**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**BELGAVI**



**DESIGN AND ANALYSIS OF ALGORITHMS**

**LABORATORY (18CSL47)**

(As per Visvesvaraya Technological University Syllabus)

**Complied By:**

**Prof. Hemanth T. D.** **Prof. Swetha K. H.**

Assistant Professor, Dept. of ISE Assistant Professor, Dept. of ISE



**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**

**ACHARYA INSTITUTE OF TECHNOLOGY**

(Affiliated to VTU, Belgavi, Approved by AICTE, New Delhi and Accredited by NAAC, New Delhi)

Acharya Dr. Sarvepalli Radhakrishnan Road, Bangalore-560107.

URL: [www.acharya.ac.in](http://www.acharya.ac.in/)

**2020-21**

**Disclaimer**

The information contained in this document is the proprietary and exclusive property of Acharya Institutes except as otherwise indicated. No part of this document, in whole or in part, may be reproduced, stored, transmitted, or used for course material development purposes without the prior written permission of Acharya Institutes.

The information contained in this document is subject to change without notice. The information in this document is provided for informational purposes only.

**Trademark**



**Edition: 2020 - 21**

**Document Owner**

The primary contact for questions regarding this document is:

|  |  |
| --- | --- |
| Author(s): | * Prof. Hemanth T. D. * Prof. Swetha K.H. |
| Department: | Information Science & Engineering |
| Contact Email(s): | hemantha@acharya.ac.in  swethakh@acharya.ac.in |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **DESIGN AND ANALYSIS OF ALGORITHM LABORATORY**  **[As per Choice Based Credit System (CBCS) scheme] (Effective from the academic year 2018 -19)**  **SEMESTER – IV** | | | | | | |
| Subject Code | | | 18CSL47 | IA Marks | 40 | |
| Number of Lecture Hours/Week | | | 01 I + 02 P | Exam Marks | 100 | |
| Total Number of Lecture Hours | | | 40 | Exam Hours | 03 | |
| **CREDITS – 02** | | | | | | |
| **Course objectives :** This course will enable students to   * Design and implement various algorithms in JAVA * Employ various design strategies for problem solving. * Measure and compare the performance of different algorithms. | | | | | | |
| **Description :** Design, develop, and implement the specified algorithms for the following problems using Java language under LINUX /Windows environment. Netbeans/Eclipse IDE tool can be used for development and demonstration. | | | | | | |
| **Experiments** | | | | | | |
| **1** | A | Create a Java class called ***Student*** with the following details as variables within it.   * USN * Name * Branch * Phone   Write a Java program to create *n Student* objects and print the USN, Name, Branch, and  Phone of 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. | | | | |
| **2** | 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 <name, dd/mm/yyyy> and display as <name, dd, mm, yyyy> using StringTokenizer class considering the delimiter character as “/”. | | | | |
| **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. | | | | |
| 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 and prints; third thread will print the value of cube of the number. | | | | |
| **4** | 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 divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case. | | | | | |
| **5** | 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. | | | | |
| **6** | Implement in Java, the **0/1 Knapsack** problem using (a) Dynamic Programming method  (b) Greedy method. | | | | |
| **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. | | | | |
| **8** | Find Minimum Cost Spanning Tree of a given connected undirected graph using **Kruskal's algorithm.** Use Union-Find algorithms in your program. | | | | |
| **9** | Find Minimum Cost Spanning Tree of a given connected undirected graph using **Prim's algorithm**. | | | | |
| **10** | Write Java programs to  (a) Implement All-Pairs Shortest Paths problem using **Floyd's algorithm**.  (b) Implement **Travelling Sales Person problem** using Dynamic programming. | | | | |
| **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. | | | | |
| **12** | Design and implement in Java to find all **Hamiltonian Cycles** in a connected undirected Graph G of *n* vertices using backtracking principle. | | | | |
| **Course Outcomes:** The students will be able to:   * Design algorithms using appropriate design techniques (brute-force, greedy, dynamic programming, etc.) * Implement a variety of algorithms such assorting, graph related, combinatorial, etc., in a high level language. * Analyze and compare the performance of algorithms using language features. * Apply and implement learned algorithm design techniques and data structuresto solve real- world problems. | | | | | |
| **Graduate Attributes:**   * Engineering Knowledge * Problem Analysis * Modern Tool Usage * Conduct Investigations of Complex Problems * Design/Development of Solutions | | | | | |
| **Conduction of Practical Examination:** | | | | | |
| 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: 15 + 70 + 15.**  **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.**

**AIM:**

To create a Java class called *Student* with the members (i) USN (ii) Name (iii) Branch (iv) Phone . To write a Java program to create *n Student* objects and print the USN, Name, Branch, and Phone of these objects with suitable headings.

**PROGRAM:**

import java.util.Scanner;

class Student {

String USN, Name, Branch, Phone;

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

void read() {

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

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

USN = input.nextLine();

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

Name = input.nextLine();

System.*out*.println("Enter Branch");

Branch = input.nextLine();

System.*out*.println("Enter Phone");

Phone = input.nextLine();

}

void display() {

System.*out*.println( USN, Name, Branch, Phone);

}

}

class StudentDetails{

public static void main(String[ ] args) {

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

System.*out*.println("Enter number of student details to be created");

int number = input.nextInt();

Student s[] = new Student[number];

// Read student details into array of student objects

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

s[i] = new Student();

s[i].read();

}

// Display student information

System.*out*.println( "USN", "NAME", "BRANCH", "PHONE");

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

System.*out*.println();

s[i].display();

}

input.close();

}

RESULT:

Thus the Student class was created and include the read and diplay methods. To created n student objects were created and its content were displayed.

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

AIM:

To implement the Stack using array with all its operations.

PROGRAM:

import java.util.\*;

class arrayStack {

int arr[];

int top, max;

arrayStack(int n) {

max = n;

arr = new int[max];

top = -1;

}

void push(int i) {

if (top == max - 1)

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

else

arr[++top] = i;

}

void pop() {

if (top == -1) {

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

} else {

int element = arr[top--];

System.out.println("Popped Element: " + element);

}

}

void display() {

System.out.print("\nStack = ");

if (top == -1) {

System.out.print("Empty\n");

return;

}

for (int i = top; i >= 0; i--)

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

System.out.println();

}

}

class Stack {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

System.out.println("Enter Size of Integer Stack ");

int n = scan.nextInt();

boolean done = false;

arrayStack stk = new arrayStack(n);

do {

System.out.println("\nStack Operations");

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

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

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

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

int choice = scan.nextInt();

switch (choice) {

case 1:

System.out.println("Enter integer element to push");

stk.push(scan.nextInt());

break;

case 2:

stk.pop();

break;

case 3:

stk.display();

break;

case 4:

done = true;

break;

default:

System.out.println("Wrong Entry \n ");

break;

}

} while (!done);

}

}

RESULT:

Thus the Stack was implementd using array with all of its operations.

**2.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 atleast 3 staff objects of all three categories.**

**AIM:**

To 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). To write a Java program to read and display atleast 3 staff objects of all three categories.

PROGRAM:

import java.util.Scanner;

class Staff {

String StaffID, Name, Phone, Salary;

Scanner input = new Scanner(System.in);

void read() {

System.out.println("Enter StaffID");

StaffID = input.nextLine();

System.out.println("Enter Name");

Name = input.nextLine();

System.out.println("Enter Phone");

Phone = input.nextLine();

System.out.println("Enter Salary");

Salary = input.nextLine();

}

void display() {

System.out.println("STAFFID: ");

System.out.println( StaffID);

System.out.println( "NAME: ");

System.out.println( Name);

System.out.println("PHONE:");

System.out.println(Phone);

System.out.println( "SALARY:");

System.out.println(Salary);

}

}

class Teaching extends Staff {

String Domain, Publication;

void read\_Teaching() {

super.read(); // call super class read method

System.out.println("Enter Domain");

Domain = input.nextLine();

System.out.println("Enter Publication");

Publication = input.nextLine();

}

void display() {

super.display(); // call super class display() method

System.out.println("DOMAIN:");

System.out.println(Domain);

System.out.println("PUBLICATION:");

System.out.println(Publication);

}

}

class Technical extends Staff {

String Skills;

void read\_Technical() {

super.read(); // call super class read method

System.out.println("Enter Skills");

Skills = input.nextLine();

}

void display() {

super.display(); // call super class display() method

System.out.println("SKILLS:");

System.out.println(Skills);

}

}

class Contract extends Staff {

String Period;

void read\_Contract() {

super.read(); // call super class read method

System.out.println("Enter Period");

Period = input.nextLine();

}

void display() {

super.display(); // call super class display() method

System.out.println("PERIOD:");

System.out.println(Period);

}

}

class Staffdetails {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.println("Enter number of staff details to be created");

int n = input.nextInt();

Teaching steach[] = new Teaching[n];

Technical stech[] = new Technical[n];

Contract scon[] = new Contract[n];

// Read Staff information under 3 categories

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

System.out.println("Enter Teaching staff information");

steach[i] = new Teaching();

steach[i].read\_Teaching();

}

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

System.out.println("Enter Technical staff information");

stech[i] = new Technical();

stech[i].read\_Technical();

}

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

System.out.println("Enter Contract staff information");

scon[i] = new Contract();

scon[i].read\_Contract();

}

// Display Staff Information

System.out.println("\n STAFF DETAILS: \n");

System.out.println("-----TEACHING STAFF DETAILS----- ");

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

steach[i].display();

}

System.out.println();

System.out.println("-----TECHNICAL STAFF DETAILS-----");

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

stech[i].display();

}

System.out.println();

System.out.println("-----CONTRACT STAFF DETAILS-----");

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

scon[i].display();

}

input.close();

}

}

**RESULT:**

Thus the concept of inheritance was implemented and the array of objects were created.

**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 <name, dd/mm/yyyy> and display as <name, dd, mm, yyyy> using StringTokenizer class considering the delimiter character as “/”.**

**AIM:**

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

**Program:**

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

**class** Customer {

String name;

**void** read() {

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

System.***out***.println("Enter Name and Date\_of\_Birth in the format <Name,DD/MM/YYYY>");

name = sc.next();

sc.close();

}

**void** display() {

StringTokenizer st = **new** StringTokenizer(name, ",/");

**int** count = st.countTokens();

**for** (**int** i = 1; i <= count && st.hasMoreTokens(); i++) {

System.***out***.print(st.nextToken());

**if** (i < count)

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

}

}

}

**public** **class** Customerdetails {

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

// **TODO** Auto-generated method stub

Customer c = **new** Customer();

c.read();

c.display();

}

}

Result:

Thus the Customer class was created and inlclude the read method to accept the name and dob and display method to show the name as per the prescribed format.

**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.**

**AIM:**

To wrte 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.

**PROGRAM:**

import java.util.Scanner;

class Exception {

public static void main(String[] args) {

int a, b, result;

Scanner input = new Scanner(System.in);

System.out.println("Input two integers");

a = input.nextInt();

b = input.nextInt();

try {

result = a / b;

System.out.println("Result = " + result);

}

catch (ArithmeticException e) {

System.out.println("Exception caught: Division by zero.");

}

}

}

**RESULT:**

Thus the exception was raised when the divisor is zero and handled.

**3.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 and prints; third thread will print the value of cube of the number.**

**AIM:**

To implement the multiple thread application that has three 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.

**PROGRAM:**

**import** java.util.Random;

**class** SquareThread **implements** Runnable {

**int** x;

SquareThread(**int** x) {

**this**.x = x;

}

**public** **void** run() {

System.***out***.println("Thread Name:Square Thread and Square of " + x + " is: " + x \* x);

}

}

**class** CubeThread **implements** Runnable {

**int** x;

CubeThread(**int** x) {

**this**.x = x;

}

**public** **void** run() {

System.***out***.println("Thread Name:Cube Thread and Cube of " + x + " is: " + x \* x \* x);

}

}

**class** RandomThread **implements** Runnable {

Random r;

**public** **void** run() {

**int** num;

r = **new** Random();

**try** {

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

num = r.nextInt(100);

Thread.*sleep*(1000);

System.***out***.println("Random Number " + num);

}

} **catch** (InterruptedException ex) {

System.***out***.println("Interrupted Exception");

}

}

}

**public** **class** MainThread {

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

// **TODO** Auto-generated method stub

System.***out***.println("Main Thread starts");

Random r = **new** Random();

**int** num;

Thread t1, t2, t3;

RandomThread thread\_obj = **new** RandomThread();

t1 = **new** Thread(thread\_obj);

t1.start();

num = r.nextInt(100);

t2 = **new** Thread(**new** SquareThread(num));

t2.start();

t3 = **new** Thread(**new** CubeThread(num));

t3.start();

}

}

**RESULT:**

Thus **t**he multiple thread application was implementded that has three 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

**4. 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 divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.**

**AIM:**

To sort a given set of random integer numbers using Quick Sort of divide and conquer method and analysis its time complexity.

**PROGRAM:**

import java.util.Scanner;

import java.util.Arrays;

import java.util.Random;

public class QSort {

static final int MAX = 100000;

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

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

int n = input.nextInt();

Random random = new Random();

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

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

System.out.println("Input Array:");

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

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

long startTime = System.nanoTime();

QuickSortAlgorithm(0, n - 1);

long stopTime = System.nanoTime();

long elapsedTime = stopTime - startTime;

System.out.println("\nSorted Array:");

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

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

System.out.println();

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);

}

}

}

RESULT :

Thus the given set of numbers were sorted by quick sort method using divide and conquer and analysed its time complexity.

**5. 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.**

**AIM:**

To sort a given set of random integer numbers by Merge Sort using divide and conquer method and analysis its time complexity.

**PROGRAM**

import java.util.Random;

import java.util.Scanner;

public class MergeSort {

static final int MAX = 100000;

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

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

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

int n = input.nextInt();

Random random = new Random();

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

a[i] = random.nextInt(10000);

System.out.println("Input Array is:");

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

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

long startTime = System.nanoTime();

MergeSortAlgorithm(0, n - 1);

long stopTime = System.nanoTime();

long elapsedTime = stopTime - startTime;

System.out.println("\*\*\*\*\*\*\*\*Sorted Array is\*\*\*\*\*\*\*\*\*\*\*");

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

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

System.out.println("Time Complexity (ms) for n = " +

n + " is : " + (double) elapsedTime / 1000000);

input.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[] b = new int[MAX];

int i, j, k,h;

i =k= low;

j = mid + 1;

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

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

b[k++] = a[i++];

else

b[k++] = a[j++];

if (i > mid)

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

b[k++] = a[h];

else

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

b[k++] = a[h];

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

a[k] = b[k];

}

}

**RESULT:**

Thus the given set of numbers were sorted by merge sort method using divide and conquer and analysed its time complexity.

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

**AIM:**

To implement the 0/1 Knapsack problem using Dynamic Programmming Method in Java.

**PROGRAM:**

import java.util.Scanner;

public class KnapsackDP {

static final int MAX = 20; // max. no. of objects

static int w[ ]; // weights 0 to n-1

static int p[ ]; // profits 0 to n-1

static int n; // no. of objects

static int M; // capacity of Knapsack

static int V[ ][ ]; // DP solution process - table

static int Keep[ ][ ]; // to get objects in optimal solution

public static void main(String args[ ]) {

w = new int[MAX];

p = new int[MAX];

V = new int [MAX][MAX];

Keep = new int[MAX][MAX];

int optsoln;

ReadObjects();

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

V[0][i] = 0;

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

V[i][0] = 0;

optsoln = Knapsack();

System.out.println("Optimal solution = " + optsoln);

}

static int Knapsack() {

int r; // remaining Knapsack capacity

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

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

if ((w[i] <= j) && (p[i] + V[i - 1][j - w[i]] > V[i - 1][j])) {

V[i][j] = p[i] + V[i - 1][j - w[i]];

Keep[i][j] = 1;

} else {

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

Keep[i][j] = 0;

}

// Find the objects included in the Knapsack

r = M;

System.out.println("Items = ");

for (int i = n; i > 0; i--) // start from Keep[n,M]

if (Keep[i][r] == 1) {

System.out.println(i + " ");

r = r - w[i];

}

System.out.println();

return V[n][M];

}

static void ReadObjects() {

Scanner scanner = new Scanner(System.in);

System.out.println("Knapsack Problem - Dynamic Programming Solution: ");

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

M = scanner.nextInt();

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

n = scanner.nextInt();

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

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

w[i] = scanner.nextInt();

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

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

p[i] = scanner.nextInt();

scanner.close();

}

}

RESULT:

Thus the 0/1 Knapsack problem using Dynamic Programmming Method was implemented in Java.

**AIM:**

To implement the 0/1 Knapsack problem using Greedy Method in Java.

**PROGRAM:**

import java.util.Scanner;

class KObject { // Knapsack object details

float w;

float p;

float r;

}

public class KnapsackGreedy {

static final int MAX = 20; // max. no. of objects

static int n; // no. of objects

static float M; // capacity of Knapsack

public static void main(String args[]) {

Scanner scanner = new Scanner(System.in);

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

n = scanner.nextInt();

KObject[] obj = new KObject[n];

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

obj[i] = new KObject();// allocate memory for members

ReadObjects(obj);

Knapsack(obj);

scanner.close();

}

static void ReadObjects(KObject obj[]) {

KObject temp = new KObject();

Scanner scanner = new Scanner(System.in);

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

M = scanner.nextFloat();

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

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

obj[i].w = scanner.nextFloat();

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

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

obj[i].p = scanner.nextFloat();

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

obj[i].r = obj[i].p / obj[i].w;

// sort objects in descending order, based on p/w ratio

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

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

if(obj[j].r < obj[j+1].r){

temp = obj[j];

obj[j] = obj[j+1];

obj[j+1] = temp;

}

scanner.close();

}

static void Knapsack(KObject kobj[]) {

float x[] = new float[MAX];

float totalprofit;

int i;

float U; // U place holder for M

U = M;

totalprofit = 0;

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

x[i] = 0;

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

if (kobj[i].w > U)

break;

else {

x[i] = 1;

totalprofit = totalprofit + kobj[i].p;

U = U - kobj[i].w;

}

}

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

if (i < n)

x[i] = U / kobj[i].w;

totalprofit = totalprofit + (x[i] \* kobj[i].p);

System.out.println("The Solution vector, x[]: ");

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

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

System.out.println("\nTotal profit is = " + totalprofit);

}

}

RESULT:

Thus the 0/1 Knapsack problem using Greedy Method was implemented in Java.

**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.**

**AIM:**

To find the shortest paths from a given vertex to other vertices in a weighted connected graph using Dijkstra's algorithm in Java**.**

**PROGRAM:**

import java.util.\*;

public class DijkstrasClass {

final static int MAX = 20;

final static int infinity = 9999;

static int n; // No. of vertices of G

static int a[ ][ ]; // Cost matrix

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

public static void main(String[] args) {

ReadMatrix();

System.out.println("Enter starting vertex: ");

int s = scan.nextInt();

Dijkstras(s); // find shortest path

}

static void ReadMatrix() {

a = new int[MAX][MAX];

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

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();

}

static void Dijkstras(int s) {

int S[] = new int[MAX];

int d[] = new int[MAX];

int u, v;

int i;

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

S[i] = 0;

d[i] = a[s][i];

}

S[s] = 1;

d[s] = 0;

i = 2;

while (i <= n) {

u = Extract\_Min(S, d);

S[u] = 1;

i++;

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

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

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

}

}

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

System.out.println(s+””+i + "=" + d[i]);

}

static int Extract\_Min(int S[ ], int d[ ]) {

int i, j = 1, min;

min = infinity;

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

if ((d[i] < min) && (S[i] == 0)) {

min = d[i];

j = i;

}

}

return (j);

}

}

RESULT:

Thus the shortest paths was found from a given vertex to other vertices in a weighted connected graph using Dijkstra's algorithm in Java**.**

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

**AIM:**

To find the Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm and Use Union-Find algorithms in Java**.**

**PROGRAM:**

import java.util.Scanner;

public class KruskalsClass {

final static int MAX = 20;

static int n; // No. of vertices of G

static int cost[][]; // Cost matrix

static int parent[] = new int[9];

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

public static void main(String[] args) {

ReadMatrix();

Kruskals();

}

static void ReadMatrix() {

int i, j;

cost = new int[MAX][MAX];

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

System.out.println("Enter the no. of vertices");

n = scan.nextInt();

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

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

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

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

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

cost[i][j] = 999;

}

}

}

static void Kruskals() {

int a = 0, b = 0, u = 0, v = 0, i, j, ne = 0, min, mincost = 0;

System.out.println("The edges of Minimum Cost Spanning Tree are");

while (ne != n-1) {

min=999;

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

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

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

min = cost[i][j];

a = u = i;

b = v = j;

}

}

}

u = find(u);

v = find(v);

if (u != v) {

union(u, v);

System.out.println("edge (" + a + "," + b + ") =" + min);

mincost += min;

ne++;

}

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

}

System.out.println("Minimum cost :" + mincost);

}

static int find(int i) {

while (parent[i] != 0)

i = parent[i];

return i;

}

static void union(int i, int j) {

parent[j] = i;

}

}

RESULT:

Thus the Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm and Use Union-Find algorithms was found in Java**.**

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

**AIM:**

To find the Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm in Java**.**

**PROGRAM:**

import java.util.Scanner;

public class Prims {

public static void main(String[ ] args) {

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

int i, j, mincost = 0;

Scanner in = new Scanner(System.in);

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

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

int n = in.nextInt();

System.out.println("Enter the 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("Minimum Spanning Tree Edges and costs are");

mincost=prims(cost,n,mincost);

System.out.println("The minimum spanning tree 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;

nearV[1]=0;

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.println(i+") Minimum edge is ("+t[i][1]);

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

}

return mincost;

}

}

RESULT:

Thus the Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm in Java**.**

**10. Write Java programs to**

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

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

**AIM:**

To implement All-Pairs Shortest Paths problem using Floyd's algorithm in Java.

**PROGRAM:**

import java.util.Scanner;

public class FloydsClass {

static final int MAX = 20; // max. size of cost matrix

static int a[][]; // cost matrix

static int n; // actual matrix size

public static void main(String args[]) {

a = new int[MAX][MAX];

ReadMatrix();

Floyds(); // find all pairs shortest path

PrintMatrix();

}

static void ReadMatrix() {

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

Scanner scanner = new Scanner(System.in);

n = scanner.nextInt();

System.out.println("Enter the Cost Matrix (999 for infinity) \n");

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

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

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

}

}

scanner.close();

}

static void Floyds() {

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

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

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

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

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

}

}

static void PrintMatrix() {

System.out.println("The All Pair Shortest Path Matrix is:\n");

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

{

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

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

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

}

}

}

**RESULT:**

Thus the All-Pairs Shortest Paths problem using Floyd's algorithm was implemented in Java.

**AIM:**

To implement Travelling Sales Person problem using Dynamic programming in Java.

**PROGRAM:**

import java.util.Scanner;

public class TravSalesPerson {

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 scanner = new Scanner(System.in);

n = scanner.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] = scanner.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");

scanner.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;

int 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;

}

}

RESULT:

Thus the Travelling Sales Person problem using Dynamic programming was implemented in Java.

**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.**

AIM:

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 was designed and implemented in Java.

**PROGRAM:**

import java.util.Scanner;

public class SumOfsubset {

final static int MAX = 10;

static int n;

static int S[];

static int soln[];

static int d;

public static void main(String args[]) {

S = new int[MAX];

soln = new int[MAX];

int sum = 0;

Scanner scanner = new Scanner(System.in);

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

n = scanner.nextInt();

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

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

S[i] = scanner.nextInt();

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

d = scanner.nextInt();

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

sum = sum + S[i];

if (sum < d || S[1] > d)

System.out.println("No Subset possible");

else

SumofSub(0, 0, sum);

scanner.close();

}

static void SumofSub(int i, int weight, int total) {

if (promising(i, weight, total) == true)

if (weight == d) {

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

if (soln[j] == 1)

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

}

System.out.println();

} else {

soln[i + 1] = 1;

SumofSub(i + 1, weight + S[i + 1], total - S[i + 1]);

soln[i + 1] = 0;

SumofSub(i + 1, weight, total - S[i + 1]);

}

}

static boolean promising(int i, int weight, int total) {

return ((weight + total >= d) && (weight == d || weight + S[i + 1] <= d));

}

}

RESULT:

Thus the subset of a given set S = {Sl, S2,.....,Sn} of n positive integers whose SUM is equal to a given positive integer d was found designed and implemented in Java.

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

**AIM:**

To design and implement,find all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle in Java.

**PROGRAM:**

import java.util.Scanner;

public class Hamiltonian {

static int *n*;

public static void main(String[] args) {

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

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

*n*=in.nextInt();

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

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

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

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

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

System.*out*.println("Entered adjacency matrix of graph is");

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

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

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

}

System.*out*.println();

}

*hamCycle*(graph);

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

}

static void printSolution(int path[]){

System.*out*.println("Solution Exists:");

System.*out*.println(" Following is one Hamiltonian Cycle ");

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

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

System.*out*.println(path[0]);

}

static boolean isSafe(int v,int graph[][],int path[],int pos){

if (graph[path[pos-1]][v] == 0)

return false;

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

if (path[i] == v)

return false;

return true;

}

/\* A recursive utility function to solve Hamiltonian cycle problem \*/

static boolean hamCycleUtil(int graph[][],int path[],int pos){

if (pos == *n*){

if (graph[path[pos-1]][path[0]] == 1)

return true;

else return false;

}

for (int v = 1; v < *n*; v++){

if (*isSafe*(v, graph, path, pos)){

path[pos] = v;

if (*hamCycleUtil* (graph, path, pos+1) == true)

return true;

path[pos] = -1;

}

}

return false;

}

/\* This function solves the Hamiltonian Cycle problem using Backtracking. It mainly uses hamCycleUtil() to solve the problem. It returns false if there is no Hamiltonian Cycle possible,

otherwise return true and prints the path.This function prints one of the feasible solutions. \*/

static boolean hamCycle(int graph[][]){

int path[] = new int[*n*+1];

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

path[i] = -1;

path[0] = 0;

if (*hamCycleUtil*(graph, path, 1) == false){

System.*out*.println("\nSolution does not exist");

return false;

}

*printSolution*(path);

return true;

}

}

RESULT:

Thus all Hamiltonian Cycles in a connected undirected Graph G of n vertices using backtracking principle was found and implemented in Java.