**Discrete Mathematics**

**Assignments**

**Program 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.**

**Ans.**

#include<iostream>

#include<cmath>

using namespace std;

void removeDupli(int a[],int& n)

{

int k=0;

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

{

int count=0;

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

{

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

{

count++;

}

}

if(count==1)

{

a[k]=a[i];

k++;

}

}

n=k;

}

void printSet(int a[],int n)

{

cout<<"{";

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

{

cout<<a[i]<<" ";

}

cout<<"}"<<endl;

}

bool isMember(int a,int A[],int n)

{

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

{

if(a==A[i])

return true;

}

return false;

}

void powerSet(int A[],int n)

{

int binArr[100];

int p=pow(2,n);

cout<<"\n{";

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

{

cout<<"{";

int temp=i;

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

{

binArr[j]=temp%2;

temp/=2;

}

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

{

if(binArr[k]==1)

{

cout<<A[k]<<",";

}

}

if(i!=0)

cout<<"\b";

cout<<"}, ";

}

cout<<"\b\b}";

}

int main()

{

int A[20];

cout<<"\nEnter the number of elements to be entered:";

int n;

cin>>n;

cout<<"Enter the elements:";

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

{

cin>>A[i];

}

removeDupli(A,n);

printSet(A,n);

cout<<"Cardinality of the set is "<<n;

cout<<"\nSelect any one of the following:"

<<"\n1-->Search for the presence of an element."

<<"\n2-->Print the power set of the set"

<<"\nEnter your choice(1 or 2):";

int choice;

while(!(cin>>choice) || choice!=1 && choice!=2)

{

cout<<"\nInvalid Input. Please enter again:";

}

switch(choice)

{

case 1:

cout<<"\nEnter the element you want to search for in the set:";

int a;

cin>>a;

if(isMember(a,A,n))

cout<<"\nThe element exist in the set";

else

cout<<"\nThe element does not exist in the set";

break;

case 2:

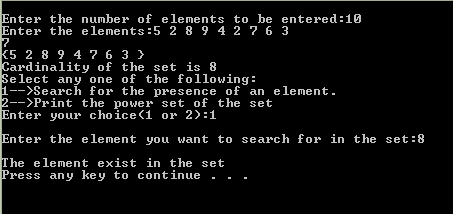
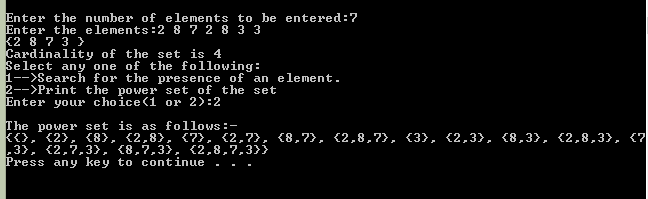
cout<<"\nThe power set is as follows:-";

powerSet(A,n);

}

return 0;

}

**Program 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.**

**Ans:**

#include<iostream>

using namespace std;

class Set

{

public:

int arr[100];

int n;

int getCardin()

{

return n;

}

int setCardin(int a)

{

n=a;

}

int getElement(int i)

{

return arr[i];

}

int setElement(int a,int n)

{

arr[n]=a;

}

void inputSet()

{

cout<<"\nEnter the number of elements:";cin>>n;

cout<<"\nEnter the elements:";

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

{

cin>>arr[i];

}

removeDupli();

}

bool isPresent(int a)

{

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

{

if(arr[i]==a)

{

return true;

}

}

return false;

}

void isSubset(Set s2)

{

int count=0;

for(int i=0;i<s2.getCardin();i++)

{

if(isPresent(s2.getElement(i)))

{

count++;

}

}

if(count==s2.getCardin())

{

cout<<"\nYes set 2 is a subset of set 1";

}

else{

cout<<"\nSet 2 is not a subset of set 1";

}

}

void removeDupli()

{

int k=0;

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

{

int count=0;

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

{

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

{

count++;

}

}

if(count==1)

{

arr[k]=arr[i];

k++;

}

}

n=k;

}

friend Set intersection(Set& s1,Set& s2);

friend Set union\_Set(Set& s1,Set& s2);

friend Set compliment(Set& univ,Set& s1);

friend Set difference(Set& s1,Set& s2);

friend void cardProd(Set& s1,Set& s2);

friend Set symDiff(Set& s1,Set& s2);

friend void printSet(Set s1);

};

void printSet(Set s1)

{

cout<<"{";

for(int i=0;i<s1.n;i++)

{

cout<<s1.arr[i]<<",";

}

cout<<"\b"<<"}";

}

Set union\_Set(Set& s1,Set& s2)

{

Set s3;

int j=0;

for(int i=0;i<s1.n;i++)

{

s3.arr[j]=s1.arr[i];

j++;

}

for(int k=0;k<s2.n;k++)

{

s3.arr[j]=s2.arr[k];

j++;

}

s3.n=j;

s3.removeDupli();

return s3;

}

Set intersection(Set& s1,Set& s2)

{

Set s3;

int j=0;

for(int i=0;i<s1.n;i++)

{

if(s2.isPresent(s1.arr[i]))

{

s3.arr[j]=s1.arr[i];

j++;

}

}

s3.n=j;

return s3;

}

Set compliment(Set& univ,Set& s1)

{

Set comp;

int j=0;

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

{

if(!(s1.isPresent(univ.arr[i])))

{

comp.arr[j]=univ.arr[i];

j++;

}

}

comp.n=j;

return comp;

}

Set difference(Set& s1,Set& s2)

{

Set s3;

int j=0;

for(int i=0;i<s1.n;i++)

{

if(!(s2.isPresent(s1.arr[i])))

{

s3.arr[j]=s1.arr[i];

j++;

}

}

s3.n=j;

return s3;

}

Set symDiff(Set& s1,Set& s2)

{

Set s3=difference(s1,s2);

Set s4=difference(s2,s1);

s3=union\_Set(s3,s4);

return s3;

}

void cartProd(Set& s1,Set& s2)

{

cout<<"{";

for(int i=0;i<s1.n;i++)

{

for(int j=0;j<s2.n;j++)

{

cout<<'('<<s1.arr[i]<<','<<s2.arr[j]<<"),";

}

}

cout<<"\b}";

}

int main()

{

Set s1,s2,s3;

cout<<"\nFor set 1";

s1.inputSet();

cout<<"\nFor set 2";

s2.inputSet();

int exit=0;

while(exit==0){

cout<<"\nSet 1:";

printSet(s1);

cout<<"\nSet 2:";

printSet(s2);

cout<<"\n1->Subset: Check whether one set is a subset of other or not."

<<"\n2->Union and Intersection of two Sets."

<<"\n3->Complement: Set 1 is universe,set 2 it's subset"

<<"\n4->Set Difference and Symmetric Difference between two SETS."

<<"\n5->Cartesian Product of Sets."

<<"\nEnter your choice:";

int choice;

cin>>choice;

switch(choice){

case 1:

s1.isSubset(s2);

break;

case 2:

s3=union\_Set(s1,s2);

cout<<"\nUnion of both sets:";

printSet(s3);

s3=intersection(s1,s2);

cout<<"\nIntersection of both sets:";

printSet(s3);

break;

case 3:

cout<<"\nCompliment of set 2:";

s3=compliment(s1,s2);

printSet(s3);

break;

case 4:

cout<<"\nSet difference(Set1 - Set2) =";

s3=difference(s1,s2);

printSet(s3);

cout<<"\nSymmetric difference((Set1 - Set2) U (Set2 - Set1)) =";

s3=symDiff(s1,s2);

printSet(s3);

break;

case 5:

cout<<"\nCartesian product is ";

cartProd(s1,s2);

}

char c;

cout<<"\nDo you want to continue(y/n):";

cin>>c;

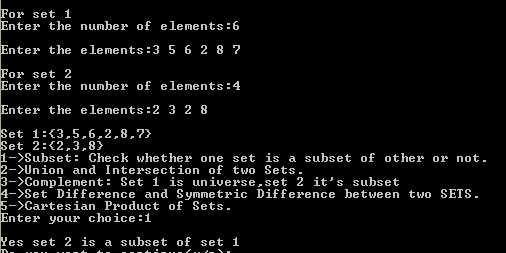
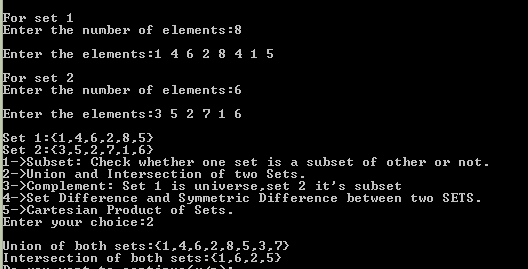
if(c=='n')

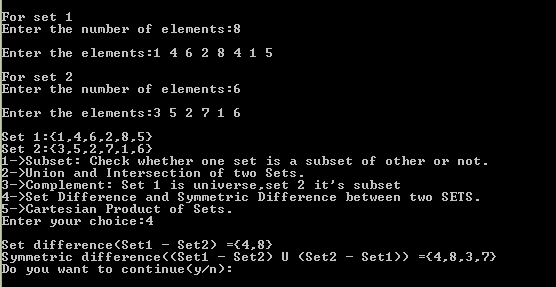
exit=1;

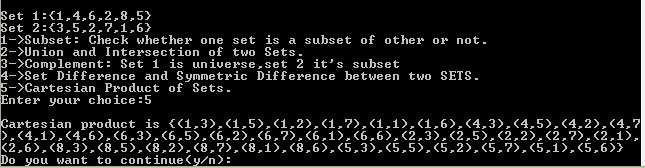
}

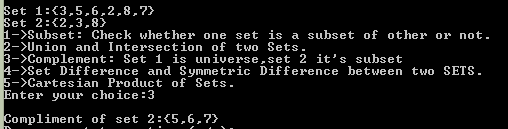
return 0;

}







**Program 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.**

**Program 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**

**Ans 3 and 4:**

#include<iostream>

#include<iomanip>

using namespace std;

class Relation

{

int n;

public:

int refFlag,symFlag,transFlag,antiSymFlag,ERFlag,PORFlag;

int relArr[10][10];

int set[10];

int getN()

{

return n;

}

int setN(int n)

{

this->n=n;

}

void inputArr(int arr[],int n)

{

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

{

cin>>arr[i];

}

}

void inputRel()

{

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

{

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

{

cin>>relArr[i][j];

}

}

}

void printRel()

{

cout<<setw(5);

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

{

cout<<set[i]<<setw(5);

}

cout<<"\n";

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

{

cout<<set[j]<<setw(4);

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

{

cout<<relArr[j][k]<<setw(5);

}

cout<<"\n";

}

}

int isReflexive()

{

int flag=0;

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

{

if(relArr[i][i]==1)

flag++;

}

if(flag==n)

return refFlag=1;

return refFlag=0;

}

int isSymmetric()

{

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

{

for(int j=0;(j<n && i>=j);j++)

{

if(relArr[i][j]!=relArr[j][i])

return symFlag=0;

}

}

return symFlag=1;

}

int isTransitive(){

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

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

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

if(relArr[i][j]==1 && relArr[j][k]==1 && relArr[i][k]==0){

return transFlag=0;

}

}

}

}

return transFlag=1;

}

int isEquivalence(){

if(refFlag && symFlag && transFlag){

return 1;

}

else

return 0;

}

int isAntisymmetric(){

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

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

if(relArr[i][j]==1 && relArr[j][i]==1 && i!=j){

return antiSymFlag=0;

}

}

}

return antiSymFlag=1;

}

int isPoset(){

if(refFlag && transFlag && antiSymFlag ){

return 1;

}

else{

return 0;

}

}

};

int main()

{

Relation r1;

int n;

cout<<"\t\t\t\* \* \* R E L A T I O N M A T R I X \* \* \*"

<<"\nEnter the number of elements of set on which the relation is applied:";

cin>>n;

r1.setN(n);

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

r1.inputArr(r1.set,n);

/\*r1.relArr={{1,1,1},

{0,1,1},

{0,0,1}}

\*/

cout<<"\nEnter the relation matrix:";

r1.inputRel();

r1.printRel();

if(r1.isReflexive())

cout<<"\nRelation is Reflexive";

else{cout<<"\nRelation is Non Reflexive";}

if(r1.isSymmetric())

cout<<"\nRelation is Symmetric";

else

cout<<"\nRelation is Non symmetric";

if(r1.isTransitive())

cout<<"\nRelation is Transitive";

else

cout<<"\nRelation is Non-Transitive";

//q4 from here

if(r1.isEquivalence()){

r1.ERFlag=1;

cout<<"\nThe relation is an Equivalence relation";

}

else{

r1.ERFlag=0;

cout<<"\nRelation is not an equivalence relation";

}

if(r1.isAntisymmetric()){

cout<<"\nThe relation is antisymmetric";

}

else{

cout<<"The relation is not antisymmetric";

}

if(r1.isPoset()){

r1.PORFlag=1;

cout<<"\nThe relation is a partially ordered relation";

}

else{

r1.PORFlag=0;

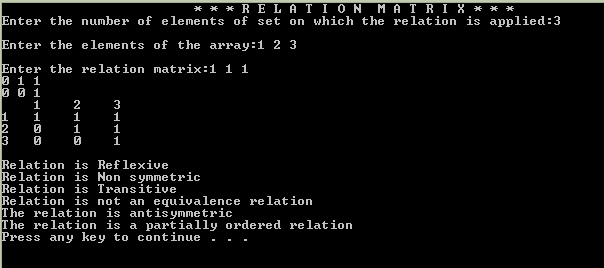
cout<<"\nThe relation is not a partially ordered relation ";

}

if(r1.PORFlag==0 && r1.ERFlag==0)

cout<<"\nThe relation is neither equivalence nor partially ordered relation.";

}



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

**Ans:**

#include<iostream>

using namespace std;

void bufferClr(){

cin.clear();

cin.ignore(100,'\n');

}

void fibonacci(int n,int a=0,int b=1)

{

if(n==0)

return;

cout<<a<<" ";

fibonacci(n-1,b,a+b);

}

int main()

{

int n;

cout<<"\t\t\t \* \* \* F I B O N A C C I \* \* \*"

<<"\nEnter the number of terms of fibonacci sequence to be printed:";

while(!(cin>>n) || n<=0)

{

bufferClr();

cout<<"\nInvalid Input.PLease enter the a number greater than 0:";

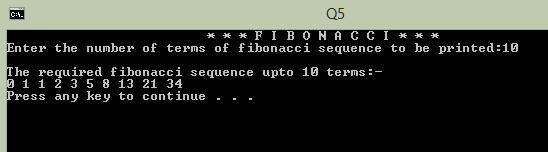
}

cout<<"\nThe required fibonacci sequence upto "<<n<<" terms:-\n";

fibonacci(n);

return 0;

}



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

**Ans:**

#include<iostream>

using namespace std;

void tow\_of\_Hanoi(int n,char from\_tower,char to\_tower,char aux\_tower)

{

if(n==1)

{

cout<<"\nMove disk 1 from tower "<<from\_tower<<" to tower "<<to\_tower;

return;

}

tow\_of\_Hanoi(n-1,from\_tower,aux\_tower,to\_tower);

cout<<"\nMove disk "<<n<<" from tower "<<from\_tower<<" to tower "<<to\_tower;

tow\_of\_Hanoi(n-1,aux\_tower,to\_tower,from\_tower);

}

int main()

{

cout<<"\t\t\t \*\*\*TOWER OF HANOI\*\*\*";

cout<<"\nEnter the number of rings in the primary tower:";

int n;

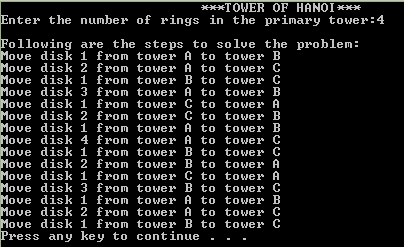
cin>>n;

cout<<"\nFollowing are the steps to solve the problem:";

tow\_of\_Hanoi(n,'A','C','B');

return 0;

}



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

**Ans:**

#include<iostream>

#include<iomanip>

using namespace std;

void drawLine()

{

cout<<"\n"<<setw(80)<<setfill('-')<<"-\n";

}

void sortArr(int array[],int length){

int i,j;

for(i=0;i<length;i++)

{

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

{

if(array[j]>array[j+1])

{

int temp=array[j];

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

array[j+1]=temp;

}

}

}

}

int binSearch(int a[],int begin,int end,int c)

{

if(begin<=end)

{

int mid=(begin+end)/2;

if(a[mid]==c)

{

cout<<"\nElement found at index "<<mid;

return 1;

}

if(a[mid]>c)

{

return binSearch(a,begin,mid-1,c);

}

if(a[mid]<c)

{

return binSearch(a,mid+1,end,c);

}

}

return 0;

}

int main()

{

int a[100];

int n;

drawLine();

cout<<"\t\t\t \* \* \*B I N A R Y S E A R C H\* \* \*";

drawLine();

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

cin>>n;

cout<<"\nEnter the elements:";

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

{

cin>>a[i];

}

sortArr(a,n);

cout<<"\nSorted array:\n";

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

{

cout<<a[i]<<" ";

}

int c;

cout<<"\nEnter the number to search for:";cin>>c;

int bin=binSearch(a,0,n,c);

if(bin==0)

{

cout<<"\nElement not found";

}

}



**Program 8 : 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.**

**Ans:**

#include<iostream>

#include<iomanip>

using namespace std;

void drawline()

{

cout<<"\n"<<setw(80)<<setfill('-')<<"-\n";

}

void bufClr()

{

cin.clear();

cin.ignore(100,'\n');

}

void printArr(int a[],int n)

{

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

{

cout<<a[i]<<" ";

}

}

void bubbleSort(int array[],int length)

{

int i,j;

for(i=0;i<length;i++)

{

int comp=0,flag=0;

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

{

comp++;

if(array[j]>array[j+1])

{

flag++;

int temp=array[j];

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

array[j+1]=temp;

}

}

drawline();

cout<<"\nPass "<<(i+1);

cout<<"\nComparisons:"<<comp<<"\n";

printArr(array,length);

if(flag==0)

break;

}

}

int main()

{

cout<<"\t\t\t \* \* \* B U B B L E S O R T \* \* \*";

cout<<"\nEnter the number of elements in the array:";

int len,arr[100];

while(!(cin>>len) || len<=0 || len>100)

{

bufClr();

cout<<"Invalid Input.Enter again:";

}

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

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

{

while(!(cin>>arr[i]))

{

bufClr();

cout<<"\nEnter a valid number:";

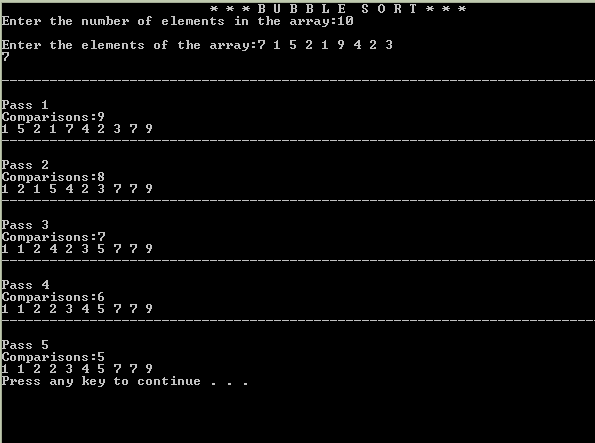
}

}

bubbleSort(arr,len);

return 0;

}



**Program 9 : 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.**

**Ans:**

#include<iostream>

#include<iomanip>

using namespace std;

void drawline()

{

cout<<"\n"<<setw(80)<<setfill('-')<<"-\n";

}

void bufClr()

{

cin.clear();

cin.ignore(100,'\n');

}

void printArr(int a[],int n)

{

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

{

cout<<a[i]<<" ";

}

}

void insertionSort(int arr[],int length)

{

int i,key,j;

for(i=1;i<length;i++)

{

int comp = 0;

key=arr[i];

j=i-1;

comp+=2;

while(j>=0 && arr[j]>key)

{

arr[j+1]=arr[j];

j=j-1;

comp+=2;

}

arr[j+1]=key;

drawline();

cout<<"\nPass "<<(i);

cout<<"\nComparisons:"<<comp<<"\n";

printArr(arr,length);

}

}

int main()

{

cout<<"\t\t\t \* \* \* I N S E R T I O N S O R T \* \* \*";

cout<<"\nEnter the number of elements in the array:";

int len,arr[100];

while(!(cin>>len) || len<=0 || len>100)

{

bufClr();

cout<<"Invalid Input.Enter again:";

}

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

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

{

while(!(cin>>arr[i]))

{

bufClr();

cout<<"\nEnter a valid number:";

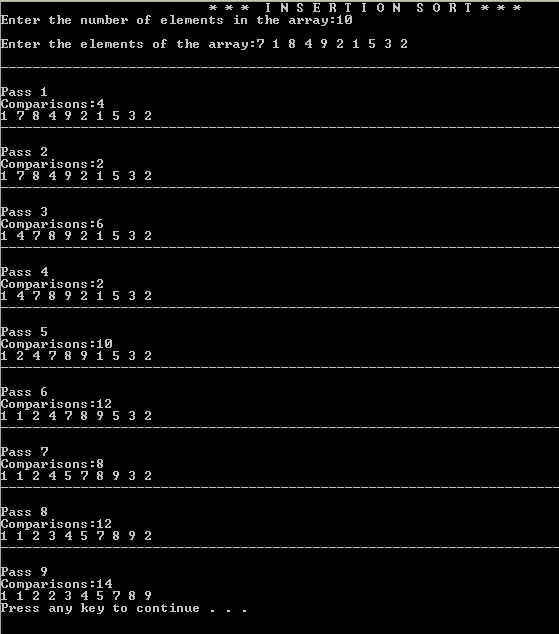
}

}

insertionSort(arr,len);

return 0;

}



**Program 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).**

**Ans:**

#include<iostream>

using namespace std;

void swap(int& i,int& j){

int t=i;

i=j;

j=t;

}

void printArr(int a[],int n)

{

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

cout<<a[i]<<" ";

}

void permute(int arr[],int start,int end){

if(start==end){

cout<<"\n";

printArr(arr,end);

}

else{

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

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

permute(arr,start+1,end);

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

}

}

}

void inputArr(int a[],int n){

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

cin>>a[i];

}

}

int main()

{

cout<<"\t\t\t \* \* \* P E R M U T A I O N S \* \* \*";

int arr[20];

cout<<"\nEnter the number of elements in the array:";

int n;

cin>>n;

cout<<"\nEnter the elements:";

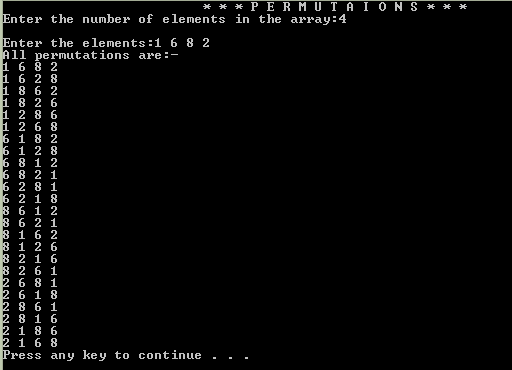
inputArr(arr,n);

cout<<"All permutations are:-";

permute(arr,0,n);

return 0;

}



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

**Ans:**

#include<iostream>

using namespace std;

void bufferClr(){

cin.clear();

cin.ignore(100,'\n');

}

long nCr(long n,long r){

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

return 1;

}

else

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

}

long nPr(long n,long r){

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

return 1;

else{

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

}

}

int main(){

cout<<"\t\*\*\* Computing permutations and combinations using recursion \*\*\*";

cout<<"\nInput n:";

int n,r;

while(!(cin>>n)){

bufferClr();

cout<<"\nInvalid Input.Enter again:";

}

cout<<"\nInput r:";

while(!(cin>>r) || r>n){

bufferClr();

cout<<"\nInvalid Input.Enter again:";

}

cout<<"\nEnter your choice: "

<<"\n\t1->Permutation\n\t2->Combination\nChoice:";

int choice;

while(!(cin>>choice) || (choice!=1 && choice!=2)){

bufferClr();

cout<<"\nInvalid input.Enter again:";

}

switch(choice){

case 1:{

cout<<"nPr="<<nPr(n,r);

break;

}

case 2:{

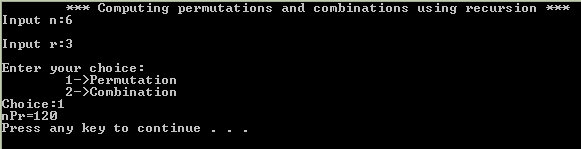
cout<<"nCr="<<nCr(n,r);

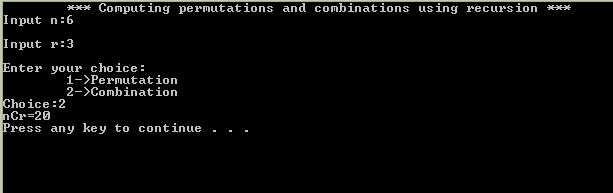
break;

}

}

}





**Program 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.**

**Program 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**

**Ans:**

#include<iostream>

using namespace std;

int conjunction(int x , int y){

return x & y;

}

int disjunction(int x,int y){

return (x | y);

}

int ex\_OR(int x,int y){

return (x & !y) | (!x & y);

}

int conditional(int x,int y){

return !x | y;

}

int biconditional(int x, int y){

return (!x & !y) | (x & y);

}

int ex\_NOR(int x,int y){

return (!x & !y) | (x & y);

}

int negation(int x){

return !x;

}

int nandOp(int x, int y){

return !(x & y);

}

int norOp(int x,int y){

return !(x | y);

}

int main(){

cout<<"\t\t\t\*\*\*BOOLEAN OPERATORS\*\*\*";

int x,y,out,choice;

cout<<"\nEnter X:";

cin>>x;

cout<<"\nEnter Y:";

cin>>y;

while(1){

cout<<"WHAT DO YOU WANT TO PERFORM?"

<<"\n1) Conjunction"

<<"\n2) Disjunction"

<<"\n3) Exclusive OR"

<<"\n4) Conditional"

<<"\n5) Bi-conditional"

<<"\n6) Exclusive NOR"

<<"\n7) Negation"

<<"\n8) NAND"

<<"\n9) NOR"

<<"\nEnter your choice:";

cin>>choice;

cout<<"\n\nTRUTH TABLE\n";

switch(choice){

case 1:

cout<<"\nZ=X^Y\n";

cout<<"\nX\tY\tZ";

cout<<"\n"<<x<<"\t"<<y<<"\t"<<conjunction(x,y);

break;

case 2:

cout<<"\nZ=X v Y\n";

cout<<"\nX\tY\tZ";

cout<<"\n"<<x<<"\t"<<y<<"\t"<<disjunction(x,y);

break;

case 3:

cout<<"\nZ=X XOR Y\n";

cout<<"\nX\tY\tZ";

cout<<"\n"<<x<<"\t"<<y<<"\t"<<ex\_OR(x,y);

break;

case 4:

cout<<"\nZ=X -> Y\n";

cout<<"\nX\tY\tZ";

cout<<"\n"<<x<<"\t"<<y<<"\t"<<conditional(x,y);

break;

case 5:

cout<<"\nZ=X <-> Y\n";

cout<<"\nX\tY\tZ";

cout<<"\n"<<x<<"\t"<<y<<"\t"<<biconditional(x,y);

break;

case 6:

cout<<"\nZ=~(X XOR Y)\n";

cout<<"\nX\tY\tZ";

cout<<"\n"<<x<<"\t"<<y<<"\t"<<ex\_NOR(x,y);

break;

case 7:

cout<<"\nZ= ~X\n";

cout<<"\nX\tZ";

cout<<"\n"<<x<<"\t"<<negation(x);

break;

case 8:

cout<<"\nZ=~(X ^ Y)\n";

cout<<"\nX\tY\tZ";

cout<<"\n"<<x<<"\t"<<y<<"\t"<<nandOp(x,y);

break;

case 9:

cout<<"\nZ=~(X v Y)\n";

cout<<"\nX\tY\tZ";

cout<<"\n"<<x<<"\t"<<y<<"\t"<<norOp(x,y);

break;

}

int c;

cout<<"\nDo you want to continue(0 or 1):";

cin>>c;

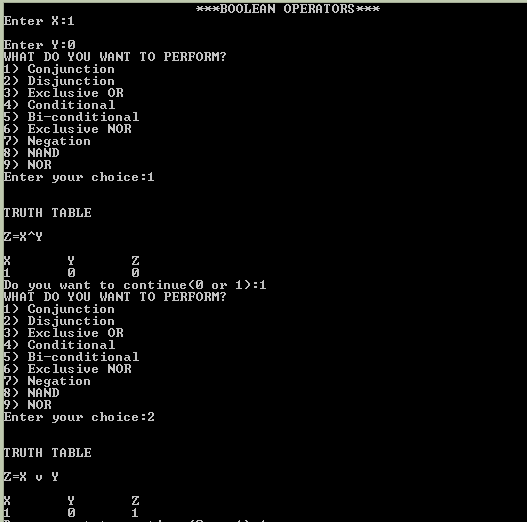
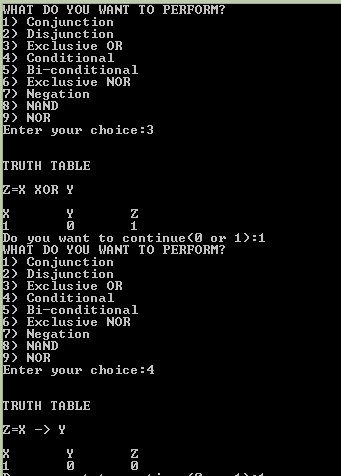
if(c==0){

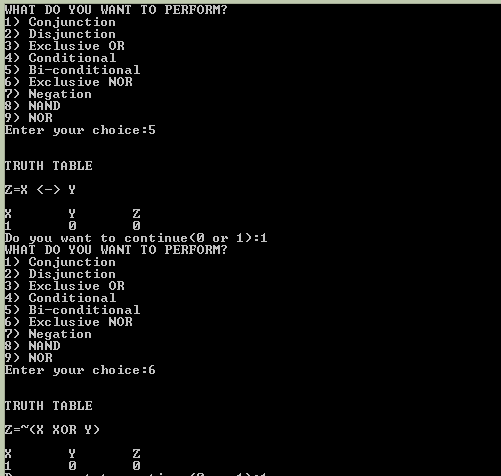
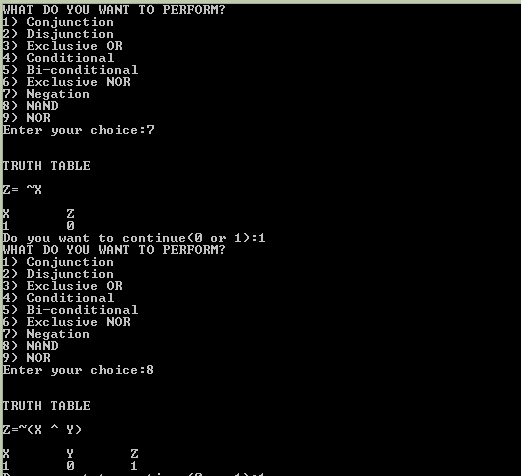
break;

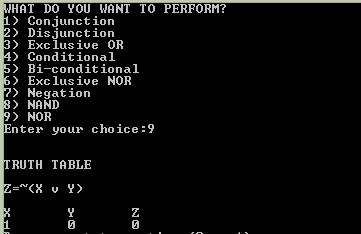
}

}

}



**Program 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.**

**Ans:**

#include<iostream>

using namespace std;

void bufferClr(){

cin.clear();

cin.ignore(100,'\n');

}

int recFunc1(int n){

if(n==0)

return 1;

return recFunc1(n-1)+n;

}

int recFunc2(int n){

if(n==0)

return 1;

return recFunc2(n-1)+n\*n;

}

int recFunc3(int n){

if(n==1){

return 1;

}

return 2\*recFunc3(n/2)+n;

}

int main(){

cout<<"\t\t\t \* \* \* R E C U R S I V E F U N C T I O N S \* \* \*"

<<"\nChoose from the following relations:"

<<"\n\t1-> T(n) = T(n-1) + n"

<<"\n\t2-> T(n) = T(n-1) + n^2"

<<"\n\t3-> T(n) = 2 \* T(n/2) + n"

<<"\nEnter your choice(1,2 or 3):";

int choice;

while(!(cin>>choice) || choice!=1 && choice!=2 && choice!=3){

bufferClr();

cout<<"\nInvalid Input.Enter again:";

}

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

int n;

while(!(cin>>n) || n<0){

bufferClr();

cout<<"\nInvalid Input.Enter again:";

}

switch(choice){

case 1:

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

cout<<"\nValue of [T(n) = T(n-1) + n] at n="<<i<<" = "<<recFunc1(i);

}

break;

case 2:

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

cout<<"\nValue of [T(n) = T(n-1) + n^2] at n="<<i<<" = "<<recFunc2(i);

}

break;

case 3:

for(int i=1;i<=n;i++){

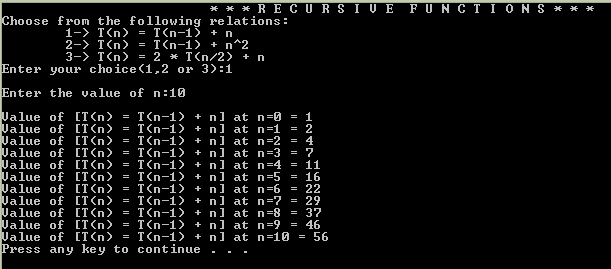
cout<<"\nValue of [T(n) = 2 \* T(n/2) + n] at n="<<i<<" = "<<recFunc3(i);

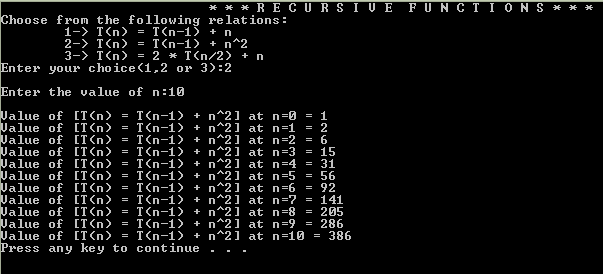
}

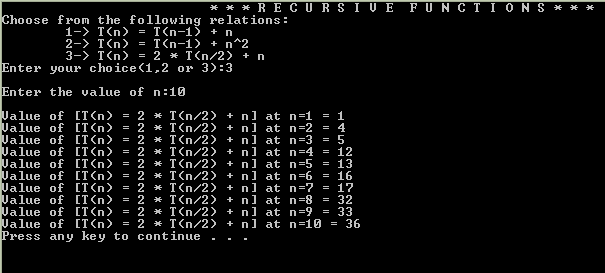
break;

}

}







**Program 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)).**

**Ans:**

#include<iostream>

#include<math.h>

using namespace std;

void bufferClr(){

cin.clear();

cin.ignore(100,'\n');

}

class Polynomial{

int deg;

int coeff[11];

int varTerm[11];

public:

Polynomial(int degree){

deg=degree;

}

void inputCoeff(){

for(int i=0;i<=deg;i++){

while(!(cin>>coeff[i])){

bufferClr();

cout<<"\nInvalid Input.Enter again:";

}

}

}

void calcVarTerms(int x){

for(int i=0;i<=deg;i++){

varTerm[i]=pow(x,deg-i);

}

}

void printPol(){

cout<<coeff[0]<<"x^"<<deg;

for(int i=1;i<=deg;i++){

if(coeff[i]!=0){

if(coeff[i]>0){

cout<<"+";

}

cout<<coeff[i];

if(i!=deg){

cout<<"x";

if(i!=deg-1){

cout<<"^"<<deg-i;

}

}

}

}

}

int calcValAt(int a){

this->calcVarTerms(a);

int sum=0;

for(int i=0;i<=deg;i++){

sum+=coeff[i]\*varTerm[i];

}

return sum;

}

};

int main(){

int deg;

cout<<"\t\t\t \* \* \* FUNCTION \* \* \*";

cout<<"\nEnter the degree of the polynomial[max:10]:";

while(!(cin>>deg) || deg<=0 || deg>10){

bufferClr();

cout<<"\nInvalid input.Enter again:";

}

Polynomial Px=Polynomial(deg);

cout<<"\nEnter the coefficients of the polynomial:";

Px.inputCoeff();

cout<<"\n\t\tf(X)=";

Px.printPol();

cout<<"\nEnter the value where the function is to be evaluated:";

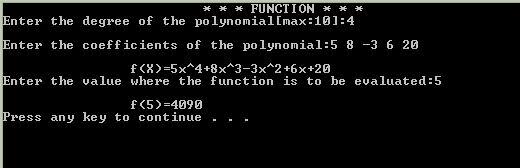
int x;

cin>>x;

cout<<"\n\t\tf("<<x<<")="<<Px.calcValAt(x);

return 0;

}



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

**Ans:**

#include<iostream>

#include<iomanip>

using namespace std;

void bufferClr(){

cin.clear();

cin.ignore(100,'\n');

}

class Graph{

int vertices;

int adjacencyMatrix[10][10];

public:

void inputAdjMat(int n){

vertices=n;

for(int i=0;i<vertices;i++){

for(int j=0;j<vertices;j++){

int element;

while(!(cin>>element)){

bufferClr();

cout<<"Invalid input.Enter again:";

}

adjacencyMatrix[i][j]=element;

}

}

}

void printAdjMat()

{

cout<<setw(5);

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

{

cout<<"v"<<i<<setw(5);

}

cout<<"\n";

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

{

cout<<"v"<<j<<setw(4);

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

{

cout<<adjacencyMatrix[j][k]<<setw(5);

}

cout<<"\n";

}

}

bool checkComplete(){

for(int i=0;i<vertices;i++){

if(adjacencyMatrix[i][i]==1){

cout<<"\nIt is not a simple graph.";

return false;

}

for(int j=0;j<vertices;j++){

if(i!=j && (adjacencyMatrix[i][j]==0)){

return false;

}

}

}

return true;

}

};

int main(){

cout<<"\t\t\t \* \* \* GRAPHS USING ADJACENCY MATRIX \* \* \*"

<<"\nEnter the number of vertices[max:10]:";

int n;

while(!(cin>>n) || n<=0){

bufferClr();

cout<<"\nInvalid Input.Enter again:";

}

cout<<"\nEnter the elements of the adjacency matrix:-\n";

Graph G1;

G1.inputAdjMat(n);

cout<<"\nThe adjacency matrix is:\n";

G1.printAdjMat();

if(G1.checkComplete()){

cout<<"\n\nThe graph is a complete graph";

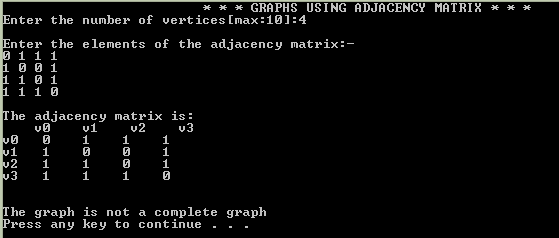
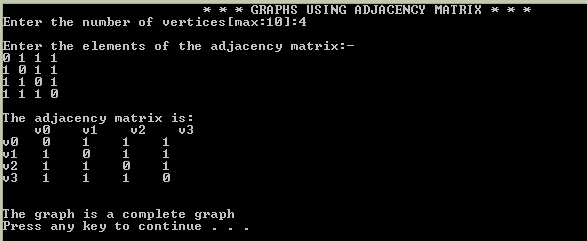
}

else{

cout<<"\n\nThe graph is not a complete graph";

}

}

**Program 17 : Write a Program to accept a directed graph G and compute the in-degree and out-degree of each vertex.**

**Ans:**

#include<iostream>

#include<iomanip>

using namespace std;

void bufferClr(){

cin.clear();

cin.ignore(100,'\n');

}

class DirectedGraph{

int vertices;

int adjMatrix[10][10];

public:

void inputAdjMat(int n){

vertices=n;

for(int i=0;i<vertices;i++){

for(int j=0;j<vertices;j++){

int element;

while(!(cin>>element)){

bufferClr();

cout<<"Invalid input.Enter again:";

}

adjMatrix[i][j]=element;

}

}

}

void printAdjMat()

{

cout<<setw(5);

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

{

cout<<"v"<<i<<setw(5);

}

cout<<"\n";

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

{

cout<<"v"<<j<<setw(4);

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

{

cout<<adjMatrix[j][k]<<setw(5);

}

cout<<"\n";

}

}

int calcIndegree(int v){

int inDegree=0;

for(int i=0;i<vertices;i++){

inDegree+=adjMatrix[i][v];

}

return inDegree;

}

int calcOutdegree(int v){

int outDegree=0;

for(int i=0;i<vertices;i++){

outDegree+=adjMatrix[v][i];

}

return outDegree;

}

};

int main(){

cout<<"\t\t \* \* \* DIRECTED GRAPHS USING ADJACENCY MATRIX \* \* \*"

<<"\n\nEnter the number of vertices[max:10]:";

int n;

while(!(cin>>n) || n<=0){

bufferClr();

cout<<"\nInvalid Input.Enter again:";

}

cout<<"\nEnter the elements of the adjacency matrix:-\n";

DirectedGraph G1;

G1.inputAdjMat(n);

cout<<"\nThe adjacency matrix for directed graph is:\n";

G1.printAdjMat();

cout<<"\n\nIn degrees:\n";

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

cout<<"\nv"<<i<<" = "<<G1.calcIndegree(i);

}

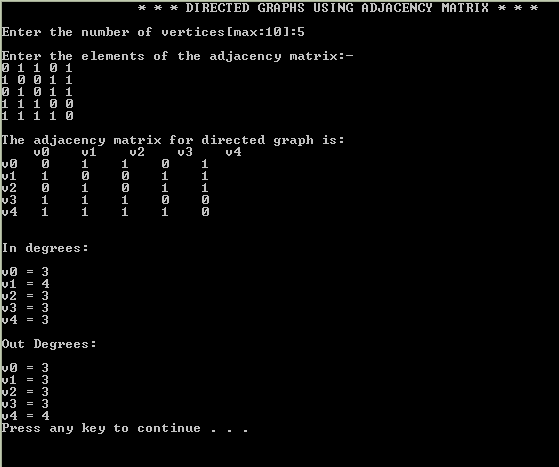
cout<<"\n\nOut Degrees:\n";

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

cout<<"\nv"<<j<<" = "<<G1.calcOutdegree(j);

}

}



**Program 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.**

**Ans:**

#include<iostream>

#include<iomanip>

using namespace std;

void bufferClr(){

cin.clear();

cin.ignore(100,'\n');

}

class PathFinderClass{

int A[10][10],A2[10][10],A0[10][10];

int n;

public:

void inputAdjMatrix(int num){

n=num;

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

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

cin>>A[i][j];

A2[i][j]=A[i][j];

}

}

}

void copyMatrix(){

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

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

A2[i][j]=A0[i][j];

}

}

}

void mulMatrix(){

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

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

int sum=0;

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

sum+=A[i][k]\*A2[k][j];

}

A0[i][j]=sum;

}

}

copyMatrix();

}

void powerMatrix(int pow){

if(pow==1)

return;

mulMatrix();

powerMatrix(pow-1);

}

int getNumberPaths(int v1,int v2,int len){

powerMatrix(len);

return A2[v1-1][v2-1];

}

void printAdjMat()

{

cout<<" ";

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

{

cout<<"\tv"<<i;

}

cout<<"\n";

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

{

cout<<"v"<<j;

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

{

cout<<'\t'<<setw(2)<<setfill(' ')<<A[j][k];

}

cout<<"\n";

}

}

void printPow(){

cout<<" ";

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

{

cout<<"\tv"<<i;

}

cout<<"\n";

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

{

cout<<"v"<<j;

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

{

cout<<'\t'<<setw(2)<<setfill(' ')<<A2[j][k];

}

cout<<"\n";

}

}

};

int main(){

PathFinderClass g1;

int n,len;

cout<<"\t\t\t \* \* \* PATH LENGTH FINDER \* \* \*"

<<"\n\nEnter the number of vertices in the graph:";

cin>>n;

cout<<"\nEnter the elements of the adjacency Matrix:\n";

g1.inputAdjMatrix(n);

cout<<"\nThe adjacency matrix is as follows:\n";

g1.printAdjMat();

cout<<"Enter the initial vertex and the final vertex number:";

int v1,v2;

cin>>v1>>v2;

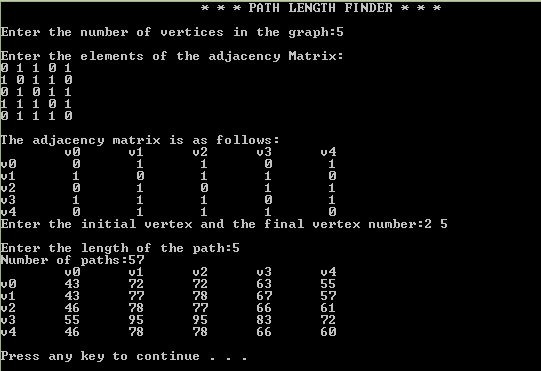
cout<<"\nEnter the length of the path:";

cin>>len;

cout<<"Number of paths:"<<g1.getNumberPaths(v1,v2,len)<<"\n";

g1.printPow();

}



**Program 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).**

**Ans:**

#include<iostream>

#include<iomanip>

using namespace std;

void bufferClr(){

cin.clear();

cin.ignore(100,'\n');

}

class EulerFinder{

int adjMatrix[10][10];

int pathArr[20];

int edges;

int n;

public:

void inputMat(int n){

this->n=n;

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

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

int element;

while(!(cin>>element)){

bufferClr();

cout<<"Invalid input.Enter again:";

}

adjMatrix[i][j]=element;

}

}

}

bool checkEdge(int v1,int v2){

if(adjMatrix[v1-1][v2-1]>=1){

adjMatrix[v1-1][v2-1]-=1;

adjMatrix[v2-1][v1-1]-=1;

return true;

}

else{

return false;

}

}

void calcNumEdge(){

this->edges=0;

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

for(int j=0;j>n;j++){

this->edges+=adjMatrix[i][j];

}

}

}

void eulerTest(){

int degrees[10];

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

degrees[i]=0;

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

if(i==j){

degrees[i]+=2\*adjMatrix[i][j];

}

else{

degrees[i]+=adjMatrix[i][j];

}

}

}

int oddCount=0;

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

cout<<"\nv"<<k<<"="<<degrees[k];

if(degrees[k]%2==1){

oddCount++;

}

}

if(oddCount==2){

cout<<"\nEuler path exists.";

}

else if(oddCount==0){

cout<<"\nEuler circuit exists.";

}

else{

cout<<"\nNeither Euler path exist nor Euler circuit exists.";

}

}

void printAdjMat()

{

cout<<" ";

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

{

cout<<"\tv"<<i;

}

cout<<"\n";

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

{

cout<<"v"<<j;

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

{

cout<<'\t'<<setw(2)<<setfill(' ')<<adjMatrix[j][k];

}

cout<<"\n";

}

}

};

int main(){

int n;

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

cin>>n;

EulerFinder g1;

cout<<"\nEnter the elements of adjacency matrix:";

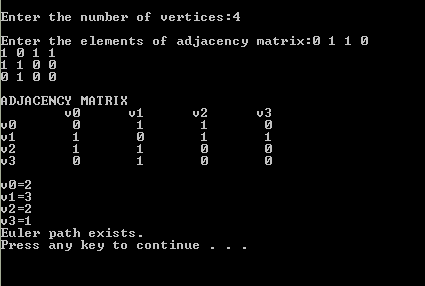
g1.inputMat(n);

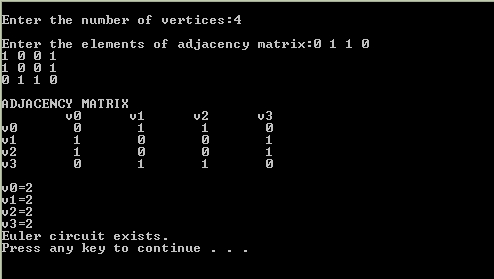
cout<<"\nADJACENCY MATRIX\n";

g1.printAdjMat();

g1.eulerTest();

}





**Program 20 : Given a full m-ary tree with i internal vertices, write a program to find the number of leaf nodes.**

**Ans:**

#include<iostream>

#include<math.h>

using namespace std;

int main(){

int m,i,l;

cout<<"\nEnter the number of childs for each node of m-ary tree:";

cin>>m;

cout<<"\nEnter the number of internal nodes:";

cin>>i;

l=i\*(m-1)+1;

cout<<"\nNumber of leaves in the tree is "<<l;

}

