1. **Program in C to implement basic set operations: UNION, INTERSECTION and DIFFERENCE**

#include <stdio.h>

int main()

{

int i,j,k,p,ch,n1,n2,set1[10],set2[10],set3[20],flag;

printf("Enter size of set1:");

scanf("%d",&n1);

printf("\nEnter elements of set 1:");

for(i=0;i<n1;i++)

{

scanf("%d",&set1[i]);

}

printf("Enter size of set2:");

scanf("%d",&n2);

printf("\nEnter elements of set 2:");

for(i=0;i<n2;i++)

{

scanf("%d",&set2[i]);

}

while(1)

{

printf("\nPress 1 for union:");

printf("\nPress 2 for intersection:");

printf("\nPress 3 for difference:");

printf("\nEnter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nThe union is:\n");

k=0;

for(i=0;i<n1;i++)

{

set3[k]=set1[i];

k++;

}

for(i=0;i<n2;i++)

{

flag=1;

for(j=0;j<n1;j++)

{

if(set2[i]==set1[j])

{

flag=0;

break;

}

}

if(flag==1)

{

set3[k]=set2[i];

k++;

}

}

p=k;

for(k=0;k<p;k++)

{

printf("%d\t",set3[k]);

}

break;

case 2:

printf("\nThe intersection is:\n");

k=0;

for(i=0;i<n2;i++)

{

flag=1;

for(j=0;j<n1;j++)

{

if(set2[i]==set1[j])

{

flag=0;

break;

}

}

if(flag==0)

{

set3[k]=set2[i];

k++;

}

}

p=k;

for(k=0;k<p;k++)

{

printf("%d\t",set3[k]);

}

break;

case 3:

printf("\nThe difference is:\n");

k=0;

for(i=0;i<n1;i++)

{

flag=1;

for(j=0;j<n2;j++)

{

if(set1[i]==set2[j])

{

flag=0;

break;

}

}

if(flag==1)

{

set3[k]=set1[i];

k++;

}

}

p=k;

for(k=0;k<p;k++)

{

printf("%d\t",set3[k]);

}

break;

default:

printf("Invalid choice:");

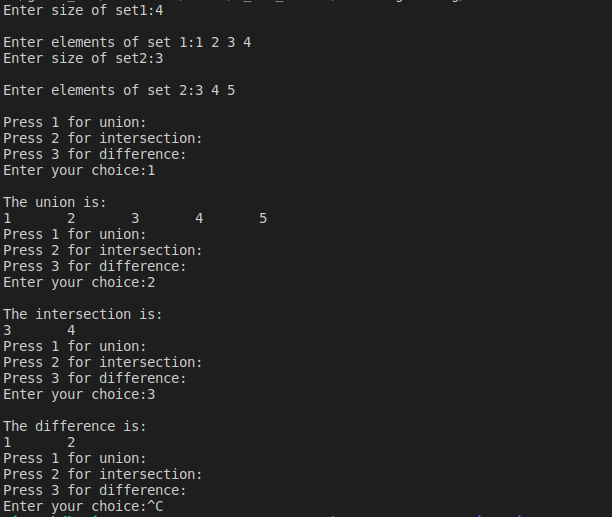
}

}

return 0;

}

**OUTPUT:**

****

1. **Program in C to find the Cartesian product of two sets.**

#include<stdio.h>

#include<conio.h>

void main()

{

int a[10],b[10],m,n,i,j;

clrscr();

printf("How many elements do you want in set a?\n");

scanf("%d",&m);

printf("Enter %d elements in set a:",m);

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

{

scanf("%d",&a[i]);

}

printf("How many elements do you want in set b?\n");

scanf("%d",&n);

printf("Enter %d elements in set b:",n);

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

{

scanf("%d",&b[j]);

}

printf("\nCartesian Product:");

printf("{");

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

{

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

{

printf("(%d,%d)",a[i],b[j]);

printf(",");

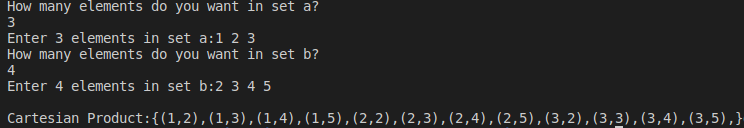
}}

printf("}");

getch();

}

**Output:**



3. **C program to find ceiling and floor value**

#include<stdio.h>

#include<math.h>

int main()

{

float val;

float fval,cval;

printf("Enter a float value:");

scanf("%f",&val);

fval=floor(val);

cval=ceil(val);

printf("Floor value =%f \n Ceiling value =%f",fval,cval);

return 0;

}

**Output:**

Z:\home\giver_kdk\Pictures\ds3.png

1. **Write a C program to find to implement fuzzy set operations**

#include<stdio.h>

#include<stdlib.h>

float min(float a,float b);

float max(float a,float b);

int main()

{

float fa,fb,fi,fu,fac;

float x = 1.0;

printf("Enter membership function of first set:\n");

scanf("%f",&fa);

printf("Enter membership function of second set:\n");

scanf("%f",&fb);

fi = min(fa,fb);

fu = max(fa,fb);

fac = x-fa;

printf("The membership function of intersection = %0.1f\n",fi);

printf("The membership function of union = %0.1f\n",fu);

printf("The membership function of complement of first set = %0.1f\n",fac);

return 0;

}

float min(float a, float b)

{

if(a<b)

return a;

else

return b;

}

float max(float a, float b)

{

if(a>b)

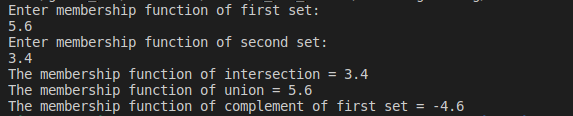
return a;

else

return b;

}

**Output:**



1. **C program to implement Boolean matrix operation join.**

#include<stdio.h>

int main()

{

int m,n,p,q,i,j,k;

int first[5][5],second[5][5],join[5][5];

printf("Enter the number of rows and columns of first matrix:\n");

scanf("%d%d",&m,&n);

printf("Enter the elements of first matrix:\n");

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

{

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

{

scanf("%d",&first[i][j]);

}

}

printf("Enter the number of rows and columns of second matrix:\n");

scanf("%d%d",&p,&q);

printf("Enter the elements of second matrix:\n");

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

{

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

{

scanf("%d",&second[i][j]);

}

}

printf("The elements of first matrix:\n");

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

{

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

{

printf("%d\t",first[i][j]);

}

printf("\n");

}

printf("The elements of second matrix:\n");

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

{

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

{

printf("%d\t",second[i][j]);

}

printf("\n");

}

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

{

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

{

join[i][j]=first[i][j]||second[i][j];

}

}

printf("Boolean join of the martices:\n");

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

{

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

{

printf("%d\t",join[i][j]);

}

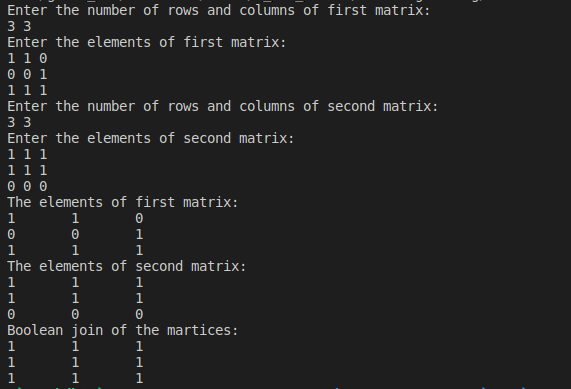
printf("\n");

}

return 0;

}

**Output:**



1. **C-Program to implement Boolean meet.**

#include<stdio.h>

int main()

{

int m,n,p,q,i,j,k;

int first[5][5],second[5][5],meet[5][5];

printf("Enter the number of rows and columns of first matrix:\n");

scanf("%d%d",&m,&n);

printf("Enter the elements of first matrix:\n");

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

{

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

{

scanf("%d",&first[i][j]);

}

}

printf("Enter the number of rows and columns of second matrix:\n");

scanf("%d%d",&p,&q);

printf("Enter the elements of second matrix:\n");

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

{

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

{

scanf("%d",&second[i][j]);

}

}

printf("The elements of first matrix:\n");

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

{

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

{

printf("%d\t",first[i][j]);

}

printf("\n");

}

printf("The elements of second matrix:\n");

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

{

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

{

printf("%d\t",second[i][j]);

}

printf("\n");

}

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

{

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

{

meet[i][j]=first[i][j]&&second[i][j];

}

}

printf("Boolean meet of the martices:\n");

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

{

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

{

printf("%d\t",meet[i][j]);

}

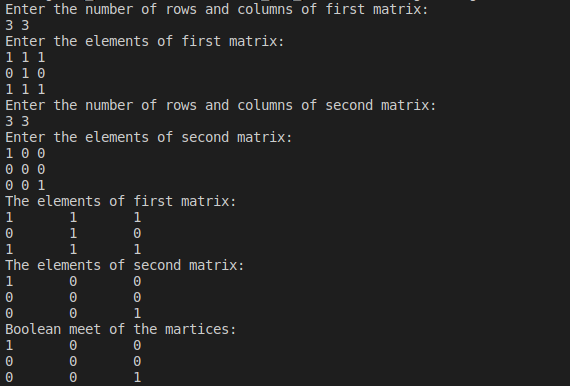
printf("\n");

}

return 0;

}

**Output:**



1. **C-Program to implement Boolean product.**

#include<stdio.h>

int main ()

{

int m,n,p,q,i,j,k,sum=0;

int first[5][5],second[5][5],multiply[5][5];

printf("Enter the number of rows and columns of first matrix:\n");

scanf("%d%d",&m,&n);

printf("\nEnter the elements of first matrix:\n");

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

{

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

scanf("%d",&first[i][j]);

}

printf("Enter the number of rows and columns of second matrix:\n");

scanf("%d%d",&p,&q);

if(n!=p)

printf("The matrices cannot be multiplied with each other.\n");

else

{

printf("\nEnter the elements of second matrix:\n");

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

{

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

scanf("%d",&second[i][j]);

}

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

{

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

{

for(k=0;k<p;k++)

{

sum=sum|| first[i][k]&&second[k][j];

}

multiply[i][j]=sum;

sum=0;

}

}

printf("Boolean Product of Matrices is:\n");

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

{

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

{

printf("%d\t",multiply[i][j]);

}

printf("\n");

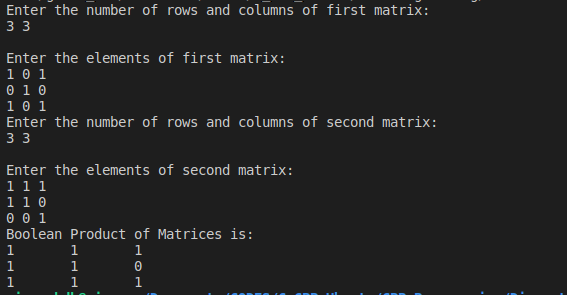
}

}

return 0;

}

**Output:**



1. **C program to implement Euclidian algorithms**

#include<stdio.h>

int gcd(int a,int b);

int main()

{

int a,b,g;

printf("Enter first number\n");

scanf("%d",&a);

printf("Enter second number\n");

scanf("%d",&b);

g =gcd(a,b);

printf("The gcd of %d and %d = %d\n",a,b,g);

getch();

return 0;

}

int gcd(int a,int b)

{

if(a==0)

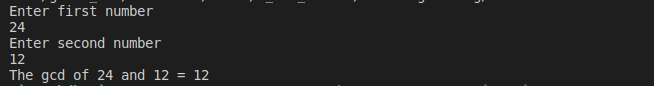
return b;

else

return gcd(b%a,a);

}

**Output:**



1. **C program to find factorial of number using recursion.**

#include <stdio.h>

int do\_factorial(int x);

int main()

{

int num, factorial;

printf("Enter the number: ");

scanf("%d", &num);

factorial = do\_factorial(num);

printf("Factorial of the number is %d", factorial);

}

int do\_factorial(int num)

{

if (num == 0) // Acts as condition to terminate recursion

{

return (1);

}

else

{

// Function repeatedly calls itself inside 'itself' causing looping effect.

return (num \* do\_factorial(num - 1));

}

}

**Output:**

Z:\home\giver_kdk\Pictures\ds9.png

1. **C program to x^y using recursion.**

#include <stdio.h>

int power\_fx(int num, int pow);

int main()

{

int number, power, result;

printf("Enter the value of x and y: ");

scanf("%d%d", &number, &power);

result= power\_fx(number, power);

printf("The %d to the power %d is %d", number, power, result);

}

int power\_fx(int num, int pow)

{

if(pow==0)

{

return(1);

}

else

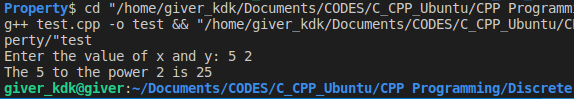
{

return(num\* power\_fx(num, pow-1)); //Direct recursion causing 'num' to multiply itself.

}

}

**Output:**



1. **C program to print truth table of AND Operation**

#include<stdio.h>

int AND(int a,int b);

int main()

{

int a,b;

printf("A\tB\tA^B\n");

for(a=0;a<=1;a++)

{

for(b=0;b<=1;b++)

{

printf("%d\t%d\t%d\n",a,b,AND(a,b));

}

}

return 0;

}

int AND(int a,int b)

{

if(a==1&&b==1)

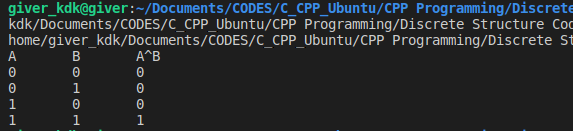
return 1;

else

return 0;

}

**Output:**



1. **C program to print truth table of OR Operation**

#include<stdio.h>

int OR(int a,int b);

int main()

{

int a,b;

printf("A\tB\tA V B\n\n");

for(a=0;a<=1;a++)

{

for(b=0;b<=1;b++)

{

printf("%d\t%d\t%d\n\n",a,b,OR(a,b));

}

}

return 0;

}

int OR(int a,int b)

{

if(a==1||b==1)

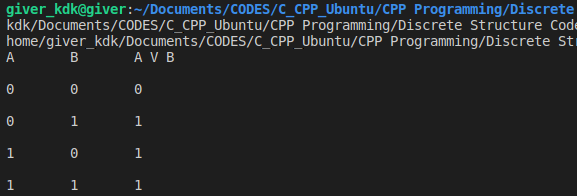
return 1;

else

return 0;

}

**Output:**



1. **C program to print truth table of NOT Operation**

#include<stdio.h>

int NOT(int a);

int main()

{

int a,b;

printf("A\t~A\n");

for(a=0;a<=1;a++)

{

printf("%d\t%d\n",a,NOT(a));

}

return 0;

}

int NOT(int a)

{

if(a==1)

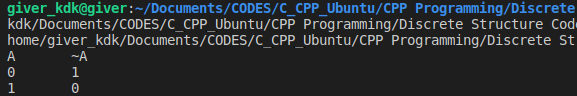
return 0;

else

return 1;

}

**Output:**

****

1. **Program to find the number of ordered arrangemnt without repetition using permutation**

#include <stdio.h>

int factorial(int num);

int main()

{

int n, r, arrangements;

printf("Enter the number of element: ");

scanf("%d", &n);

printf("Enter the number of elements to be arranged: ");

scanf("%d", &r);

arrangements = factorial(n) / factorial(n - r);

printf("Possible number of ordered arrangement without repetition: %d \n", arrangements);

}

int factorial(int num)

{

int fact = 1;

for(int i = 1; i <= num; i++ )

{

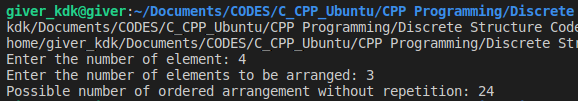
fact = fact \* i;

}

return fact;

}

**Output:**

****

1. **Program to find the number of unordered arrangemnt using combination**

#include <stdio.h>

int factorial(int num);

int main()

{

int n, r, arrangements;

printf("Enter the number of element: ");

scanf("%d", &n);

printf("Enter the number of elements to be arranged: ");

scanf("%d", &r);

arrangements = factorial(n) / (factorial(n - r) \* factorial(r));

printf("Possible number of unordered arrangement: %d \n", arrangements);

}

int factorial(int num)

{

int fact = 1;

for(int i = 1; i <= num; i++ )

{

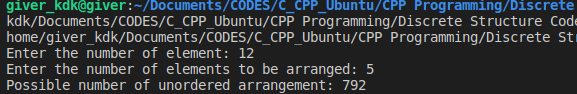
fact = fact \* i;

}

return fact;

}

**Output:**

****

1. **C-Program to determine the properties of a relation**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int reflexive = 0, irreflexive = 0, nonReflexive = 0, count = 0, verified = 0;

int symmetric = 0, asymmetric = 0, antiSymmetric = 0, transitive = 0, antiCount = 0;

int i, j, k, l, setA[10], sizeA;

int relation[100][2], relationSize, pair1[2], pair2[2], pair3[2];

printf("How many elemnets do you want in set?\n");

scanf("%d", &sizeA);

printf("Enter %d elements in set: ", sizeA);

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

{

scanf("%d", &setA[i]);

}

printf("How many pairs do you want in the relation?\n");

scanf("%d", &relationSize);

printf("Enter total %d elements in the relation:\n", relationSize \* 2);

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

{

printf("Enter two elements for pair %d: ", i + 1);

for (j = 0; j < 2; j++)

{

scanf("%d", &relation[i][j]);

}

}

// Relation elements check with respect to set

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

{

for (j = 0; j < 2; j++)

{

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

{

if(setA[k] == relation[i][j])

{

verified = 1;

}

}

if(!verified)

{

printf("Elements on the relation doesn't belong to the set!\n");

exit(0);

}

else

{verified = 0;}

}

}

// Reflexivity test

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

{

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

{

for (k = 0; k < 2; k++)

{

pair1[k] = relation[j][k];

}

if(pair1[0] == setA[i])

{

if(pair1[0] == pair1[1])

{

count++;

break;

}

}

}

}

if(count == sizeA)

{

reflexive = 1;

}

else if(count == 0)

{

irreflexive = 1;

}

else

{

nonReflexive =1;

}

count = 0;

// Symmetry test

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

{

for (j = 0; j < 2; j++)

{

pair1[j] = relation[i][j];

}

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

{

for(k = 0; k < 2; k++)

{

pair2[k] = relation[j][k];

}

if((pair1[0] == pair2[1]) && (pair1[1] == pair2[0]))

{

count++;

if(pair1[0] == pair1[1])

{

antiCount++;

}

}

}

}

if(count == antiCount)

{

antiSymmetric = 1;

}

if(count == relationSize)

{

symmetric = 1;

}

else if(count == 0)

{

asymmetric = 1;

}

count = 0;

// Transitive test

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

{

for (j = 0; j < 2; j++)

{

pair1[j] = relation[i][j];

}

if(pair1[0] != pair1[1])

{

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

{

for(k = 0; k < 2; k++)

{

pair2[k] = relation[j][k];

}

if(pair2[0] != pair2[1])

{

if((pair2[0] == pair1[1]) && (pair1[0] != pair2[1]))

{

for(k = 0; k < relationSize; k++)

{

for(l = 0; l < 2; l++)

{

pair3[l] = relation[k][l];

}

if(pair3[0] != pair3[1])

{

if((pair3[0] == pair1[0]) && (pair3[1] == pair2[1]))

{

transitive = 1;

}

}

}

if(!transitive)

{

goto exit;

}

}

}

}

}

}

exit:

// Final Description

printf("Relation is %s", (nonReflexive ? "Non-Reflexive.\n" : "Reflexive or Irreflexive.\n"));

printf("Relation is %s", (reflexive ? "Reflexive.\n" : "not Reflexive.\n"));

printf("Relation is %s", (irreflexive ? "Irreflexive.\n" : "not Irreflexive.\n"));

printf("Relation is %s", (symmetric ? "Symmetric.\n" : "not Symmetric.\n"));

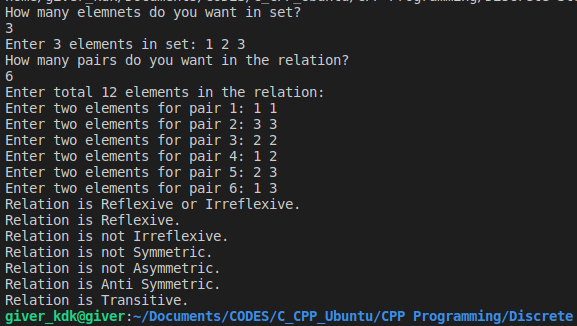
printf("Relation is %s", (asymmetric ? "Asymmetric.\n" : "not Asymmetric.\n"));

printf("Relation is %s", (antiSymmetric ? "Anti Symmetric.\n" : "not Anti Symmetric.\n"));

printf("Relation is %s", (transitive ? "Transitive.\n" : "not Transitive.\n"));

}

**Output:**

****

1. **Program to represent the graph using adjacency matrix.**

#include <stdio.h>

#include <stdlib.h>

int dirgraph();

int undirgraph();

int readgraph(int adjmat[50][50], int n );

int main()

{

int option;

do

{

printf("\n A Program to represent a Graph by using an Adjacency Matrix method \n ");

printf("\n 1. Directed Graph ");

printf("\n 2. Un-Directed Graph ");

printf("\n 3. Exit ");

printf("\n\n Select a proper option : ");

scanf("%d", &option);

switch(option)

{

case 1 :dirgraph();

break;

case 2 :undirgraph();

break;

case 3 : exit(0);

}

}while(1);

}

int dirgraph()

{

int adjmat[50][50];

int n;

int indeg, outdeg, i, j;

printf("\n How Many Vertices ? : ");

scanf("%d", &n);

readgraph(adjmat, n);

printf("\n Vertex \t InDegree \t OutDegree \t TotalDegree ");

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

{

indeg = outdeg = 0;

for ( j = 1 ; j <= n ; j++ )

{

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

indeg++;

}

for ( j = 1 ; j <= n ; j++ )

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

outdeg++;

printf("\n\n %5d\t\t\t%d\t\t%d\t\t%d\n\n",i,indeg,outdeg,indeg+outdeg);

}

return 0;

}

int undirgraph()

{

int adjmat[50][50];

int deg, i, j, n;

printf("\n How Many Vertices ? : ");

scanf("%d", &n);

readgraph(adjmat, n);

printf("\n Vertex \t Degree ");

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

{

deg = 0;

for ( j = 1 ; j <= n ; j++ )

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

deg++;

printf("\n\n %5d \t\t %d\n\n", i, deg);

}

return 0;

}

int readgraph( int adjmat[50][50], int n )

{

int i, j;

char reply;

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

{

for ( j = 1 ; j <= n ; j++ )

{

if ( i == j )

{

adjmat[i][j] = 0;

continue;

}

printf("\n Vertices %d & %d are Adjacent ? (Y/N) :",i,j);

fflush(stdin);

scanf("%c", &reply);

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

adjmat[i][j] = 1;

else

adjmat[i][j] = 0;

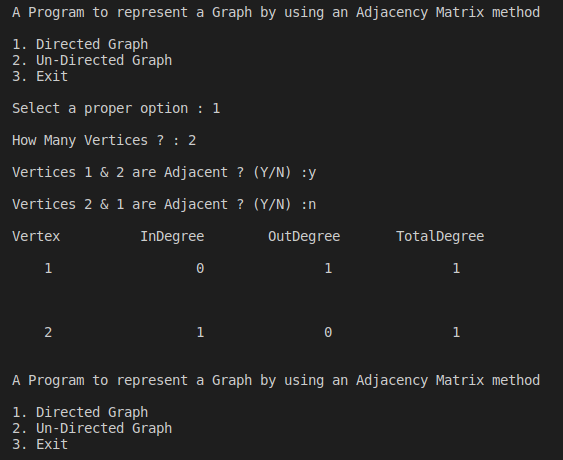
}

}

return 0;

}

**Output:**

****

1. **C-Program to implement Kruskal's Algorithm for MST.**

#include<stdio.h>

int i,j,k,a,b,u,v,n,ne=1;

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

int main()

{

printf("\n Implementation of Kruskal's algorithm\n");

printf("\n Enter no of vertices:");

scanf("%d",&n);

printf("\nEnter the cost of adjacency matrix \n");

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

{

for(j=1;j<=n;j++)

{

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

}

printf("The edges of minimum spanning tree are \n");

while(ne<n)

{

for(i=1,min=999;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(cost [i][j]<min)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}//end for

}//end for

u=find(u);

v=find(v);

if(uni(u,v))

{

printf("%d edge(%d,%d)=%d\n",ne++,a,b,min);

mincost+=min;

}

cost[a][b]=cost[b][a]=999;

}//end while

printf("\n\tMinimal cost=%d\n",mincost);

return 0;

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

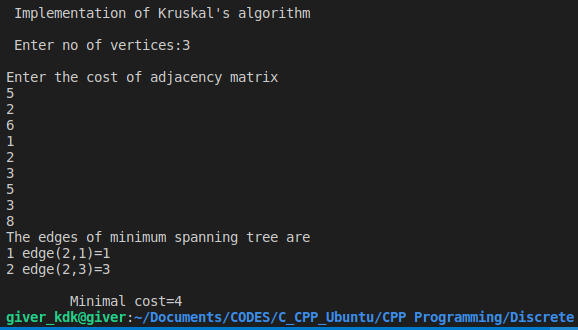
return 1;

}

return 0;

}

**Output:**

****