (isSubset(A.set, B.set) == 1)

        {

            arr = (complement(A.set, B.set));

            cout << "\nComplement Set of Set B : ";

            for (auto element : arr)

            {

                cout << element << " ";

            }

            cout << endl;

        }

        else

        {

            cout << "\nB is not subset of A , therefore its complement cannot be determined." << endl;

        }

        break;

    case 5:

        arr = (SetDiff(A.set, B.set));

        cout << "\nSet Difference of Set B from Set A : ";

        for (auto element : arr)

        {

            cout << element << " ";

        }

        cout << endl;

        break;

    case 6:

        arr = (SymmDiff(A.set, B.set));

        cout << "\nSymmetric Diff of Set A and Set B : ";

        for (auto element : arr)

        {

            cout << element << " ";

        }

        cout << endl;

        break;

    case 7:

        cartesianP(A.set , B.set);

        break;

    default:

        cout<<"\nInvalid Input"<<endl;

        break;

    }

    return menu(A,B);

}

int main(){

    int n,m;

    cout<<"\nEnter no of elements of set A : ";

    cin>>n;

    SET A(n);

    cout<<"\nEnter no of elements of set B : ";

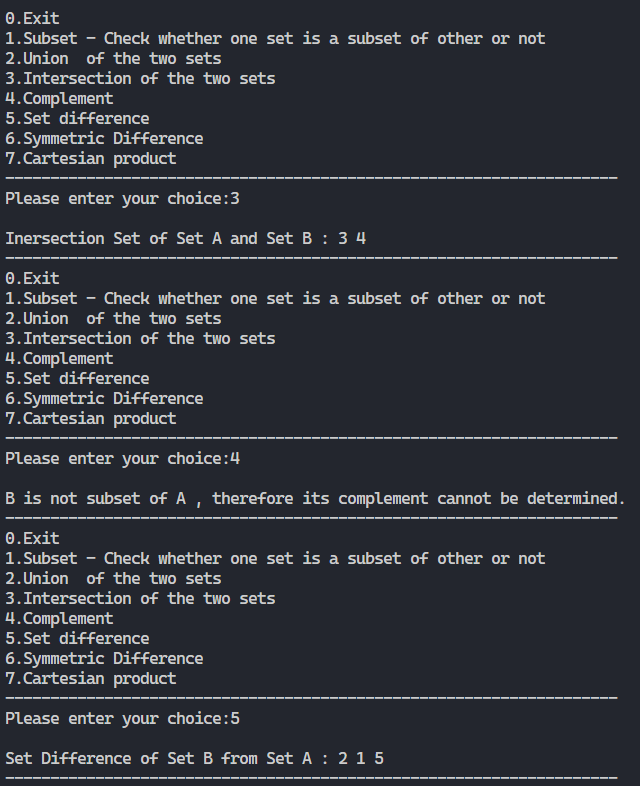
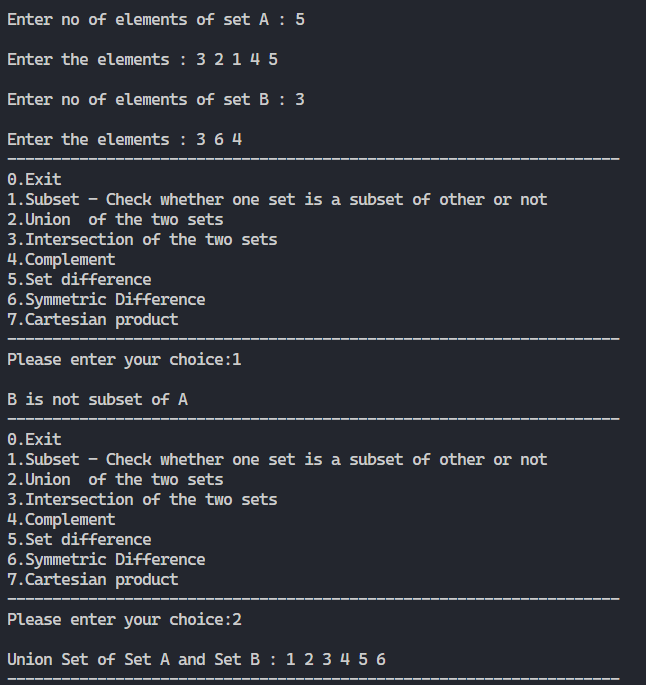
    cin>>m;

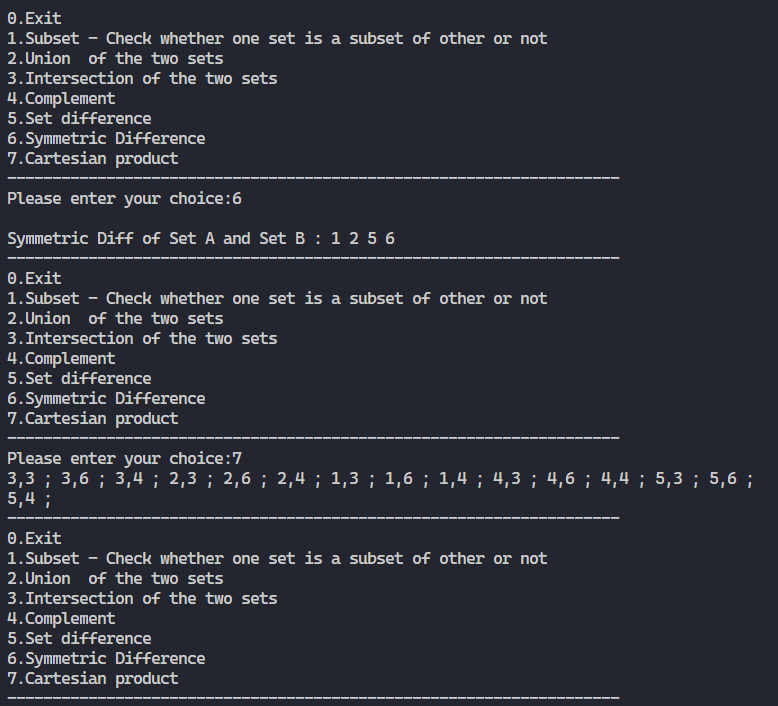
    SET B(m);

    menu(A,B);

    return 0;

}

**Output:**

****

**3. 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 "bits/stdc++.h"

using namespace std;

class Relation {

    public:

    vector<vector<int>> rel;

    Relation(int n){

        cout<<"Enter Matrix Representation of Relation :"<<endl;

        for (int i = 0; i < n; i++)

        {

            vector<int> temp;

            for(int j=0 ; j<n ; j++){

                int a;

                cin>>a;

                temp.push\_back(a);

            }

            rel.push\_back(temp);

        }

    }

    void display(){

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

        {

            for (int j = 0; j < rel[i].size(); j++)

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

            cout << endl;

        }

    }

    bool isReflexive(){

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

        {

            if(rel[i][i] != 1){

                return false;

            }

        }

        return true;

    }

    bool isSymmetric(){

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

        {

            for (int j = 0; j < rel[i].size(); j++)

            {

                if (rel[i][j] == 1 && rel[j][i] != 1)

                {

                    return false;

                }

            }

        }

        return true;

    }

    bool isTransitive(){

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

        {

            for (int j = 0; j < rel[i].size(); j++)

            {

                if (rel[i][j] == 1 && rel[i][j+1]==1 && rel[i+1][j+1] !=1 )

                {

                    return false;

                }

            }

        }

        return true;

    }

};

int main(){

    int n;

    cout<<"Enter size of matrix : ";

    cin>>n;

    Relation r1(n);

    // r1.display();

    if(r1.isReflexive()){

        cout<<"Relation is reflexive"<<endl;

    }

    else{

        cout<<"Relation is not reflexive"<<endl;

    }

    if(r1.isSymmetric()){

        cout << "Relation is Symmetric" << endl;

    }

    else{

        cout << "Relation is not Symmetric" << endl;

    }

    if(r1.isTransitive()){

        cout << "Relation is Transitive" << endl;

    }

    else{

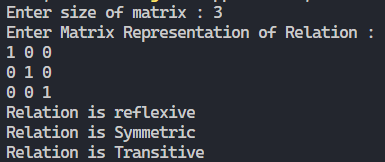
        cout << "Relation is not Transitive" << endl;

    }

    return 0;

}

**Output:**

****

**4. 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 "bits/stdc++.h"

using namespace std;

bool isEquivalent(int \*arr, int s)

{

    if (isTransitive(arr, s) == 1 && isReflexive(arr, s) == 1 && isSymmetric(arr, s) == 1)

        return true;

    else

        return false;

}

bool isPartialOrderRelation(int \*arr, int s)

{

    if (isTransitive(arr, s) == 1 && isReflexive(arr, s) == 1 && isAntiSymmetric(arr, s) == 1)

        return true;

    else

        return false;

}

int main()

{

    int s, e;

    vector<int> set;

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

    cin >> s;

    int arr[s][s];

    for (int i = 0; i < s; i++) // initializing the whole matrix to 0

    {

        for (int j = 0; j < s; j++)

        {

            arr[i][j] = 0;

        }

    }

    cout << "Enter the  set on which the relation is defined :" << endl;

    for (int i = 0; i < s; i++)

    {

        cin >> e;

        set.push\_back(e);

    }

    int n;

    cout << "Enter the number of pairs in the relation :";

    cin >> n;

    pair<int, int> rSet[n];

    for (int i = 0; i < n; i++)

    {

        cout << "Enter the elements of the relation (pairwise) :" << endl;

        cin >> rSet[i].first;

        cin >> rSet[i].second;

    }

    int a, b;

    for (int i = 0; i < n; i++)

    {

        a = rSet[i].first;

        b = rSet[i].second;

        arr[a][b] = 1;

    }

    cout << "Matrix Representation of the above relation :" << endl;

    for (int i = 0; i < s; i++)

    {

        for (int j = 0; j < s; j++)

            cout << arr[i][j] << " ";

        cout << endl;

    }

    string ans1, ans2;

    ans1 = isEquivalent((int \*)arr, s) ? "Relation is Eqivalent" : "Relation is not Equivalent";

    cout << ans1 << endl;

    ans2 = isPartialOrderRelation((int \*)arr, s) ? "Relation is Partial Order Relation" : "Relation is not a  Partial Order Relation";

    cout << ans2 << endl;

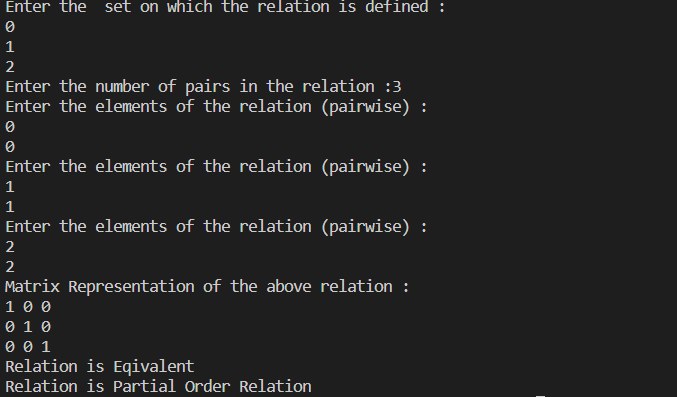
    if (ans1 == "Relation is not Equivalent" and ans2 == "Relation is not a  Partial Order Relation")

        cout << "Relation is neither Equivalent nor Partial Order Relation. " << endl;

    return 0;

}

**Output:**



**5. Write a Program to generate the Fibonacci Series using recursion.**

**Code:**

#include <iostream>

using namespace std;

int fibbo(int n){

    if(n < 2){

        return n;

    }

    return fibbo(n-1) + fibbo(n-2);

}

int main(){

    int n;

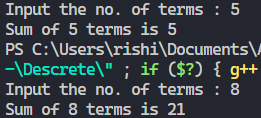
    cout<<"Input the no. of terms : ";

    cin>>n;

    cout<<"Sum of "<<n<<" terms is "<<fibbo(n)<<endl;

}

**Output:**

****

**6. Write a Program to implement Tower of Hanoi using recursion.**

**Code:**

#include <iostream>

using namespace std;

void towerOfHanoi(int n, char src, char dest, char help){

    if(n==0){

        return;

    }

    towerOfHanoi(n-1,src,help,dest);

    cout<<"Moved from "<<src<<" to "<<dest<<endl;

    towerOfHanoi(n-1,help,dest,src);

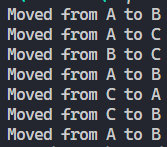
}

int main(){

    towerOfHanoi(3,'A','B','C');

}

**Output:**

****

**7. Write a Program to implement binary search using recursion.**

**Code:**

#include <iostream>

using namespace std;

int binarySearch(int arr[],int start, int end, int key){

    int mid = start + (end-start)/2;

    if(start>end)

        return -1;

    if(arr[mid]==key)

        return mid;

    else if(arr[mid]<key)

        binarySearch(arr,mid+1,end,key);

    else

        binarySearch(arr,start,mid-1,key);

    return -1;

}

int main(){

    int n;

    cout<<"Enter size of array : ";

    cin>>n;

    int arr[n];

    cout<<"Enter elements of the array :\n";

    for(int i=0 ; i<n ; i++){

        cin>>arr[i];

    }

    int key;

    cout<<"Enter an element to search : ";

    cin>>key;

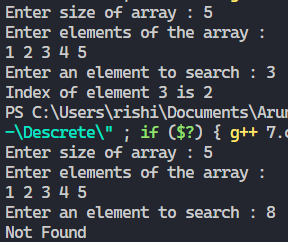
    int ans = binarySearch(arr,0,n,key);

    if(ans == -1)

        cout<<"Not Found"<<endl;

    else

        cout<<"Index of element "<<key<<" is "<<ans<<endl;

****    return 0;

}

**Output:**

**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). (One method is given in Liu)**

**Code:**

#include "bits/stdc++.h"

using namespace std;

vector<vector<int>> ans;

void permute(vector<int> &a, int idx)

{

    if (idx == a.size())

    {

        ans.push\_back(a);

        return;

    }

    for (int i = idx; i < a.size(); i++)

    {

        swap(a[i], a[idx]);

        permute(a, idx + 1);

        swap(a[i], a[idx]);

    }

    return;

}

int main()

{

    int n;

    cout << "Enter lenght of set : ";

    cin >> n;

    vector<int> a(n);

    cout << "Enter the elements : ";

    for (auto &i : a)

    {

        cin >> i;

    }

    permute(a, 0);

    cout << "Permutations : \n";

    for (auto v : ans)

    {

        for (auto i : v)

        {

            cout << i << " ";

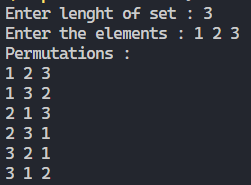
        }

        cout << endl;

    }

}

**Output:**

****

**11. Write a Program to calculate Permutation and Combination for an input value n and r using recursive formula of nCr and nPr .**

**Code:**

#include "bits\stdc++.h"

using namespace std;

int permutation(int n, int r){

    if (r == 0)

        return 1;

    if (r > n)

        return 0;

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

}

int combination(int n, int r){

    if (r == 0 || r == n)

        return 1;

    return combination(n - 1, r) + combination(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 << "): " << permutation(n, r);

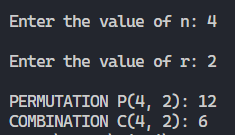
    cout << "\nCOMBINATION "

         << "C(" << n << ", " << r << "): " << combination(n, r);

    return 0;

}

**Output:**

****

**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 countSolutions(int n, int val)

{

    int total = 0;

    if (n == 1 && val >= 0)

        return 1;

    for (int i = 0; i <= val; i++)

    {

        total += countSolutions(n - 1, val - i);

    }

    return total;

}

int main()

{

    int n = 5;

    int val = 20;

    cout << countSolutions(n, val);

}

**Output:**

****

**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 "bits/stdc++.h"

using namespace std;

void disp(vector<int> x, vector<int> y, vector<int> res){

    cout << "x\ty\tres"<<endl;

    for (int i = 0; i < 4; i++)

        cout << x[i] << "\t" << y[i] << "\t" << res[i]<<endl;

}

void conjuction(vector<int> x, vector<int> y)

{

    vector<int> res;

    for (int i = 0; i < 4; i++)

    {

        res.push\_back(x[i] \* y[i]);

    }

    disp(x,y,res);

}

void disjunction(vector<int> x, vector<int> y)

{

    vector<int> res;

    for (int i = 0; i < 4; i++)

    {

        res.push\_back(x[i] || y[i]);

    }

    disp(x,y,res);

}

void negation(vector<int> x, vector<int> y)

{

    vector<int> res;

    for (int i = 0; i < 4; i++)

    {

        res.push\_back(!x[i]);

    }

    for (int i = 0; i < 4; i++)

    {

        res.push\_back(!y[i]);

    }

    disp(x,y,res);

}

void nAND(vector<int> x, vector<int> y)

{

    vector<int> res;

    for (int i = 0; i < 4; i++)

    {

        res.push\_back(!(x[i] \* y[i]));

    }

    disp(x,y,res);

}

void nOR(vector<int> x, vector<int> y)

{

    vector<int> res;

    for (int i = 0; i < 4; i++)

    {

        res.push\_back(!(x[i] || y[i]));

    }

    disp(x,y,res);

}

void xOR(vector<int> x, vector<int> y)

{

    // odd 1's counter , this valid for 2 input xOR gate

    vector<int> res;

    for (int i = 0; i < 4; i++)

    {

        if (x[i] == y[i])

            res.push\_back(0);

        else

            res.push\_back(1);

    }

    disp(x,y,res);

}

void xNOR(vector<int> x, vector<int> y)

{

    vector<int> res;

    for (int i = 0; i < 4; i++)

    {

        if (x[i] == y[i])

            res.push\_back(1);

        else

            res.push\_back(0);

    }

    disp(x,y,res);

}

void biConditional(vector<int> x, vector<int> y)

{

    vector<int> res;

    for (int i = 0; i < 4; i++)

    {

        res.push\_back((!x[i] + y[i]) \* (!y[i] + x[i]));

    }

    disp(x,y,res);

}

void conditnl(vector<int> x, vector<int> y)

{

    vector<int> res;

    for (int i = 0; i < 4; i++)

    {

        res.push\_back(!x[i] || y[i]);

    }

    disp(x,y,res);

}

void menu(vector<int> x, vector<int> y)

{

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

    cout << "Please select an operation from the list below:" << endl;

    cout << "1.Conjuction - AND" << endl;

    cout << "2.Disjunction - OR " << endl;

    cout << "3.Negation " << endl;

    cout << "4.NAND " << endl;

    cout << "5.NOR" << endl;

    cout << "6.Exclusive OR " << endl;

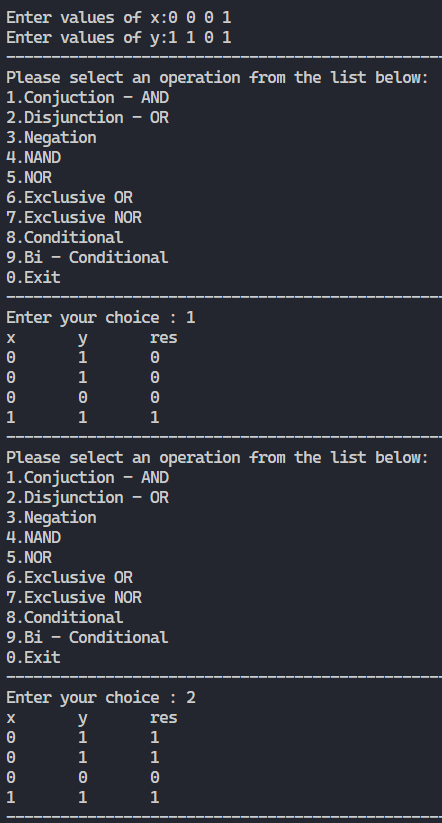
    cout << "7.Exclusive NOR" << endl;

    cout << "8.Conditional " << endl;

    cout << "9.Bi - Conditional " << endl;

    cout << "0.Exit" << endl;

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

****

    int ch;

    cout<<"Enter your choice : ";

    cin>>ch;

    switch (ch)

    {

    case 1:

        conjuction(x,y);

        break;

    case 2:

        disjunction(x,y);

        break;

    case 3:

        negation(x,y);

        break;

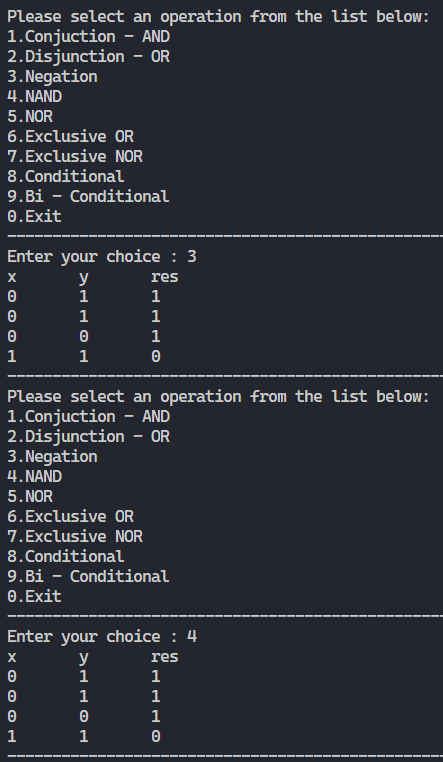
    case 4:

        nAND(x,y);

        break;

    case 5:

        nOR(x,y);

****        break;

    case 6:

        xOR(x,y);

        break;

    case 7:

        xNOR(x,y);

        break;

    case 8:

        conditnl(x,y);

        break;

    case 9:

        biConditional(x,y);

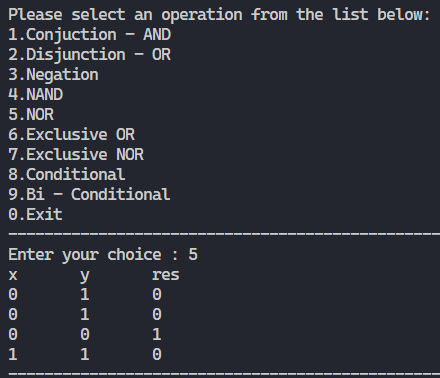
        break;

    case 0:

        return;

    default:

        cout<<"Invalid Input\n";

        break;

    }

    return menu(x,y);

}

int main(){

    vector<int> x(4);

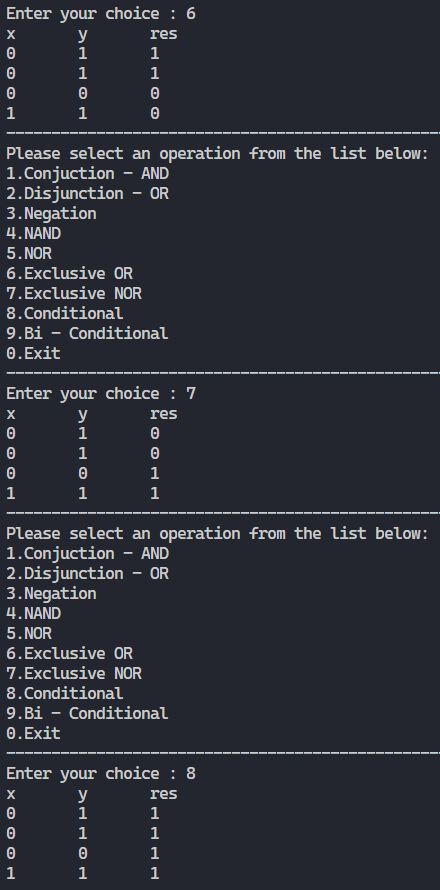
    vector<int> y(4);

    cout << "\nEnter values of x:";

    for (auto &i : x)

    {

        cin >> i;

    }

    cout << "Enter values of y:";

    for (auto &i : y)

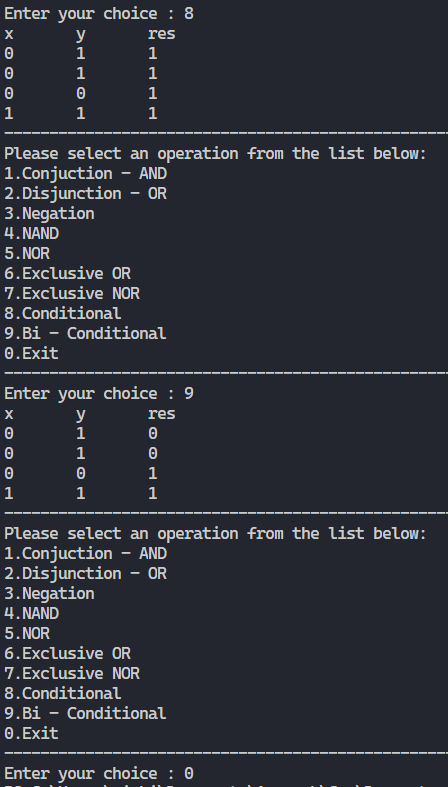
    {

        cin >> i;

    }

    menu(x,y);

}

**Output:**

**15. Write a Program to store a function (polynomial/exponential), and then evaluate thepolynomial. (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;

int main(){

    int n;

    cout<<"Enter the highest power of x : ";

    cin>>n;

    int arr[n];

    cout<<"Enter the coefficient of x (degree 0 to "<<n<<") : ";

    for(int i=0 ; i<n ; i++){

        cin>>arr[i];

    }

    int k;

    cout<<"Enter the value of x : ";

    cin>>k;

    int ans = 0;

    for(int i=0 ; i<=n ; i++){

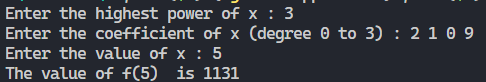
        ans += arr[i]\*pow(k,i);

    }

    cout<<"The value of f("<<k<<")  is "<<ans<<endl;

}

**Output:**

****

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

**Code:**

/\*

    I have cerated 2 vectors. The first vector will take edge as

    input from from user. Then by using createAdjMAtrix function

    we are converting the first vector into Adjacency Matrix(vector 2)

\*/

#include <iostream>

using namespace std;

int main()

{

    int e, n;

    cout << "Enter number of vertices:";

    cin >> n;

    cout << "Enter total number of edges:";

    cin >> e;

    int arr[n][n];

    for (int i = 0; i < n; i++) // initializing the whole matrix to 0

    {

        for (int j = 0; j < n; j++)

        {

            arr[i][j] = 0;

        }

        cout << endl;

    }

    int v;

    int u;

    for (int i = 0; i < e; i++)

    {

        cout << "Enter the edge:" << endl;

        cin >> u >> v;

        arr[v][u] = 1;

        arr[u][v] = 1;

    }

    cout << "Adjacency matrix for the data provided:" << endl;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

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

        }

        cout << endl;

    }

    // Checking whether the graph is complete or not.

    // Principal diagonal elements will be 0.

    bool isComplete = true;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            if (arr[i][i] != 0)

                isComplete = false;

        }

    }

    if (isComplete)

        cout << "The graph is a complete graph." << endl;

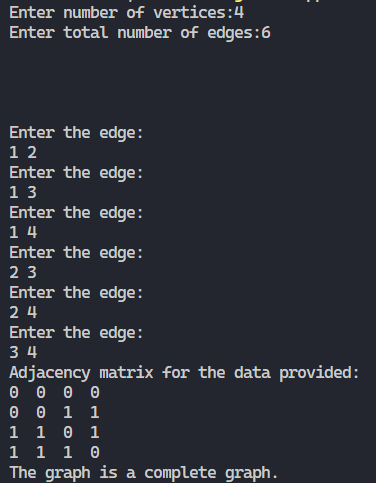
    else

        cout << "Graph is an incomplete graph." << endl;

    return 0;

}

**Output:**

****

**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;

class Edge

{

public:

    int startingVertex;

    int endingVertex;

    void addEdge(int v, int u)

    {

        startingVertex = v;

        endingVertex = u;

    }

    void dispEdge()

    {

        cout << "(" << startingVertex << "," << endingVertex << ")"

             << "  ";

    }

};

int main()

{

    int n;

    cout << "Enter total number of vertices in the graph:";

    cin >> n;

    int vertexList[n];

    for (int i = 0; i < n; i++)

    {

        vertexList[i] = i;

    }

    int e;

    cout << "Enter total number of edges in the graph:";

    cin >> e;

    Edge edgeList[e];

    int incidenceMatrix[n][e];

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < e; j++)

        {

            incidenceMatrix[i][j] = 0;

        }

    }

    int v = 0, u = 0;

    for (int i = 0; i < e; i++)

    {

        cout << "Enter edge:" << endl;

        cin >> v >> u;

        edgeList[i].addEdge(v, u);

    }

    cout << "You entered the following edges:" << endl;

    for (int i = 0; i < e; i++)

    {

        edgeList[i].dispEdge();

    }

    cout << endl;

    /\* Populating the incidence  matrix for the graph entered

        --> n- number of vertices e- number of edges

    \*/

    int i = 0, j = 0;

    for (int k = 0; k < e; k++)

    {

        i = edgeList[k].startingVertex;

        j = edgeList[k].endingVertex;

        incidenceMatrix[i][k] = 1;

        incidenceMatrix[j][k] = -1;

    }

    // displaying the adjacency matrix of the given graph

    cout << "Incidence  Matrix for the provided digraph" << endl;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < e; j++)

        {

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

        }

        cout << "\n";

    }

    // calulating the in degree and out degree of each vertex in the graph using the adjacency matrix generated

    for (int i = 0; i < n; i++)

    {

        int outdeg = 0;

        int indeg = 0;

        for (int j = 0; j < e; j++)

        {

            if (incidenceMatrix[i][j] == 1)

                outdeg++;

            if (incidenceMatrix[i][j] == -1)

                indeg++;

        }

        cout << vertexList[i] << "\t"

             << "Indegree:" << indeg << " |"

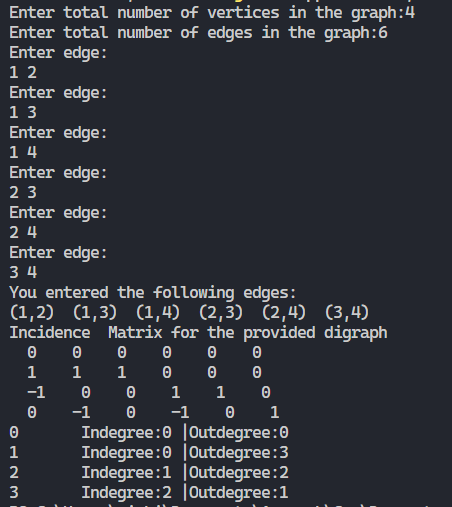
             << "Outdegree:" << outdeg << endl;

    }

    return 0;

}

**Output:**

****

**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;

class Edge

{

public:

    int startingVertex;

    int endingVertex;

    void addEdge(int v, int u)

    {

        startingVertex = v;

        endingVertex = u;

    }

    void dispEdge()

    {

        cout << "(" << startingVertex << "," << endingVertex << ")"

             << "  ";

    }

};

void power(int \*adjMatrix, int \*I, int \*res, int n)

{

    for (int i = 0; i < n; ++i)

    {

        for (int j = 0; j < n; ++j)

        {

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

            {

                \*(res + i \* n + j) += (\*(adjMatrix + i \* n + k)) \* (\*(I + k \* n + j));

            }

        }

    }

    // Updating identity matrix

    for (int i = 0; i < n; ++i)

    {

        for (int j = 0; j < n; ++j)

        {

            \*(I + i \* n + j) = \*(res + i \* n + j);

        }

    }

}

void multiply(int \*adjMatrix, int \*I, int \*res, int n, int k)

{

    for (int i = 1; i <= k; ++i)

    {

        power((int \*)adjMatrix, (int \*)I, (int \*)res, n);

    }

}

int main()

{

    int e, n;

    cout << "Enter number of vertices:";

    cin >> n;

    int adjMatrix[n][n];

    int I[n][n];

    int res[n][n];

    for (int i = 0; i < n; i++) // Identity Matrix

    {

        for (int j = 0; j < n; j++)

        {

            if (i == j)

                I[i][j] = 1;

            else

                I[i][j] = 0;

        }

        cout << endl;

    }

    for (int i = 0; i < n; i++) // initializing the whole matrix to 0

    {

        for (int j = 0; j < n; j++)

        {

            adjMatrix[i][j] = 0;

            res[i][j] = 0;

        }

        cout << endl;

    }

    cout << "Enter total number of edges:";

    cin >> e;

    Edge edgeList[e];

    int v = 0, u = 0;

    for (int i = 0; i < e; i++)

    {

        cout << "Enter edge:" << endl;

        cin >> v >> u;

        edgeList[i].addEdge(v, u);

    }

    int i = 0, j = 0;

    for (int k = 0; k < e; k++)

    {

        i = edgeList[k].startingVertex;

        j = edgeList[k].endingVertex;

        adjMatrix[i][j] = 1;

        adjMatrix[j][i] = 1;

    }

    cout << "Adjacency matrix for the data provided:" << endl;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

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

        }

        cout << endl;

    }

    // finding the path of length k from src to dest.

    int src = 0, dest = 0, k = 0;

    cout << "Enter the source:";

    cin >> src;

    cout << "Enter the destination:";

    cin >> dest;

    cout << "Enter the length of the path:";

    cin >> k;

    multiply((int \*)adjMatrix, (int \*)I, (int \*)res, n, k);

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

    cout << "Adjacency matrix raised to the power " << k << endl;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

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

        }

        cout << endl;

    }

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

    cout << "Number of paths of length " << k << " between the nodes " << src << " and " << dest << " are: " << res[src][dest] << endl;

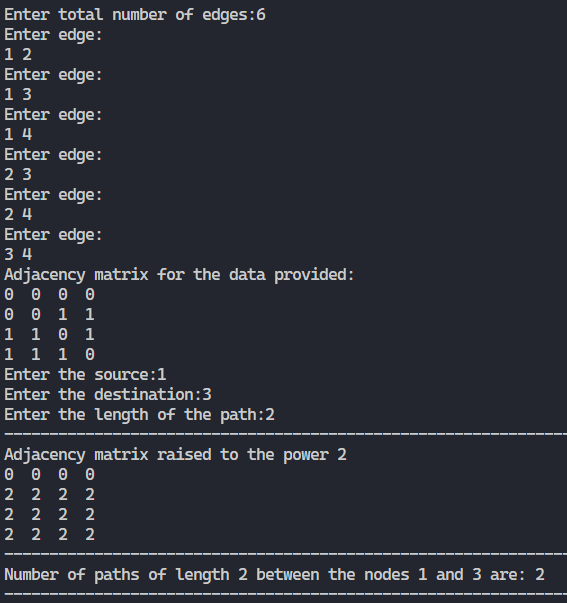
    cout << "-------------------------------------------------------------------------------------" << endl;

    return 0;

}

**Output:**

****

****

**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;

void check(int degList[], int n)

{

    int evenDeg = 0;

    int oddDeg = 0;

    for (int i = 0; i < n; i++)

    {

        if (degList[i] % 2 == 0)

            evenDeg += 1;

        else

            oddDeg += 1;

    }

    if (evenDeg == n)

        cout << "The graph is both an euler path and an euler circuit." << endl;

    if (oddDeg == 2)

        cout << "The graph is only an euler path not an euler circuit." << endl;

}

void calcDeg(int \*adjMatrix, int n, int degList[])

{

    for (int i = 0; i < n; i++)

    {

        int deg = 0;

        for (int j = 0; j < n; j++)

        {

            if (\*(adjMatrix + i \* n + j) == 1)

            {

                deg += 1;

                cout << i << j << endl;

            }

        }

        degList[i] = deg;

    }

}

int main()

{

    int n;

    cout << "Enter number of vertices:";

    cin >> n;

    int adjMatrix[n][n];

    int degList[n];

    for (int i = 0; i < n; i++) // initializing the whole matrix to 0

    {

        for (int j = 0; j < n; j++)

        {

            adjMatrix[i][j] = 0;

        }

        cout << endl;

    }

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

            cin >> adjMatrix[i][j];

        }

        cout << "A row of data has been entered." << endl;

    }

    cout << "\nAdjacency matrix for the data provided:" << endl;

    for (int i = 0; i < n; i++)

    {

        for (int j = 0; j < n; j++)

        {

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

        }

        cout << endl;

    }

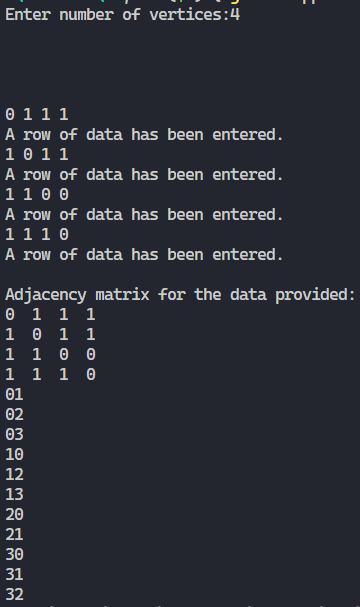
    calcDeg((int \*)adjMatrix, n, degList);

    check(degList, n);

    return 0;

}

**Output:**

****

**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 numOfLeafNodes(int m, int v)

{

    int i = (v - 1) / m;

    int l = (m - 1) \* i + 1;

    return l;

}

int main()

{

    int m, v;

    cout << " m is the maximum number of child nodes a parent node can have." << endl;

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

    cin >> m;

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

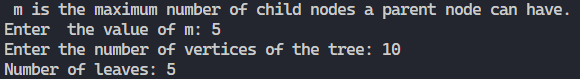
    cin >> v;

    cout << "Number of leaves: " << numOfLeafNodes(m, v);

    return 0;

}

**Output:**

****