**Program 1: Create n Student Objects**

import java.util.Scanner;

class Student

{

int usn;

String name;

String branch;

long phone;

Student(int no, String nm, String br,long ph)

{

usn=no;

name=nm;

branch=br;

phone=ph;

}

void print()

{

System.out.print("USN : "+usn+"\t");

System.out.print("NAME "+name+"\t");

System.out.print("BRANCH "+branch+"\t");

System.out.println("PHONE "+phone);

}

}

public class TestStudent {

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.println("Enter number of students : ");

int n = sc.nextInt();

int no;

String name, branch;

long phone;

Student s[] = new Student[n];

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

{

System.out.println("Enter "+(i+1)+ " Student Details");

System.out.print(" Usn:");

no = sc.nextInt();

System.out.print(" Name:");

name = sc.next();

System.out.print(" Branch:");

branch = sc.next();

System.out.print(" Phone:");

phone = sc.nextLong();

s[i] = new Student(no, name,branch, phone);

}

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

{

System.out.println("Details of the student "+(i+1)+":");

s[i].print();

}

}

}

**Program 2: Stack operation**

import java.util.\*;

import java.util.Scanner;

class Stack

{

int top, size, s[];

Stack()

{

top=-1;

size=5;

s=new int[size];

}

Stack(int x)

{

top=-1;

size=x;

s=new int[size];

}

boolean stackEmpty()

{

if(top==-1)

return true;

else

return false;

}

boolean stackFull()

{

if(top==size-1)

return true;

else

return false;

}

void push(int ele)

{

if(stackFull())

System.out.println(" stack full");

else

s[++top]=ele;

}

int pop()

{

return s[top--];

}

void display()

{

if(stackEmpty())

{

System.out.println("Stack is Empty..");

return;

}

System.out.println("Stack elements ..");

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

{

System.out.println(s[i]);

}

}

}

public class TestStack

{

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter the size= ");

int n = sc.nextInt();

Stack s=new Stack(n);

boolean cont=true;

int choice, ele;

while(cont)

{

System.out.println("Stack Operations");

System.out.println("1. push");

System.out.println("2. pop");

System.out.println("3. display");

System.out.println("4. Exit");

System.out.println("Enter the choice=");

choice=sc.nextInt();

switch (choice)

{

case 1 : System.out.println("Enter int ele=");

ele=sc.nextInt();

s.push(ele);

break;

case 2 : if(s.stackEmpty())

System.out.println(" Stack empty");

else

System.out.println("Popped Element = " + s.pop());

break;

case 3 : s.display();

break;

case 4 : cont=false; break;

default : System.out.println("Wrong Entry Try again");

break;

}

}

}

}

**Program 3: Read and Display details of 3 Staff objects and Customer data**

class Staff

{

int staffid,phone,salary; String name;

public Staff(int id , int no, int sal, String na)

{

staffid=id;

phone=no;

salary=sal;

name=na;

}

void display()

{

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

System.out.println("Staff ID:"+ " "+ staffid);

System.out.println("Staff Phone number:" + " "+phone);

System.out.println("Staff Salary:" +" "+ salary);

System.out.println("Staff Name:" +" "+ name);

}

}

class Teaching extends Staff

{

String domain;

int no\_of\_publications;

public Teaching(int id, int no, int sal, String na,String d,int nop)

{

super(id,no,sal,na);

domain=d; no\_of\_publications=nop;

}

void Tdisplay()

{

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

System.out.println("Teaching Staff Details"); super.display();

System.out.println("Domain :" +" "+domain);

System.out.println("No\_of\_publications:"+" "+no\_of\_publications);

}

}

class Technical extends Staff

{

String skills;

public Technical(int id , int no, int sal, String na,String sk)

{

super(id,no,sal,na);

skills=sk;

}

void Tedisplay()

{

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

System.out.println("Technical Staff Details");

super.display();

System.out.println("Skills :" + " "+skills);

}

}

class Contract extends Staff

{

int period;

public Contract(int id , int no, int sal, String na,int pd)

{

super(id,no,sal,na);

period=pd;

}

void Cdisplay()

{

System.out.println("-------------------------------------"); System.out.println("Contract Staff Details");

super.display();

System.out.println("ContractPeriod:" + " "+period + "years");

}

}

public class Multilevel

{

public static void main(String args[])

{

Teaching t1=new Teaching(11,998765434,31000,"Anil","CSE",10); Teaching t2=new Teaching(12,996655546,30000,"Anu","ISE",9); Teaching t3=new Teaching(13,999933442,32000,"Anusha","EEE",8); t1.Tdisplay();

t2.Tdisplay();

t3.Tdisplay();

Technical te1=new Technical(21,994433221,22000,"Kumar","C");

Technical te2=new Technical(22,998877665,28000,"Krisna","Java");

Technical te3=new Technical(23,991654321,33000,"Kiran","Java"); te1.Tedisplay();

te2.Tedisplay();

te3.Tedisplay();

Contract ct1=new Contract(31,998765434,35000,"Anil",3); Contract ct2=new Contract(32,912345678,39000,"Meghana",2); Contract ct3=new Contract(33,992233445,30000,"Uma",4); ct1.Cdisplay();

ct2.Cdisplay();

ct3.Cdisplay();

}

}

**Program 4: String Tokenizer class**

import java.util.\*;

public class Strtoken

{

private String name,date\_of\_birth; public void input()

{

Scanner s=new Scanner(System.in); System.out.println("enter the name"); name=s.nextLine();

System.out.println("enter the date of birth in format dd/mm/yyyy");

date\_of\_birth=s.nextLine();

}

public void display()

{

System.out.println("name is:"+name);

StringTokenizer st=new StringTokenizer(date\_of\_birth,"/"); System.out.print("Date of birth is:");

while(st.hasMoreTokens()) System.out.print(st.nextToken("/")+",");

}

public static void main(String[] args)

{

Strtoken st=new Strtoken();

st.input();

st.display();

}

}

**Program 5: Exception handling**

import java.util.Scanner;

public class ExceptionProgram

{

public static void main(String[] args)

{

int a,b;

float res;

try

{

Scanner inn=new Scanner(System.in);

System.out.println("Input Dividend value ");

a=inn.nextInt();

System.out.println("Input Divisor ");

b=inn.nextInt();

res=a/b;

System.out.println("Quotient is="+res);

int arr[]= new int[b];

arr[a]=b;

}

catch(ArithmeticException e)

{

System.out.println("Divide by zero error "+e);

}

catch(ArrayIndexOutOfBoundsException e)

{

System.out.println("Array index out-of-bounds: " + e);

}

}

}

**Program 6: Multithreading**

import java.util.Random;

class Square extends Thread

{

int x;

Square(int n)

{

x = n;

}

public void run()

{

int sqr = x \* x;

System.out.println("Square of " + x + " = " + sqr );

}

}

class Cube extends Thread

{

int x;

Cube(int n)

{

x = n;

}

public void run()

{

int cub = x \* x \* x;

System.out.println("Cube of " + x + " = " + cub );

}

}

class Number extends Thread

{

public void run()

{

Random random = new Random();

for(int i =0; i<10; i++)

{

int randomInteger = random.nextInt(100); System.out.println("Random Integer generated : " +randomInteger); Square s = new Square(randomInteger);

s.start();

Cube c = new Cube(randomInteger);

c.start();

try {

Thread.sleep(1000);

}

catch (InterruptedException ex)

{

System.out.println(ex);

}

}

}

}

public class Multi

{

public static void main(String args[])

{

System.out.println("10 Random Integer");

Number n = new Number();

n.start();

}

}

**Program 7: Quick sort**

import java.util.Scanner;

import java.util.Arrays;

import java.util.Random;

public class QuickSortComplexity

{

static final int MAX = 200000;

static int[] a;

static Random random = new Random();

public static void main(String[] args)

{

Scanner sc = new Scanner(System.in);

System.out.print("Enter Max array size: ");

int n = sc.nextInt();

a= new int[n];

System.out.println("Enter the array elements: ");

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

a[i] = random.nextInt(1000);

System.out.println("---The Best Case and AVG Case Scenario---"); TimeComplexity(n);

System.out.println("---The Worst Case Scenario---");

Arrays.sort (a);

TimeComplexity(n);

sc.close();

}

public static void TimeComplexity(int n)

{

long startTime=0;

startTime= System.nanoTime();

QuickSortAlgorithm(0, n - 1);

long stopTime=0;

stopTime = System.nanoTime();

long elapsedTime = stopTime - startTime;

System.out.println("Time Complexity in ms for n=" + n + " is: " + (double) elapsedTime / 1000000);

}

public static void QuickSortAlgorithm(int p, int r)

{

int i, j, temp, pivot;

if (p < r)

{

i = p;

j = r + 1;

pivot = a[p]; // mark first element as pivot

while (true)

{

i++;

while (a[i] < pivot && i < r) i++;

j--;

while (a[j] > pivot) j--;

if (i < j)

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

else

break; // partition is over

}

a[p] = a[j]; a[j] = pivot;

QuickSortAlgorithm(p, j - 1);

QuickSortAlgorithm(j + 1, r);

}

}

}

**Program 8: Merge Sort**

import java.util.Random;

import java.util.Scanner;

public class MergeSort

{

static final int *MAX* = 200000;

static int[] *a* = new int[*MAX*];

public static void main(String[] args)

{

Scanner sc = new Scanner(System.*in*);

System.*out*.print("Enter Max array size: ");

int n = sc.nextInt();

Random random = new Random();

//System.out.println("Enter the array elements: ");

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

{

*a*[i] = random.nextInt(1000);

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

}

long startTime = System.*nanoTime*();

*MergeSortAlgorithm*(0, n - 1);

long stopTime = System.*nanoTime*();

long elapsedTime = stopTime - startTime;

System.*out*.println("---The Best Case, AVG Case and Worst CaseScenario---");

System.*out*.println("Time Complexity (ms) for n = " + n + " is: " + (double) elapsedTime / 1000000);

//System.out.println("Sorted Array (Merge Sort):");

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

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

}

public static void MergeSortAlgorithm(int low, int high)

{

int mid;

if (low < high)

{

mid = (low + high) / 2;

*MergeSortAlgorithm*(low, mid);

*MergeSortAlgorithm*(mid + 1, high);

*Merge*(low, mid, high);

}

}

public static void Merge(int low, int mid, int high)

{

int[] temp = new int[*MAX*];

int i, j, k;

k = i = low; j = mid + 1;

while ((i <= mid) && (j <= high))

{

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

{

temp[k]=*a*[i];

i++;

k++;

}

else

{

temp[k]=*a*[j];

j++;

k++;

}

}

while(i<=mid)

{

temp[k]=*a*[i];

i++;

k++;

}

while(j<=high)

{

temp[k]=*a*[j];

j++;

k++;

}

for(k=low;k<=high;k++)

{

*a*[k]=temp[k];

}

}

}

**Program 9: 0/1 knapsack problem**

1. **Dynamic Programming**

import java.util.Scanner;

public class Knapsack\_DP{

static int max(int a, int b)

{

return (a>b)? a:b;

}

static int knapSack(int W, int wt[], int val[], int n)

{

int i, w;

int [][] K = new int[n+1][W+1];

// Build table K[][] in bottom up manner

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

{

for (w = 0; w<= W; w++)

{

if (i==0 || w==0)

K[i][w] = 0;

else if (wt[i-1] <= w)

K[i][w] = *max*(val[i-1] + K[i-1][w-wt[i-1]], K[i-1][w]);

else

K[i][w] = K[i-1][w];

}

}

return K[n][W];

}

public static void main(String args[])

{

Scanner sc = new Scanner(System.*in*);

System.*out*.println("Enter the number of Objects: ");

int n = sc.nextInt();

System.*out*.println("Enter the object's weights: ");

int []wt = new int[n];

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

wt[i] = sc.nextInt();

System.*out*.println("Enter the object's profits: ");

int []val = new int[n];

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

val[i] = sc.nextInt();

System.*out*.println("Enter the maximum capacity: "); int W = sc.nextInt();

System.*out*.println("The Maximum value that can be put in a knapsack of capacity W is: " + *knapSack*(W, wt, val, n));

sc.close();

}

}

1. **Greedy Method**

import java.io.IOException;

import java.util.Scanner;

class Fractional\_Knapsack

{

public static void main(String args[]) throws IOException

{

int i,j=0,max\_qty,m,n;

float sum=0,max;

Scanner sc = new Scanner(System.*in*);

int array[][]=new int[2][20];

System.*out*.println("Enter no of items");

n=sc.nextInt();

System.*out*.println("Enter the weights of each items");

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

array[0][i]=sc.nextInt();

System.*out*.println("Enter the values of each items");

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

array[1][i]=sc.nextInt();

System.*out*.println("Enter maximum volume of knapsack :");

max\_qty=sc.nextInt();

m=max\_qty;

while(m>=0)

{

max=0;

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

{

if(((float)array[1][i])/((float)array[0][i])>max)

{

max=((float)array[1][i])/((float)array[0][i]);

j=i;

}

}

if(array[0][j]>m)

{

System.*out*.println("Quantity of item number: " + (j+1) + " added is " +m);

sum+=m\*max;

m=-1;

}

else

{

System.*out*.println("Quantity of item number: " + (j+1) + " added is " + array[0][j]);

m-=array[0][j];

sum+=(float)array[1][j];

array[1][j]=0;

}

}

System.*out*.println("The total profit is " + sum);

sc.close();

}

}

**Program 10: Implementation of Dijkstra’s algorithm**

import java.util.Scanner;

public class Dijstra

{

public static void main(String[] args)

{

System.*out*.println("DIJKSTRA");

Scanner sc = new Scanner(System.*in*);

//no. of nodes

int cost[][] = new int[10][20]; //Adjacency matrix of graph

//source node

int v[] = new int[10]; //visited array. keeps track to nodes visited and not //visited

int d[] = new int[10]; //distance array. keeps latest shortest distance from //source

int i, j; //index variables

System.*out*.println("Enter n: ");

int n = sc.nextInt();//2. Read cost adjacency matrix of graph

System.*out*.println("Enter Cost matrix: \n");

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

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

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

//3. Read source

System.*out*.println("Enter source: ");

int source = sc.nextInt();

//4. Initialise d[] to distance from source to each node

//Initialise v[] to 0, indicating none of the nodes are visited

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

{

d[i] = cost[source][i];

v[i] = 0;

}

//5. Call function to compute shortest distance

*dijk*(cost, n, source, v, d);

//6. Print Shortest distance from source to all other nodes

System.*out*.println("Shortest distance from source " +source);

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

System.*out*.println(source+ " -->" +i+ "=" +d[i]);

}

public static void dijk(int cost[][],int n,int source,int v[],int d[])

{

int least, i, j, u=0;

//A. Mark source node as visited

v[source] = 1;

//B. From each node find shortest distance to nodes not visited

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

{

//B1. Assume least as infinity

least = 999;

//B2. Find u and d(u) such that d(u) is minimum i.e., Find the next nearest //node

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

{

if(v[j] == 0 && d[j] < least)

{

least = d[j];

u = j;

}

}

//B3. Mark u as visited (mark nearest node as visited)

v[u] = 1;

//B4. For remaining nodes, find shortest distance through u (update d array)

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

{

if(v[j] == 0 && (d[j] > (d[u] + cost[u][j])) )

d[j] = d[u] + cost[u][j]; }

}//end for outer

}

}

**Program 11: Implementation of Kruskal’s algorithm**

import java.util.Scanner;

class Kruskal

{

int ne=0; int size;

int min,mincost=0;

int cost[][];

int parent[];

Kruskal(int n)

{

size=n;

cost=new int[size][size];

parent=new int[size];

}

int find(int i)

{

while(parent[i]!=0)

i=parent[i];

return i;

}

boolean uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

return true;

}

return false;

}

void spanningTree()

{

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

int s=0;

Scanner sc=new Scanner(System.*in*);

System.*out*.println("Implementation of Kruskal's algorithm\n\n");

System.*out*.println("Adjacency matrix for "+size+" x"+size+"\n");

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

{

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

{

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

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

cost[i][j]=999;

}

}

System.*out*.println("Adjacency matrix for "+size+" x"+size+"\n");

System.*out*.println("\nThe edges of Minimum Cost Spanning Tree are\n\n");

while(ne<size-1)

{

min=999;

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

{

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

{

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

{

min=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

if(uni(u,v))

{

s=s+min;

ne++;

System.*out*.println(ne+" edge= "+a+", "+b+" mincost="+s);

}

cost[a][b]=999;

cost[b][a]=999;

}

System.*out*.println("\n\n");

System.*out*.println("Minimum cost = "+s);

}

}

public class Prg8

{

public static void main(String args[])

{

System.*out*.println("Enter the rows and columns of matrix");

Scanner sc=new Scanner(System.*in*);

int s=sc.nextInt();

Kruskal kobj=new Kruskal(s);

kobj.spanningTree();

}

}

**Program 12: Implementation of Prim’s algorithm**

import java.util.Scanner;

public class Prims

{

public static void main(String[] args)

{

int source; //source node

int i, j; //index variable

//1. Read no. of nodes

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

Scanner sc = new Scanner(System.*in*);

int a[] = new int[1000];

System.*out*.println("Enter n: ");

int n = sc.nextInt();

//2. Read cost adjavcency matrix

System.*out*.println("Enter cost matrix: ");

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

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

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

//3. Read source

System.*out*.println("Enter Source: ");

source = sc.nextInt();

//4. Call prims function

*prims*(n,cost,source);

}

public static void prims(int n,int cost[][],int source)

{

int i, j; //index variables

int u=0, least, sum=0;

int v[] = new int[1000];

int d[] = new int[1000];

int vertex[] = new int[1000];

//1. Initialisation

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

{

v[i] = 0; //visited array

d[i] = cost[source][i]; //distance array

vertex[i] = source; //nearest node to spanning tree

}

v[source] = 1; //mark source node as visited

//2. n iteration

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

{

least = 999;

//2a) Find u and d(u) such that d(u) is least (find nearset neighbour)

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

{

if(v[j] == 0 && d[j] < least)

{

least = d[j];

u = j;

}

}

//2b) mark node u as visited

v[u] = 1;

sum += d[u];

System.out.println("%d --> %d = %d Sum = %d\n\n",vertex[u],u,d[u],sum);

System.*out*.println(vertex[u]+ " -->" +u+ "=" +d[u]+ "Sum = " +sum);

//2c) update d[] array. Explore paths through node u.

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

{

if(v[j] == 0 && cost[u][j] < d[j])

{

d[j] = cost[u][j];

vertex[j] = u;

}

}//end for inner

}//end for outer

//3. Print total sum

System.*out*.println("Total cost: " +sum);

}//end prims

}

**Program 13: Implementation of Floyd’s algorithm**

import java.util.Scanner;

public class Floyd

{

private int dmat[][];

private int n;

public static final int *INFINITY* = 999;

public Floyd(int n)

{

dmat = new int[n + 1][n + 1]; this.n = n;

}

public void floyd\_compute(int adjmat[][]){ for (int i = 1; i <= n; i++)

{

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

{

dmat[i][j] = adjmat[i][j];

}

}

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

{

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

{

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

{

if (dmat[i][k] + dmat[k][j] < dmat[i][j])

dmat[i][j] = dmat[i][k] + dmat[k][j];

}

}

}

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

System.*out*.print("\t" + i);

System.*out*.println();

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

{

System.*out*.print(i + "\t");

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

{

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

}

System.*out*.println();

}

}

public static void main(String[] arg)

{

int adjmat[][];

int n;

Scanner scan = new Scanner(System.*in*);

System.*out*.println("Enter the number of vertices");

n = scan.nextInt();

adjmat = new int[n + 1][n + 1];

System.*out*.println("Enter the Weighted Matrix for the graph");

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

{

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

{

adjmat[i][j] = scan.nextInt();

if (i == j)

{

adjmat[i][j] = 0; continue;

}

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

{

adjmat[i][j] = *INFINITY*;

}

}

}

System.*out*.println("The All Pairs Shortest Distance of the Graph is: ");

Floyd floydobj = new Floyd(n);

floydobj.floyd\_compute(adjmat);

scan.close();

}

}

**Program 14: Travelling sales Person problem**

import java.util.Scanner;

public class TravellingSalesPerson

{

static int *MAX* = 100;

static final int *infinity* = 999;

public static void main(String args[])

{

int cost = *infinity*;

int c[][] = new int[*MAX*][*MAX*]; // cost matrix

int tour[] = new int[*MAX*]; // optimal tour

int n; // max. cities

System.*out*.println("Travelling Salesman Problem using Dynamic Programming\n");

System.*out*.println("Enter number of cities: ");

Scanner sc = new Scanner(System.*in*);

n = sc.nextInt();

System.*out*.println("Enter Cost matrix:\n"); for (int i = 0; i < n; i++)

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

{

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

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

c[i][j] = 999;

}

for (int i = 0; i < n; i++) tour[i] = i;

cost = *tspdp*(c, tour, 0, n); // print tour cost and tour

System.*out*.println("Minimum Tour Cost: " + cost);

System.*out*.println("\nTour:");

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

{

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

}

System.*out*.println(tour[0] + "\n"); sc.close();

}

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

{

int i, j, k;

int temp[] = new int[*MAX*];

int mintour[] = new int[*MAX*];

int mincost, cost;

if (start == n - 2)

return c[tour[n - 2]][tour[n - 1]] + c[tour[n - 1]][0];

mincost = *infinity*;

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

{

for (j = 0; j < n; j++) temp[j] = tour[j];

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

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

if (c[tour[start]][tour[i]] + (cost = *tspdp*(c, temp, start + 1, n)) < mincost)

{

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

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

mintour[k] = temp[k];

}

}

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

tour[i] = mintour[i];

return mincost;

}

}

**Program 15: Implementation of subset problem**

import java.util.Scanner;

public class Subset {

public static void main(String[] args)

{

//Required subset sum

int i; //index variable

int sum = 0;

Scanner sc = new Scanner(System.in);

int s[] = new int[1000];

//1. Read no. of elements in set

System.out.println("Enter n: ");

int n = sc.nextInt();

//2. Read the elements in the set

System.out.println("Enter the set in increasing order");

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

{

s[i] = sc.nextInt();

sum += s[i];

}

//3. Read required subset sum

System.out.println("Enter the maximum subset value of d: ");

int d = sc.nextInt();

//4. Call function

if(sum < d)

System.out.println("Solution NOT possible.\n");

else

subset(n,d,s);

}

public static void subset(int n, int d, int s[])

{

int x[] = new int[1000]; //Shows elements in subset (0 - Absent 1 - //Present)

int sum; //Stores current sumset sum

int i, k; //index variables

//Initialise x[] to 0. (None of the elements in set are selected)

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

x[i] = 0;

sum = 0;

k = 1; //Take first element

x[k] = 1; //Add first element to subset

while(true)

{

if(k <= n && x[k] == 1) //k in range(1 to n) & kth element //selected

{

if(sum+s[k] == d)//If required subset sum found, print //solution

{

System.out.println("Solution is \n");

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

{

if(x[i] == 1)

System.out.println(s[i]);

}

System.out.println(" ");

x[k] = 0; //Proceed to find next solution

}

else if(sum + s[k] < d) //If subset sum is lessthan required sum,

sum += s[k]; //then add the current element to //subset

else

x[k] = 0; //If subset sum is morethan required //sum,

//then remove the current element

}

else

{

k--; //Bring k within range

while(k > 0 && x[k] == 0) //Find previously added element to //subset

k--;

if(k == 0) break; //No more solutions exists, so quit

x[k] = 0; //Remove recently added element from //subset

sum = sum - s[k];

}

k = k + 1; //Take next element

x[k] = 1; //Add next element to subset

} //end while

} //end function

}

**Program 16: Implementation of Hamiltonian cycles**

import java.util.Scanner;

import java.util.Arrays;   
 /\*\* Class HamiltonianCycle \*\*/  
 public class HamiltonianCycle  
 {  
 private int V, pathCount;  
 private int[] path;   
 private int[][] graph;  
 /\*\* Function to find cycle \*\*/  
 public void findHamiltonianCycle(int[][] g)  
 {  
 V = g.length;  
 path = new int[V];  
 Arrays.fill(path, -1);  
 graph = g;   
 try  
 {   
 path[0] = 0;  
 pathCount = 1;   
 solve(0);  
 System.out.println("No solution");  
 }  
 catch (Exception e)  
 {  
 System.out.println(e.getMessage());  
 display();  
 }  
 }  
  
 /\*\* function to find paths recursively \*\*/  
 public void solve(int vertex) throws Exception  
 {  
 /\*\* solution \*\*/  
 if (graph[vertex][0] == 1 && pathCount == V)  
 throw new Exception("Solution found");  
 /\*\* all vertices selected but last vertex not linked to 0 \*\*/  
 if (pathCount == V)  
 return;  
 for (int v = 0; v < V; v++)  
 {  
 /\*\* if connected \*\*/  
 if (graph[vertex][v] == 1 )  
 {  
 /\*\* add to path \*\*/   
 path[pathCount++] = v;   
 /\*\* remove connection \*\*/   
 graph[vertex][v] = 0;  
 graph[v][vertex] = 0;  
 /\*\* if vertex not already selected solve recursively \*\*/  
 if (!isPresent(v))  
 solve(v);  
 /\*\* restore connection \*\*/  
 graph[vertex][v] = 1;  
 graph[v][vertex] = 1;  
 /\*\* remove path \*\*/  
 path[--pathCount] = -1;   
 }  
 }  
 }   
 /\*\* function to check if path is already selected \*\*/  
 public boolean isPresent(int v)  
 {  
 for (int i = 0; i < pathCount - 1; i++)  
 if (path[i] == v)  
 return true;  
 return false;   
 }  
 /\*\* display solution \*\*/  
 public void display()  
 {  
 System.out.print("\nPath : ");  
 for (int i = 0; i <= V; i++)  
 System.out.print(path[i % V] +" ");  
 System.out.println();  
 }   
 /\*\* Main function \*\*/  
 public static void main (String[] args)   
 {  
 Scanner scan = new Scanner(System.in);  
 System.out.println("HamiltonianCycle Algorithm Test\n");  
 /\*\* Make an object of HamiltonianCycle class \*\*/  
 HamiltonianCycle hc = new HamiltonianCycle();  
 /\*\* Accept number of vertices \*\*/  
 System.out.println("Enter number of vertices\n");  
 int V = scan.nextInt();  
 /\*\* get graph \*\*/  
 System.out.println("\nEnter matrix\n");  
 int[][] graph = new int[V][V];  
 for (int i = 0; i < V; i++)  
 for (int j = 0; j < V; j++)  
 graph[i][j] = scan.nextInt();  
 hc.findHamiltonianCycle(graph);   
 }   
 }