acm International Collegiate Programming Contest 2008 IBM event sponsor

Soluciones y Explicación de los problemas ACM International Collegiate Programming Contest 2008

South American Regional Contests

Autor M.Sc. Jorge Teran

Elaboración: Enero, 2009

Version 2

Agradecimiento por los aportes al solucioanrio

Estas soluciones se han realizado con el aporte de las siguientes personas:

- **Hernán Payrumani** que contribuyó con las soluciones de Candy, DNA Sequences, Travel Salesman Problem, Poligons y Bases.
  email: hpayrumani@acm.org

- **Gabriel Rea Velasco** que contribuyó con las soluciones de Bora Bora y Higgs Boson.
  email: gareve25@hotmail.com

Introducción

El South American Regional Contests organizado por ACM International Collegiate Programming Contest año 2008 se realizó en la ciudad de Cochabamba bajo el auspicio de la Universidad Mayor de San Simón. En dicho concurso se presentaron dos ejercicios para la sesión de práctica y 11 para la competencia. Estos problemas fueron:

Sesión de práctica:

1. Og
2. He is offside!

Competencia:

1. Almost Shortest Path
2. Candy
3. DNA Subsequences
4. Electricity
5. Feynman
6. Pole Position
7. Higgs Boson
8. Bora Bora
9. Shrinking Polygons
10. Traveling Shoemaker Problem

El documento ha sido organizado como sigue:

- Enunciado tal como fue presentado en el concurso.
- Explicación de la solución propuesta.
- Programa en Java, C o ambos.

Espero que éste solucionario sea útil para seguir incursionando en esta apasionante área de resolver problemas. Para mejorar y completar este solucionario, espero sus comentarios mi correo es teranj@acm.org

M.Sc. Jorge Teran P.

Problem A
Og (Warmup)

Source file name: `og.c`, `og.cpp` *or* `og.java`

Og is a caveman with many children, and he wants to count them all. Og counts his sons with his left hand and his daughters with his right hand.

However, Og is very dumb, and can't add the two counts, so he asked you to write him a program that will do the addition.

Input

The input contains several test cases. Each test case consists of a single line containing two integers L and R, separated by a single space, indicating respectively the number of sons and daughters ($1 \leq L, R \leq 5$).

The end of input is indicated by a line containing only two zeros, separated by a single space.

The input must be read from file `og.in`.

Output

For each test case in the input print a line containing a single integer indicating how many children Og has.

The output must be written to standard output.

Sample input	**Output for the sample input**
`2 2`	`4`
`2 3`	`5`
`5 5`	`10`
`1 1`	`2`
`0 0`	

Análisis del Problema A

Este problema es muy sencillo, solo pide que se sumen los valores leídos y se imprimen en la pantalla, no requiere ninguna explicación adicional.

Programa Java que soluciona el problema

```
import java.io.*;
import java.util.*;

class Main {
public static void main(String[] args) throws Exception {
InputStreamReader stdin = new InputStreamReader(System.in);
Scanner s = new Scanner(stdin);
        while (true) {
             int A = s.nextInt();
             int B = s.nextInt();
             if (A == 0 && B == 0) break;
             System.out.println(A + B);
             }
        }
}
```

Programa C que soluciona el problema

```
#include <stdio.h>

int L, R;

int main(){
    while (scanf(" %d %d", &L, &R) && !(L == 0 && R == 0)) {
        printf("%d\n", L + R);
        }
    return 0;
}
```

Problem B
He is offside! (Warmup)

Source file name: `he.c`, `he.cpp` *or* `he.java`

Hemisphere Network is the largest television network in Tumbolia, a small country located east of South America (or south of East America). The most popular sport in Tumbolia, unsurprisingly, is soccer; many games are broadcast every week in Tumbolia.

Hemisphere Network receives many requests to replay dubious plays; usually, these happen when a player is deemed to be offside by the referee. An attacking player is *offside* if he is nearer to his opponents' goal line than the second last opponent. A player is not offside if

- he is level with the second last opponent or
- he is level with the last two opponents.

Through the use of computer graphics technology, Hemisphere Network can take an image of the field and determine the distances of the players to the defending team's goal line, but they still need a program that, given these distances, decides whether a player is offside.

Input

The input file contains several test cases. The first line of each test case contains two integers A and D separated by a single space indicating, respectively, the number of attacking and defending players involved in the play ($2 \leq A, D \leq 11$). The next line contains A integers B_i separated by single spaces, indicating the distances of the attacking players to the goal line ($1 \leq B_i \leq 10^4$). The next line contains D integers C_j separated by single spaces, indicating the distances of the defending players to the goal line ($1 \leq C_j \leq 10^4$).

The end of input is indicated by a line containing only two zeros, separated by a single space.

The input must be read from file he.in*.*

Output

For each test case in the input print a line containing a single character: "`Y`" (uppercase) if there is an attacking player offside, and "`N`" (uppercase) otherwise.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
2 3
500 700
700 500 500
2 2
200 400
200 1000
3 4
530 510 490
480 470 50 310
0 0
```
	```
	N
	Y
	N
	```

Análisis del Problema B

Este problema es muy sencillo, fue propuesto el 2007 y se usó de problema de entrenamiento para el 2008. Analizando los datos de entrada nos damos cuenta que existen un máximo de 11 atacantes y 11 defensores.

Ordenando los datos de acuerdo a la distancia, podemos comparar la distancia del primer atacante con la distancia del segundo defensor. Esto corresponde a las posiciones 0 y 1. más detalle puede ver en el código adjunto.

Programa Java que soluciona el problema

```
import java.io.*;
import java.util.*;

class main {
static int[] ataque = new int[11], defensa = new int[11];
public static final void main(String[] args) throws Exception {
Scanner s = new Scanner(new InputStreamReader(System.in));

while (true) {
int A = s.nextInt();
int D = s.nextInt();

if (A == 0 && D == 0)
break;

for (int i = 0; i < A; i++) {
ataque[i] = s.nextInt();
}
for (int j = 0; j < D; j++) {
defensa[j] = s.nextInt();
}
Arrays.sort(ataque);
Arrays.sort(defensa);
if (ataque[0] < defensa[1]) {
System.out.println("Y");
} else {
System.out.println("N");
}
}
}
}
```

Programa C que soluciona el problema

```
#include <stdio.h>
#include <algorithm>

int A, D;
int B[11];
int C[11];

using namespace std;

int main()
{
 int i;
 //freopen("he.in", "r", stdin);
 scanf("%d %d", &A, &D);
 while(A || D)
 {
  for(i=0; i<A; i++)
   scanf("%d", B + i);
  for(i=0; i<D; i++)
   scanf("%d", C + i);
  sort(B, B + A);
  sort(C, C + D);
  printf("%c\n", B[0] < C[1] ? 'Y' : 'N');
  scanf("%d %d", &A, &D);
 }
 return 0;
}
```

Problem C
Almost Shortest Path

Source file name: `almost.c`, `almost.cpp` *or* `almost.java`

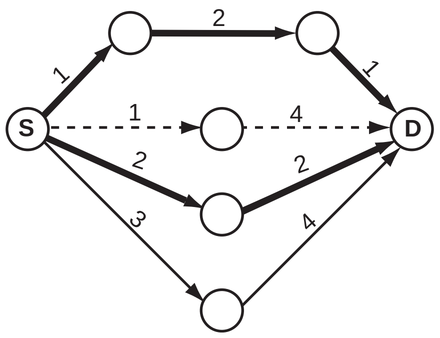
Finding the shortest path that goes from a starting point to a destination point given a set of points and route lengths connecting them is an already well known problem, and it's even part of our daily lives, as shortest path programs are widely available nowadays.

Most people usually like very much these applications as they make their lives easier. Well, maybe not that much easier.

Now that almost everyone can have access to GPS navigation devices able to calculate shortest paths, most routes that form the shortest path are getting slower because of heavy traffic. As most people try to follow the same path, it's not worth it anymore to follow these directions.

With this in his mind, your boss asks you to develop a new application that only he will have access to, thus saving him time whenever he has a meeting or any urgent event. He asks you that the program must answer not the shortest path, but the almost shortest path. He defines the almost shortest path as the shortest path that goes from a starting point to a destination point such that no route between two consecutive points belongs to any shortest path from the starting point to the destination.

For example, suppose the figure below represents the map given, with circles representing location points, and lines representing direct, one-way routes with lengths indicated. The starting point is marked as `S` and the destination point is marked as `D`. The bold lines belong to a shortest path (in this case there are two shortest paths, each with total length 4). Thus, the almost shortest path would be the one indicated by dashed lines (total length 5), as no route between two consecutive points belongs to any shortest path. Notice that there could exist more than one possible answer, for instance if the route with length 3 had length 1. There could exist no possible answer as well.



Input

The input contains several test cases. The first line of a test case contains two integers N ($2 \leq N \leq 500$) and M ($1 \leq M \leq 10^4$), separated by a single space, indicating respectively the number of points in the map and the number of existing one-way routes connecting two points directly. Each point is identified by an integer between 0 and $N-1$. The second line contains two integers S and D, separated by a single space, indicating respectively the starting and the destination points ($S \neq D$; $0 \leq S, D < N$). Each one of the following M lines contains three integers U, V and P ($U \neq V$; $0 \leq U, V < N$; $1 \leq P \leq 10^3$), separated by single spaces, indicating the existence of a one-way route from U to V with distance P. There is at most one route from a given point U to a given point V, but notice that the existence of a route from U to V does not imply there is a route from V to U, and, if such road exists, it can have a different length. The end of input is indicated by a line containing only two zeros separated by a single space.

The input must be read from file almost.in.

Output

For each test case in the input, your program must print a single line, containing `-1` if it is not possible to match the requirements, or an integer representing the length of the almost shortest path found.

The output must be written to standard output.

Sample input

```
7 9
0 6
0 1 1
0 2 1
0 3 2
0 4 3
1 5 2
2 6 4
3 6 2
4 6 4
5 6 1
4 6
0 2
0 1 1
1 2 1
1 3 1
3 2 1
2 0 3
3 0 2
6 8
0 1
0 1 1
0 2 2
0 3 3
2 5 3
3 4 2
4 1 1
5 1 1
3 0 1
0 0
```

Output for the sample input

```
5
-1
6
```

Análisis del Problema C

Para poder resolver éste problema es necesario tener conocimientos previos en la teoría de grafos. Como hallar todos los caminos más cortos y hallar caminos intermedios.

En la solución presentada, se utiliza el algoritmo de Dijkstra. Solo se presenta la explicación de este algoritmo.

Representamos los nodos con una matriz de distancia:

	0	1	2	3	4	5	6
0	0	1	1	2	3	0	0
1	0	0	0	0	0	2	0
2	0	0	0	0	0	0	4
3	0	0	0	0	0	0	2
4	0	0	0	0	0	0	4
5	0	0	0	0	0	0	1
6	0	0	0	0	0	0	0

Creamos un vector de distancia iniciado en infinito. En el código se puede poner un número muy grande, o el máximo valor aceptable.

0	1	2	3	4	5	6
inf	inf	inf	inf	inf	inf	inf

Luego se crea un vector donde se marcará cada uno de los nodos procesados. Inicialmente se fijan estos valores en falso.

0	1	2	3	4	5	6
false	false	false	false	false	false	false
0	0	0	0	0	0	0

Se comienza con el origen y anotamos las distancias a los lugares directamente conectados. Si la distancia es menor a la registrada se cambia, por esto se inicializó todo en infinito.

En la primera iteración el vector de distancias queda como sigue:

0	1	2	3	4	5	6
false	false	false	false	false	false	false
0	1	1	2	3	inf	inf

Se escoge el mínimo y se marca como procesado verdadero.

0	1	2	3	4	5	6
true	false	false	false	false	false	false
0	1	1	2	3	inf	inf

continuamos del mínimo encontrado y continuamos con el proceso en forma iterativa obteniendo:

0	1	2	3	4	5	6
true	false	false	false	false	false	false
0	1	1	2	3	3	inf

Veamos que en el nodo 5 se ha hallado el valor 3 que es la suma del costo para llegar a 1 más el costo de llegar de 1 a 5, dando 3. Tomamos el valor mínimo en entre infinito y 3.

Marcamos el mínimo como procesado. Continuando el proceso cuando se han marcado todos como procesados, llegamos al resultado:

0	1	2	3	4	5	6
true	true	true	true	true	true	true
0	1	1	2	3	3	4

El valor mínimo para llegar al nodo 6 es 4.

Este algoritmo se ha utilizado en la solución para hallar el resultado. Primero se halla el mínimo, luego se hace lo mismo con la matriz transpuesta, y finalmente con ambas soluciones, se crea una matriz que nos permitirá hallar los valores próximos al mínimo, que es el resultado pedido.

Programa Java que soluciona el problema

```
/*
 * Problema: Almost
 * Solution by Alessandro de Luna Almeida
 *
 * Dijkstra
 *
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.StreamTokenizer;
import java.util.Arrays;


public class sol {
    static int m, n, s, d, grafo[][], grafo2[][];
    static StreamTokenizer in;
    static PrintStream out;
    static int[] dist, dist2;

    static boolean readdata() throws IOException{
        in.nextToken();
        n = (int)in.nval;
        if(n==0)
            return false;
        in.nextToken();
        m = (int)in.nval;
        in.nextToken();
        s = (int)in.nval;
        in.nextToken();
        d = (int)in.nval;
        grafo = new int[n][n];
        grafo2 = new int[n][n];
        dist = new int[n];
        dist2 = new int[n];
        Arrays.fill(dist,Integer.MAX_VALUE/3);
        Arrays.fill(dist2,Integer.MAX_VALUE/3);
        return true;
}

    static void process() throws IOException{
        for(int i =0;i<m;i++){
            in.nextToken();
            int u = (int)in.nval;
            in.nextToken();
            int v = (int)in.nval;
            in.nextToken();
            int p = (int)in.nval;
            grafo[u][v]=p;
            grafo2[v][u]=p;
        }
        dijkstra(s,grafo,dist);
        int mindist = dist[d];
        if(mindist == Integer.MAX_VALUE/3) {
            out.println(-1);
            return;
        }
        dijkstra(d,grafo2,dist2);
        for(int i =0;i<n;i++){
            for(int j=0;j<n;j++){
                if(grafo[i][j]>0){
                    if(dist[i] + grafo[i][j] + dist2[j] == mindist)
                        grafo[i][j] = 0;
                }
            }
        }
        Arrays.fill(dist,Integer.MAX_VALUE/3);
        dijkstra(s,grafo,dist);
        out.println(dist[d]==Integer.MAX_VALUE/3?-1:dist[d]);
    }

    static void dijkstra(int source, int[][] g, int[] dis) {
        boolean mark[] = new boolean[n];
        dis[source] = 0;
        int v;
        while((v = getmin(dis, mark))!= -1){
            mark[v] = true;
            for(int w = 0; w<n;w++){
                if(g[v][w]>0 && !mark[w] && (dis[v] + g[v][w] < dis[w])) {
                    dis[w] = dis[v] + g[v][w];
                }
            }
        }
    }

    static int getmin(int[] dis, boolean[] mark) {
        int ret = -1, min = Integer.MAX_VALUE;
        for(int i = 0; i<n; i++) {
            if(!mark[i] && dis[i] < min) {
                min = dis[i];
                ret = i;
            }
        }
        return ret;
    }

    public static void main(String[] args) throws IOException {
        in = new StreamTokenizer(new BufferedReader(
                new InputStreamReader(System.in)));
        out = System.out;
        while(readdata())
            process();
        out.close();
    }
}
```

Problem D
Bases

Source file name: `bases.c`, `bases.cpp` *or* `bases.java`

What do you get if you multiply 6 by 9? The answer, of course, is 42, but only if you do the calculations in base 13.

Given an integer $B \geq 2$, the *base B numbering system* is a manner of writing integers using only digits between 0 and $B-1$, inclusive. In a number written in base B, the rightmost digit has its value multiplied by 1, the second rightmost digit has its value multiplied by B, the third rightmost digit has its value multiplied by B^2, and so on.

Some equations are true or false depending on the base they are considered in. The equation $2+2=4$, for instance, is true for any $B \geq 5$ — it does not hold in base 4, for instance, since there is no digit '4' in base 4. On the other hand, an equation like $2+2=5$ is never true.

Write a program that given an equation determines for which bases it holds.

Input

Each line of the input contains a test case; each test case is an equation of the form "`EXPR=EXPR`", where both "`EXPR`" are arithmetic expressions with at most 17 characters.

All expressions are valid, and contain only the characters '`+`', '`*`' and the digits from '`0`' to '`9`'. No expressions contain leading plus signs, and no numbers in it have leading zeros.

The end of input is indicated by a line containing only "`=`".

The input must be read from file `bases.in`.

Output

For each test case in the input your program should produce a single line in the output, indicating for which bases the given equation holds.

If the expression is true for infinitely many bases, print "`B+`", where B is the first base for which the equation holds.

If the expression is valid only for a finite set of bases, print them in ascending order, separated by single spaces.

If the expression is not true in any base, print the character '`*`'.

The output must be written to standard output.

Sample input	**Output for the sample input**
`6*9=42`	`13`
`10000+3*5*334=3*5000+10+0`	`6 10`
`2+2=3`	`*`
`2+2=4`	`5+`
`0*0=0`	`2+`
`=`	

Análisis del Problema D

Recordemos como pasar de una base x a base decimal, por ejemplo sea $x = 2$ base binaria, tenemos `10101111` y deseamos pasarlo a decimal:

$$1 * 2^7 + 0 * 2^6 + 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 \tag{1}$$

el resultado de (1) es $128 + 0 + 32 + 0 + 8 + 4 + 2 + 1 = 175$

Sea $x = 8$ base octal deseamos obtener la equivalencia de `257` en base decimal:

$$2 * 8^2 + 5 * 8^1 + 7 * 8^0 \tag{2}$$

el resultado de (2) es $128 + 40 + 7 = 175$

Ahora un polinomio de grado n es una función $P_{(x)}$ de la forma:

$P_{(x)} = \sum_{k=0}^{k=n} c_k * x^k$ - Definición de Horner

$P_{(x)} = c_n x^n + c_{n-1} x^{n-1} + \ldots + c_2 x^2 + c_1 x^1 + c_0 x^0$

Donde c_k es una constante

Como se aprecia existe una similitud entre la representación de un polinomio y la representación de una base x en base decimal. Siguiendo la descripción del problema tenemos al termino independiente 7 multiplicado por 1 o por B^0, luego a 5 multiplicado por B y finalmente a 2 multiplicado por B^2 donde $B = 8$.

En nuestro problema tenemos una ecuación de la forma $P_{(x)} = Q_{(x)}$.

El problema se reduce a buscar todas las raíces enteras del polinomio donde x es la base.

El algoritmo es el siguiente:

- Expander cada numero, ejemplo $432 = 4x^2 + 3x^1 + 2$.
- Multiplicar o sumar el polinomio en cada lado de la ecuación.
- Igualar la ecuación a cero, $P_{(x)} - Q_{(x)} = 0$.
- Y finalmente buscamos las raíces enteras del polinomio resultante.

Tenemos $6 * 9 = 42$

Lo expandimos: $6x^0 * 9x^0 = 4x^1 + 2x^0$

Multiplicamos y sumamos: $54 = 4x + 2$

Igualamos a cero: $54 - 4x - 2 = 0 \rightarrow 52 - 4x = 0$

Y encontramos que: $x = 13$

Programa C que soluciona el problema

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX 17

typedef struct {
int c[MAX];
} poly;

poly poly_add(poly A, poly B) {
    int i;
    poly R;
    for (i = 0; i < MAX; i++) {
        R.c[i] = A.c[i] + B.c[i];
    }
    return R;
}

poly poly_mul(poly A, poly B) {
    int i, j, k;
    poly R;
    memset(&R, 0, sizeof(R));
    for (i = 0; i < MAX; i++) {
        for (j = 0; j < MAX; j++) {
            k = i - j;
            if (k < 0 || MAX <= k) continue;
            R.c[i] += A.c[j] * B.c[k];
        }
    }
    return R;
}

poly poly_read(char* p) {
    char *q;
    int i, d;
    poly acc, cur, tmp;
    memset(&acc, 0, sizeof(acc));
    memset(&cur, 0, sizeof(cur)); cur.c[0] = 1;
    q = p;
    do {
        p++;
        if (!isdigit(*p)) {
            d = p - q;
            memset(&tmp, 0, sizeof(tmp));
            for (i = 0; i < d; i++) {
                tmp.c[i] = *(p - (i+1)) - '0';
            }
            cur = poly_mul(cur, tmp);
            if (*p != '*') {
                acc = poly_add(acc, cur);
                memset(&cur, 0, sizeof(cur)); cur.c[0] = 1;
            }
            q = p+1;
        }
    } while (*p != '=' && *p != '\0');
    return acc;
}

int div[1920], ndiv;

int cmp_int(const void *p, const void *q) {
    return *(int*)p - *(int*)q;
}

void find_div(int N) {
    int i, p, old;
    if (N < 0) N = -N;
    ndiv = 0;
    div[ndiv++] = 1;
    for (p = 2; p <= 46337 && p*p <= N; p++) {
        old = ndiv;
        while (N % p == 0) {
            for (i = 0; i < old; i++) {
                div[ndiv] = div[ndiv - old] * p;
                ndiv++;
            }
            N /= p;
        }
    }
    if (N > 1) {
        old = ndiv;
        for (i = 0; i < old; i++) {
            div[ndiv] = div[ndiv - old] * N;
            ndiv++;
        }
    }
    qsort(div, ndiv, sizeof(int), cmp_int);
}

poly lhs, rhs;
```

```
int check(int x) {
    int acc, i;
    acc = 0;
    for (i = 0; i < MAX; i++) {
        acc /= x;
        acc += lhs.c[i];
        if (acc % x != 0) {
            return 0;
        }
    }
    return (acc == 0);
}

int main() {
    int TC;
    int h, i, j, max, fail;
    char *p;
    char buf[2*MAX+2];

    for (;;) {
        gets(buf);
        if (buf[0] == '=') {
            break;
        }
        max = 1;
        for (p = buf; *p; p++) {
            if (*p == '=') {
                continue;
            }
            if (*p - '0' > max) {
                max = *p - '0';
            }
        }
        lhs = poly_read(buf);
        for (p = buf; *p != '='; p++)
            ;
        rhs = poly_read(++p);
        for (i = 0; i < MAX; i++) {
            lhs.c[i] -= rhs.c[i];
        }
        for (j = 0; j < MAX; j++) {
            if (lhs.c[j]) break;
        }
        if (j == MAX) {
            printf("%d+\n", max+1);
            continue;
        }
        memcpy(&rhs, &lhs, sizeof(rhs));
        memset(&lhs, 0, sizeof(lhs));
        for (i = 0; i < MAX - j; i++) {
            lhs.c[i] = rhs.c[i+j];
        }
        find_div(lhs.c[0]);
        fail = 1;
        for (i = 0; i < ndiv; i++) {
            if (div[i] <= max)
                continue;
            if (check(div[i])) {
                if (!fail) {
                    printf(" ");
                }
                fail = 0;
                printf("%d", div[i]);
            }
        }
        if (fail) {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

Problem E
Candy

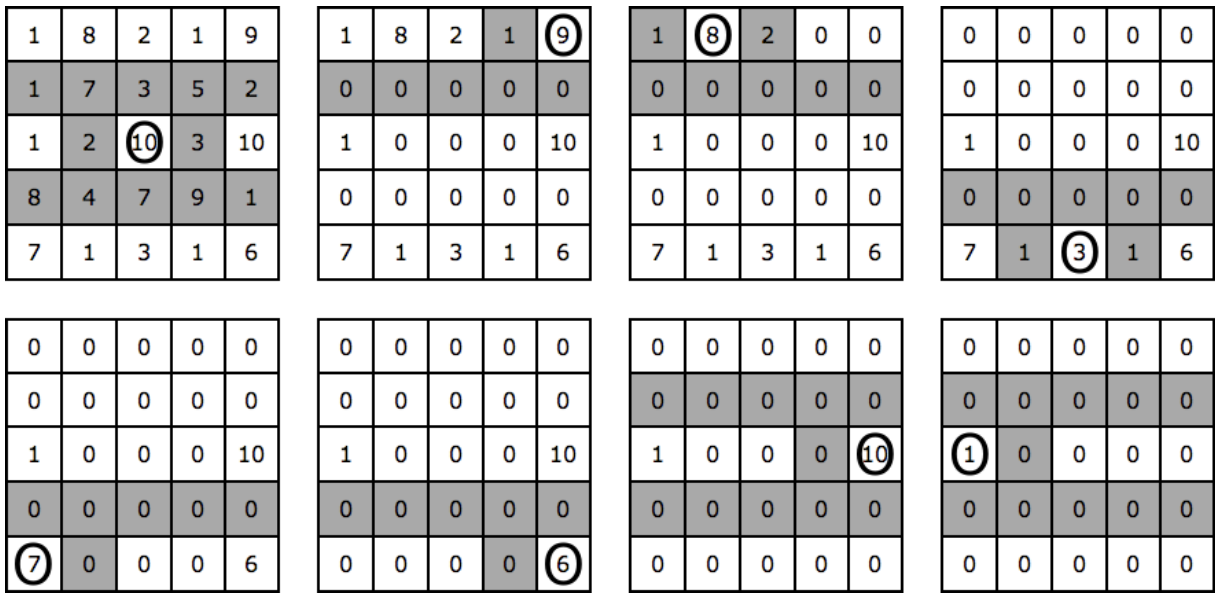
Source file name: `candy.c`, `candy.cpp` *or* `candy.java`

Little Charlie is a nice boy addicted to candies. He is even a subscriber to All Candies Magazine and was selected to participate in the International Candy Picking Contest.

In this contest a random number of boxes containing candies are disposed in M rows with N columns each (so, there are a total of $M \times N$ boxes). Each box has a number indicating how many candies it contains.

The contestant can pick a box (any one) and get all the candies it contains. But there is a catch (there is always a catch): when choosing a box, all the boxes from the rows immediately above and immediately below are emptied, as well as the box to the left and the box to the right of the chosen box. The contestant continues to pick a box until there are no candies left.

The figure bellow illustrates this, step by step. Each cell represents one box and the number of candies it contains. At each step, the chosen box is circled and the shaded cells represent the boxes that will be emptied. After eight steps the game is over and Charlie picked $10 + 9 + 8 + 3 + 7 + 6 + 10 + 1 = 54$ candies.



For small values of M and N, Charlie can easily find the maximum number of candies he can pick, but when the numbers are really large he gets completely lost. Can you help Charlie maximize the number of candies he can pick?

Input

The input contains several test cases. The first line of a test case contains two positive integers M and N ($1 \leq M \times N \leq 10^5$), separated by a single space, indicating the number of rows and columns respectively. Each of the following M lines contains N integers separated by single spaces, each representing the initial number of candies in the corresponding box. Each box will have initially at least 1 and at most 10^3 candies.

The end of input is indicated by a line containing two zeroes separated by a single space.

The input must be read from file candy.in*.*

Output

For each test case in the input, your program must print a single line, containing a single value, the integer indicating the maximum number of candies that Charlie can pick.

The output must be written to standard output.

Sample input

```
5 5
1 8 2 1 9
1 7 3 5 2
1 2 10 3 10
8 4 7 9 1
7 1 3 1 6
4 4
10 1 1 10
1 1 1 1
1 1 1 1
10 1 1 10
2 4
9 10 2 7
5 1 1 5
0 0
```

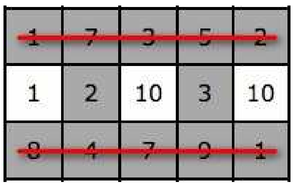
Output for the sample input

```
54
40
17
```

Análisis del Problema E

Es posible ver que es el mismo problema para cada una de las filas, que si se tomaran todas las filas a la vez. Ya que al elegir una fila, descartamos sus filas adyacentes.

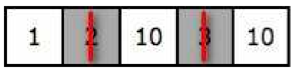
Si tomamos la tercera fila del ejemplo, descartamos la segunda y la cuarta fila.



~~1~~	~~7~~	~~3~~	~~5~~	~~2~~
1	2	10	3	10
~~8~~	~~4~~	~~7~~	~~9~~	~~1~~

Ya hemos reducido el problema a buscar la solución para una sola fila.

Cuando escogemos una celda de la fila no es posible elegir ninguna de sus celdas adyacentes.



1	~~2~~	10	~~3~~	10

Si tenemos nuestra fila de dulces en un vector `D[]` entonces la cantidad máxima que podemos escoger es `D[actual] + D[actual - 2]` pero solo para la posición actual de la celda, y debemos recorrer toda la fila para ver cual es la cantidad máxima a escoger.

Para resolver el problema utilizaremos Programación Dinámica.

En el vector `D[]` se tiene una secuencia de enteros $E : e_1, e_2, e_3 \ldots e_n$, de donde deseamos buscar la respuesta para la subsecuencia $e_1 \ldots e_k$, esto nos lleva a una recursión que toma la siguiente forma `D[k] = max(D[k - 1], D[k] + D[k - 2])`, poniéndolo en palabras es guardar en la posición actual el valor máximo entre un valor ya calculado (`D[k - 1]`) y el calculado para la posición actual (`D[k] + D[k - 2]`), con esto obtenemos el valor máximo que puede darnos a elegir la fila, sin saber que valores han sido elegidos ya que estos no son requeridos, una vez calculado para una fila debemos realizar esta tarea para todas las filas restantes.

Analicemos el primer caso en donde M = 5 y N = 5;

Para no tener problemas con `k - 1` o `k - 2` cuando `k < 1` o `k < 2` simplemente añadimos dos celdas vacías al inicio y cuando se consulte la cantidad en `k - 1` o `k - 2` esta sea `0`, como vemos en el siguiente ejemplo, donde `D[][]` es nuestra matriz que guarda las cantidades de dulces y al mismo tiempo nuestra matriz de programación dinámica.

```
int[][] D = new int[M][N+2];
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        D[i][j+2] = in.nextInt();
    }
}
```

$$D_{ij} = \begin{pmatrix} 0 & 0 & 1 & 8 & 2 & 1 & 9 \\ 0 & 0 & 1 & 7 & 3 & 5 & 2 \\ 0 & 0 & 1 & 2 & 10 & 3 & 10 \\ 0 & 0 & 8 & 4 & 7 & 9 & 1 \\ 0 & 0 & 7 & 1 & 3 & 1 & 6 \end{pmatrix}$$

Buscamos la solución para cada fila.

```
for (int i = 0; i < M; i++) {
    for (int j = 2; j < N+2; j++) {
        D[i][j] = max(D[i][j-1], D[i][j]+D[i][j-2]);
    }
}
```

$$D_{ij} = \begin{pmatrix} 0 & 0 & 1 & 8 & 8 & 9 & 17 \\ 0 & 0 & 1 & 7 & 7 & 12 & 12 \\ 0 & 0 & 1 & 2 & 11 & 11 & 21 \\ 0 & 0 & 8 & 8 & 15 & 17 & 17 \\ 0 & 0 & 7 & 7 & 10 & 10 & 16 \end{pmatrix}$$

Una vez tengamos el valor máximo de cada fila, debemos realizar la misma tarea con esos valores, para ésto guardamos las soluciones en un vector `sol[]` de tamaño igual al número de filas más dos vacías.

$$sol_{ij} = \begin{pmatrix} 0 & 0 & 17 & 12 & 21 & 17 & 16 \end{pmatrix}$$

```
int[] sol = new int[M+2];
for (int i = 0; i < M; i++) {/guardamos las soluciones de cada fila
    sol[i+2] = D[i][N+1];
}
for (int i = 2; i < M+2; i++) {//
    sol[i] = max(sol[i-1], sol[i]+sol[i-2]);
}
```

$$sol_{ij} = \begin{pmatrix} 0 & 0 & 17 & 17 & 38 & 38 & 54 \end{pmatrix}$$

La respuesta del problema se encuentra en el último valor calculado en `sol[M+1] = 54`.

Si realizamos los mismos cálculos el momento de leer podemos crear una version mucho mas simple. Esta version se presenta codificada en lenguaje c.

Programa Java que soluciona el problema

```
import java.util.Scanner;
import static java.lang.Math.max;
class candy {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int M, N;
        while(true){
            M = in.nextInt();
            N = in.nextInt();
            if(M == 0 && N == 0) break;
            int[][] D = new int[M][N + 2];
            for (int i = 0; i < M; i++) {
                for (int j = 0; j < N; j++) {
                    D[i][j + 2] = in.nextInt();
                }
            }
            for (int i = 0; i < M; i++) {
                for (int j = 2; j < (N + 2); j++) {
                    D[i][j] = max(D[i][j - 1], D[i][j] + D[i][j - 2]);
                }
            }
            int[] sol = new int[M + 2];
            for (int i = 0; i < M; i++) {
                sol[i + 2] = D[i][N + 1];
            }
            for (int i = 2; i < (M + 2); i++) {
                sol[i] = max(sol[i - 1], sol[i] + sol[i - 2]);
            }
            System.out.println(sol[M + 1]);
        }
    }
}
```

Programa C que soluciona el problema

```
#include <stdio.h>

#define max(_x,_y) ((_x) > (_y) ? (_x) : (_y))

int main(void)
{
    int N,M,i,j,a,b,c,A,B,C;
    while(scanf("%d %d",&N,&M) == 2 && N)
    {
        for(i = A = B = 0; i < N; i++)
        {
            for(j = a = b = 0; j < M; j++)
            {
                scanf("%d",&c);
                c = max(a+c,b);
                a = b;
                b = c;
            }
            C = max(A+c,B);
            A = B;
            B = C;
        }
        printf("%d\n",C);
    }

    return(0);
}
```

Problem F
DNA Subsequences

Source file name: `sequence.c`, `sequence.cpp` *or* `sequence.java`

Thomas, a computer scientist that works with DNA sequences, needs to compute longest common subsequences of given pairs of strings. Consider an alphabet Σ of letters and a word $w = a_1 a_2 \cdots a_r$, where $a_i \in \Sigma$, for $i = 1, 2, \ldots, r$. A *subsequence* of w is a word $x = a_{i_1} a_{i_2} \cdots a_{i_s}$ such that $1 \leq i_1 < i_2 < \ldots < i_s \leq r$. Subsequence x is a *segment* of w if $i_{j+1} = i_j + 1$, for $j = 1, 2, \ldots, s - 1$. For example the word `ove` is a segment of the word `lovely`, whereas the word `loly` is a subsequence of `lovely`, but not a segment.

A word is a *common subsequence* of two words w_1 and w_2 if it is a subsequence of each of the two words. A *longest common subsequence* of w_1 and w_2 is a common subsequence of w_1 and w_2 having the largest possible length. For example, consider the words $w_1 =$ `lovxxelyxxxxx` and $w_2 =$ `xxxxxxxlovely`. The words $w_3 =$`lovely` and $w_4 =$ `xxxxxxx`, the latter of length 7, are both common subsequences of w_1 and w_2. In fact, w_4 is their longest common subsequence. Notice that the empty word, of length zero, is always a common subsequence, although not necessarily the longest.

In the case of Thomas, there is an extra requirement: the subsequence must be formed from common segments having length K or more. For example, if Thomas decides that $K = 3$, then he considers `lovely` to be an acceptable common subsequence of `lovxxelyxxxxx` and `xxxxxxxlovely`, whereas `xxxxxxx`, which has length 7 and is also a common subsequence, is not acceptable. Can you help Thomas?

Input

The input contains several test cases. The first line of a test case contains an integer K representing the minimum length of common segments, where $1 \leq K \leq 100$. The next two lines contain each a string on lowercase letters from the regular alphabet of 26 letters. The length ℓ of each string satisfies the inequality $1 \leq \ell \leq 10^3$. There are no spaces on any line in the input. The end of the input is indicated by a line containing a zero.

The input must be read from file sequence.in.

Output

For each test case in the input, your program must print a single line, containing the length of the longest subsequence formed by consecutive segments of length at least K from both strings. If no such common subsequence of length greater than zero exists, then `0` must be printed.

The output must be written to standard output.

Sample input	**Output for the sample input**

`3`
`lovxxelyxxxxx`
`xxxxxxxlovely`
`1`
`lovxxelyxxxxx`
`xxxxxxxlovely`
`3`
`lovxxxelxyxxxx`
`xxxlovelyxxxxxxx`
`4`
`lovxxxelyxxx`
`xxxxxxlovely`
`0`
	`6`
	`7`
	`10`
	`0`

Análisis del Problema F

Empecemos explicando el problema de la Subsecuencia común mas larga, si bien la descripción del problema nos da una definición matemática bastante exacta no esta demás ver ejemplos.

Este problema clásico de la Bioinformática busca la subsecuencia mas larga que se encuentre de forma creciente dentro de dos secuencias, la búsqueda de su solución por medio de fuerza bruta esta demostrada que tiene una complejidad No Polinomial, pero usando programación dinámica lo reducimos a un problema de complejidad Polinomial.

Ejemplo 1

Sea $A =$ `a1b2c3d4e` y $B =$ `tt1uu2vv3ww4xx`, la única subsequencias común es `1234` entonces decimos que la subsecuencia común mas larga de B en A es 4.

Ejemplo 2

Sea $A =$ `xyz` y $B =$ `xzy`, existen dos subsequencias la primera es `xy` y la segunda `xz` entonces decimos que la subsecuencia común mas larga de B en A es 2.

Ejemplo 3

Sea $A =$ `lovxxelyxxxxx` y $B =$ `xxxxxxxlovely`, la subsecuencia `lovely` el tamaño es de 6 y en la subsecuencia `xxxxxxx` el tamaño es de 7 entonces decimos que la subsecuencia común mas larga de B en A es 7.

En la siguiente función se muestra un algoritmo para encontrar la subsecuencia común mas larga, siendo `C` la matriz de programación dinámica.

```
function  LCS(A[1..m], B[1..n])
    C = array(0..m, 0..n)
    for i := 0..m
       C[i,0] = 0
    for j := 0..n
       C[0,j] = 0
    for i := 1..m
        for j := 1..n
            if A[i] = B[j]
                C[i,j] := C[i-1,j-1] + 1
            else:
                C[i,j] := max(C[i,j-1], C[i-1,j])
    return C[m,n]
```

El problema habla de un requerimiento extra, que es buscar la subsecuencia común mas larga formada desde un segmento K o mas largo, el algoritmo anterior solo resuelve el problema para segmentos de tamaño 1, veamos los tres casos de prueba que nos proporcionan.

Caso 1

Donde $k = 3$ la primera cadena es `lovxxelyxxxxx` y la segunda cadena es `xxxxxxxlovely`, nos pide la mayor subsecuencia que empiece por una segmento común de tamaño tres o mas, en este caso `lov` es el segmento que buscamos y la subsecuencia de este segmento es `lovely` que tiene una longitud de 6, notese que la subsecuencia común mas larga es `xxxxxxx` de longitud 7.

Caso 2

Donde $k = 1$ la primera cadena es `lovxxelyxxxxx` y la segunda cadena es `xxxxxxxlovely`, nos pide la mayor subsecuencia común que empiece por una segmento común de tamaño uno o mas y esto es lo mismo que mostrarle la subsecuencia común mas larga que conocemos, esta es `xxxxxxx` que tiene una longitud de 7.

Caso 3

Donde $k = 3$ la primera cadena es `lovxxxelxyxxxx` y la segunda cadena es `xxxlovelyxxxxxxx`, el segmento común que buscamos de tamaño tres es `lov` la subsecuencia formada a partir de este segmento es `lovelyxxxx` que tiene una longitud de 10, también existe otra en donde el segmento es `xxx` y la subsecuencia es xxxelyxxxx de igual longitud a la anterior.

Caso 4

Donde $k = 4$ la primera cadena es `lovxxxelyxxx` y la segunda cadena es `xxxxxxlovely`, en este caso no existe un segmento común que tenga el tamaño de cuatro o mas, a lo mas tiene un segmento común de tres.

La solución en java que planteo es bastante similar al algoritmo anterior, pero en vez comparar un caracter con otro:

```
    if A[i] = B[j]
        C[i,j] := C[i-1,j-1] + 1
```

la comparación es entre los k últimos caracteres, viendo que todos sean iguales para esto agrego un ciclo adicional en el cual veo si la solución parcial cumple con el requerimiento extra.

Este problema se puede considerar como de gran complejidad si uno no esta familiarizado con técnicas de programación dinámica.

Programa Java que soluciona el problema

```
import java.util.Scanner;
import static java.lang.Math.max;
class sequence {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String first, second;
        int k;
        while(true){
            k = in.nextInt();
            if(k == 0) break;
            first = in.next();
            second = in.next();
            System.out.println(lcs(first, second, k));
        }
    }
    public static int lcs(String s1, String s2, int k) {
        s1 = "#" + s1;
        s2 = "#" + s2;
        int C[][] = new int[s1.length()][s2.length()];
        for(int a = k; a < s1.length(); a++) {
            for(int b = k; b < s2.length(); b++) {
                C[a][b] = max(C[a][b - 1], C[a - 1][b]);
                for(int c = 0; c <= a; c++){// or c <=b
                    if(c >= k) {
                        C[a][b] = max(C[a][b], C[a - c][b - c] + c);
                    }
                    if(s1.charAt(a - c) != s2.charAt(b - c)) break;
                }
            }
        }
        return C[s1.length()-1][s2.length()-1];
    }
}
```

Programa C que soluciona el problema

```
#include <stdio.h>

#define max(x,y) ((x)>(y))?(x):(y)

int main(void) {
  int L[1001][1001]; // max. tamaño del prefijo comun
  int Sol[1001][1001]; // max. tamaño de la subsequencia
  int K; // segmento comun minimo
  char X[1001], Y[1001]; // sequencias
  int m, n; // longitud de sequencias
  int i,j,k;
  for (i = 0; i <= 1000; i++) {
       L[i][0] = L[0][i] = 0;
       Sol[i][0] = Sol[0][i] = 0;
}
   while (1) {
        scanf("%d\n", &K);
        if (!K) break;
        scanf("%s%n\n", X, &m);
        scanf("%s%n\n", Y, &n);

        for (i = 1; i <= m; i++) {
           for (j = 1; j <= n; j++) {
              L[i][j] = (X[i-1] == Y[j-1]) ? (L[i-1][j-1] + 1) : 0;
              sol[i][j] = max(Sol[i-1][j], Sol[i][j-1]);
                  for (k = K; k <= L[i][j]; k++) {
                      Sol[i][j] = max(Sol[i][j], Sol[i-k][j-k] + k);
                      }
               }
         }
         printf("%d\n", Sol[m][n]);
}
return 0;
}
```

Problem G
Electricity

Source file name: `electricity.c`, `electricity.cpp` *or* `electricity.java`

Martin and Isa stopped playing crazy games and finally got married. It's good news! They're pursuing a new life of happiness for both and, moreover, they're moving to a new house in a remote place, bought with most of their savings.

Life is different in this new place. In particular, electricity is very expensive, and they want to keep everything under control. That's why Martin proposed to keep a daily record of how much electricity has been consumed in the house. They have an electricity meter, which displays a number with the amount of KWh (kilowatt-hour) that has been consumed since their arrival.

At the beginning of each day they consult the electricity meter, and write down the consumption. Some days Martin does it, and some days Isa does. That way, they will be able to look at the differences of consumption between consecutive days and know how much has been consumed.

But some days they simply forget to do it, so, after a long time, their register is now incomplete. They have a list of dates and consumptions, but not all of the dates are consecutive. They want to take into account only the days for which the consumption can be precisely determined, and they need help.

Input

The input contains several test cases. The first line of each test case contains one integer N indicating the number of measures that have been taken ($2 \le N \le 10^3$). Each of the N following lines contains four integers D, M, Y and C, separated by single spaces, indicating respectively the day ($1 \le D \le 31$), month ($1 \le M \le 12$), year ($1900 \le Y \le 2100$), and consumption ($0 \le C \le 10^6$) read at the beginning of that day. These N lines are increasingly ordered by date, and may include leap years. The sequence of consumptions is strictly increasing (this is, no two different readings have the same number). You may assume that D, M and Y represent a valid date.

Remember that a year is a leap year if it is divisible by 4 and not by 100, or well, if the year is divisible by 400.

The end of input is indicated by a line containing only one zero.

The input must be read from file electricity.in.

Output

For each test case in the input, your program must print a single line containing two integers separated by a single space: the number of days for which a consumption can be precisely determined, and the sum of the consumptions for those days.

The output must be written to standard output.

Sample input

```
5
9 9 1979 440
29 10 1979 458
30 10 1979 470
1 11 1979 480
2 11 1979 483
3
5 5 2000 6780
6 5 2001 7795
7 5 2002 8201
8
28 2 1978 112
1 3 1978 113
28 2 1980 220
1 3 1980 221
5 11 1980 500
14 11 2008 600
15 11 2008 790
16 12 2008 810
0
```

Output for the sample input

```
2 15
0 0
2 191
```

Análisis del Problema G

El problema nos pide determinar en la entrada si dos fechas son consecutivas. En este caso se acumulan los kwatts que se consumieron, que son la diferencia entre ambas entradas.

La dificultad está en controlar los años bisiestos que se determinan con los datos dados:

1. El año debe ser múltiplo de 4 y no por 100
2. o divisible por 400

Si y es el año el código siguiente nos dice si el año es bisiesto.

```
( ((y%4)==0 && (y%100)!=0) || (y%400)==0 )
```

Para determinar si un día es conse-cutivo con otro hay dos formas:

1. Calcular el día Juliano y restar ambos días si la distancia es uno son consecutivos
2. La segunda opción es la de comparar los días, meses y años para determinar si son consecutivos. Por ejemplo si el mes es enero 31 y el siguiente es febrero 1 son consecutivos, si están en el mismo año,

Utilizando la segunda opción usaríamos el código utilizado en la versión java.

```
public static boolean consecutive(int d1, int m1, int y1, int d2, int m2,
                                  int y2) {
if(m1 == 1 || m1 == 3 || m1 == 5 || m1 == 7 || m1 == 8 || m1 == 10 ||
   m1 == 12) {
       if(d1<=30 && d2==d1+1 && m2==m1 && y2==y1) {
            return true;
        }
        if(d1==31 && m1<12 && d2==1 && m2==m1+1 && y2==y1) {
            return true;
        }
        if(d1==31 && m1==12 && d2==1 && m2==1 && y2==y1+1) {
            return true;
        }
    }
    else if(m1 == 4 || m1 == 6 || m1 == 9 || m1 == 11) {
         if(d1<=29 && d2==d1+1 && m2==m1 && y2==y1) {
            return true;
           }
         if(d1==30 && d2==1 && m2==m1+1 && y2==y1) {
            return true;
          }
         }
```

```
      else if(m1 == 2) {
           if( (d1<=27 || (d1==28 && isLeap(y1))) && d2==d1+1 && m2==m1 &&
              y2==y1) {
              return true;
            }
            if(((d1==29 && isLeap(y1)) || (d1==28 && !isLeap(y1))) && d2==1 &&
              m2==3 && y2==y1) {
              return true;
            }
         }
                  return false;
}
```

Para hallar el día Juliano una alternativa es la siguiente:

Crear un vector con los datos siguientes:

```
const int f[13] = {-1, 306, 337, 0, 31, 61, 92, 122, 153, 184, 214, 245, 275};

      if (mes <= 2) --ano;
      a = ano / 4;
      b = ano / 100;
      c = ano / 400;
      n = 0L + dia + f[mes] + ano * 365 + a - b + c;
```

En este caso *n* contiene el dia Juliano.

Se han codificado ambas estrategias, una ea java y la otra en c. Cualquier estrategia puede utilizarse en cualquier lenguaje.

Programa Java que soluciona el problema

```
import java.util.Scanner;
public class Main {
  public static int N;
  public static int sum;
  public static int nDays;

public static boolean isLeap(int y) {
      return ( ((y%4)==0 && (y%100)!=0) || (y%400)==0 );
}
public static void main (String[] args) {
      Scanner sc = new Scanner(System.in);
      int d1, m1, y1, d2, m2, y2;
      int c1, c2;
      N = sc.nextInt();
      while(N != 0) {
          nDays = 0;
          sum = 0;
          d1 = sc.nextInt();
          m1 = sc.nextInt();
          y1 = sc.nextInt();
          c1 = sc.nextInt();
          for(int i=1; i<N; i++) {
              d2 = sc.nextInt();
              m2 = sc.nextInt();
              y2 = sc.nextInt();
              c2 = sc.nextInt();
              if(consecutive(d1,m1,y1,d2,m2,y2)) {
                  nDays++;
                  sum += (c2-c1);
               }
               d1 = d2;
               m1 = m2;
               y1 = y2;
               c1 = c2;
           }
        System.out.println(nDays + " "+ sum);
        N = sc.nextInt();
       }

  }

}
```

Programa C que soluciona el problema

```
#include <stdio.h>

typedef struct {
  long long int n;
  int consumo;
} Dado;

const int f[13] = {-1, 306, 337, 0, 31, 61, 92, 122, 153, 184, 214, 245, 275};

int main (int argc, char *noargs[]) {
  int N, i, res, total, dia, mes, ano, a, b, c;
  Dado ant, cor;

  for (;;) {
    scanf ("%d", &N);
    if (N == 0) break;
    ant.n = -1;
    res = total = 0;
    for (i = 0; i < N; ++i) {
      scanf ("%d %d %d %d", &dia, &mes, &ano, &cor.consumo);
      if (mes <= 2) --ano;
      a = ano / 4;
      b = ano / 100;
      c = ano / 400;
      cor.n = 0L + dia + f[mes] + ano * 365 + a - b + c;
      if (cor.n == ant.n + 1) {
++res;
total += cor.consumo - ant.consumo;
      }
      ant = cor;
    }
    printf ("%d %d\n", res, total);
  }
  return 0;
}
```

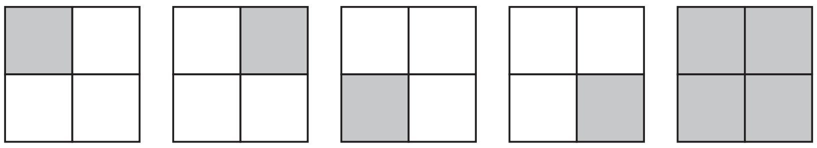
Problem H
Feynman

Source file name: `feynman.c`, `feynman.cpp` *or* `feynman.java`

Richard Phillips Feynman was a well known American physicist and a recipient of the Nobel Prize in Physics. He worked in theoretical physics and also pioneered the field of quantum computing. He visited South America for ten months, giving lectures and enjoying life in the tropics. He is also known for his books “Surely You're Joking, Mr. Feynman!” and “What Do You Care What Other People Think?”, which include some of his adventures below the equator.

His life-long addiction was solving and making puzzles, locks, and cyphers. Recently, an old farmer in South America, who was a host to the young physicist in 1949, found some papers and notes that is believed to have belonged to Feynman. Among notes about mesons and electromagnetism, there was a napkin where he wrote a simple puzzle: “how many different squares are there in a grid of $N \times N$ squares?”.

In the same napkin there was a drawing which is reproduced below, showing that, for $N = 2$, the answer is 5.



Input

The input contains several test cases. Each test case is composed of a single line, containing only one integer N, representing the number of squares in each side of the grid ($1 \leq N \leq 100$).

The end of input is indicated by a line containing only one zero.

The input must be read from file feynman.in.

Output

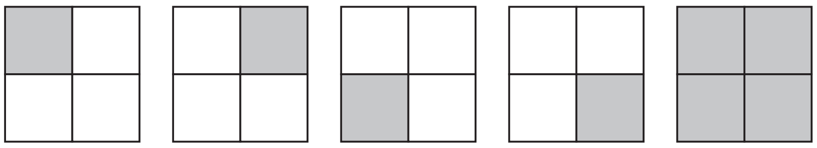
For each test case in the input, your program must print a single line, containing the number of different squares for the corresponding input.

The output must be written to standard output.

Sample input	Output for the sample input
2 1 8 0	5 1 204

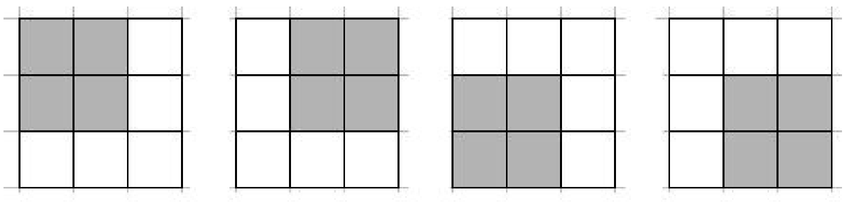
Análisis del Problema H

Para entender el problema primeramente vamos explicar como se forman los 5 cuadrados de la imagen



Inicialmente tenemos el cuadrado del contorno de la imagen y luego cada uno de los cuadrados. Representando ésto vemos que el número total de cuadrados es $1 + 2 * 2$.

Consideremos un cuadrado de $3 * 3$



Inicialmente tenemos los cuadrados interiores $3*3 = 9$ más los cuatro cuadrados de $2*2$ y el de contorno, haciendo un total de 14. Si analizamos estos resultados vemos $14 = 3*3+4+1$. Esto nos lleva a ver que es que es una recurrencia clásica $f(n) = \sum_{i=1}^{n} f(n-1) + n^2$ con $f(0) = 0$.

Para escribir el programa utilizamos los parámetros dados en la definición del problema $n \leq 100$ y tenemos dos alternativas:

1. Hallar un vector con todos los valores precalculados. Luego para cada n podemos imprimir su resultado.
2. Hallar la expresión de la suma y calcular el resultado que es $n = i * (i+1) * (2i+1)/6$. Probablemente no se recuerde éste valor de esta sumatoria en la competencia.

Utilizaremos la primera alternativa para escribir el código que resuelve el problema.

Programa Java que soluciona el problema

```
import java.util.Scanner;

class Main
{
   public static void main( String args[] ) {
      Scanner in = new Scanner( System.in );
  int V[] = new int[101];
  int n;
  for( n=1; n<=100; n++)
```

```
     V[n]=V[n-1]+n*n;
  while( true ) {
     n = in.nextInt();
 if( n == 0 )
    break;
 System.out.println( V[n] );
  }
   }
}
```

Programa C que soluciona el problema

```
#include<stdio.h>

int main()
{
    long V[101];
long n;
for( V[0]=0,n=1; n<=100; n++)
    V[n] = V[n-1] + n*n;
while( scanf("%ld",&n) && n )
    printf("%ld\n", V[n] );
return 0;
}
```

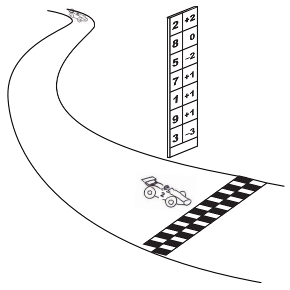
Problem I
Pole Position

Source file name: `pole.c`, `pole.cpp` *or* `pole.java`

In car races, there is always a high pole next to the finish line of the track. Before the race starts, the pole is used to display the starting grid. The number of the first car in the grid is displayed at the top of the pole, the number of the car in second place is shown below that, and so on.

During the race, the pole is used to display the current position of each car: the car that is winning the race has its number displayed at the top of the pole, followed by the car that is in second place, and so on.

Besides showing the current position of a car, the pole is also used to display the number of positions the cars have won or lost, relative to the starting grid. This is done by showing, side by side to the car number, an integer number. A positive value v beside a car number in the pole means that car has won v positions relative to the starting grid. A negative value v means that car has lost v positions relative to the starting grid. A zero beside a car number in the pole means the car has neither won nor lost any positions relative to the starting grid (the car is in the same position it started).



We are in the middle of the Swedish Grand Prix, the last race of the World Championship. The race director, Dr. Shoo Makra, is getting worried: there have been some complaints that the software that controls the pole position system is defective, showing information that does not reflect the true race order.

Dr. Shoo Makra devised a way to check whether the pole system is working properly. Given the information currently displayed in the pole, he wants to reconstruct the starting grid of the race. If it is possible to reconstruct a valid starting grid, he plans to check it against the real starting grid. On the other hand, if it is not possible to reconstruct a valid starting grid, the pole system is indeed defective.

Can you help Dr. Shoo Makra?

Input

The input contains several test cases. The first line of a test case contains one integer N indicating the number of cars in the race ($2 \le N \le 10^3$). Each of the next N lines contains two integers C and P, separated by one space, representing respectively a car number ($1 \le$

$C \leq 10^4$) and the number of positions that car has won or lost relative to the starting grid ($-10^6 \leq P \leq 10^6$), according to the pole system. All cars in a race have different numbers.

The end of input is indicated by a line containing only one zero.

The input must be read from file pole.in.

Output

For each test case in the input, your program must print a single line, containing the reconstructed starting grid, with car numbers separated by single spaces. If it is not possible to reconstruct a valid starting grid, the line must contain only the value `-1`.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
4
1 0
3 1
2 -1
4 0
4
22 1
9 1
13 0
21 -2
3
19 1
9 -345
17 0
7
2 2
8 0
5 -2
7 1
1 1
9 1
3 -3
0
```
	```
	1 2 3 4
	-1
	-1
	5 8 2 3 7 1 9
	```

Análisis del Problema I

Las posiciones iniciales están dadas por la posición actual sumando lo que ganó o perdió en la posición anterior. ésto se puede hacer sumando el valor de la posición actual.

Al momento de leer tenemos la posición actual sumando al desplazamiento tenemos la posición anterior. Guardamos ésto en un vector.

Al inicio éste vector debe estar en cero. Si alguna posición se repite es un error. Si una posición está fuera de la cantidad de coches participantes es un error. El código siguiente resuelve lo explicado:

```
for (i=0;i<n;i++) {
      scanf("%d %d",&c,&p);
      tmp=i+p;
      if (tmp>=0 && tmp<n && polepos[tmp]==0)
        polepos[tmp]=c;
      else
        ok=FALSE;
```

Programa Java que soluciona el problema

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Main {
    private static int[] pole;
    private static int n,c,p,tmp;
    private static boolean ok;

    public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    n = in.nextInt();
    while (n > 0) {
       pole = new int[n];
       ok = true;
       for (int i = 0; i<n; i++){
           c = in.nextInt();
           p = in.nextInt();
   tmp=i+p;
           if (tmp>=0 && tmp<n && pole[tmp]==0)
               pole[tmp]=c;
           else
               ok = false;
        }
       if (ok) {
          System.out.print(pole[0]);
          for (int i=1;i<n;i++)
     System.out.print(" "+pole[i]);

          System.out.println("");
       } else
       System.out.println("-1");

       n = in.nextInt();
   }
   }
}
```

Programa C que soluciona el problema

```
#include <stdio.h>
#include <strings.h>

#define TRUE 1
#define FALSE 0
#define MAX 1000

int n,c,p;
int polepos[MAX];

int main (void)
{
  int i,tmp,ok;

  while (scanf("%d",&n)==1 && n>0) {
    bzero(polepos,n*sizeof(int));
    ok=TRUE;
    for (i=0;i<n;i++) {
      scanf("%d %d",&c,&p);
      tmp=i+p;
      if (tmp>=0 && tmp<n && polepos[tmp]==0)
polepos[tmp]=c;
      else
ok=FALSE;
    }
    if (ok) {
      printf("%d",polepos[0]);
      for (i=1;i<n;i++) {
printf(" %d",polepos[i]);
```

```
      }
      printf("\n");
    }
    else
      printf("-1\n");
  }
  return(0);
}
```

Problem J
Higgs Boson

Source file name: `higgs.c`, `higgs.cpp` *or* `higgs.java`

It's been 100 years since the detection of the first Higgs boson and now particle physics is a mainstream subject in all high schools. Obviously, kids love the fact that they can create tiny black holes using only their portable particle accelerators and show off to their friends and colleagues. Although the creation of big black holes that could swallow the whole planet is possible even with these portable particle accelerators, the devices are programmed to only thrown particles when this undesirable side effect is impossible.

Your granddaughter is trying to create her own black holes with a portable accelerator kit, which is composed of two small particle accelerators that throw, each one, a boson-sized particle. Both particles are thrown at the same time, and a black hole appears when the particles collide. However, your granddaughter doesn't know how much time she'll have to wait before this happens. Fortunately, each accelerator can predict the particle's trajectory, showing four integer values into its display, called A, B, C and D. Each value can be replaced into the following equations:

$$r = At + B$$
$$\theta = Ct + D$$

in order to determine the trajectory of the particle, in polar coordinates. The radius (r) is represented in distance units and the angle (θ) in degrees. The time (t) is given in time units and it is always a rational value which can be represented by an irreducible fraction. Your granddaughter knows that in polar coordinates a point has infinite representations. In general, the point (r, θ) can be represented as $(r, \theta \pm k \times 360^o)$ or $(-r, \theta \pm (2k + 1) \times 180^o)$, where k is any integer. Besides, the origin $(r = 0)$ can be represented as $(0, \theta)$ for any θ.

Using these parameters informed by each particle accelerator, your granddaughter wants to determine whether the particles will eventually collide and, if they do, the time when they will collide. After the first collision it is impossible to predict the particle's trajectory, therefore, only the first possible collision should be considered.

Although your granddaughter is really intelligent and has a deep knowledge of particle physics, she does not know how to program computers and is looking for some notes in her grandfather's (or grandmother's) ICPC notebook (don't forget, she is *your* granddaughter!). Fortunately for you, there is a note on your notebook which says that you wrote that code during the 2008 ICPC South America Regional Contest (or, to be more specific, *this* contest).

Input

The input consists of several test cases, one per line. Each test case contains eight integer numbers separated by single spaces, $A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2$ $(-10^4 \leq A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2 \leq 10^4)$. The first four input values (A_1, B_1, C_1, D_1) correspond to the four parameters displayed by the first portable particle accelerator and the following input values (A_2, B_2, C_2, D_2) correspond to the four parameters displayed by the second portable particle accelerator when both particles are thrown. The end of the input is represented by $A_1 = B_1 = C_1 = D_1 = A_2 =$

$B_2 = C_2 = D_2 = 0$, which should not be processed as a test case, since these are the values displayed by the particle accelerators when a big black hole would be created if the particles were trown. Although the end of input is represented by a line with eight zeroes, note that the number zero is a possible input value.

The input must be read from file higgs.in.

Output

For each test case, your program must output a line containing two non-negative integers t_a and t_b separated by a single space. If there is no possibility of collision, $t_a = t_b = 0$, otherwise, t_a/t_b must be an irreducible fraction representing the earliest collision time. Even if the fraction results in an integer value, you still must output the number 1 as the denominator (see samples below).

The output must be written to standard output.

Sample input	**Output for the sample input**
`1 1 180 0 2 0 180 360`	`1 1`
`10 10 360 0 -24 18 180 72`	`0 0`
`5 5 180 0 -12 9 10 40`	`4 17`
`-9 5 5 180 2 5 5 180`	`0 1`
`0 0 0 0 0 0 0 0`	

Análisis del Problema J

Nos dan dos trayectorias en forma de coordenadas polares de la forma:

$$r = At + B$$
$$\theta = Ct + D$$

Nuestro objetivo es encontrar t para las 4 ecuaciones, de modo que r y θ identifique el mismo punto en ambos sistemas de ecuaciones.

Esto equivale a escribir

$$r_1 = A_1t + B_1 \text{ y } r_2 = A_2t + B_2$$
$$\theta_1 = C_1t + D_1 \text{ y } \theta_2 = C_2t + D_2$$

Para que se genere una colisión deben cumplirse las condiciones

$$r_1 = r_2$$
$$\theta_1 = \theta_2 \pm k \times 360$$

Resolviendo las ecuaciones podemos hallar el valor de t

$$t = \frac{B_2 - B_1}{A_1 - A_2}$$
$$t = \frac{D_1 - D_2 \pm k \times 360}{C_2 - C_1}$$

En el enunciado nos hace notar que podemos escribir de dos formas sin alterar el resultado:

1. $(r, \theta + -k * 360)$ o $(-r, \theta + -(2k+1) * 180)$ para k entero.
2. si r es cero, entonces θ puede tomar cualquier valor.
3. La solución debe ser mínima y no negativa.

Así tenemos dos formas de encontrar el resultado: $r1 == r2$ $r1 == -r2$ y sumarle a θ 180^o

Empezamos a desarrollar la solución de la forma $\frac{numerador}{denominador}$, esta solución la encontramos en las ecuaciones de r y de acuerdo a ciertas condiciones las comparamos posiblemente en θ.

Si el denominador es cero:

- si el numerador también es cero, entonces tenemos infinitos resultados, así que buscamos la solución en θ.
- si el numerador es diferente de cero, entonces no existe solución.

Si el denominador no es cero:

- si esta solución es negativa entonces no existe solución.

- si es mayor o igual a cero, se verifica si también es solución de θ.

Si con nuestra solución hacemos de $r1 == 0$ (y por consiguiente $r2 == 0$), entonces podemos decir que es solución sin necesidad de comprobarla en θ (por 2).

Para encontrar los resultados en θ, podemos sumarle o restarle múltiplos de 360 para hacer que la solución sea positiva.

Para verificar nuestra posible solución en θ se debe cumplir que una vez que hayamos remplazado t con nuestra solución: $(\theta_1 - \theta_2)\%(360 * denominador) == 0$

Finalmente si es que tenemos alguna solución, dividimos el numerador y el denominador entre el máximo común divisor e imprimimos el menor en caso de que tengamos dos soluciones.

Programa C que soluciona el problema

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <cstdlib>
#define INFINITO 99999
using namespace std;

long numerador,denominador;//posible solucion
long a1,b1,c1,d1,a2,b2,c2,d2;//datos de entrada

long GCD(int a,int b){//maximo comun divisor
if(b==0)return a;
else return GCD(b,a%b);
}
void reducir(){//hace que la solucion sea una fraccion irreducible
    long gcd=GCD(numerador,denominador);
    numerador/=gcd;denominador/=gcd;
}
bool busca_solucion_theta(){ //en el caso de que la solucion se encuentre en theta
    if(d1<0)d1=360*(long)ceil(d1/-360.0)+d1;//haciendo que 0<= d1,d2 <360
    else d1=d1%360;
    if(d2<0)d2=360*(long)ceil(d2/-360.0)+d2;
    else d2=d2%360;
    numerador=d2-d1;//creando la posible solucion
    denominador=c1-c2;
    if(denominador<0){
        numerador*=-1;
        denominador*=-1;
    }
//en el caso de que el numerador sea negativo, se le suma 360 hasta que sea positivo
    if(numerador<0)numerador=360*(long)ceil(numerador/-360.0)+numerador;
    if(denominador==0){
        if(numerador==0)denominador=1;
//infinitas soluciones, asi que buscamos el menor (0,1)
        else return false;//no existe solucion
    }else if(numerador==0)denominador=1;
//para evitar respuestas del tipo 0/358... y convertirlas a 0/1
    else reducir();//volviendo a fraccion irreducible.
    return true;//entonces si se llego a una solucion
}
void obtener_solucion(){//buscando la solucion en las ecuaciones de r
    bool ok=true;//por ahora existe solucion
    numerador=b2-b1;//creando una posible solucion
    denominador=a1-a2;
    if(denominador==0){
        if(numerador==0)ok=busca_solucion_theta();
//infinitas soluciones, asi que buscamos en theta
        else ok=false;
//no existen soluciones
    }else if(numerador*denominador>=0){
//hay solucion si es positiva
        if(numerador==0)denominador=1;
//para evitar respuestas del tipo 0/358... y convertirlas a 0/1
        else{
            numerador= labs(numerador);denominador= labs(denominador);
// volviendolos positivos, evita solucion -2,-6
            reducir(); //volviendo a fraccion irreducible.
        }
        if(ok){ //si hasta ahora no ha fallado nada, se comprueba la solucion en thet
            long th1=c1*numerador+d1*denominador;
            long th2=c2*numerador+d2*denominador;
            ok=(a1*numerador+b1*denominador==0)||((th1-th2)%(360*denominador)==0);
//verdad si r1==0 o es solucion de theta
        }
    }else ok=false;//solucion negativa, se descarta
    if(!ok){
        numerador=INFINITO;//si no hay solucion, se la pone en infinita
        denominador=1;
    }
}
int main(){
    long num1,num2,den1,den2;
    while((cin>>a1>>b1>>c1>>d1>>a2>>b2>>c2>>d2)&&
(a1||b1||c1||d1||a2||b2||c2||d2)){
        obtener_solucion();//obteniendo solucion de la manera normal
        num1=numerador,den1=denominador;//guardando solucion

        a2*=-1;b2*=-1;d2+=180;//obteniendo solucion de la segunda manera
        obtener_solucion();
        num2=numerador,den2=denominador;//guardando solucion
//mostrando la menor solucion.
        if(num1==INFINITO&&num2==INFINITO)printf("0 0\n");
        else if(num1*den2>num2*den1)printf("%ld %ld\n",num2,den2);
        else printf("%ld %ld\n",num1,den1);
    }
}
```

Problem K
Traveling Shoemaker Problem

Source file name: `tsp.c`, `tsp.cpp` *or* `tsp.java`

Once upon a time there was a very peaceful country named Nlogonia. Back then, Poly the Shoemaker could come to the country and travel freely from city to city doing his job without any harassment. This task was very easy, as every city in Nlogonia had a direct road to every other city in the country. He could then easily travel the whole country visiting each city exactly once and fixing everybody's shoes.

But not anymore. The times have changed and war has come to Nlogonia. The age when people could travel freely is over.

Confederations identified by colors were formed among the cities all over the country, and now each city belongs to at least one and at most two confederations. When trying to enter a city, you must give to the border officer a ticket from one of the confederations this city belongs to. When leaving the city, you receive a ticket from the other confederation the city belongs to (i.e. different from the one you gave when entering) or from the same confederation if the city only belongs to one.

As Poly the Shoemaker is a long time friend of Nlogonia, he is allowed to choose a ticket and a city he wants to enter as the first city in the country, but after that he must obey the confederations rules. He wants to do the same routine he did before, visiting each city exactly once in Nlogonia, but now it's not easy for him to do this, even though he can choose where to start his journey.

For example, suppose there are four cities, labeled from 0 to 3. City 0 belongs to confederations *red* and *green*; city 1 belongs only to *red*; city 2 belongs to *green* and *yellow*; and city 3 belongs to *blue* and *red*. If Poly the Shoemaker chooses to start at city 0, he can enter it carrying either the *red* or the *green* ticket and leave receiving the other. Should he choose the *red* ticket, he will leave with a *green* ticket, and then there is only city 2 he can travel to. When leaving city 2 he receives the *yellow* ticket and now can't go anywhere else. If he had chosen the *green* ticket as the first he would receive the *red* one when leaving, and then he could travel to cities 1 or 3. If he chooses city 3, when leaving he will receive the *blue* ticket and again can't go anywhere else. If he chooses city 1, he receives the *red* ticket again when leaving (city 1 belongs only to the *red* confederation) and can only travel to city 3 and will never get to city 2. Thus, it is not possible to visit each city exactly once starting at city 0. It is possible, however, starting at city 2 with the *yellow* ticket, leaving the city with the *green* ticket, then visiting city 0, leaving with *red* ticket, then visiting city 1, leaving with *red* ticket again and, at last, visiting city 3.

As you can see, it got really difficult for Poly the Shoemaker to accomplish the task, so he asks you to help him. He wants to know if it's possible to choose a city to start such that he can travel all cities from Nlogonia exactly once.

Can you help Poly the Shoemaker?

Input

The input contains several test cases. The first line of a test case contains two integers N and C, separated by one space, indicating respectively the number of cities ($1 \leq N \leq 500$) and confederations ($1 \leq C \leq 100$) in the country. Each of the next C lines describes a confederation. It starts with one integer K ($0 \leq K \leq N$) and then K integers representing the cities which belong to this confederation. All integers are separated by single spaces and cities are numbered from 0 to $N - 1$. Each city will appear at least once and at most twice and no city will be repeated on the same confederation.

The end of input is indicated by a line containing two zeroes separated by a single space.

The input must be read from file `tsp.in`.

Output

For each test case in the input, your program must print a single line, containing the integer `-1` if it's not possible to match the requirements or one integer representing the city where Poly the Shoemaker can start his journey. If there are multiple correct answers, print the smallest one.

The output must be written to standard output.

Sample input	**Output for the sample input**

4 4
1 3
3 0 1 3
2 0 2
1 2
3 4
1 0
3 0 1 2
1 1
1 2
3 4
1 1
2 1 0
2 0 2
1 2
0 0
	2
	-1
	1

Análisis del Problema K

Uno de los primeros problemas de teoria de grafos es el de los puentes königsberg que nos dice ¿Es posible dar un paseo empezando por una cualquiera de las cuatro partes de tierra firme, cruzando cada puente una sola vez y volviendo al punto de partida?, donde la tierra firme representan los nodos y los puentes los arcos del grafo.

En este problema el zapatero debe visitar una cuidad exactamente una vez, este paseo o rrecorrido es conocido como camino euleriano, para que exista este rrecorrido en un grafo debe cumplirse:

- No deben existir nodos aislados (grafo conexo).
- Si todos los nodos del grafo tienen grado par.
- Si tiene a lo mas dos nodos de grado impar.
- El grafo debe estar totalmente conectado.

En principio es necesario transformar el grafo que se nos proporciona, en uno en donde las confederaciones representan un nodo y una ciudad representa un arco que es la conexión entre dos confederaciones, si esta ciudad solo pertenece a una confederación debemos crear un ciclo es decir duplicar la confederación o hacer una conexión entre a si mismo.

Tenemos un camino euleriano si todos los nodos están conectados, y deben existir a lo mas dos nodos de grado impar.

```
n_imp = 0;//numero de nodos de grado impar
for (v = 1; v <= N; ++v) {//d[v] grado del vértice v
    if ((d[v] % 2 == 1) && (++n_imp > 2)) break;
}
if (n_imp > 2 || !connected(-1)) res = -1;
```

Si el grafo tiene grado par, tenemos un camino euleriano y la respuesta sera 0, que es el indice del nodo inicial.

```
else if (n_imp == 0) {
    if (connected (-1)) res = 1;//res-1 = 0
    else res = -1;
}
```

Si existen nodos de grado impar, debemos iniciar nuestro recorrido en cualquiera de ellos, y terminar el en el otro, si no podemos iniciar en cualquier nodo. El arco que esta conectado a uno a uno de estos nodos con el menor indice es la respuesta, otra cosa que debemos tomar en cuenta es que el nodo que elegimos al inicio no puede ser un puente en el grafo, a no ser de que sea la única elección, con una simple búsqueda en profundidad es posible encontrar los puentes del grafo.

Programa Java que soluciona el problema

```
import java.util.*;
class tsp {

    static final int MAX_N = 100;
    static final int MAX_M = 500;
    static int M, N;
    static int[][] edge = new int[MAX_M + 1][2];
    static int[][] inc = new int[MAX_N + 1][MAX_M];
    static int[] d = new int[MAX_N + 1];
    static int forbidden_edge, n_visited;
    static int[] visited = new int[MAX_N + 1];

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int K, e, v, n_imp, j, res;
        for (;;) {
            M = in.nextInt();
            N = in.nextInt();
            if (M == 0) break;
            Arrays.fill(d, 0);
            for(int a=0;a<=MAX_M;a++){ edge[a][0] = edge[a][1] = 0;}
            for (v = 1; v <= N; ++v) {
                K = in.nextInt();
                for (j = 0; j < K; ++j) {
                    e = in.nextInt();
                    e++;
                    inc[v][d[v]++] = e;
                    if (edge[e][0] == 0) edge[e][0] = v;
                    else edge[e][1] = v;
                }
            }
            for (e = 1; e <= M; ++e) {
                if (edge[e][1] == 0) {
                v = edge[e][0];
                edge[e][1] = v;
                inc[v][d[v]++] = e;
                }
            }
            n_imp = 0;
            for (v = 1; v <= N; ++v) {
                if ((d[v] % 2 == 1) && (++n_imp > 2)) break;
            }
            if (n_imp > 2 || !connected(-1)) res = -1;
            else if (n_imp == 0) {
                if (connected (-1)) res = 1;
                else res = -1;
                }
                else {
                    res = -1;
                    for (e = 1; e <= M; ++e) {
                        if ((d[edge[e][0]] % 2 == 0) && (d[edge[e][1]] % 2 == 0))
                            continue;
                        if ((d[edge[e][0]] == 1) || (d[edge[e][1]] == 1) || connected
                            res = e;
                            break;
                        }
                    }
                }
            if (res == -1) System.out.printf("-1\n");
            else System.out.printf("%d\n", res-1);
        }
    }

    public static void dfs (int v) {
        int e, i, w;
        visited[v] = 1;
        ++n_visited;
        for (i = 0; i < d[v]; ++i) {
            e = inc[v][i];
            if (e == forbidden_edge) continue;
            w = edge[e][0];
            if (w == v) w = edge[e][1];
            if (visited[w] != 0) continue;
            dfs (w);
        }
    }

    public static boolean connected (int e) {
        boolean res;
        int i, empty;
        forbidden_edge = e;
        n_visited = 0;
        Arrays.fill(visited, 0);
        for(i = 1, empty = 0; i <= N; i++)
            if (d[i] == 0) empty++;
        for(i = 1; d[i] == 0; i++);
        dfs (i);
        res = n_visited == N-empty;;
        return res;
    }
}
```

Problem L
Bora Bora

Source file name: `bora.c`, `bora.cpp` *or* `bora.java`

Bora Bora is a simple card game for children, invented in the South Pacific Island of the same name. Two or more players can play, using a deck of standard cards. Cards have the usual ranks: Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen and King. Each card has also one of four suits: Clubs, Diamonds, Hearts and Spades.

Players sit on a circle around the table and play by turns. The next player to play may be the one to the left (clockwise direction) or to the right (counter-clockwise direction) of the current player, depending on the cards played, as we will see. At the start, the direction of play is clockwise.

The deck is shuffled and each player is dealt a hand of cards. The remaining of the deck is placed, face down, on the table; this is called the *stock* pile. Then the first (topmost) card is removed from the stock and placed on the table, face up, starting another pile, called the *discard* pile.

The objective of the game is for a player to discard all his cards. At each turn, a player discards at most one card. A card can be discarded only if it has the same rank or the same suit as the topmost card on the discard pile. A player discards a card by placing it, face up, in the discard pile (this card becomes the topmost). If a player does not have a suitable card to discard on his turn, he must draw one card from the stock and add it to his hand; if he can discard that card, he does so, otherwise he does nothing else and his turn ends. A player always discards the highest valued card he possibly can. The *value* of a card is determined first by the card rank and then by the card suit. The rank order is the rank itself (Ace is the lowest, King is the highest), and the suit order is, from lowest to highest, Clubs, Diamonds, Hearts and Spades. Therefore, the highest valued card is the King of Spades and the lowest valued card is the Ace of Clubs. As an example, a Queen of Diamonds has a higher value than a Jack (any suit) but has a lower value than a Queen of Hearts.

Some of the discarded cards affect the play, as follows:

- when a Queen is discarded, the direction of play is reversed: if the direction is clockwise, it changes to counter-clockwise, and vice-versa;
- when a Seven is discarded, the next player to play must draw two cards from the stock (the number of cards in his hand increases by two), and misses his turn (does not discard any card);
- when an Ace is discarded, the next player to play must draw one card from the stock (the number of cards in his hand increases by one), and misses his turn (does not discard any card);
- when a Jack is discarded, the next player to play misses his turn (does not discard any card).

Notice that the penalty for the first card in the discard pile (the card draw from the stock at the beginning) is applied to the first player to play. For example, if the first player to play is p and the first card on the discard pile is an Ace, player p draws a card from the stock and does not discard any card on his first turn. Also notice that if the first card is a Queen, the direction of play is reversed to counter-clockwise, but the first player to play remains the same.

The winner is the player who first discards all his cards (the game ends after the winner discards his last card).

Given the description of the shuffled deck and the number of players, write a program to determine who will win the game.

Input

The input contains several test cases. The first line of a test case contains three integers P, M and N, separated by single spaces, indicating respectively the number of players ($2 \leq P \leq 10$), the number of cards distributed to each of the players at the beginning of the game ($1 \leq M \leq 11$) and the total number of cards in the shuffled deck ($3 \leq N \leq 300$). Each of the next N lines contains the description of one card. A card is described by one integer X and one character S, separated by one space, representing respectively the card rank and the card suite. Card ranks are mapped to integers from 1 to 13 (Ace is 1, Jack is 11, Queen is 12 and King is 13). Card suits are designated by the suit's first letter: '`C`' (Clubs), '`D`' (Diamonds), '`H`' (Hearts) or '`S`' (Spades).

Players are identified by numbers from 1 to P, and sit on a circle, in clockwise direction, $1, 2 \ldots P, 1$. The first $P \times M$ cards of the deck are dealt to the players: the first M cards to the first player (player 1), the next M to the second player (player 2), and so on. After dealing the cards to the players, the next card on the deck — the $(P \times M + 1)$-th card — is used to start the discard pile, and the remaining cards form the stock. The $(P \times M + 2)$-th card to appear on the input is the topmost card on the stock, and the last card to appear on the input (the N-th card) is the bottommost card of the stock (the last card that can be drawn). Player 1 is always the first to play (even when the card used to start the discard pile is a Queen). All test cases have one winner, and in all test cases the number of cards in the deck is sufficient for playing to the end of the game.

The end of input is indicated by a line containing only three zeros, separated by single spaces.

The input must be read from file `bora.in`.

Output

For each test case in the input, your program must print a single line, containing the number of the player who wins the game.

The output must be written to standard output.

Sample input	**Output for the sample input**

```
2 2 10
1 D
7 D
1 S
3 C
13 D
1 S
5 H
12 D
7 S
2 C
3 2 11
1 S
7 D
11 D
3 D
7 D
3 S
11 C
8 C
9 H
6 H
9 S
3 3 16
1 H
10 C
13 D
7 C
10 H
2 S
2 C
10 S
8 S
12 H
11 C
1 C
1 C
4 S
5 D
6 S
0 0 0
```
	```
	1
	3
	2
	```

Análisis del Problema L

Problema de simulación de un juego de cartas (similar a UNO), donde gana el jugador que se deshace de todas sus cartas, en el que con ciertas cartas se puede girar el sentido de los turnos, hacerle perder el turno al compañero y/o adicionarle cartas a su mano.

Para el desarrollo del programa, se creó una clase jugador, donde se almacenan sus cartas y un indicador de cuántas le quedan.

El programa recrea el juego paso a paso,

Notas para entender el código:

1. Para simular el mazo de cartas, se utilizará una cola.
2. El operador & manda la dirección del objeto en sí, y no una copia
3. $pair < int, int >$ es una clase que almacena dos datos enteros llamados first y second.
4. $suit == palo$

En esta solución, se usan más funciones de las necesarias en favor de la comprensión

Programa C que soluciona el problema

```
#include <iostream>
#include <cstdio>
#include <queue>
#include <vector>

using namespace std;

queue<pair<int,int> > mazo;  //palo,numero
pair<int,int> actual; //ultima carta en ser descartada
int p,n,m,direccion;
bool perdio_su_turno;

typedef struct{ //datos de un jugador
    int cartas[4][14];
    int n_cartas;
}jugador;

int convertir_palo(char c){
    if(c=='C')return 0;
    else if(c=='D')return 1;
    else if(c=='H')return 2;
    else return 3;
}
```

```
void inicializar_jugador(jugador *x){//inicializando todo a cero
    for(int i=1;i<=13;i++){
        x->cartas[0][i]=0;
        x->cartas[1][i]=0;
        x->cartas[2][i]=0;
        x->cartas[3][i]=0;
    }
    x->n_cartas=0;
}
void insertar_carta(jugador *x,int p,int c){
    x->cartas[p][c]++;
    x->n_cartas++;
}
int max_por_numero(jugador *x,int p){//seleccionando la mayor carta en relacion al pa
    for(int i=13;i>=1;i--)if(x->cartas[p][i]>0)return i;
    return -1;
}
int max_por_palo(jugador *x,int c){//seleccionando la mayor carta en relacion al nume
    if(x->cartas[3][c]>0)return 3;
    else if(x->cartas[2][c]>0)return 2;
    else if(x->cartas[1][c]>0)return 1;
    else if(x->cartas[0][c]>0)return 0;
    else return -1;
}
void descartar_carta(jugador *x,int p,int c){//descartando carta
    x->cartas[p][c]--;
    x->n_cartas--;
    if(c==12)direccion*=(-1);           //Reina invierte el sentido del juego
    perdio_su_turno=(c==1||c==7||c==11);//as siete o jack
    actual.first=p;                     //actualizamos la carta a ser descargada
    actual.second=c;
}
bool intentando_descartar(jugador *x,int p1,int c1,int p2,int c2){
//devuelve true si es que se pudo
    if(c1!=-1||c2!=-1){ //descartar una carta
        if(c1>c2)descartar_carta(x,p1,c1);//primero por numero
        else if(c1==c2){                  //despues por palo
            if(p1>p2)descartar_carta(x,p1,c1);
            else descartar_carta(x,p2,c2);
        }else descartar_carta(x,p2,c2);
        return true;
    }return false;
}
void resuelva(){
    jugador jugadores[p];
    int carta,sw=0,p1,p2,c1,c2;
    char palo;
```

```
pair<int,int> aux;
direccion=1;//sentido normal, -1 en el caso de que se invierta
perdio_su_turno=false;

for(int i=0;i<p;i++){//insercion de cartas a cada jugador
    inicializar_jugador(&jugadores[i]);
    for(int j=1;j<=m;j++){
        scanf("%d %c",&carta,&palo);
        insertar_carta(&jugadores[i],convertir_palo(palo),carta);
    }
}

n-=(p*m);
while(!mazo.empty())mazo.pop();
for(int i=1;i<=n;i++){//insertando cartas a la cola
    scanf("%d %c",&carta,&palo);
    mazo.push(make_pair(convertir_palo(palo),carta));
}
actual=mazo.front();  //sacando la primera carta
mazo.pop();
perdio_su_turno=(actual.second==1||actual.second==7||actual.second==11);// por si
if(actual.second==12)direccion*=(-1);// as,siete,jack o reina

while(true){
    if(!perdio_su_turno){//validando si el jugador ha perdido su turno
        p1=actual.first;c1=max_por_numero(&jugadores[sw],actual.first);
           // 1°candidato en relacion al numero
        p2=max_por_palo(&jugadores[sw],actual.second);c2=actual.second;
           // 2°candidato en relacion al palo

        //-1 si es que no se encontro algun candidato
        if(c1==-1)p1=-1;    //Para que la funcion de comparacion
        if(p2==-1)c2=-1;    //funcione bien.

        if(!intentando_descartar(&jugadores[sw],p1,c1,p2,c2)){
           //si no descarta se intenta una vez mas
            aux=mazo.front();
            mazo.pop();
            insertar_carta(&jugadores[sw],aux.first,aux.second);
            p1=actual.first;c1=max_por_numero(&jugadores[sw],actual.first);
            p2=max_por_palo(&jugadores[sw],actual.second);c2=actual.second;
            if(c1==-1)p1=-1;
            if(p2==-1)c2=-1;
            intentando_descartar(&jugadores[sw],p1,c1,p2,c2);
                 //intentando descartar por 2° vez
        }else if(jugadores[sw].n_cartas==0){
                //en el caso de que haya descartado en la
```

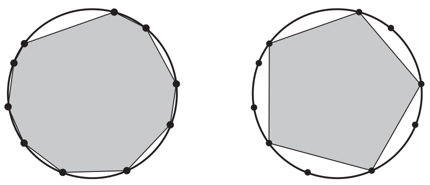
```
                    printf("%d\n",sw+1);
                     //primera vez, se verifica si ya no tiene
                    return; //cartas, si es asi, imprime y termina
                }
            }else{ //si el jugador actual perdio su turno
                if(actual.second==1){//as hace que saque una carte
                    aux=mazo.front();
                    mazo.pop();
                    insertar_carta(&jugadores[sw],aux.first,aux.second);
                }else if(actual.second==7){//siete hace que saque dos cartas
                    aux=mazo.front();
                    mazo.pop();
                    insertar_carta(&jugadores[sw],aux.first,aux.second);
                    aux=mazo.front();
                    mazo.pop();
                    insertar_carta(&jugadores[sw],aux.first,aux.second);
                }
                perdio_su_turno=false;
            }
            sw+=direccion;//pasando al siguiente jugador
            if(sw<0)sw=p-1;//igual a la funcion %p
            else if(sw>=p)sw=0;
        }
    }
int main(){
    while(scanf("%d %d %d",&p,&m,&n)&&(p||m||n))resuelva();
}
```

Problem M
Shrinking Polygons

Source file name: `polygons.c`, `polygons.cpp` *or* `polygons.java`

A polygon is said to be *inscribed* in a circle when all its vertices lie on that circle. In this problem you will be given a polygon inscribed in a circle, and you must determine the minimum number of vertices that should be removed to transform the given polygon into a *regular polygon*, i.e., a polygon that is equiangular (all angles are congruent) and equilateral (all edges have the same length).

When you remove a vertex v from a polygon you first remove the vertex and the edges connecting it to its adjacent vertices w_1 and w_2, and then create a new edge connecting w_1 and w_2. Figure (a) below illustrates a polygon inscribed in a circle, with ten vertices, and figure (b) shows a pentagon (regular polygon with five edges) formed by removing five vertices from the polygon in (a).



In this problem, we consider that any polygon must have at least three edges.

Input

The input contains several test cases. The first line of a test case contains one integer N indicating the number of vertices of the inscribed polygon ($3 \leq N \leq 10^4$). The second line contains N integers X_i separated by single spaces ($1 \leq X_i \leq 10^3$, for $0 \leq i \leq N-1$). Each X_i represents the length of the arc defined in the inscribing circle, clockwise, by vertex i and vertex $(i+1) \bmod N$. Remember that an *arc* is a segment of the circumference of a circle; do not mistake it for a *chord*, which is a line segment whose endpoints both lie on a circle.

The end of input is indicated by a line containing only one zero.

The input must be read from file polygons.in.

Output

For each test case in the input, your program must print a single line, containing the minimum number of vertices that must be removed from the given polygon to form a regular polygon. If it is not possible to form a regular polygon, the line must contain only the value `-1`.

The output must be written to standard output.

Sample input

```
3
1000 1000 1000
6
1 2 3 1 2 3
3
1 1 2
10
10 40 20 30 30 10 10 50 24 26
0
```

Output for the sample input

```
0
2
-1
5
```

Análisis del Problema M

Como nos dice la descripción del problema, un polígono regular es un polígono en el que todos los lados tienen el mismo tamaño y todos los ángulos interiores tienen la misma medida.

Existen dos propiedades de los polígonos regulares que nos ayudará a resolver el problema:

- El perímetro debe ser un número entero.
- El tamaño del arco entre vértices consecutivos en el polígono buscado debería ser divisor del perímetro.

Para esto dentro de un vector de tipo booleano que es del tamaño de todo el perímetro, lleno de `true` en los lugares en donde se encuentra un vértice.

```
for(i = 0; i < N; i++) {//per: perímetro
    per[verLen] = true;
    verLen = verLen + X[i];//verLen: tamaño de cada vértice
}
```

Inicio un contador i que empieza en N, el número total de vértices, y se reduce hasta 3 que es el número mínimo de vértices de un polígono, luego veo que el número de vértices sea un divisor del perímetro (el perímetro debe ser un número entero), inicio *arcLen* que es el tamaño del arco entre dos vértices consecutivos del polígono regular. Solo es posible tener *arcLen* puntos iniciales a tratar, el límite de mi segundo contador j. Para cada punto inicial, veo si hay vértices alrededor del círculo separados por *arcLen*, ésto último cumple con la condición de que todos los lados tienen el mismo tamaño.

```
for(i = N; i >= 3; i--) {
    if((totPer % i) == 0)  {
        arcLen = totPer / i;
        for(j = 0; j < arcLen; j++) {
            isPos = true;//supongo es posible construirlo
            k = j;
            while(k < totPer) {
                if(per[k] == false) isPos = false;//mala suposición
                    k = k + arcLen;
                }
            if(isPos == true) //es posible contruirlo quitando i vertices
        }
    }
    isPos = false;//no es posible contruirlo
}
```

Una segunda alternativa más eficiente utilizando menos memoria es hallar el tamaño del lado y el número de lados. Si sumando las distancias consecutivas podemos construir éste numero de lados tenemos una solución. Esta solución se incluye en lenguaje c.

Programa Java que soluciona el problema

```
import java.util.*;
class polygons {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int N, i, totPer, verLen, j, arcLen, k;
        int[] X;
        boolean[] per;//imeter size
        boolean isPos = true;//ible
        while(true) {
            N = in.nextInt();
            if(N == 0) break;
            X = new int[N];
            totPer = 0;
            for(i = 0; i < N; i++) {
                X[i] = in.nextInt();
                totPer = totPer + X[i];
            }
            per = new boolean[totPer];
            Arrays.fill(per, false);
            verLen = 0;
            for(i = 0; i < N; i++) {
                per[verLen] = true;
                verLen = verLen + X[i];
            }
FOR:        for(i = N; i >= 3; i--) {
                if((totPer % i) == 0)  {
                    arcLen = totPer / i;
                    for(j = 0; j < arcLen; j++) {
                        isPos = true;
                        k = j;
                        while(k < totPer) {
                            if(per[k] == false) isPos = false;
                            k = k + arcLen;
                        }
                        if(isPos == true) break FOR;
                    }
                }
                isPos = false;
            }//
            if(isPos == true) System.out.println(N - i);
            else System.out.println("-1");
        }
    }
}
```

Programa C que soluciona el problema

```
#include<stdio.h>

#define MAXVERTICES 100000
#define TRUE 1
#define FALSE 0

int distancia[MAXVERTICES+1];
int lado,circunferencia;
int n;


int  check(int k) {
  int i,j,jj,in,next,numlados,distanciaActual;

  lado=circunferencia/k;
  in=distanciaActual=0;
  distanciaActual=distancia[in];
  while (distanciaActual<lado) {
    in++;
    distanciaActual+=distancia[in];
  }
  for (i=0;i<=in;i++) {
    numlados=distanciaActual=0;
    next=lado;
    j=i;
    for (jj=0;jj<n;jj++) {
      distanciaActual+=distancia[j];
      if (distanciaActual==next) {
        next+=lado;
        numlados++;
      }
      else if (distanciaActual>next)
        break;
      j=(j+1) % n;
    }
    if (numlados==k)
      return TRUE;
  }
  return FALSE;
}

int main(void) {
  int i;

  while (scanf("%d",&n)==1 && n) {
   circunferencia=0;
    for (i=0;i<n;i++) {
      scanf("%d",&distancia[i]);
     circunferencia+=distancia[i];
    }
    for (i=n;i>2;i--)
      if (circunferencia%i==0 && check(i)) {
printf("%d\n",n-i);
break;
      }
    if (i==2)
      puts("-1");
  }
  return 0;
}
```