Q1:

Code:

import java.util.LinkedList;  
import java.util.Queue;  
import java.util.Scanner;  
  
  
public class PathExist {  
  
 public static boolean hasValidPath(int[][] adjacencyMatrix, int source, int destination) {  
 int n = adjacencyMatrix.length;  
 boolean[] visited = new boolean[n];  
  
 Queue<Integer> queue = new LinkedList<>();  
 queue.add(source);  
 visited[source] = true;  
  
 while (!queue.isEmpty()) {  
 int currentVertex = queue.poll();  
  
 for (int i = 0; i < n; i++) {  
 if (adjacencyMatrix[currentVertex][i] == 1 && !visited[i]) {  
 if (i == destination) {  
 return true; // Valid path found  
 }  
 queue.add(i);  
 visited[i] = true;  
 }  
 }  
 }  
  
 return false; // No valid path found  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.print("Enter the number of vertices: ");  
 int n = scanner.nextInt();  
  
 int[][] adjacencyMatrix = new int[n][n];  
  
 System.*out*.println("Enter the adjacency matrix: ");  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n; j++) {  
 adjacencyMatrix[i][j] = scanner.nextInt();  
 }  
 }  
  
 System.*out*.print("Enter the source vertex: ");  
 int source = scanner.nextInt();  
  
 System.*out*.print("Enter the destination vertex: ");  
 int destination = scanner.nextInt();  
  
 boolean result = *hasValidPath*(adjacencyMatrix, source, destination);  
  
 System.*out*.println(result);  
  
 scanner.close();  
 }  
}

Outpt:

A screen shot of a computer

Description automatically generated

Q2:

DOB: 17-03-1999

Last five digits: 31999

Values are

4

3

7

8

9

Again,

Adding 1 to all(1st didit)

wright

5

4

8

9

10

knapsack is the sum = 12

* b.

public class KnapsackDP {  
  
 public static int knapsack(int[] weights, int[] values, int maxWeight) {  
 int n = weights.length;  
 int[][] dp = new int[n + 1][maxWeight + 1];  
  
 for (int i = 1; i <= n; i++) {  
 for (int w = 1; w <= maxWeight; w++) {  
 if (weights[i - 1] <= w) {  
 dp[i][w] = Math.*max*(dp[i - 1][w], values[i - 1] + dp[i - 1][w - weights[i - 1]]);  
 } else {  
 dp[i][w] = dp[i - 1][w];  
 }  
 }  
 }  
  
 return dp[n][maxWeight];  
 }  
  
 public static void main(String[] args) {  
 int[] weights = {5, 4, 8, 9, 10};  
 int[] values = {4, 3, 7, 8, 9};  
 int maxWeight = 12;  
  
 int maxValue = *knapsack*(weights, values, maxWeight);  
  
 System.*out*.println("Maximum value: " + maxValue);  
 }  
}

output:

A screen shot of a computer

Description automatically generated

* C

import java.util.Arrays;  
import java.util.Comparator;  
  
class Item {  
 int weight;  
 int value;  
  
 public Item(int weight, int value) {  
 this.weight = weight;  
 this.value = value;  
 }  
}  
  
public class FractionalKnapsack {  
  
 public static double fractionalKnapsack(int[] weights, int[] values, int maxWeight) {  
 int n = weights.length;  
 Item[] items = new Item[n];  
  
 // Create an array of items with value-to-weight ratios  
 for (int i = 0; i < n; i++) {  
 items[i] = new Item(weights[i], values[i]);  
 }  
  
 // Sort items based on the value-to-weight ratio in descending order  
 Arrays.*sort*(items, Comparator.*comparingDouble*((Item item) -> (double) item.value / item.weight).reversed());  
  
 double maxValue = 0.0;  
 int remainingWeight = maxWeight;  
  
 // Greedily fill the knapsack  
 for (Item item : items) {  
 if (item.weight <= remainingWeight) {  
 // Take the whole item  
 maxValue += item.value;  
 remainingWeight -= item.weight;  
 } else {  
 // Take a fraction of the item  
 double fraction = (double) remainingWeight / item.weight;  
 maxValue += fraction \* item.value;  
 break; // Knapsack is full  
 }  
 }  
  
 return maxValue;  
 }  
  
 public static void main(String[] args) {  
 int[] weights = {5, 4, 8, 9, 10};  
 int[] values = {4, 3, 7, 8, 9};  
 int maxWeight = 12;  
  
 double maxValue = *fractionalKnapsack*(weights, values, maxWeight);  
  
 System.*out*.println("Maximum value fraction: " + maxValue);  
 }  
}