//Assignment 1

import java.util.\*;

class Main {

public static void main(String[] args) {

String s1 = "", s2 = "";

int ch = 0;

int cho;

Scanner sc = new Scanner(System.in);

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

System.out.println("Select From Menu : ");

System.out.println("1. Multiplication (20-digit numbers) \n2. Square (20-digit number)\n3. Exit");

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

System.out.print("Enter Your Choice: ");

cho = sc.nextInt();

sc.nextLine(); // Consume newline

switch (cho) {

case 1:

System.out.print("Enter First 20-digit Number: ");

s1 = sc.nextLine();

System.out.print("Enter Second 20-digit Number: ");

s2 = sc.nextLine();

System.out.println("Multiplication of " + s1 + " and " + s2 + " is: " + findMulti(s1,

s2));

break;

case 2:

System.out.print("Enter a 20-digit Number to find its Square: ");

s1 = sc.nextLine();

System.out.println("Square of " + s1 + " is: " + findMulti(s1, s1));

break;

case 3:

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

break;

default:

System.out.println("Invalid choice. Please try again.");

break;

}

System.out.print("Do you want to continue? (1 for Yes / 0 for No): ");

ch = sc.nextInt();

sc.nextLine(); // Consume newline

} while (ch == 1 && cho != 3);

sc.close();

}

// Function to multiply two large numbers using Karatsuba Algorithm

private static String findMulti(String A, String B) {

if (A.length() < B.length()) {

String temp = A;

A = B;

B = temp;

}

// Append zeros to make the length of both numbers equal

int len = A.length();

while (B.length() < len) {

B = '0' + B;

}

// Base case for recursion

if (len == 1) {

int ans = Integer.parseInt(A) \* Integer.parseInt(B);

return Integer.toString(ans);

}

// If length is odd, pad with zero

if (len % 2 == 1) {

A = '0' + A;

B = '0' + B;

len++;

}

// Split numbers into two halves

String Al = A.substring(0, len / 2);

String Ar = A.substring(len / 2);

String Bl = B.substring(0, len / 2);

String Br = B.substring(len / 2);

// Recursively calculate p = Al \* Bl, q = Ar \* Br

String p = findMulti(Al, Bl);

String q = findMulti(Ar, Br);

// Calculate middle term r = (Al + Ar) \* (Bl + Br) - p - q

String r = findDiff(findMulti(findSum(Al, Ar), findSum(Bl, Br)), findSum(p, q));

// Shift results by multiplying by powers of 10

p += "0".repeat(len);

r += "0".repeat(len / 2);

// Final result = p + r + q

String result = findSum(findSum(p, r), q);

// Remove leading zeros

return result.replaceFirst("^0+(?!$)", "");

}

// Function to find the sum of two large numbers represented as strings

private static String findSum(String s1, String s2) {

if (s1.isEmpty()) return s2;

if (s2.isEmpty()) return s1;

if (s1.length() > s2.length()) {

String temp = s1;

s1 = s2;

s2 = temp;

}

StringBuilder result = new StringBuilder();

String str1 = new StringBuilder(s1).reverse().toString();

String str2 = new StringBuilder(s2).reverse().toString();

int carry = 0;

for (int i = 0; i < s1.length(); i++) {

int sum = (str1.charAt(i) - '0') + (str2.charAt(i) - '0') + carry;

result.append(sum % 10);

carry = sum / 10;

}

for (int i = s1.length(); i < s2.length(); i++) {

int sum = (str2.charAt(i) - '0') + carry;

result.append(sum % 10);

carry = sum / 10;

}

if (carry != 0) {

result.append(carry);

}

return result.reverse().toString();

}

// Function to find the difference of two large numbers represented as strings

private static String findDiff(String s1, String s2) {

if (s1.length() < s2.length() || (s1.length() == s2.length() && s1.compareTo(s2) < 0)) {

String temp = s1;

s1 = s2;

s2 = temp;

}

StringBuilder result = new StringBuilder();

String str1 = new StringBuilder(s1).reverse().toString();

String str2 = new StringBuilder(s2).reverse().toString();

int borrow = 0;

for (int i = 0; i < s1.length(); i++) {

int digit1 = str1.charAt(i) - '0';

int digit2 = (i < s2.length()) ? str2.charAt(i) - '0' : 0;

int diff = digit1 - digit2 - borrow;

if (diff < 0) {

diff += 10;

borrow = 1;

} else {

borrow = 0;

}

result.append(diff);

}

return result.reverse().toString().replaceFirst("^0+(?!$)", "");

}

}

//ASSIGNMENT 2

import java.util.\*;

class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

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

int n=sc.nextInt();

Tasks[] tasks=new Tasks[n];

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

System.out.print("Enter task "+(i+1)+" profit:");

int p=sc.nextInt();

System.out.print("Enter task "+(i+1)+" deadline:");

int d=sc.nextInt();

Tasks t=new Tasks(i+1,p,d);

tasks[i]=t;

}

int maxDeadline=0;

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

if(tasks[i].deadline>maxDeadline)

maxDeadline=tasks[i].deadline;

}

int result[]=new int[maxDeadline];

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

result[j]=-1;

}

Arrays.sort(tasks,(a,b)->b.profit-a.profit);

int totalProfit=0;

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

for(int j=tasks[i].deadline-1;j>=0;j--){

if(result[j]==-1){

totalProfit+=tasks[i].profit;

result[j]=tasks[i].id;

break;

}

}

}

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

System.out.println("Time Slot | Task");

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

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

System.out.print(" " +(i+1)+"|");

//print time slot

if(result[i]!=-1)

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

else

System.out.println(" Empty");

}

System.out.println("Total profit is:"+totalProfit);

}

}

class Tasks{

int id,profit,deadline;

//parameterized constructor

Tasks(int id,int profit,int deadline){

this.id=id;

this.profit=profit;

this.deadline=deadline;

}

}\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//ASSIGNMENT 3

import java.util.Arrays;

import java.util.Scanner;

class Main {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

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

int n=sc.nextInt();

int[][] costMatrix=new int[n][n];

System.out.println("Enter the Cost Matrix:(-1 if there is no direct path between offices)");

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

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

if(i==j)

costMatrix[i][j]=0;

else {

System.out.printf("Cost from office %d to office %d:",i,j);

int cost=sc.nextInt();

costMatrix[i][j]=(cost==-1)?Integer.MAX\_VALUE:cost;

}

}

}

int[][] minCostMatrix=floydWarshall(costMatrix,n);

System.out.println("The minimum cost Matrix is:");

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

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

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

}

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

}

sc.close();

}

public static int[][] floydWarshall(int[][] costMatrix,int n) {

int [][] dist=new int[n][n];

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

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

dist[i][j]= costMatrix[i][j];

}

}

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

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

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

if(dist[i][k]!=Integer.MAX\_VALUE&& dist[k][j]!=Integer.MAX\_VALUE){

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

}

}

}

}

return dist;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

//ASSIGNMENT 4  
import java.util.Scanner;

class Main {

static int V;

public static void main(String[] args) {

Scanner sc= new Scanner(System.in);

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

V=sc.nextInt();

int graph[][]=new int[V][V];

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

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

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

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

}

}

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

int source=sc.nextInt();

Main sp=new Main();

sp.dijkstrasAlgorithm(graph,source);

sc.close();

}

void dijkstrasAlgorithm(int graph[][],int source) {

int distances[]=new int[V];

Boolean visitedSet[]=new Boolean[V];

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

distances[i]= Integer.MAX\_VALUE;

visitedSet[i]=false;

}

distances[source]=0;

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

int u=findMinDistance(distances,visitedSet);

visitedSet[u]=true;

for(int v=0;v<V;v++){

if(!visitedSet[v] && graph[u][v] != 0 && distances[u] != Integer.MAX\_VALUE

&& distances[u] + graph[u][v] < distances[v])

distances[v]=distances[u]+graph[u][v];

}

}

printShortestPaths(distances);

}

int findMinDistance(int distances[], Boolean visitedSet[]) {

int min=Integer.MAX\_VALUE,minIndex=-1;

for(int v=0;v<V;v++)

if(!visitedSet[v] && distances[v]<= min) {

min=distances[v];

minIndex=v;

}

return minIndex;

}

void printShortestPaths(int distances[]) {

System.out.println("Vertex \t\t Distance");

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

System.out.println(i+" \t\t "+distances[i]);

}

}

//Assignment 5

import java.util.Scanner;

public class Main {

static int n; // Board size will now be dynamic

// Function to check if a given position (i, j) is valid on the chessboard

public static boolean isValid(int i, int j, int sol[][]) {

return i >= 1 && i <= n && j >= 1 && j <= n && sol[i][j] == -1;

}

// Recursive function to perform the Knight's Tour

public static boolean knightTour(int sol[][], int i, int j, int stepCount, int xMove[], int yMove[]) {

// If all squares are visited, return true

if (stepCount == n \* n) {

return true;

}

// Try all next moves from the current coordinate

for (int k = 0; k < 8; k++) {

int nextI = i + xMove[k];

int nextJ = j + yMove[k];

// Check if the next move is valid

if (isValid(nextI, nextJ, sol)) {

// Mark the square as visited

sol[nextI][nextJ] = stepCount;

// Recur to explore the next move

if (knightTour(sol, nextI, nextJ, stepCount + 1, xMove, yMove)) {

return true; // If a valid solution is found, return true

}

sol[nextI][nextJ] = -1; // Backtrack by marking the square as unvisited

}

}

return false; // No valid solution found

}

// Function to initiate the Knight's Tour

public static boolean startKnightTour(int startRow, int startCol) {

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

// Initialize the chessboard with all squares unvisited

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

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

sol[i][j] = -1;

}

}

sol[startRow][startCol] = 1; // Place the knight at the starting point

// Possible moves for the knight

int xMove[] = {2, 1, -1, -2, -2, -1, 1, 2};

int yMove[] = {1, 2, 2, 1, -1, -2, -2, -1};

// Call the recursive function to find the Knight's Tour

if (knightTour(sol, startRow, startCol, 2, xMove, yMove)) {

// Print the solution if found

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

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

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

}

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

}

return true;

}

return false; // No solution found

}

// Main function to take user input and initiate the Knight's Tour

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take user input for the board size

System.out.print("Enter the board size (e.g., 8 for an 8x8 board): ");

n = scanner.nextInt();

// Take user input for the starting position

System.out.print("Enter the starting row (1 to " + n + "): ");

int startRow = scanner.nextInt();

System.out.print("Enter the starting column (1 to " + n + "): ");

int startCol = scanner.nextInt();

// Check if the starting position is valid

if (startRow < 1 || startRow > n || startCol < 1 || startCol > n) {

System.out.println("Invalid starting point. The starting point should be within the chessboard.");

} else {

// Initiate the Knight's Tour

if (!startKnightTour(startRow, startCol)) {

System.out.println("No solution found for the given board size and starting position.");

}

}

scanner.close();

}

}

//ASSIGNMENT 6

import java.util.\*;

// Node class represents a club assignment

class Node {

Node parent; // parent node

int pathCost; // cost to reach this node

int cost; // lower bound cost

int studentID; // student ID

int clubID; // club ID

boolean assigned[]; // keeps track of assigned clubs

public Node(int N) {

assigned = new boolean[N]; // initialize assigned clubs array

}

}

public class NClubs {

// Function to create a new search tree node

static Node newNode(int x, int y, boolean assigned[], Node parent, int N) {

Node node = new Node(N);

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

node.assigned[j] = assigned[j];

}

if (y != -1) {

node.assigned[y] = true;

}

node.parent = parent;

node.studentID = x;

node.clubID = y;

return node;

}

// Function to calculate the least promising cost of a node

static int calculateCost(int costMatrix[][], int x, int y, boolean assigned[], int N) {

int cost = 0;

boolean available[] = new boolean[N];

Arrays.fill(available, true);

for (int i = x + 1; i < N; i++) {

int min = Integer.MAX\_VALUE, minIndex = -1;

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

if (!assigned[j] && available[j] && costMatrix[i][j] < min) {

minIndex = j;

min = costMatrix[i][j];

}

}

cost += min;

available[minIndex] = false;

}

return cost;

}

// Function to print club assignment

static void printAssignments(Node min) {

if (min.parent == null) {

return;

}

printAssignments(min.parent);

System.out.println("Assign student " + (char) (min.studentID + 'A') + " to club " + (min.clubID + 1));

}

// Function to solve club Assignment Problem using Branch and Bound

static int findMinCost(int costMatrix[][], int N) {

PriorityQueue<Node> pq = new PriorityQueue<>(Comparator.comparingInt(node -> node.cost));

boolean assigned[] = new boolean[N];

Node root = newNode(-1, -1, assigned, null, N);

root.pathCost = root.cost = 0;

root.studentID = -1;

pq.add(root);

while (!pq.isEmpty()) {

Node min = pq.poll();

int i = min.studentID + 1;

if (i == N) {

printAssignments(min);

return min.cost;

}

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

if (!min.assigned[j]) {

Node child = newNode(i, j, min.assigned, min, N);

child.pathCost = min.pathCost + costMatrix[i][j];

child.cost = child.pathCost + calculateCost(costMatrix, i, j, child.assigned, N);

pq.add(child);

}

}

}

return 0;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Prompt for matrix size

System.out.print("Enter the number of students/clubs (N): ");

int N = sc.nextInt();

if (N <= 0) {

System.out.println("Invalid matrix size. Exiting.");

return;

}

// Initialize the cost matrix

int[][] costMatrix = new int[N][N];

// Prompt for matrix elements

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

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

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

System.out.print("Cost for student " + (char) ('A' + i) + " to club " + (j + 1) + ": ");

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

}

}

// Display the input matrix

System.out.println("\nCost Matrix:");

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

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

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

}

System.out.println();

}

// Solve the problem

System.out.println("\nOptimal Cost is " + findMinCost(costMatrix, N));

}

}