Sample Problem Number 1: The Return of the Roman Empire

Write a program that accepts a Roman numeral and converts it to decimal form. Remember, I=1, V=5, X=10, L=50, C=100, D=500, and M=1000. Furthermore, there are the following digraphs: IV=4, IX=9, XL=40, XC=90, CD=400, CM=900. The program should reject improperly formed numerals.

Sample Input:
MCMXCVIII
Program Output:
1998

Sample Input:
CCM
Program Output:
This is not a valid number
Sample Problem Number 2: To Boldly Go

Space ships in the 25th century are driven by "warp" drive. Unfortunately, if two warp drives operate simultaneously within 10 light years of each other both space ships blow up leaving a really nasty mess that has to be cleaned up. You are to implement software for space traffic control (STC) to insure the safe operation of space ships using warp drive. All space ships must file a flight plan with STC. STC reviews each request in the order it was received and issues flight clearance to the space ship if its operation will not interfere with any previously cleared space ships. If the space ship interferes with another cleared space ship then clearance is denied and the space ship is not allowed to operate.

A space ship that starts moving at the exact same instance when another space ship stops operation does not cause a problem. Your program is to read a number n, the number of clearance requests followed by n clearance requests, one per line. A clearance request consists of a string space ship name followed by x1, y1, z1, t1 coordinates of the origination point in space-time followed by x2, y2, z2, t2 coordinates of the destination coordinates. All distances are in light years and time is given in minutes. t2 is always greater than t1. The proposed flight plan starts at x1, y1, z1 at time t1 and moves at constant velocity to x2, y2, z2, arriving at t2.

Your program is to consider the clearance requests in order and to print for each one the name of the space ship followed by the word "granted" if the flight will never come within 10 light years of any previously cleared flight or the word "denied" if the proposed flight plan will come within 10 light years of another operating space ship.

Sample Input
2
ship1 0 0 0 0 15 0 0 1
ship2 0 15 0 0 0 0 0 1
Sample Output
ship1 granted
ship2 granted

Sample Input
2
ship1 0 0 0 0 14 0 0 1
ship2 0 14 0 0 0 0 0 1
Sample Output
ship1 granted
ship2 denied

Sample Input
3
ship1 0 0 0 0 14 0 0 1
ship2 0 14 0 0 0 0 0 1
ship3 0 14 0 0 0 15 0 1
Sample Output
ship1 granted
ship2 denied
ship3 granted

Sample Input
2
ship1 0 0 0 0 0 0 0 1
ship2 11 0 0 0 11 0 0 1
Sample Output
ship1 granted
ship2 granted
Sample Problem 3: Swamp Things

Hugh F. Oh, in his never-ending quest to prove the existence of extraterrestrials, has gotten hold of a number of nighttime photographs taken by a research group that is examining glowing swamp gas.

Hugh wants to see if any of the photos show, not swamp gas, but Little Grey Men in glowing suits. The photographs consist of bright dots appearing against a black background. Unfortunately, at the time the photos were taken, trains were traveling through the area (there is a train trestle over the swamp), and occasional lights from the train windows also appear in the photographs. Hugh, being a fastidious researcher, wants to eliminate these spots from the images. He can’t tell from the photos exactly where the tracks are, or from what direction the photos were taken, but he knows that the tracks in that area are perfectly straight, so he’s decided on the following approach: he will find the line with the maximum number of spots lying on it and, if there are four or more spots on the line, he will eliminate those points from his calculations, assuming that those are windows on the train. If two or more lines have the maximum number of points, Hugh will just randomly select one such set and delete it from the photo (he’s not all that fastidious – after all, he believes in Little Grey Men). If there are fewer than four points lying along a common line, Hugh will assume that there is no train in the photograph and won’t delete any points. Please write a program for him to process a set of photographs.

Input
There will be a series of test cases. Each test case is one photograph described by a line containing a positive integer n (<= 1000), the number of distinct spots in the photograph, followed by n lines containing the integer coordinates of the spots, one (x, y) pair per line. All coordinates are between 0 and 10000. The last photo description is followed by a line containing a zero, marking the end of the input. This line should not be processed.

Output
For each test case, output the photo number followed by the number of points eliminated from the photograph. Imitate the sample output below.

Sample Input
6
0 1
0 2
1 2
2 2
4 5
5 6
4
3 5
4 4
6 5
7 4
0

Sample Output
Photo 1: 4 points eliminated
Photo 2: 0 points eliminated
Sample Problem 4: Queens, Knights and Pawns

You all are familiar with the famous 8-queens problem which asks you to place 8 queens on a chess board so no two attack each other. In this problem, you will be given locations of queens and knights and pawns and asked to find how many of the unoccupied squares on the board are not under attack from either a queen or a knight (or both). We’ll call such squares “safe” squares. Here, pawns will only serve as blockers and have no capturing ability. The board below has 6 safe squares. (The shaded squares are safe.)

Recall that a knight moves to any unoccupied square that is on the opposite corner of a 2x3 rectangle from its current position; a queen moves to any square that is visible in any of the eight horizontal, vertical, and diagonal directions from the current position. Note that the movement of a queen can be blocked by another piece, while a knight’s movement can not.

Input
There will be multiple test cases. Each test case will consist of 4 lines. The first line will contain two integers n and m, indicating the dimensions of the board, giving rows and columns, respectively. Neither integer will exceed 1000. The next three lines will each be of the form

\[ k \ r1 \ c1 \ r2 \ c2 \cdots \ rk \ ck \]

indicating the location of the queens, knights and pawns, respectively. The numbering of the rows and columns will start at one. There will be no more than 100 of any one piece. Values of \( n = m = 0 \) indicate end of input.

Output
Each test case should generate one line of the form

Board b has s safe squares.

where \( b \) is the number of the board (starting at one) and you supply the correct value for \( s \).

Sample Input
4 4
2 1 4 2 4
1 1 2
1 2 3
2 3
1 1 2
1 1 1
0
1000 1000
1 3 3
0
0
0

Sample Output
Board 1 has 6 safe squares.
Board 2 has 0 safe squares.
Board 3 has 996998 safe squares.
SOLUTIONS FOR PROBLEM 3

C++

// Swamp.cpp : Defines the entry point for the console application.
//

#include <iostream>
using namespace std;

const int SIZE = 1000;

struct point {
    int x, y;
};

int main()
{
    int i, j, k, icase=0;
    cin >> npoints;
    while (npoints != 0) {
        icase++;
        for(i=0; i<npoints; i++) {
            cin >> points[i].x >> points[i].y;
        }
        int max = 0;
        for(i=0; i<npoints-2; i++) {
            for(j=i+1; j<npoints-1; j++) {
                // determine eqn ax+by=c
                int a = points[i].y - points[j].y;
                int b = points[j].x - points[i].x;
                int c = points[j].x*points[i].y - points[j].y*points[i].x;
                int count = 2;
                for(k=j+1; k<npoints; k++) {
                    if (a*points[k].x + b*points[k].y == c) {
                        count++;
                    }
                }
                if (count > max)
                    max = count;
            }
        }
        if (max >= 4)
            cout << "Photo " << icase << ": " << max << " points eliminated" << endl;
        else
            cout << "Photo " << icase << ": 0 points eliminated" << endl;
        cin >> npoints;
    }
    return 0;
}

////////////////
SOLUTIONS FOR PROBLEM 3

JAVA

// Solution to "Swamp Things" by Bob Roos
// This is a straightforward triple-nested-loop with no attempt at
// improving efficiency; at n = 1000 it crawls.

import java.io.*;
import java.util.*;
public class SwampFinal {
    public static BufferedReader in;
    public static int photoNum;
    public static int n;
    public static int[] x;
    public static int[] y;

    public static int process() {
        int maxCount = 0;
        for (int i = 0; i < n-1; i++) {
            for (int j = i+1; j < n; j++) {
                int rise = y[j] - y[i];
                int run = x[j] - x[i];
                for (int k = j+1; k < n; k++) {
                    if (k == i || k == j)
                        continue;
                    int risel = y[k] - y[i];
                    int run1 = x[k] - x[i];
                    if (rise*run1 == risel*run)
                        count++;
                }
            }
        }
        return maxCount;
    }

    public static void main(String[] args) throws IOException {
        in = new BufferedReader(new InputStreamReader(System.in));
        photoNum = 0;
        String line;
        while (!((line = in.readLine().trim()).equals("0"))) {
            photoNum++;
            n = Integer.parseInt(line);
            x = new int[n];
            y = new int[n];
            for (int i = 0; i < n; i++) {
                line = in.readLine().trim();
                StringTokenizer tok = new StringTokenizer(line);
                x[i] = Integer.parseInt(tok.nextToken());
                y[i] = Integer.parseInt(tok.nextToken());
            }
            System.out.println("Photo "+ photoNum + ": "+ process() + " points eliminated");
        }
    }
}

public class SwampFinal {

    public static BufferedReader in;
    public static int photoNum;
    public static int n;
    public static int[] x;
    public static int[] y;

    public static void main(String[] args) throws IOException {
        in = new BufferedReader(new InputStreamReader(System.in));
        photoNum = 0;
        String line;
        while (!((line = in.readLine().trim()).equals("0"))) {
            photoNum++;
            n = Integer.parseInt(line);
            x = new int[n];
            y = new int[n];
            for (int i = 0; i < n; i++) {
                line = in.readLine().trim();
                StringTokenizer tok = new StringTokenizer(line);
                x[i] = Integer.parseInt(tok.nextToken());
                y[i] = Integer.parseInt(tok.nextToken());
            }
            System.out.println("Photo "+ photoNum + ": "+ process() + " points eliminated");
        }
    }
}
maxCount = count;
}
}
return maxCount;
}
SOLUTIONS FOR PROBLEM 4 in C++

```cpp
#include <iostream>
using namespace std;

const int MAXSIZE = 1000;

char board[MAXSIZE+4][MAXSIZE+4]; // +2 to allow for easy processing of knight moves
bool safe[MAXSIZE+4][MAXSIZE+4];
int numsafe;

void process(int i, int j)
{
    if (safe[i][j])
        numsafe--;
    safe[i][j] = false;
}

int main()
{
    int n, m, i, j, k;
    int nboard = 0;
    cin >> n >> m;
    while (n != 0) {
        nboard++;
        for(i=2; i<n+2; i++) {
            for(j=2; j<m+2; j++) {
                board[i][j] = ' ';  
                safe[i][j] = true;
            }
        }
        for(i=0; i<n+4; i++) {
            board[i][0] = board[i][1] = board[i][m+2] = board[i][m+3] = '+';
            safe[i][0] = safe[i][1] = safe[i][m+2] = safe[i][m+3] = false;
        }
        for(j=0; j<m+4; j++) {
            board[0][j] = board[1][j] = board[n+2][j] = board[n+3][j] = '+';
            safe[0][j] = safe[1][j] = safe[n+2][j] = safe[n+3][j] = false;
        }
        numsafe = n*m;
        int num;
        cin >> num;
        for(i=0; i<num; i++) {
            int r, c;
            cin >> r >> c;
            board[r+1][c+1] = 'q';
        }
        cin >> num;
        for(i=0; i<num; i++) {
            int r, c;
            cin >> r >> c;
            board[r+1][c+1] = 'k';
        }
        cin >> num;
        for(i=0; i<num; i++) {
            int r, c;
            cin >> r >> c;
            board[r+1][c+1] = 'p';
        }
        for(i=2; i<n+2; i++) {
            for(j=2; j<m+2; j++) {
                switch (board[i][j]) {
                    case 'p':
```
process(i, j);
break;

case 'k' :
    process(i, j);
    process(i+1, j+2);
    process(i+1, j-2);
    process(i-1, j+2);
    process(i-1, j-2);
    process(i+2, j+1);
    process(i+2, j-1);
    process(i-2, j+1);
    process(i-2, j-1);
    break;

    for(k=i-1; k>=2; k--) {
        if (board[k][j] != ' ')
            break;
    }
    for(k=i+1; k<n+2; k++) {
        if (board[k][j] != ' ')
            break;
    }
    for(k=j-1; k>=2; k--) {
        if (board[i][k] != ' ')
            break;
    }
    for(k=j+1; k<m+2; k++) {
        if (board[i][k] != ' ')
            break;
    }
    for(k=i-1; k>=2; k--) {
        if (board[k][j+k-i] != ' ')
            break;
    }
    for(k=i+1; k<n+2; k++) {
        if (board[k][j+k-i] != ' ')
            break;
    }
    break;
}

cout << "Board " << nboard << " has " << numsafe << " safe squares." << endl;
cin >> n >> m;

return 0; }