**PROGRAM 8:**

**Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal’s algorithm. Use Union-Find algorithms in your program.**

**Kruskal’s Algorithm**

Algorithm Kruskal(G)

//Kruskal’s Algorithm for constructing a minimum spanning tree.

//Input: A weighted connected graph G=( V,E)

//Output: ET, the set of edges composing a minimum spanning tree of G

Sort E in non-decreasing order of the edge weights

W(ei1)  ………  W( ei |E|)

ET  ; //initialize the set of tree edges

ecounter 0 //inititalize the size of tree

k 0 //initialize the no of processed edges

while ecounter < |V| -1

k  k + 1

if ET U{eik} is a cyclic

ET ET U{eik}

Ecounter ecounter + 1

return ET

With an efficient sorting algorithm the

**Time complexity of Kruskal’s Algorithm is O( |E| log |E| )**

**Code:**

package maverick;

import java.util.\*;

public class Lp8 {

public static void main(String[] args) {

// TODO Auto-generated method stub

int cost[][]=new int[10][10];

int i,j,mincost=0;

Scanner in=new Scanner(System.in);

System.out.println("kruskal's");

System.out.println("enter no. of nodes");

int n=in.nextInt();

System.out.println("enter cost matrix");

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

{

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

{

cost[i][j]=in.nextInt();

}

}

System.out.println("The entered cost matrix is");

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

{

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

{

System.out.print(cost[i][j]+"\t");

}

System.out.println();

}

System.out.println("The MST edges and costs are");

mincost=Kruskal(cost,n,mincost);

System.out.println("The MST cost is");

System.out.println(mincost);

}

static int Kruskal(int cost[][],int n,int mincost)

{

int ne=1,min;

int a=0,b=0,u=0,v=0;

int parent[]=new int[10];

while(ne<n)

{

min=999;

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

{

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

{

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

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u,parent); //u=first element,Find absolute parent of u and parent subset

v=find(v,parent); //v= second element,Find absolute parent of subset

if(union(u,v,parent)!=0)

{

System.out.print((ne++)+") min edge is ");

System.out.println("("+a+","+b+") and cost is "+min);

mincost+=min;

parent[v]=u;

}

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

}

return mincost;

}

// Find the root of the set in which element `i` belongs

static int find(int i,int parent[])

{

while(parent[i]!=0)

i=parent[i];

return i;

}

    // Perform Union of two subsets

static int union(int i,int j,int parent[])

{

if(i!=j)

{

parent[j]=i;

return 1;

}

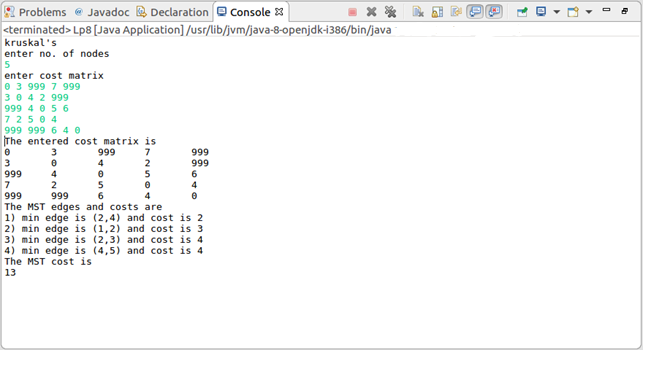
else

return 0;

}

}

OUTPUT:



**PROGRAM 9**

**Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim’s algorithm.**

Algorithm PRIM(G)

//Purpose: Prim’s algorithm for constructing a minimum spanning tree

//Input: A weighted connected graph G = (V,E)

//Output: ET, the set of edges composing a minimum spanning tree of G

VT  { V0} // the set of tree vertices can be initialized with any vertex.

ET  

For i 1 to |V| -1 do

//find a minimum weight edge e\* (V\*,U\*)

//among all the edges (V,U) such that V in VT and U is in V-VT

VT  VT U {U\*}

ET  ET U {e\*}

return ET

**Time complexity of Prims Algorithm is ( |V2| )**

**Code:**

import java.util.\*;

public class Lp9 {

public static void main(String[] args) {

// TODO Auto-generated method stub

int cost[][]=new int[10][10];

int i,j,mincost=0;

Scanner in=new Scanner(System.in);

System.out.println("PRIMS");

System.out.println("enter no. of nodes");

int n=in.nextInt();

System.out.println("enter cost matrix");

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

{

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

{

cost[i][j]=in.nextInt();

}

}

System.out.println("the entered cost matrix is");

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

{

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

{

System.out.print(cost[i][j]+"\t");

}

System.out.println();

}

System.out.println("MST edges and cost are ");

mincost=Prims(cost,n,mincost);

System.out.println("MST cost is");

System.out.println(+mincost);

System.out.println("----------");

}

static int Prims(int cost[][],int n,int mincost)

{

int nearV[]=new int[10],t[][]=new int[10][3],u=0,i,j,k;

for(i=2;i<=n;i++)

nearV[i]=1; //**a vector nearV[i] to store theleast cost details of the edges**

nearV[1]=0; //source node

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

{

int min=999;

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

{

if(nearV[j]!=0&&cost[j][nearV[j]]<min)

{

min=cost[j][nearV[j]];

u=j;

}

}

t[i][1]=u;

t[i][2]=nearV[u];

mincost+=min;

nearV[u]=0;

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

{

if(nearV[k]!=0&&cost[k][nearV[k]]>cost[k][u])

nearV[k]=u;

}

System.out.print(i+")min edge is ("+t[i][1]);

System.out.println(","+t[i][2]+") and cost is "+min);

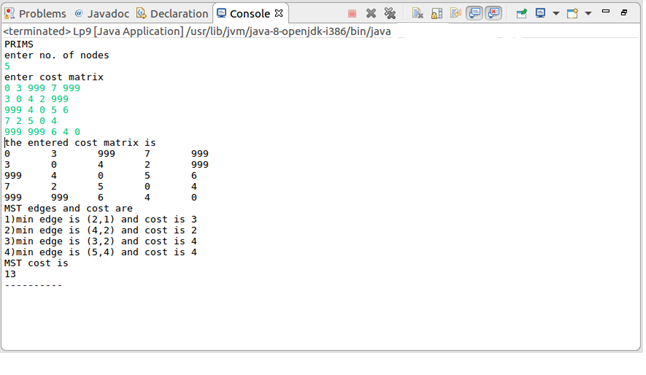
}

return mincost;

}

}

OUTPUT:



**PROGRAM 10A**

**Write Java programs to (a) Implement All-Pairs Shortest Paths problem using Floyd’s algorithm.**

**/ Floyd’s Algorithm**

Algorithm Floyd (W[1……n, 1….n])

//Purpose: Implements Floyd’s algorithm for the all pairs shortest paths problem.

//Input: The weight matrix W of a graph.

//Output : The distance matrix of the shortest paths length.

D = W // is not necessary if W can be overwritten

for k 1 to n do

for i 1 to n do

for j 1 to n do

D[i,j]  min {D [i,j], D[i,k]+D[k,j]}

return D

# Time complexity of Floyd’s Algorithm is ( n3 )

**Code:**

import java.util.\*;

public class Lp10A {

public static void main(String[ ] args) {

// TODO Auto-generated method stub

int a[ ][ ]=new int[10][10];

int i,j;

Scanner in=new Scanner(System.in);

System.out.println("Floyd's");

System.out.println("enter no. of nodes");

int n=in.nextInt();

System.out.println("Enter the adjacency matrix");

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

{

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

{

a[i][j]=in.nextInt();

}

}

System.out.println("The entered adjacency matrix is");

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

{

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

{

System.out.print(a[i][j]+"\t");

}

System.out.println();

}

Floyd(a,n);

System.out.println("All pair shortest path matrix");

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

{

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

{

System.out.print(a[i][j]+"\t");

}

System.out.println();

}

}

static void Floyd(int a[][],int n)

{

int i,j,k;

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

{

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

{

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

{

a[i][j]=min(a[i][j],a[i][k]+a[k][j]);

}

}

}

}

static int min(int a,int b)

{

if(a>b)

return b;

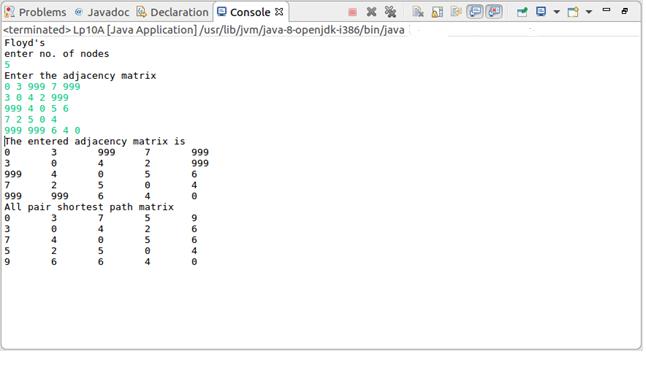
else

return a;

}

}

**OUTPUT:**



**PROGRAM 10B**

**Implement Travelling Sales Person problem using Dynamic programming.**

**Traveling salesman Problem**

Algorithm TSP(G,n)

for k:2 to n do

c({i,k},k) := d1,k

end for

for s=3 to n do

for all S{1,2,…..,n} ||S||=S do

for all k to S do

{

c(S,k)= minm6=k,n S[c(S={k},m)+dm,k]

}

opt= mink6=1[c({1,2,3,…..n},k)+d1,k

end for

end for

end for

return (opt)

end

**Code:**

import java.util.\*;

public class Lp10B {

public static void main(String[] args) {

// TODO Auto-generated method stub

int c[][]=new int[10][10],tour[]=new int[10];

int i,j,cost;

Scanner in=new Scanner(System.in);

System.out.println("TSP by Dynamic Programming");

System.out.println("Enter no. of cities");

int n=in.nextInt();

if(n==1)

{

System.out.println("Path not possible");

System.exit(0);

}

System.out.println("Enter the cost matrix");

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

{

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

{

c[i][j]=in.nextInt();

}

}

System.out.println("The entered cost matrix is");

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

{

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

{

System.out.print(c[i][j]+"\t");

}

System.out.println();

}

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

tour[i]=i;

cost=tsp(c,tour,1,n);

System.out.println("The accurate path is");

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

System.out.print(tour[i]+"->");

System.out.println("1");

System.out.println("The accurate cost is "+cost);

}

static int tsp(int c[][],int tour[],int start,int n)

{

int mintour[]=new int[10],temp[]=new int[10],mincost=999,ccost,i,j,k;

if(start==n-1)

{

return(c[tour[n-1]][tour[n]]+c[tour[n]][1]);

}

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

{

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

temp[j]=tour[j];

temp[start+1]=tour[i];

temp[i]=tour[start+1];

if((c[tour[start]][tour[i]]+(ccost=tsp(c,temp,start+1,n)))<mincost)

{

mincost=c[tour[start]][tour[i]]+ccost;

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

mintour[k]=temp[k];

}

}

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

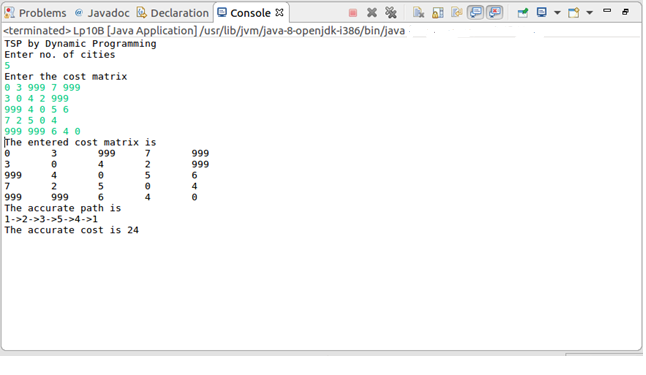
tour[i]=mintour[i];

return mincost;

}

}

OUTPUT:



**PROGRAM 11**

**Design and implement in Java to find a subset of a given set S = {Sl, S2,.....,Sn} of n positive integers whose SUM is equal to a given positive integer d. For example, if S ={1, 2, 5, 6, 8} and d= 9, there are two solutions {1,2,6}and {1,8}. Display a suitable message, if the given problem instance doesn’t have a solution.**

**Description:**

This problem is one of the most popular backtracking question. Recursion is the key in backtracking programming. As the name suggests we back- track to find the solution. We start with one possible move out of many available moves and try to solve the problem if we are able to solve the problem with the selected move then we will print the solution else we will backtrack and select some other move and try to solve it. If none if the moves work out we will claim that there is no solution for the problem.

## Algorithm: Subset(s, k, r)



//create left child with kth element

1. x[k]=1
2. if(s+w[k]==d)
3. for(int i=0;i<=k;i++)
4. if(x[i]==1)
5. print w[i]
7. else if((s+w[k]+w[k+1])<=d)
8. Subset(s+w[k],k+1,r-w[k]);

//create right child with out kth object

1. if((s+r-w[k])>=d&&(s+w[k+1])<=d)
2. {
3. x[k]=0;
4. Subset(s,k+1,r-w[k]);
5. }
7. end Subset

.

import java.util.\*;

public class Lp11 {

static int c=0;

public static void main(String[] args) {

// TODO Auto-generated method stub

int w[]=new int[10];

int n,d,i,sum=0;

int x[]=new int[10];

Scanner in=new Scanner(System.in);

System.out.println("SUBSET");

System.out.println("enter the no. of elements");

n=in.nextInt();

System.out.println("enter elements in increasing order");

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

w[i]=in.nextInt();

System.out.println("enter value of d");

d=in.nextInt(); //read the subset max value d

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

sum=sum+w[i];

System.out.println("sum="+sum);

if(sum<d)

{

System.out.println("subset not possible");

System.exit(0);

}

Subset(0,0,sum,x,w,d); //passing initial valu(cs)e,index(k),final sum(r)

if(c==0)

System.out.println("subset not posssible");

}

static void Subset(int cs,int k,int r,int x[],int w[],int d)

{

x[k]=1; //number of subsets

if(cs+w[k]==d)

{

c++;

System.out.print("\nSolution "+c+" is {");

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

if(x[i]==1)

{

System.out.print(w[i]+" ");

}

System.out.println("}");

}

else if((cs+w[k]+w[k+1])<=d)

Subset(cs+w[k],k+1,r-w[k],x,w,d);

if((cs+r-w[k])>=d&&(cs+w[k+1])<=d)

{

x[k]=0;

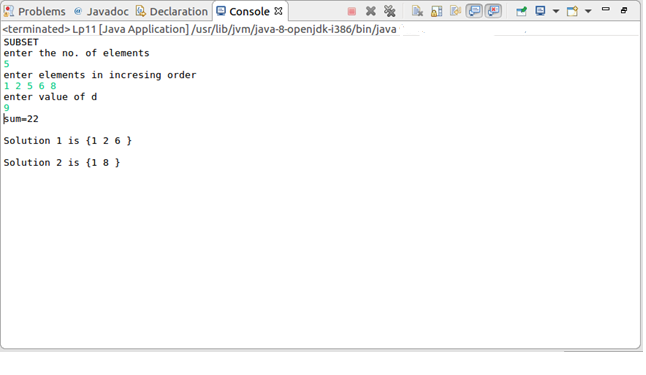
Subset(cs,k+1,r-w[k],x,w,d);

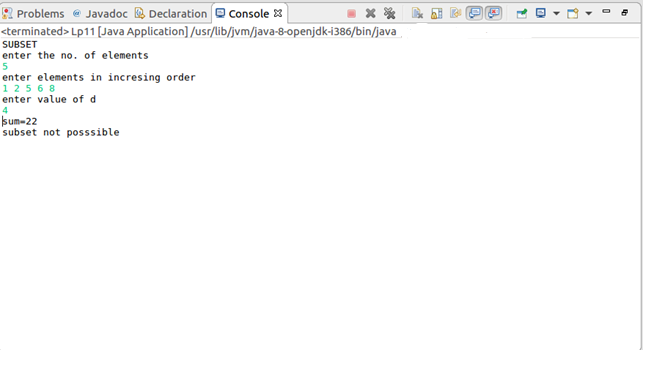
}

}

}

OUTPUT:





**PROGRAM 12**

**Design and implement in Java to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.**

**Description :**

Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in graph) from the last vertex to the first vertex of the Hamiltonian Path. Determine whether a given graph contains Hamiltonian Cycle or not. If it contains, then print the path.

## Algorithm:

Generate all possible configurations of vertices and print a configuration that satisfies the given constraints. There will be n! (n factorial) configurations.

while there are untried configurations

{

generate the next configuration

if ( there are edges between two consecutive vertices of this configuration and there is an edge from the last vertex to the first ).

{

print this configuration; break;

}

import java.util.\*;

public class Lp12 {

int count;

int path=1;

public static void main(String[] args) {

// TODO Auto-generated method stub

int graph[ ][ ]=new int[10][10];

Scanner in=new Scanner(System.in);

System.out.println("Hamiltonian Cycles");

System.out.println("enter no. of nodes");

int n=in.nextInt();

System.out.println("enter the adjacency matrix");

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

{

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

{

graph[i][j]=in.nextInt();

}

}

int arr[ ]=new int[n];

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

arr[i]=i;

System.out.println("All possible Hamiltonian Cycles are");

new Lp12().Permute(arr,graph);

}

void Permute(int arr[ ],int graph[ ][ ])

{

Permute(arr,0,arr.length-1,graph);

}

void Permute(int arr[ ],int i,int n,int cost[ ][ ])

{

int j;

if(i==n)

{

HamCycle(arr,cost);

}

else

{

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

{

Swap(arr,i,j);

Permute(arr,i+1,n,cost);

Swap(arr,i,j);

}

}

}

void HamCycle(int a[],int graph[][])

{

count=0;

for(int i=0;i<a.length-1;i++)

{

if(graph[a[i]][a[i+1]]!=0)

count++;

}

if(count==a.length-1&&graph[a[a.length-1]][a[0]]==1)

{

System.out.println("cycle no."+path+"->");

for(int i=0;i<a.length;i++)

System.out.print(a[i]+" ");

System.out.println(a[0]);

System.out.println();

path++;

}

}

void Swap(int a[ ],int i,int j)

{

int temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

OUTPUT:

