**Design and Analysis of Algorithms Laboratory**

**Subject Code: 18CSL47 I.A. Marks : 20**

**Hours/Week : 01 I + 02 P Exam Hours: 03**

**Total Hours : 40 Exam Marks: 80**

|  |  |
| --- | --- |
| **COURSE OUTCOME:** | |
| After the completion of this course the students will be able to: | |
| CO1: | Design algorithms using appropriate design techniques (brute-force, greedy, dynamic programming, etc.) |
| CO2: | Implement a variety of algorithms such assorting, graph related, combinatorial, etc., in a high level language. |
| CO3: | Analyze and compare the performance of algorithms using language features |
| CO4: | Apply and implement learned algorithm design techniques and data structuresto solve realworld problems. |

1. A. Create a Java class called Studentwith the following details as variables within it. (i) USN (ii) Name (iii) Branch (iv) Phone Write a Java program to create nStudent objects and print the USN, Name, Branch, and Phoneof these objects with suitable headings.

B. Write a Java program to implement the Stack using arrays. Write Push(), Pop(), and Display() methods to demonstrate its working.

1. A. Design a superclass called Staff with details as StaffId, Name, Phone, Salary. Extend this class by writing three subclasses namely Teaching (domain, publications), Technical (skills), and Contract (period). Write a Java program to read and display at least 3 staff objects of all three categories.

B. Write a Java class called Customer to store their name and date\_of\_birth. The date\_of\_birth format should be dd/mm/yyyy. Write methods to read customer data as and display as using StringTokenizer class considering the delimiter character as “/”.

1. A. Write a Java program to read two integers a andb. Compute a/b and print, when b is not zero. Raise an exception when b is equal to zero.

B. Write a Java program that implements a multi-thread application that has three threads. First thread generates a random integer for every 1 second; second thread computes the square of the number andprints; third thread will print the value of cube of the number.

1. Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divideand-conquer method works along with its time complexity analysis: worst case, average case and best case.
2. Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divideand-conquer method works along with its time complexity analysis: worst case, average case and best case.
3. Implement in Java, the 0/1 Knapsack problem using (a) Dynamic Programming method (b) Greedy method.
4. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in Java.
5. Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal'salgorithm. Use Union-Find algorithms in your program.
6. Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.
7. Write Java programs to

(a) Implement All-Pairs Shortest Paths problem using Floyd's algorithm.

(b) Implement Travelling Sales Person problem using Dynamic programming.

1. 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.
2. Design and implement in Java to find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle.

**Note: All laboratory experiments (Twelve problems) are to be included for practical examination. Students are allowed to pick one experiment from the lot. To generate the data set use random number generator function. Strictly follow the instructions as printed on the cover page of answer script for breakup of marks Marks distribution: Procedure + Conduction + Viva: 20 + 50 + 10 (80). Change of experiment is allowed only once and marks allotted to the procedure**

**1 a. Create a Java class called Student with the following details as variables within it. (i) USN (ii) Name (iii) Branch (iv) Phone Write a Java program to create n Student objects and print the USN, Name, Branch, and Phone of these objects with suitable headings.**

**import** java.util.Scanner;

**class** Student

{

String usn, name,branch,phone;

**void** read()

{

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

System.***out***.println("Enter Student USN");

usn=sobj.next();

System.***out***.println("Enter Student Name:");

name=sobj.next();

System.***out***.println("Enter Student branch:");

branch=sobj.next();

System.***out***.println("Enter Student phone no:");

phone=sobj.next();

}

**void** display()

{

System.***out***.println(usn+"..."+name+"..." +branch+"..."+phone);

}

}

**public** **class** Pgm1a {

**public** **static** **void** main(String[] args)

{

**int** n;

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

System.***out***.println("Enter the Number of Students");

n=sobj.nextInt();

//array of objects

Student[] st=**new** Student[n];

System.***out***.println("Please enter the student details");

//creation of n objects

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

{

st[i]=**new** Student();

}

//read student data

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

{

st[i].read();

}

//print student data

System.***out***.println("USN | Name || USN | Name");

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

{

st[i].display();

}

}

}

**OUTPUT**

Enter the Number of Students

2

Please enter the student details

Enter Student USN

is01

Enter Student Name:

abhi

Enter Student branch:

ise

Enter Student phone no:

9876543210

Enter Student USN

is02

Enter Student Name:

manu

Enter Student branch:

ise

Enter Student phone no:

9098765432

USN | Name || USN | Name

is01...abhi...ise...9876543210

is02...manu...ise...9098765432

**1 b. Write a Java program to implement the Stack using arrays. Write Push(), Pop(), and Display() methods to demonstrate its working.**

**import** java.util.Scanner;

**class** Stack

{

**final** **int** size=5;

**int** arr[] = **new** **int**[size];

**int** top = -1;

**public** **void** push(**int** item)

{

**if**(top < size-1)

{

top++;

arr[top]=item;

System.***out***.println("The " + item + "is pushed into the stack");

}

**else**

{

System.***out***.println("Error !Stack Overflow ");

}

}

**public** **void** pop()

{

**if**(top==-1)

{

System.***out***.println("error stack underflow");

}

**else**

{

**int** item;

item =arr[top];

System.***out***.println("The " + arr[top] + " is poped out of the stack");

top--;

}

}

**public** **void** display()

{

**if**(top==-1)

{

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

}

**else**

{

System.***out***.println("Elements in stack ");

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

{

System.***out***.println(arr[i]);

}

}

}

}

**public** **class** Stack\_Demo

{

**public** **static** **void** main(String[] args)

{

Stack s= **new** Stack();

**int** x;

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

**int** ch;

System.***out***.println("press 1 to push element");

System.***out***.println("press 2 to pop element");

System.***out***.println("press 3 to display elements");

System.***out***.println("press 4 to exit ");

**do**

{

System.***out***.println("Enter your choice: ");

ch=sobj.nextInt();

**switch**(ch)

{

**case** 1:

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

x=sobj.nextInt();

s.push(x);

**break**;

**case** 2:

s.pop();

**break**;

**case** 3:

s.display();

**break**;

**default**: System.***out***.println("Invalid Choice ");

**break**;

}

}**while** (ch!=4);

}

}

**OUTPUT**

press 1 to push element

press 2 to pop element

press 3 to display elements

press 4 to exit

Enter your choice:

1

Enter element:

2

The 2is pushed into the stack

Enter your choice:

1

Enter element:

3

The 3is pushed into the stack

Enter your choice:

1

Enter element:

3

The 3is pushed into the stack

Enter your choice:

3

Elements in stack

2

3

3

Enter your choice:

1

Enter element:

4

The 4is pushed into the stack

Enter your choice:

1

Enter element:

5

The 5is pushed into the stack

Enter your choice:

1

Enter element:

6

Error !Stack Overflow

Enter your choice:

2

The 5 is poped out of the stack

Enter your choice:

2

The 4 is poped out of the stack

Enter your choice:

2

The 3 is poped out of the stack

Enter your choice:

2

The 3 is poped out of the stack

Enter your choice:

2

The 2 is poped out of the stack

Enter your choice:

2

error stack underflow

Enter your choice:

2

error stack underflow

Enter your choice:

3

Stack Empty

Enter your choice:

4

Invalid Choice

**2 a. Design a super class called Staff with details as StaffId, Name, Phone, Salary. Extend this class by writing three subclasses namely Teaching (domain, publications), Technical (skills), and Contract (period). Write a Java program to read and display at least 3 staff objects of all three categories.**

**import** java.util.Scanner;

**class** Staff

{

**int** staffid,salary;

String name, phone;

Staff(**int** staffid, **int** salary,String name,String phone )

{

**this**.staffid=staffid;

**this**.salary=salary;

**this**.name=name;

**this**.phone=phone;

}

**void** display()

{

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

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

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

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

}

}

**class** Teaching **extends** Staff

{

String domain, publication;

Teaching(**int** staffid,**int** salary,String name,String phone,String dom,String pub)

{

**super**(staffid,salary,name,phone);

domain=dom;

publication=pub;

}

**void** displayTeach()

{

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

System.***out***.println("Publication:"+publication);

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

}

}

**class** Technical **extends** Staff

{

String skills;

Technical(**int** staffid,**int** salary,String name,String phone,String skill)

{

**super**(staffid,salary,name,phone);

skills=skill;

}

**void** displayTechnical()

{

System.***out***.println("Skils:"+skills);

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

}

}

**class** Contract **extends** Staff

{

**int** period;

Contract(**int** staffid,**int** salary,String name,String phone,**int** per)

{

**super**(staffid,salary,name,phone);

period=per;

}

**public** **void** displayCont()

{

System.***out***.println("Period:"+period);

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

}

}

**public** **class** Pgm2a

{

**public** **static** **void** main(String[] args)

{

**int** ch, sid,salary,period;

String name,phone,domain,publication,skill;

System.***out***.println("Enter your category:1. Teaching, 2. Technical. 3.Contract");

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

ch=sobj.nextInt();

**switch**(ch)

{

**case** 1: Teaching[] te=**new** Teaching[3];

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

{

System.***out***.println("Enter SID,Salary,Name,Phone,domain and Publications");

sid=sobj.nextInt();

salary=sobj.nextInt();

name=sobj.next();

phone=sobj.next();

domain=sobj.next();

publication=sobj.next();

te[i]=**new** Teaching(sid,salary,phone,name,domain,publication);

}

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

{

te[i].display();

te[i].displayTeach();

}

**break**;

**case** 2: Technical[] tech=**new** Technical[3];

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

{

System.***out***.println("Enter SID,Salary,Name,Phone and Skills");

sid=sobj.nextInt();

salary=sobj.nextInt();

name=sobj.next();

phone=sobj.next();

skill=sobj.next();

tech[i]=**new**

Technical(sid,salary,phone,name,skill);

}

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

{

tech[i].display();

tech[i].displayTechnical();

}

**break**;

**case** 3: Contract[] con=**new** Contract[3];

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

{

System.***out***.println("Enter SID,Salary,Name,Phone and period");

sid=sobj.nextInt();

salary=sobj.nextInt();

name=sobj.next();

phone=sobj.next();

period=sobj.nextInt();

con[i]=**new** Contract(sid,salary,phone,name,period);

}

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

{

con[i].display();

con[i].displayCont();

}

**break**;

**default**: System.***out***.println("Invalid option");

}

}

}

**OUTPUT 1:**

Enter your category:1. Teaching, 2. Technical. 3.Contract

1

Enter SID,Salary,Name,Phone,domain and Publications

1

10000

abhi

9090909875

networks

ieee

Enter SID,Salary,Name,Phone,domain and Publications

2

11000

ramesh

8909890987

datamining

ijret

Enter SID,Salary,Name,Phone,domain and Publications

3

12000

suma

7898767899

programming

iccstar

Staff ID:1

Salary:10000

Name:9090909875

Phone:abhi

Domain:networks

Publication:ieee

-----------------------------

Staff ID:2

Salary:11000

Name:8909890987

Phone:ramesh

Domain:datamining

Publication:ijret

-----------------------------

Staff ID:3

Salary:12000

Name:7898767899

Phone:suma

Domain:programming

Publication:iccstar

-----------------------------

**OUTPUT 2:**

Enter your category:1. Teaching, 2. Technical. 3.Contract

3

Enter SID,Salary,Name,Phone and period

1

10000

rani

9876567898

2

Enter SID,Salary,Name,Phone and period

2

11000

rama

9870987890

3

Enter SID,Salary,Name,Phone and period

3

12000

raghava

9098767876

4

Staff ID:1

Salary:10000

Name:9876567898

Phone:rani

Period:2

-----------------------------

Staff ID:2

Salary:11000

Name:9870987890

Phone:rama

Period:3

-----------------------------

Staff ID:3

Salary:12000

Name:9098767876

Phone:raghava

Period:4

**OUTPUT 3:**

Enter your category:1. Teaching, 2. Technical. 3.Contract

2

Enter SID,Salary,Name,Phone and Skills

1

10000

ram

9876554322

java

Enter SID,Salary,Name,Phone and Skills

2

11000

yash

9870987655

c++

Enter SID,Salary,Name,Phone and Skills

3

12000

mary

7898765432

python

Staff ID:1

Salary:10000

Name:9876554322

Phone:ram

Skils:java

-----------------------------

Staff ID:2

Salary:11000

Name:9870987655

Phone:yash

Skils:c++

-----------------------------

Staff ID:3

Salary:12000

Name:7898765432

Phone:mary

Skils:python

-----------------------------

**2 b. Write a Java class called Customer to store their name and date\_of\_birth. The date\_of\_birth format should be dd/mm/yyyy. Write methods to read customer data as and display as using StringTokenizer class considering the delimiter character as “/”.**

import java.util.Scanner;

import java.util.StringTokenizer;

class Customer

{

String cname,dob;

Scanner sobj=new Scanner(System.in);

void read()

{

System.out.println("Enter Customer name:");

cname=sobj.next();

System.out.println("Enter Customer DOB in the format dd/mm/yyy");

dob=sobj.next();

}

void display()

{

StringTokenizer st = new StringTokenizer(dob, "/");

System.out.print(cname+",");

while(st.hasMoreTokens())

{

String val = st.nextToken();

System.out.print(val);

if(st.countTokens()!=0)

System.out.print(","+" ");

}

}

}

public class Pgm2b {

public static void main(String[] args) {

Customer cobj=new Customer();

cobj.read();

System.out.println("Customer Details");

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

cobj.display();

}

}

**OUTPUT**

Enter Customer name:

Abhinava

Enter Customer DOB in the format dd/mm/yyy

01/07/1992

Customer Details

---------------------

Abhinava,01, 07, 1992

**3 a. Write a Java program to read two integers a and b. Compute a/b and print, when b is not zero. Raise an exception when b is equal to zero.**

**import** java.util.Scanner;

**public** **class** Exception\_Divide

{

**public** **static** **void** main(String[] args)

{

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

System.***out***.print("Please enter first number(numerator): ");

**int** numerator = inputDevice.nextInt();

System.***out***.print("Please enter second number(denominator): ");

**int** denominator = inputDevice.nextInt();

**try** {

**new** Exception\_Divide().doDivide(numerator, denominator);

}

**catch** (Exception e)

{

System.***out***.println("Exception Condition Program is ending ");

}

}

**public** **void** doDivide(**int** a, **int** b) **throws** Exception

{

**float** result = a/b;

System.***out***.println("Division result of "+ a +"/"+b +"= " +result);

}

}

**OUTPUT 1:**

Please enter first number(numerator): 10

Please enter second number(denominator): 2

Division result of 10/2= 5.0

**OUTPUT 2:**

Please enter first number(numerator): 10

Please enter second number(denominator): 0

Exception Condition Program is ending

**3. b Write a Java program that implements a multi-thread application that hashtree threads. First thread generates a random integer for every 1 second; second thread computes the square of the number and prints; third thread will print the value of cube of the number.**

import java.util.Random;

import java.util.Scanner;

class first extends Thread

{

public void run()

{

int num=0;

Random r=new Random();

try

{

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

{

num=r.nextInt(10);

System.out.println("first thread generated num is ="+num);

Thread t2 = new Thread(new second(num));

t2.start();

Thread.sleep(1000);

Thread t3 = new Thread(new third(num));

t3.start();

Thread.sleep(1000);

}

}

catch(Exception e)

{

System.out.println(e.getMessage());

System.out.println(" ");

}

}

}

class second implements Runnable

{

int x;

public second(int x)

{

this.x=x;

}

public void run()

{

System.out.println("Second thread: Square of 2 num is"+x\*x);

}

}

class third implements Runnable

{

public int x;

public third(int x)

{

this.x=x;

}

public void run()

{

System.out.println("Third thread: Cube of num is"+x\*x\*x);

}

}

public class Multithreading {

public static void main(String[] args)

{

first a=new first();

a.start();

}

}

**OUTPUT**

first thread generated num is =3

Second thread: Square of 2 num is 9

Third thread: Cube of num is 27

first thread generated num is =5

Second thread: Square of 2 num is 25

Third thread: Cube of num is125

first thread generated num is =6

Second thread: Square of 2 num is 36

Third thread: Cube of num is216

first thread generated num is =4

Second thread: Square of 2 num is 16

Third thread: Cube of num is64

first thread generated num is =8

Second thread: Square of 2 num is 64

Third thread: Cube of num is512

first thread generated num is =2

Second thread: Square of 2 num is 4

Third thread: Cube of num is 8

first thread generated num is =1

Second thread: Square of 2 num is 1

Third thread: Cube of num is 1

first thread generated num is =3

Second thread: Square of 2 num is 9

Third thread: Cube of num is 27

first thread generated num is =6

Second thread: Square of 2 num is 36

Third thread: Cube of num is 216

first thread generated num is =9

Second thread: Square of 2 num is 81

Third thread: Cube of num is 729

first thread generated num is =6

Second thread: Square of 2 num is 36

Third thread: Cube of num is 216

**4 Sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n > 5000, and record the time taken to sort. Plot a graph of the time taken versus n on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.**

import java.util.Random;

import java.util.Scanner;

public class merge\_sort

{

static int max=30000;

public static void main(String[] args)

{

int a[]=new int[max];

long start,end;

Scanner sobj=new Scanner (System.in);

System.out.println("\*\*\*\*\*\*\*\*MERGE SORT ALGORITHM\*\*\*\*\*\*\*\*");

System.out.println("Enter the no. of elements to be sorted :");

int n=sobj.nextInt();

Random rand=new Random();

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

{

a[i]=rand.nextInt(100);

}

System.out.println("Array elements to be sorted are :");

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

{

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

}

start=System.nanoTime();

mergesort(a,0,n-1);

end=System.nanoTime();

System.out.println("\nThe sorted elements are :");

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

{

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

}

System.out.println("\nThe time taken to sort is "+(end-start)+"ns");

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

}//end of main

static void mergesort(int a[],int low,int high)

{

int mid;

if(low<high)

{

mid=(low+high)/2;

mergesort(a,low,mid);//recursively sort left part

mergesort(a,mid+1,high);//recursively sort right part

merge(a,low,mid,high); // merge two sorted parts

}

}

static void merge(int a[],int low,int mid,int high)

{

int i,j,h,k;

int b[]=new int[max];

h=low;//h points to first element in first half a[low:mid]

i=low;

j=mid+1;//j points to first element in second half a[mid+1:high]

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

{

if(a[h]<a[j])

{

b[i]=a[h];

h=h+1;

}

else

{

b[i]=a[j];

j=j+1;

}

i=i+1;

}

if(h>mid)

{

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

{

b[i]=a[k];

i=i+1;

}

}

else

{

for(k=h;k<=mid;k++)

{

b[i]=a[k];

i=i+1;

}

}

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

a[k]=b[k];

}//end of merge

}//end of class merge

**OUTPUT 1:**

\*\*\*\*\*\*\*\*MERGE SORT ALGORITHM\*\*\*\*\*\*\*\*

Enter the no. of elements to be sorted :

5

Array elements to be sorted are :

52 59 15 12 37

The sorted elements are :

12 15 37 52 59

The time taken to sort is 445502ns

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**OUTPUT 2:**

\*\*\*\*\*\*\*\*MERGE SORT ALGORITHM\*\*\*\*\*\*\*\*

Enter the no. of elements to be sorted :

10

Array elements to be sorted are :

17 92 59 22 36 8 62 27 95 47

The sorted elements are :

8 17 22 27 36 47 59 62 92 95

The time taken to sort is 870356ns

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Table that accounts the values for no of elements and time taken to sort**

|  |  |  |
| --- | --- | --- |
| **Sl No** | **No of elements(n)** | **Time Taken in ns(T(n))** |
| 1 | 1000 | 96716619 |
| 2 | 5000 | 465834410 |
| 3 | 10000 | 931645022 |
| 4 | 15000 | 1379909926 |
| 5 | 20000 | 1868495032 |

**5. Sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n > 5000 and record the time taken to sort. Plot a graph of the time taken versus n on graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using Java how the divideand-conquer method works along with its time complexity analysis: worst case, average case and best case**

import java.util.Random;

import java.util.Scanner;

public class quicksort

{

static int max=30000;

public static void main(String[] args)

{

int a[]=new int[max];

long start,end;

Scanner sobj=new Scanner (System.in);

System.out.println("\*\*\*\*\*\*\*\*QUICK SORT ALGORITHM\*\*\*\*\*\*\*\*");

System.out.println("Enter the no. of elements to be sorted :");

int n=sobj.nextInt();

Random rand=new Random();

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

{

a[i]=rand.nextInt(100);

}

System.out.println("Array elements to be sorted are :");

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

{

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

}

a[n]=9999;

start=System.nanoTime();

qsort(a,0,n-1);

end=System.nanoTime();

System.out.println("\nThe sorted elements are :");

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

{

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

}

System.out.println("\nThe time taken to sort is "+(end-start)+"ns");

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

}//end of main

static void qsort(int a[],int low,int high)

{

int s;

if(low<high)

{

s=partition(a,low,high);//s is the final position of pivot element in

a[low:high]

qsort(a,low,s-1);

qsort(a,s+1,high);

}

}

static int partition(int a[],int low,int high)

{

int pivot,i,j;

pivot=a[low];

i=low;

j=high;

while(i<=j)

{

while(a[i]<=pivot)

i++;

while(a[j]>pivot)

j--;

if(i<j)

swap(a,i,j);

}

a[low]=a[j];

a[j]=pivot;

return j;

}

static void swap(int a[],int i,int j)

{

int temp;

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

**OUTPUT 1:**

\*\*\*\*\*\*\*\*QUICK SORT ALGORITHM\*\*\*\*\*\*\*\*

Enter the no. of elements to be sorted:

5

Array elements to be sorted are:

96 71 67 52 82

The sorted elements are:

52 67 71 82 96

The time taken to sort is 11899ns

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**OUTPUT 2:**

\*\*\*\*\*\*\*\*QUICK SORT ALGORITHM\*\*\*\*\*\*\*\*

Enter the no. of elements to be sorted:

10

Array elements to be sorted are:

96 9 33 35 16 38 37 7 66 98

The sorted elements are:

7 9 16 33 35 37 38 66 96 98

The time taken to sort is 16098ns

**Table that accounts the values for no of elements and time taken to sort**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **No. of elements(n)** | **Time Taken in ns(T(n))** |
| 1 | 1000 | 609634 |
| 2 | 5000 | 990393 |
| 3 | 10000 | 2115521 |
| 4 | 15000 | 4135854 |
| 5 | 20000 | 12681933 |

**6 a) Implement in Java, the 0/1 Knapsack problem using Dynamic Programming method**

import java.util.Scanner;

public class knapsackDP

{

public void solve(int[] wt, int[] val, int W, int N)

{

int i,j;

int sol[][] = new int[N + 1][W + 1];

int selected[] = new int[N+1];

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

{

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

{

if(i==0||j==0)

sol[i][j]=0;

else if(wt[i]>j)

sol[i][j]=sol[i-1][j];

else

sol[i][j]=Math.*max*(sol[i-1][j],(sol[i - 1][j - wt[i]] + val[i]));

}

}

System.*out*.println("The profit table is:: ");

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

{

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

System.*out*.print(sol[i][j]+" ");

System.*out*.println();

}

System.*out*.println("The optimal profit obtained is "+sol[N][W]);

i=N;

j=W;

while (i>0&&j>0)

{

if (sol[i][j] !=sol[i-1][j])

{

selected[i] = 1;

j = j - wt[i];

}

i--;

}

System.*out*.println("\nItems selected : ");

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

if (selected[i] == 1)

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

System.*out*.println();

}

public static void main(String[] args)

{

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

knapsackDP ks=new knapsackDP();

System.*out*.println("\*\*\*\*\* KNAPSACK PROBLEM - DYNAMIC PROGRAMMING \*\*\*\*\*");

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

int n = scan.nextInt();

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

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

System.*out*.println("\nEnter weight for "+ n +" elements");

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

wt[i] = scan.nextInt();

System.*out*.println("\nEnter profit value for "+ n +" elements");

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

val[i] = scan.nextInt();

System.*out*.println("\nEnter knapsack weight ");

int W = scan.nextInt();

ks.solve(wt, val, W, n);

}

}

**OUTPUT 1:**

\*\*\*\*\* KNAPSACK PROBLEM - DYNAMIC PROGRAMMING \*\*\*\*\*

Enter number of elements

4

Enter weight for 4 elements

2

1

2

1

Enter profit value for 4 elements

12

10

20

15

Enter knapsack weight

5

The profit table is::

0 0 0 0 0 0

0 0 12 12 12 12

0 10 12 22 22 22

0 10 20 30 32 42

0 15 25 35 45 47

The optimal profit obtained is 47

Items selected :

1 3 4

**6 b) Implement in Java, the 0/1 Knapsack problem using Greedy method.**

import java.util.Scanner;

public class FractionalKnapsack

{

double weight[];

double profit[];

double ratio[];

double cap;

int nItems;

FractionalKnapsack()

{

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

System.*out*.println("\*\*\*\*\*\*\*\*\* KNAPSACK PROBLEM-GREEDY METHOD \*\*\*\*\*\*\*\*\*");

System.*out*.println("Enter the number of items in the store: ");

nItems = scan.nextInt();

System.*out*.println("Enter the (weight and profit) of items: ");

weight = new double[nItems];

profit = new double[nItems];

ratio = new double[nItems];

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

weight[i] = scan.nextDouble();

profit[i] = scan.nextDouble();

ratio[i] = profit[i] / weight[i];

}

System.*out*.println("Enter the capacity of the knapsack: ");

cap = scan.nextDouble();

}

int getNext()

{

double max = 0;

int index = -1;

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

{

if (ratio[i] > max)

{

max = ratio[i];

index = i;

}

}

return index;

}

void fill()

{

double cW = 0; //current weight

double cP = 0; //current profit

double select[]=new double[nItems];//marking item selection

while (cW < cap)

{

int item = getNext(); //next max ratio

if (item == -1)

{

//No items left

break;

}

if (cW + weight[item] <= cap)

{

cW += weight[item];

cP += profit[item];

ratio[item] = 0; //mark as used for the getNext() (ratio) function

select[item]=1;

}

else

{

select[item]=(cap-cW)/weight[item];

cP += (ratio[item] \* (cap - cW));

cW += (cap - cW);

break; //the knapsack is full

}

}

System.*out*.println("\nItems Selected Fraction Selected(0/1/Partial) ");

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

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

System.*out*.println("\t"+(i+1)+"\t\t"+select[i]);

System.*out*.println("\nMax Profit = " + cP + ", Max Weight = " + cW);

}

public static void main(String[] args) {

FractionalKnapsack fk = new FractionalKnapsack();

fk.fill();

}

}

**OUTPUT 1:**

\*\*\*\*\*\*\*\*\* KNAPSACK PROBLEM-GREEDY METHOD \*\*\*\*\*\*\*\*\*

Enter the number of items in the store:

3

Enter the (weight and profit) of items:

18 25

15 24

10 15

Enter the capacity of the knapsack:

20

Items Selected Fraction Selected(0/1)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1 0.0

2 1.0

3 0.0

Max Profit = 24.0, Max Weight = 15.0

**OUTPUT 2:**

\*\*\*\*\*\*\*\*\* KNAPSACK PROBLEM-GREEDY METHOD \*\*\*\*\*\*\*\*\*

Enter the number of items in the store:

3

Enter the (weight and profit) of items:

18 25

15 24

10 15

Enter the capacity of the knapsack:

20

Items Selected Fraction Selected(0/1/Partial)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1 0.0

2 1.0

3 0.5

Max Profit = 31.5, Max Weight = 20.0

**7. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in Java.**

import java.util.Scanner;

class Dijkstra

{

int n,src;

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

void read\_cost\_adjacency\_matrix()

{

System.*out*.println("\*\*\*\*\*\*\*\*\* DIJKSTRA'S ALGORITHM \*\*\*\*\*\*\*\*\*");

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

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

n=sobj.nextInt();

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

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

{

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

{

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

}

}

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

src=sobj.nextInt();

sobj.close();

}

void find\_short\_distance\_path()

{

int i,j,v,min,u=0;

int d[]=new int[10];

int p[]=new int[10];

int s[]=new int[10];

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

{

d[i]=a[src][i];

p[i]=src;

s[i]=0;

}

s[src]=1;

d[src]=0;

//find shortest distance & the path to other vertices

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

{

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

{

if(s[j]==0 && d[j]<min)

{

min=d[j];

u=j;

}

}//end of j for loop

s[u]=1;

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

{

if(s[v]==0 && d[u]+a[u][v]<d[v])

{

d[v]=d[u]+a[u][v];

p[v]=u;

}

}//end of v for loop

}//end of i for loop

System.*out*.println("The shortest path and distance is shown below:");

System.*out*.println("DEST VERTEX<-(Intermediate vertices)<-SOURCE=DISTANCE");

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

{

if(d[j]==999)

System.*out*.println(j+"<-"+src+"="+d[j]);

else if(d[j]==0)

System.*out*.println(j+"<-"+src+"="+d[j]);

else

{

i=j;

while(i!=src)

{

System.*out*.print(i+"<-");

i=p[i];

}

System.*out*.println(i+"="+d[j]);

}

}//end of j for loop

}

}

public class Shortest\_path\_dijkstra

{

public static void main(String[] args)

{

Dijkstra ob=new Dijkstra();

ob.read\_cost\_adjacency\_matrix();

ob.find\_short\_distance\_path();

}

}

**OUTPUT 1:**

\*\*\*\*\*\*\*\*\* DIJKSTRA'S ALGORITHM \*\*\*\*\*\*\*\*\*

Enter no. of nodes :

6

Enter cost adjacency matrix :

0 15 10 999 45 999

999 0 15 999 20 999

20 999 0 20 999 999

999 10 999 0 35 999

999 999 999 30 0 999

999 999 999 4 999 0

Enter source vertex :

1

The shortest path and distance is shown below:

DEST VERTEX<-(Intermediate vertices)<-SOURCE=DISTANCE

1<-1=0

2<-1=15

3<-1=10

4<-3<-1=30

5<-2<-1=35

6<-1=999

**OUTPUT 2:**

\*\*\*\*\*\*\*\*\* DIJKSTRA'S ALGORITHM \*\*\*\*\*\*\*\*\*

Enter no. of nodes :

5

Enter cost adjacency matrix :

0 3 999 7 999

3 0 4 2 999

999 4 0 5 6

7 2 5 0 4

999 999 6 4 0

Enter source vertex :

1

The shortest path and distance is shown below:

DEST VERTEX<-(Intermediate vertices)<-SOURCE=DISTANCE

1<-1=0

2<-1=3

3<-2<-1=7

4<-2<-1=5

5<-4<-2<-1=9

**8. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm. Implement the program in Java language.**

import java.util.Scanner;

class Kruskal\_algo

{

int n;

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

void read\_cost\_adjacency\_matrix()

{

System.*out*.println("\*\*\*\*\*\*\*\*\* KRUSKAL'S ALGORITHM \*\*\*\*\*\*\*\*\*");

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

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

n=scan.nextInt();

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

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

{

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

{

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

}

}

scan.close();

}

void find\_minimum\_spanningtree()

{

int parent[]=new int[10];

int t[][]=new int[10][3];

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

{

parent[i]=i;

}

int count=0,sum=0, k=0,u=0,v=0;

while(count!=n-1)

{

int min=999;

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

{

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

{

if(a[i][j]!=0 && a[i][j]<min)

{

min=a[i][j];

u=i;

v=j;

}

}

}

if(min==999)

break;

int i=u;

while(parent[i]!=i)

i=parent[i];

int j=v;

while(parent[j]!=j)

j=parent[j];

if(i!=j)

{

t[k][0]=u;

t[k][1]=v;

t[k][2]=min;

k++;

sum=sum+min;

parent[j]=i;

count++;

}

a[u][v]=a[v][u]=999;

}

if(count==n-1)

{

System.*out*.println("The min cost spanning tree with edges is");

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

System.*out*.println("Edge"+"\t"+"Weight");

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

for(int i=0;i<n-1;i++)

System.*out*.println(t[i][0]+"->"+t[i][1]+"\t"+t[i][2]);

System.*out*.println("Cost of the Spanning tree="+sum);

}

else

System.*out*.println("Spanning tree does not exist");

}

}

public class kruskal

{

public static void main(String[] args)

{

Kruskal\_algo k=new Kruskal\_algo();

k.read\_cost\_adjacency\_matrix();

k.find\_minimum\_spanningtree();

}

}

**OUTPUT 1:**

\*\*\*\*\*\*\*\*\* KRUSKAL'S ALGORITHM \*\*\*\*\*\*\*\*\*

Enter number of nodes

4

Enter the cost adjacency matrix

0 1 5 2

1 0 999 999

5 999 0 3

2 999 3 0

The min cost spanning tree with edges is

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Edge Weight

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1->2 1

1->4 2

3->4 3

Cost of the Spanning tree=6

**OUTPUT 2:**

\*\*\*\*\*\*\*\*\* KRUSKAL'S ALGORITHM \*\*\*\*\*\*\*\*\*

Enter number of nodes

4

Enter the cost adjacency matrix

0 1 999 999

1 0 2 999

999 2 0 999

999 999 999 0

Spanning tree does not exist

**9. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm. Implement the program in Java language.**

import java.util.Scanner;

class Prims\_algo

{

int n;

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

void read\_adjacency\_matrix()

{

System.*out*.println("\*\*\*\*\*\*\*\*\* PRIMS ALGORITHM \*\*\*\*\*\*\*\*\*");

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

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

n=scan.nextInt();

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

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

{

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

{

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

}

}

scan.close();

}

void find\_minimum\_spanning\_tree()

{

int min,u=0,v=0,k=0,count=0,cost=0,i,j;

int visited[]=new int[20];

int t[][]=new int[20][3];

visited[1]=1;

while(count!=(n-1))

{

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

{

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

{

if(visited[i]==1 && visited[j]==0 && min > a[i][j])

{

min=a[i][j];

u=i;

v=j;

}

}

}

if(min==999)

break;

t[k][0]=u;

t[k][1]=v;

t[k][2]=min;

visited[v]=1;

cost+=min;

k++;

count++;

}//end of while

if(count==n-1)

{

System.*out*.println("The min cost spanning tree with edges is");

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

System.*out*.println("Edge"+"\t"+"Weight");

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

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

System.*out*.println(t[i][0]+"->"+t[i][1]+"\t"+t[i][2]);

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

System.*out*.println("cost of spanning tree="+cost);

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

}

else

System.*out*.println("spanning tree does not exist");

}

}

public class prim

{

public static void main(String[] args)

{

Prims\_algo p=new Prims\_algo();

p.read\_adjacency\_matrix();

p.find\_minimum\_spanning\_tree();

}

}

**OUTPUT 1:**

\*\*\*\*\*\*\*\*\* PRIMS ALGORITHM \*\*\*\*\*\*\*\*\*

Enter number of nodes

4

Enter the cost adacency matrix

0 1 5 2

1 0 999 999

5 999 0 3

2 999 3 0

The min cost spanning tree with edges is

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Edge Weight

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1->2 1

1->4 2

4->3 3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

cost of spanning tree=6

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**OUTPUT 2:**

\*\*\*\*\*\*\*\*\* PRIMS ALGORITHM \*\*\*\*\*\*\*\*\*

Enter number of nodes

4

Enter the cost adacency matrix

0 1 999 999

1 0 2 999

999 2 0 999

999 999 999 0

spanning tree does not exist

**10 a. Write Java program to Implement All-Pairs Shortest Paths problem using Floyd's algorithm.**

import java.lang.\*;

import java.util.Scanner;

class Floyd

{

int d[][]=new int[10][10];

public void dis\_path(int n, int a[][])

{

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

{

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

{

d[i][j]=a[i][j];

}

}

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

{

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

{

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

{

d[i][j]=Math.min(d[i][j],(d[i][k]+d[k][j]));

}

}

}

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

{

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

{

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

}

System.out.println();

}

}

}

public class Floyed {

public static void main(String[] args)

{

int n;

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

Scanner sobj=new Scanner(System.in);

Floyd f=new Floyd();

System.out.println("\*\*\*\*\* FLOYD'S ALGORITHM \*\*\*\*\*");

System.out.println("ENTER THE NUMBER OF NODES:");

n=sobj.nextInt();

System.out.println("ENTER THE COST ADJECENCY MATRIX:");

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

{

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

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

}

System.out.println("RESULTANT SHORTEST PATH MATRIX IS:");

f.dis\_path(n,a);

sobj.close();

}

}

**OUTPUT:**

\*\*\*\*\* FLOYD'S ALGORITHM \*\*\*\*\*

ENTER THE NUMBER OF NODES:

4

ENTER THE COST ADJECENCY MATRIX:

0 999 3 999

2 0 999 999

999 7 0 1

6 999 999 0

RESULTANT SHORTEST PATH MATRIX IS:

0 10 3 4

2 0 5 6

7 7 0 1

6 16 9 0

**10 b. Travelling Sales Person problem using Dynamic programming:**

import java.util.Scanner;

public class TSP

{

public static void main(String[] args)

{

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

Scanner in = new Scanner(System.in);

int i, j,cost;

System.out.println("\*\*\*\* TSP DYNAMIC PROGRAMMING \*\*\*\*\*\*\*");

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

int n = in.nextInt();

if(n==1)

{

System.out.println("Path is 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 = tspdp(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 mincost is "+cost);

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

}

static int tspdp(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=tspdp(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 :**

\*\*\*\*\*\*\* TSP DYNAMIC PROGRAMMING \*\*\*\*\*\*\*

Enter the number of cities: 4

Enter the cost matrix

0 1 3 6

1 0 2 3

3 2 0 1

6 3 1 0

The entered cost matrix is

0 1 3 6

1 0 2 3

3 2 0 1

6 3 1 0

The accurate path is 1->2->4->3->1

The accurate mincost is 8

\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**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.**

**import** java.util.Scanner;

**public** **class** subset

{

**static** **int** *c*=0;

**static** **int** *w*[]=**new** **int**[10];

**static** **int** *x*[]=**new** **int**[10];

**static** **int** *n*,*d*,*i*,*sum*=0;

**public** **static** **void** main(String[] args)

{

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

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

*n*=in.nextInt();

System.***out***.println("Enter the elements in increasing order:");

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

*w*[*i*]=in.nextInt();

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

*d*=in.nextInt();

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

*sum*=*sum*+*w*[*i*];

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

**if**(*sum*<*d* || *w*[0]>*d*)

{

System.***out***.println("Subset is not possible!");

System.*exit*(0);

}

*subset*(0,0,*sum*);

**if**(*c*==0)

System.***out***.println("Subset is not possible!");

}

**static** **void** subset(**int** wsf,**int** k,**int** trw )

{

**int** i;

*x*[k]=1;

**if**(wsf+*w*[k]==*d*)

{

System.***out***.println("Subset solution="+(++*c*));

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

{

**if**(*x*[i]==1)

System.***out***.println(*w*[i]);

}

**return**;

}

**if**(wsf+*w*[k]+*w*[k+1]<=*d*)

*subset*(wsf+*w*[k],k+1,trw-*w*[k]);

**if**((wsf+trw-*w*[k]>=*d*) && (wsf+*w*[k+1]<=*d*))

{

*x*[k]=0;

*subset*(wsf,k+1,trw-*w*[k]);

}

}

}

**OUTPUT**

Enter number of elements:

5

Enter the elements in increasing order:

1 2 3 4 5

Enter the value of d:

6

SUM=15

Subset solution=1

1

2

3

Subset solution=2

1

5

Subset solution=3

2

4

**12 .Design and implement the presence of Hamiltonian Cycle in an undirected Graph G of *n* vertices.**

**import** java.util.\*;

**class** Hamiltoniancycle

{

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

**int** x[]=new int[10];

**int** n;

**public** Hamiltoniancycle()

{

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

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

n=src.nextInt();

x[1]=1; // **the cycle starts from vertex 1**

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

x[i]=0;

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

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

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

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

}

**public void** nextVertex (**int** k)

{

**int** j=1;

**while**(**true**)

{

x[k]=(x[k]+1)%(n+1);

**if** (x[k]==0)

**return**;

**if** (a[x[k-1]][x[k]]==1) **//there exists a edge between k and (k-1) vertex**

**for** (j=1; j<k; j++) **// check whether the vertices are distinct**

**if** (x[j]==x[k])

**break**;

**if** (j==k) **// if there is distinction**

**if** (k<n || (k==n &&a[x[n]][1]==1))

**return**;

}**//end of while**

}**// end of the method**

**public void** getHCycle(**int** k)

{

**while**(**true**)

{

nextVertex(k);

**if** (x[k]==0)

**return**;

**if** (k==n)

{

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

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

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

System.*out*.println(1);

}

**else**

getHCycle(k+1);

}

}**//end of while**

}**//end of class Hamiltoniancycle**

**public class** HamiltoniancycleExp

{

**public static void** main(String args[])

{

Hamiltoniancycle obj=**new** Hamiltoniancycle(); obj.getHCycle(2);

}

}

**OUTPUT**

Enter the number of nodes

6

Enter the adjacency matrix

0 1 1 1 0 0

1 0 1 0 0 1

1 1 0 1 1 0

1 0 1 0 1 0

0 0 1 1 0 1

0 1 0 0 1 0

Solution:

1 2 6 5 3 4 1

Solution:

1 2 6 5 4 3 1

Solution:

1 3 2 6 5 4 1

Solution:

1 3 4 5 6 2 1

Solution:

1 4 3 5 6 2 1

Solution:

1 4 5 6 2 3 1