1.

package practice;

import java.util.\*;

public class binarysearch {

public static int binarySearch(int[] array, int left, int right, int target) {

while (left <= right) {

int middle = left + (right - left) / 2;

if (array[middle] == target) {

return middle;

}

if (array[middle] < target) {

left = middle + 1;

} else {

right = middle - 1; }

}

return -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

int[] array = new int[n];

System.out.println("Enter sorted elements:");

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

array[i] = scanner.nextInt(); }

System.out.print("Enter lower bound index: ");

int left = scanner.nextInt();

System.out.print("Enter upper bound index: ");

int right = scanner.nextInt();

System.out.print("Enter target value: ");

int target = scanner.nextInt();

int result = binarySearch(array, left, right, target);

if (result == -1) {

System.out.println("Element not found");

} else {

System.out.println("Element found at index " + result);

}

scanner.close(); } }

2.

package unit\_1;

import java.util.Scanner;

public class mergesort {

void merge(int arr[],int lb,int mid,int ub ) {

int n1=mid-lb+1;

int n2=ub-mid;

int larr[]=new int[n1];

int rarr[]=new int[n2];

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

larr[i]=arr[lb+i];

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

rarr[j]=arr[mid+1+j];

}

}

int i=0,j=0,k=lb;

while(i<n1&&j<n2) {

if(larr[i]<=rarr[j]) {

arr[k++]=larr[i++];

}

else {

arr[k++]=rarr[j++];

}

}

while(i<n1) {

arr[k++]=larr[i++];

}

while(j<n2) {

arr[k++]=rarr[j++];

}

}

void sort(int arr[],int lb,int ub) {

if(lb<ub) {

int mid=(lb+ub)/2;

sort(arr,lb,mid);

sort(arr,mid+1,ub);

merge(arr,lb,mid,ub);

}

}

static void printArray(int arr[]) {

int n=arr.length;

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

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

System.out.println();

}

public static void main(String[] args) {

int n;

Scanner sc=new Scanner(System.in);

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

n=sc.nextInt();

int arr[]=new int[n];

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

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

arr[i]=sc.nextInt();

}

System.out.println("Given array elements are : ");

printArray(arr);

mergesort ms =new mergesort();

ms.sort(arr, 0, n-1);

System.out.println("\nSorted aray elements are : ");

printArray(arr);

}

}

3.

package ada\_lab;

import java.util.Scanner;

public class exp\_3 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n,m;

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

n = sc.nextInt();

int weight[] = new int[n];

int profit[] = new int[n];

System.out.println("Enter the profits of all objects : ");

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

profit[i] = sc.nextInt();

}

System.out.println("Enter the weights of all objects : ");

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

weight[i] = sc.nextInt();

}

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

m = sc.nextInt();

double p\_w[] = new double [n];

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

p\_w[i] = (double)profit[i]/(double)weight[i];

}

System.out.print("\nObject : ");

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

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

}

System.out.print("\nProfit : ");

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

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

}

System.out.print("\nWeight : ");

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

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

}

System.out.print("\np/w : ");

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

System.out.printf("%.2f ",p\_w[i] );

}

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

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

if(p\_w[i] <p\_w[j]) {

double temp = p\_w[j];

p\_w[j] = p\_w[i];

p\_w[i] = temp;

int temp1 = profit[j];

profit[j] = profit[i];

profit[i] = temp1;

int temp2 = weight[j];

weight[j] = weight[i];

weight[i] = temp2;

}

}

}

System.out.println("\nAfter sorting according to profit/weight : ");

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

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

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

}

System.out.print("\nProfit : ");

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

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

}

System.out.print("\nWeight :");

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

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

}

System.out.print("\np\_w : ");

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

System.out.printf("%.2f",p\_w[i]);

}

int k =0;

double sum = 0;

System.out.println();

while(m>0) {

if(weight[k]<m) {

sum+=1\*profit[k];

m = m-weight[k];

}else {

sum = sum+(float)m/weight[k]\*profit[k];

m = 0;

}

k++;

}

System.out.println("Final profit is : "+sum);

}

}

4.

package unit2;

import java.util.\*;

public class JSD

{

public static void main(String args[])

{

Scanner sc =new Scanner(System.in);

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

int n=sc.nextInt();

String job[]=new String[n];

int p[]=new int[n];

int d[]=new int[n];

System.out.println("Enter the Jobs:");

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

{

job[i]=sc.next();

}

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

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

{

p[i]=sc.nextInt();

}

System.out.println("Enter the DeadLines:");

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

{

d[i]=sc.nextInt();

}

System.out.println("Jobs in the given order:");

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

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

{

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

}

System.out.print("\nProfit: ");

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

{

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

}

System.out.print("\nDeadLine:");

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

{

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

}

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

{

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

{

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

{

int temp=p[i];

p[i]=p[j];

p[j]=temp;

temp=d[i];

d[i]=d[j];

d[j]=temp;

String temp1=job[i];

job[i]=job[j];

job[j]=temp1;

}

}

}

System.out.println();

System.out.println("Jobs in the decreasing order of Profits:");

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

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

{

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

}

System.out.print("\nProfit: ");

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

{

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

}

System.out.print("\nDeadLine:");

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

{

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

}

System.out.println();

int max=d[0];

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

{

if(d[i]>max)

{

max=d[i];

}

}

String s[]=new String[max];

int profit=0;

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

{

int dl=d[i];

dl=dl-1;

if(s[dl]==null )

{

s[dl]=job[i];

profit+=p[i];

}

else

{

while(dl!=-1)

{

if(s[dl]==null)

{

s[dl]=job[i];

profit+=p[i];

break;

}

dl=dl-1;

}

}

}

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

{

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

}

System.out.println();

System.out.print("Profit Earned: "+profit);

}

}

5.

import java.util.Iterator;

import java.util.LinkedList;

import java.util.Scanner;

public class Kruskal {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

int nv, ne, i, j, mincost = 0;

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

nv = sc.nextInt();

System.out.print("Enter number of edges: ");

ne = sc.nextInt();

Edge ed[] = new Edge[ne + 1];

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

ed[i] = new Edge();

System.out.println("Enter the vertices and the weight of edge " + i + ":");

int v1 = sc.nextInt();

int v2 = sc.nextInt();

int wt = sc.nextInt();

// Input validation for vertex range

if (v1 < 1 || v1 > nv || v2 < 1 || v2 > nv) {

System.out.println("❌ Invalid vertex number! Please enter values between 1 and " + nv);

i--; // retry the same edge

continue;

}

ed[i].v1 = v1;

ed[i].v2 = v2;

ed[i].wt = wt;

}

// Sort edges by weight (Bubble Sort)

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

for (j = 1; j <= ne - i; j++) {

if (ed[j].wt > ed[j + 1].wt) {

Edge temp = ed[j];

ed[j] = ed[j + 1];

ed[j + 1] = temp;

}

}

}

Graph g = new Graph(nv);

System.out.println("\nMINIMUM SPANNING TREE:");

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

g.addEdge(ed[i].v1, ed[i].v2);

if (!g.isCycle()) {

System.out.println(ed[i].v1 + " - " + ed[i].v2);

mincost += ed[i].wt;

} else {

g.removeEdge(ed[i].v1, ed[i].v2);

}

}

System.out.println("MINIMUM COST = " + mincost);

}

}

class Edge {

int v1, v2, wt;

}

class Graph {

private int nv;

private LinkedList<Integer>[] adj;

@SuppressWarnings("unchecked")

Graph(int n) {

nv = n;

adj = new LinkedList[n + 1];

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

adj[i] = new LinkedList<>();

}

void addEdge(int u, int v) {

adj[u].add(v);

adj[v].add(u);

}

void removeEdge(int u, int v) {

adj[u].remove(Integer.valueOf(v));

adj[v].remove(Integer.valueOf(u));

}

boolean isCycle() {

boolean[] visited = new boolean[nv + 1];

for (int u = 1; u <= nv; u++) {

if (!visited[u]) {

if (cycleCheck(u, -1, visited))

return true;

}

}

return false;

}

boolean cycleCheck(int node, int parent, boolean[] visited) {

visited[node] = true;

for (int i : adj[node]) {

if (!visited[i]) {

if (cycleCheck(i, node, visited))

return true;

} else if (i != parent) {

return true;

}

}

return false;

}

}

6.

package unit3;

import java.util.Scanner;

public class APSP

{

public static void main(String[] args)

{

int n,i,j,k;

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

Scanner sc=new Scanner(System.in);

n=sc.nextInt();

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

System.out.println("Enter the weighted matrix:");

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

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

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

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

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

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

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

System.out.println("The shortest path matrix is:");

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

{

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

{

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

}

System.out.println();

}

}

}

7.

import java.util.Scanner;

public class TSP

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

int n,i,j,cost;

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

n=sc.nextInt();

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

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

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

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

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

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

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

tour[i]=i;

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

System.out.print("The Optimal Tour is: ");

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

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

System.out.println("1");

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

}

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

{

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

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

int mincost=Integer.MAX\_VALUE,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];

ccost=tspdp(c,temp,start+1,n);

if((c[tour[start]][tour[i]]+ccost)<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;

}

}

8.

import java.util.Iterator;

import java.util.LinkedList;

import java.util.Scanner;

public class DFS {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int nv, ne, i, s, v1, v2;

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

nv = sc.nextInt();

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

ne = sc.nextInt();

Graph g = new Graph(nv);

System.out.println("Enter the edges (1-based indexing):");

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

v1 = sc.nextInt() - 1; // Adjust for 0-based index

v2 = sc.nextInt() - 1;

if (v1 < 0 || v1 >= nv || v2 < 0 || v2 >= nv) {

System.out.println("❌ Invalid vertex! Please enter vertices between 1 and " + nv);

i--; // retry same edge

continue;

}

g.addEdge(v1, v2);

}

System.out.println("Enter the starting vertex (1-based):");

s = sc.nextInt() - 1;

if (s < 0 || s >= nv) {

System.out.println("❌ Invalid starting vertex! Must be between 1 and " + nv);

return;

}

System.out.println("Depth First Traversal (starting from " + (s + 1) + "):");

boolean visited[] = new boolean[nv];

g.DFSTraversal(s, visited);

}

}

class Graph {

private int nv;

private LinkedList<Integer>[] adj;

@SuppressWarnings("unchecked")

Graph(int n) {

nv = n;

adj = new LinkedList[nv];

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

adj[i] = new LinkedList<>();

}

void addEdge(int u, int v) {

adj[u].add(v);

}

void DFSTraversal(int v, boolean visited[]) {

visited[v] = true;

System.out.print((v + 1) + " "); // Display as 1-based

Iterator<Integer> i = adj[v].listIterator();

while (i.hasNext()) {

int n = i.next();

if (!visited[n])

DFSTraversal(n, visited);

}

}

}

9.

public class NQueens {

static void printSolution(int[][] board, int n) {

for (int[] row : board) {

for (int cell : row) {

System.out.print((cell == 1 ? "Q " : ". "));

}

System.out.println();

}

System.out.println();

}

static boolean isSafe(int[][] board, int row, int col, int n) {

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

if (board[i][col] == 1)

return false;

for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)

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

return false;

for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++)

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

return false;

return true;

}

static boolean solveNQueensUtil(int[][] board, int row, int n) {

if (row >= n) {

printSolution(board, n);

return true;

}

boolean res = false;

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

if (isSafe(board, row, col, n)) {

board[row][col] = 1;

res = solveNQueensUtil(board, row + 1, n) || res;

board[row][col] = 0;

}

}

return res;

}

public static void solveNQueens(int n) {

int[][] board = new int[n][n];

if (!solveNQueensUtil(board, 0, n)) {

System.out.println("No solution exists.");

}

}

public static void main(String[] args) {

int n = 4;

solveNQueens(n);

}

}

10.

import java.util.Arrays;

public class TSPBranchAndBound {

static int N = 4;

static int[][] finalPath = new int[N + 1][];

static boolean[] visited = new boolean[N];

static int finalRes = Integer.MAX\_VALUE;

static int firstMin(int[][] cost, int i) {

int min = Integer.MAX\_VALUE;

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

if (cost[i][k] < min && i != k)

min = cost[i][k];

return min;

}

static int secondMin(int[][] cost, int i) {

int first = Integer.MAX\_VALUE, second = Integer.MAX\_VALUE;

for (int j = 0; j < N; j++) {

if (i == j) continue;

if (cost[i][j] <= first) {

second = first;

first = cost[i][j];

} else if (cost[i][j] < second)

second = cost[i][j];

}

return second;

}

static void copyToFinal(int[] currPath) {

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

finalPath[i] = new int[]{currPath[i]};

finalPath[N] = new int[]{currPath[0]};

}

static void TSPRec(int[][] cost, int currBound, int currWeight, int level, int[] currPath) {

if (level == N) {

if (cost[currPath[level - 1]][currPath[0]] != 0) {

int currRes = currWeight + cost[currPath[level - 1]][currPath[0]];

if (currRes < finalRes) {

copyToFinal(currPath);

finalRes = currRes;

}

}

return;

}

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

if (cost[currPath[level - 1]][i] != 0 && !visited[i]) {

int temp = currBound;

currWeight += cost[currPath[level - 1]][i];

if (level == 1)

currBound -= ((firstMin(cost, currPath[level - 1]) + firstMin(cost, i)) / 2);

else

currBound -= ((secondMin(cost, currPath[level - 1]) + firstMin(cost, i)) / 2);

if (currBound + currWeight < finalRes) {

currPath[level] = i;

visited[i] = true;

TSPRec(cost, currBound, currWeight, level + 1, currPath);

}

currWeight -= cost[currPath[level - 1]][i];

currBound = temp;

Arrays.fill(visited, false);

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

visited[currPath[j]] = true;

}

}

}

static void TSP(int[][] cost) {

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

int currBound = 0;

Arrays.fill(currPath, -1);

Arrays.fill(visited, false);

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

currBound += (firstMin(cost, i) + secondMin(cost, i));

currBound = (currBound % 2 == 1) ? (currBound / 2) + 1 : currBound / 2;

visited[0] = true;

currPath[0] = 0;

TSPRec(cost, currBound, 0, 1, currPath);

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

System.out.print("Path Taken: ");

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

System.out.print(finalPath[i][0] + " ");

}

public static void main(String[] args) {

int[][] cost = {

{0, 10, 15, 20},

{10, 0, 35, 25},

{15, 35, 0, 30},

{20, 25, 30, 0}

};

TSP(cost);

}

}