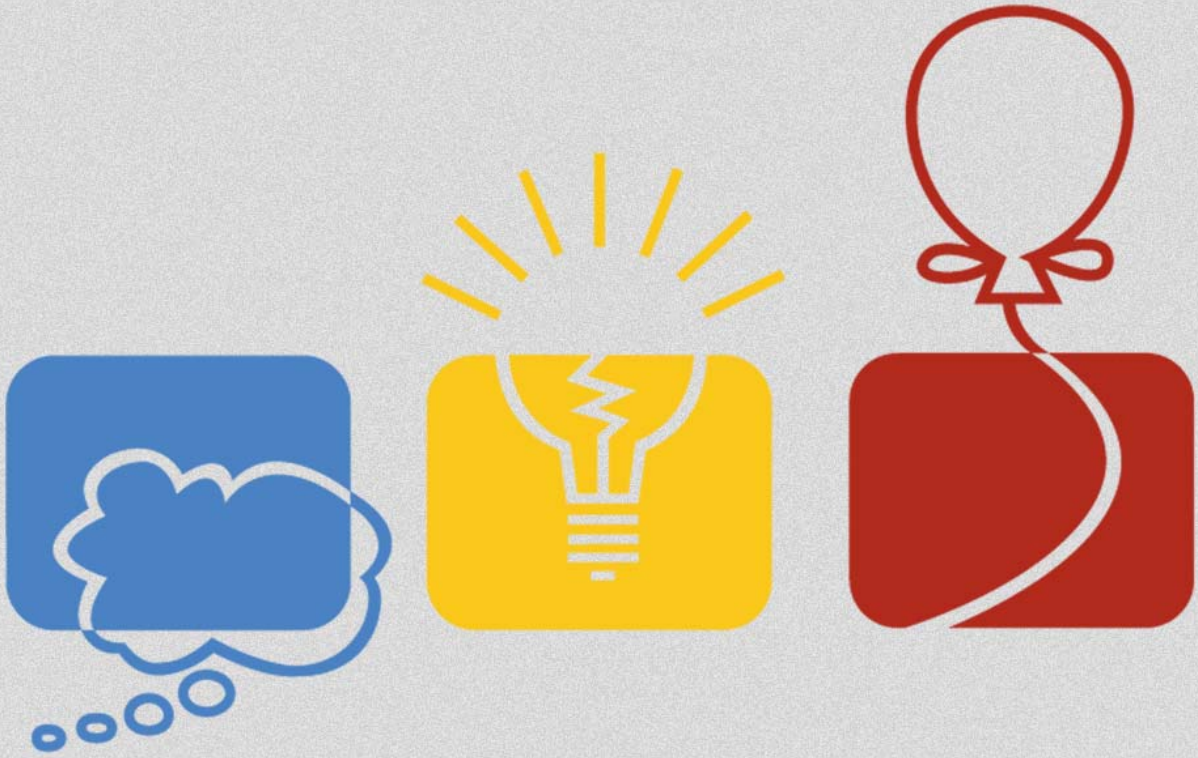


acm International Collegiate Programming Contest

PROBLEMAS RESUELTOS

Segunda Edición

La Paz – Bolivia, Octubre 2010

Jorge Terán Pomier
<teranj@acm.org>

Alberto Suxo Riveros
<albertosuxo@gmail.com>

Gabriel Rivera Safadi
<gabrielriverasafadi@hotmail.com>

Índice

PROBLEMAS DE RESOLUCIÓN PROPIA

Cara o Cruz

John y Mary han sido amigos desde la guardería. Desde entonces, ellos han compartido una rutina juguetona: cada vez que ellos se reúnen ellos juegan cara o cruz con una moneda, y quien gana tiene el privilegio de decidir que van a jugar durante el día. Mary siempre escoge cara, y John siempre escoge cruz.

Hoy en día ellos están en la universidad, pero continúan siendo buenos amigos. Siempre que ellos se encuentran, ellos todavía juegan cara o cruz y el ganador decide qué película mirar, o qué restaurante para tomar la cena juntos, y así sucesivamente.

Ayer Mary confió a John que ella tiene en su custodia un registro de los resultados de cada juego desde que ellos empezaron, en la guardería. Vino como una sorpresa para John! Pero puesto que John está estudiando informática, él decidió que era una oportunidad buena de mostrarle sus habilidades a Mary programando, escribiendo un programa para determinar el número de veces en que cada uno de ellos ganó el juego durante los años.

Entrada

La entrada contiene algunos casos de prueba. La primera línea de un caso de prueba contiene un entero N indicando el número de juegos jugados ($1 \leq N \leq 10000$). La siguiente línea contiene N enteros Ri, separados por un espacio, describiendo la lista de resultados. Si Ri=0 significa que Mary ganó el i-ésimo juego, si Ri=1 significa que John ganó el i-ésimo juego ($1 \leq i \leq N$). El final de las entradas está indicado por N=0;

La entrada debe leerse desde la entrada estándar.

Salida

Para cada caso de prueba en la entrada tu programa debe mostrar una línea conteniendo la frase “Mary won X times and John won Y times”, donde $X \geq 0$ y $Y \geq 0$.

La salida debe escribirse en la salida estándar.

Ejemplo de entrada

```
5
0 0 1 0 1
6
0 0 0 0 0 1
0
```

Ejemplo de salida

```
Mary won 3 times and John won 2 times
Mary won 5 times and John won 1 times
```

Análisis y Solución

Por: Alberto Suxo

El problema consiste únicamente en contar cuantas veces ganó Mary y Cuantas veces ganó John.

O sea, contar cuantos 0's y 1's hay.

Ejemplo

```
6           <= 6 juegos jugados
0 0 0 0 0 1 <= cinco 0's y un 1
```

Código Fuente en C:

```c
/* Problema : Cara o Cruz
 * Lenguaje : ANSI C (version: 4.0 )
 * Por      : Alberto Suxo
 ************************/

#include<stdio.h>

int main(){
   int N, coin, John, Mary;
   int i;

   while( 1 ){
      scanf( "%d", &N );
      if( N==0 )
         break;
      John = Mary = 0;
      for( i=0; i<N; i++ ){
         scanf( "%d", &coin );
         if( coin==1 )
            John++;
         else
            Mary++;
      }
      printf( "Mary won %d times and John won %d times\n", Mary, John);
   }
   return 0;
}
```

Código Fuente en Java:

```
/* Problema : Cara o Cruz
 * Lenguaje : JAVA (version: 1.5 )
 * Por      : Alberto Suxo
 *************************/

import java.util.Scanner;

public class F{

   public static void main(String args[]){
      int N, coin, John, Mary;
      int i;
      Scanner in = new Scanner( System.in );
      while( true ){
         N = in.nextInt();
         if( N==0 )
            break;
         John = Mary = 0;
         for( i=0; i<N; i++ ){
            coin = in.nextInt();
            if( coin==1 )
               John++;
            else
                Mary++;
         }
         System.out.println("Mary won " + Mary + " times and John won "
                            + John + " times" );
      }
   }
}
```

¡Él está en Fuera de Juego!

La Red Hemisferio es la mayor red televisiva en Tumbolia, un pequeño país ubicado al este de Sud América (o al sur de Este América). El juego más popular en Tumbolia, no es sorpresa, es el fútbol; varios juegos son transmitidos cada semana en Tumbolia.

La Red Hemisferio recibe varias solicitudes para volver a mostrar jugadas dudosas; usualmente, esto pasa cuando el árbitro determina que un jugador está en Fuera de Juego.

Un jugador atacante está en *fuera de juego* si el está más cerca a la línea de meta contraria en línea con el penúltimo oponente. Un jugador no está en fuera de juego si

- está al mismo nivel que el penúltimo oponente o
- está al mismo nivel que los dos últimos oponentes.

A través del uso de la tecnología de computación gráfica, la Red Hemisferio puede tomar una imagen del campo y determinar las distancias de los jugadores hacia la línea de meta defensora, pero ellos todavía necesitan un programa que, dadas estas distancias, decida si un jugador está en fuera de juego.

Entrada

El archivo de entrada contiene varios casos de prueba. La primera línea de cada caso de prueba contiene dos enteros A y D separados por un solo espacio indicando, respectivamente, el número de jugadores atacantes y defensores involucrados en la jugada (2 ≤ A, D ≤ 11). La siguiente línea contiene A enteros Bi separados por un solo espacio, indicando las distancias de los jugadores atacantes a la línea de meta (1 ≤ B_i ≤ 10^4). Las siguientes líneas contienen D enteros C_j separados por un solo espacio, indicando las distancias de los jugadores defensores a la línea de meta (1 ≤ C_j ≤ 10^4).

El final de la entrada está indicado por una línea conteniendo solo dos ceros, separados por un solo espacio.

Salida

Para cada caso de prueba en la entrada, imprimir una línea conteniendo un simple caracter: “Y” (mayúscula) si hay un jugador atacante en fuera de juego, y “N” (mayúscula) en otro caso.

Ejemplo de Entrada

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

Ejemplo de Salida

```
N
Y
N
```

Análisis y Solución

Por: Jorge Terán

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.

Código Fuente en Java:

```
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");
         }
      }
   }

}
```

Código Fuente en C++:

```cpp
#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;
}
```

Impares o Pares

Existen Muchas versiones de pares ó impares, un juego qué muchos competidores realizan para decidir muchas cosas, por ejemplo, quien resolverá este problema. En una variación de este juego los competidores comienzan escogiendo ya sea pares ó impares. Después a la cuenta de tres extiende una mano mostrando un número de dedos que pueden ser de cero hasta cinco. Luego se suman la cantidad escogida por los competidores. Si suma par la persona que escogió par gana. Similarmente ocurre lo mismo con la persona qué escoge impar, si suma impar gana.

Juan y Miaría jugaron muchos juegos durante el día. En cada juego Juan escogió pares (y en consecuencia Miaría escogió impares). Durante los juegos se cada jugador anoto en unas tarjetas el numero de dedos que mostró. Miaría utilizo tarjetas azules, y Juan tarjetas rojas. El objetivo era el de revisar los resultados posteriormente. Sin embargo al final del día Juan hizo caer todas las tarjetas. Aún cuando se podían separar por colores no podían ser colocadas en orden original.

Dado un conjunto de números escritos en tarjetas rojas y azules, usted debe escribir un programa para determinar el mínimo de juegos que Miaría con certeza gano.

Entrada

La entrada consiste de varios casos de prueba. La primera línea de la prueba consiste en un entero N que representa el numero de juegos ($1 \leq N \leq 100$). La segunda línea es un caso de prueba que contiene N enteros X_i, Indicando el numero de dedos que mostró Miaría en cada uno de los juegos ($0 \leq X_i \leq 5$, para $1 \leq i \leq N$). La tercera línea contiene N enteros Y_i, el número de dedos que escogió Juan en cada juego. ($0 \leq Y_i \leq 5$, para $1 \leq i \leq N$). El fin de archivo se indica con N = 0.

La entrada se debe leer de standard input (teclado).

Salida

Para cada caso de prueba su programa debe escribir en una línea un numero de entero, indicando el mí mino número de juegos que pudo haber ganado Miaría.

La salida debe ser standard output (pantalla).

Ejemplo de Entrada

```
3
1 0 4
3 1 2
9
0 2 2 4 2 1 2 0 4
1 2 3 4 5 0 1 2 3
0
```

Ejemplo de Salida

```
0
3
```

Análisis y Solución

Por: Jorge Terán

Analicemos el primer ejemplo. Aquí Miaría escogió 1, 0, 4 y Juan 3, 1, 2. Analizando todas las posibilidades vemos que estas son tres

Miaría	Juan	Suma
par	par	par
par	impar	impar
impar	par	impar
impar	impar	par

Veamos, todo número par puede escribirse como 2n y todo numero impar como 2n + 1, de donde se puede fácilmente deducir la tabla anterior.

Si analizamos cuantos impares escogió Miaría, se ve que es solo el numero 1. La única posibilidad de ganar seria cuando Juan escogió par o sea 2. En el caso de que Miaría hubiese escogido par o sea 0 o 4 solo podría ganar cuando Juan escogió 1.

Contando los casos tenemos:

	Pares	Impares
Miaría	2	1
Juan	1	2

El mínimo número de juegos que Miaría podría ganar es 1 – 1.

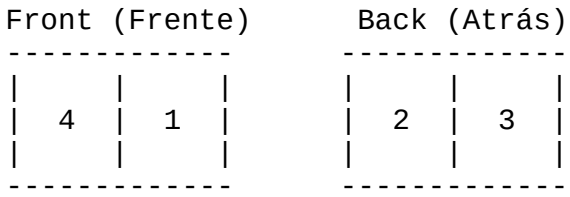
Código Fuente en Java:

La solución completa del problema es la siguiente:

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

public class Odds {
   public static void main(String[] args) {
      int Juan = 0, Maria = 0, x, n, i;
      Scanner in = new Scanner(System.in);
      while ((n = in.nextInt()) > 0) {
         Juan = 0;
         Maria = 0;
         for (i = 0; i < n; i++) {
            x = in.nextInt();
            if ((x & 1) == 1)
               Maria++;
         }
         for (i = 0; i < n; i++) {
            x = in.nextInt();
            if ((x & 1) == 0)
               Juan++;
         }
      }
      System.out.println(Juan > Maria ? Juan - Maria : Maria - Juan);
   }
}
```

Imprimiendo Folleto

Cuando imprimimos un documento, normalmente la primera página es impreso primero, luego la segunda, luego la tercera y así hasta en final. Sin embargo, cunado creamos un folleto plegado, el orden de la impresión debe ser alterado. Un folleto plegado tiene cuatro páginas por hoja, con dos en el frente y dos atrás. Cuando apila todas las hojas en orden, se pliegan y las páginas aparecen en el orden correcto con en un libro regular. Por ejemplo, un folleto de 4 páginas es impreso en una hoja conteniendo en el frente las páginas 4 y 1, y en el reverso las páginas 2 y 3.



Tu trabajo es escribir un programa que dado un número de páginas para imprimir, generar el orden de impresión.

La entrada contiene uno o más casos de prueba, seguidos por una línea con un número 0 que indica el final de la entrada. Cada caso de prueba consiste en un entero positivo n en cada línea, donde n es el número de páginas a imprimir; n no es mayor a 100.

Por cada caso de prueba, despliegue un reporte que indique que páginas son impresas en cada página, como se muestra en el ejemplo. Si el número de páginas no llena completamente las hojas, imprima la palabra Blank en lugar de un número. Si el frente o atrás de una hoja es completamente blanca, no genere su salida. La salida debe estar en orden ascendente por hojas, iniciando por el frente hasta el final.

Entrada

```
1
14
4
0
```

Salida

```
Printing order for 1 pages:
Sheet 1, front: Blank, 1
Printing order for 14 pages:
Sheet 1, front: Blank, 1
Sheet 1, back : 2, Blank
Sheet 2, front: 14, 3
Sheet 2, back : 4, 13
Sheet 3, front: 12, 5
Sheet 3, back : 6, 11
Sheet 4, front: 10, 7
Sheet 4, back : 8, 9
Printing order for 4 pages:
Sheet 1, front: 4, 1
Sheet 1, back : 2, 3
```

Análisis y Solución

Por: Alberto Suxo

Primero debemos saber cuantas páginas necesitaremos para cada caso de prueba, será de esta forma

páginas	hojas
1	1
2 a 4	2
5 a 8	4
9 a 12	6
13 a 16	8
17 a 20	10

El número de páginas estará dado por:

```
pages=(n+3)/4;
pages*=2;
```

para el caso de n = 5

```
0 1
2 0
0 3
4 5
```

Donde los 0's representan a las páginas en blanco. Ahora si creamos una variable x que será igual a 2*pages+1, la razón de esta variable x es, cunado hagamos un recorrido por cada una de las páginas que tendremos (pages), si x – #page > n, entonces sabemos que esta página estará en blanco, caso contrario existe, por lo que la imprimimos. Posterior a esto, solo es necesario saber en orden en el que deben ser impresos.

Veamos código fuente para comprenderlo de mejor forma.

Código Fuente en C:

```c
/* Problema : Imprimiendo Folleto
 * Lenguaje : ANSI C
 * Por      : Alberto Suxo
 ************************/
#include<stdio.h>

int main(){
      int n, pages;
      int i, j, x;

   while( 1 ) {
      scanf( "%d", &n );
      if( !n )
         break;
      pages=(n+3)/4;
      pages*=2;
      x = pages*2 +1;
      printf( "Printing order for %d pages:\n", n );
      if( n==1 )
         printf( "Sheet 1, front: Blank, 1\n" );
      else{
         for( i=1; i<=pages; i++ ) {
            printf( "Sheet %d, %s: ",(i+1)/2, (i&1)?"front":"back " );
            if( i&1 ) {
               if( x-i>n )
                  printf( "Blank, %d\n", i );
               else
                  printf( "%d, %d\n", x-i, i );
            } else {
               if( x-i>n )
                  printf( "%d, Blank\n", i );
               else
                  printf( "%d, %d\n", i, x-i );
            }
         }
      }
   }
   return 0;
}
```

Lotería de Fin de Semana

Algunas personas están contra loterías por razones morales, algunos gobiernos prohíben éste tipo de juegos, pero con la aparición del Internet esta forma de juego popular va prosperando, que comenzó en China y ayudo financiar la "Gran Muralla".

Las probabilidades de ganar una lotería nacional se dan, y por lo tanto sus compañeros de clase de colegio decidieron organizar una pequeña lotería privada, con el sorteo cada viernes. La lotería está basada en un estilo popular: un estudiante que quiere apostar escoge C números distintos de 1 a K y paga 1.00 Bs. (Las loterías tradicionales como la lotería estadounidense utilizan C = 6 y K = 49).

El viernes durante el almuerzo C números (de 1 a K) son extraídos. El estudiante cuya apuesta tiene el número más grande de aciertos recibe la cantidad de las apuestas. Esta cantidad es compartida en caso de empate y se acumulad a la próxima semana si nadie adivina cualquiera de los números extraídos.

Algunos de sus colegas no creen en las leyes de probabilidad y desean que usted escriba un programa que determine los números a escoger, considerando los números que menos salieron en sorteos anteriores, de modo que ellos puedan apostar a aquellos números.

Entrada

La entrada contiene varios casos de prueba. La primera línea de un caso de prueba contiene tres números enteros N, C y K que indica respectivamente el número de sorteos que ya han pasado ($1 \leq N \leq 10000$), cuantos números comprenden una apuesta ($1 \leq C \leq 10$) y el valor máximo de los números que pueden ser escogidos en una apuesta ($C < K \leq 100$). Cada una de las N líneas siguientes contiene C números enteros distintos X_i que indica los números extraídos en cada sorteo anterior ($1 \leq X_i \leq K$; para$1 \leq i \leq C$). Los valores finales de entrada se indica por N = C = K = 0.

Salida

Para cada caso de prueba en la entrada su programa debe escribir una línea de salida, conteniendo el juego de números que han sido han salido la menor cantidad de veces. Este juego debe ser impreso como una lista, en orden creciente de números. Inserte un espacio en blanco entre dos números consecutivos en la lista.

Ejemplo de entrada	**Ejemplo de salida**
5 4 6 6 2 3 4 3 4 6 5 2 3 6 5 4 5 2 6 2 3 6 4 4 3 4 3 2 1 2 1 4 4 3 2 1 4 3 0 0 0	1 1 2 3 4

Análisis y Solución

Por: Jorge Terán

Analizando el problema

Si revisamos el problema vemos que lo se pide es hallar la frecuencia de los valores que vienen en el archivo.

En el ejemplo la entrada 5 4 6 indica que han existido 5 sorteos, lo que implica leer 5 datos, el numero máximo de estos es 6, y en cada apuesta hay cuatro números. Esto significa leer 5 filas de cuatro números.

Al leer contamos cuantas veces ha salido cada número y luego se imprimen los números menores al número que haya salido menos veces.

Para realizar esta cuenta se puede utilizar el siguiente código

```
X = in.nextInt();
count[x]++;
```

Código Fuente en Java:

La solución completa del problema es la siguiente:

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

public class Lottery {
   /**
    * J. Teran requieres jdk 1.5
    */
   public static final int MAX_N = 10000;

   public static final int MAX_K = 100;

   public static void main(String[] args) {

      int[] count;
      int n = 1, c, k;
      int i, j, x, min;
      boolean first;

      Scanner in = new Scanner(System.in);
      // n=in.nextInt();

      while ((n = in.nextInt()) > 0) {
         c = in.nextInt();
         k = in.nextInt();
         count = new int[MAX_K + 1];
         for (i = 0; i < n; i++) {
            for (j = 0; j < c; j++) {
               x = in.nextInt();
               count[x]++;
            }
         }
         min = n;
```

```
            for (i = 1; i <= k; i++)
               if (count[i] < min)
                  min = count[i];
            first = true;
            for (i = 1; i <= k; i++) {
               if (count[i] == min) {
                  if (!first)
                     System.out.print(" ");
                  first = false;
                  System.out.print(i);
               }
            }
            System.out.print("\n");
            // n=in.nextInt();
         }
      }
   }
```

Mapas Burbuja

Bubble Inc Esta desarrollando una nueva tecnología para recorrer un mapa en diferentes niveles de zoom. Su tecnología asume que la región **m** se mapea a una región rectangular plana y se divide en sub - regiones rectangulares que representan los niveles de zoom.

La tecnología de Bubble Inc. representa mapas utilizando la estructura conocida como *quad-tree*.

En un quad-tree, una regio rectangular llamada *x* puede ser dividida a la mitad, tanto horizontalmente como verticalmente resultando en cuatro sub regiones del mismo tamaño. Estas sub regiones se denominan regiones hijo de *x*, y se llaman *xp* para la superior izquierda, *xq* para la superior derecha, *xr* de abajo y a la derecha y *xs* para las de abajo a la izquierda *xc* representa la concatenación de la cadena *x* y carácter c = 'p', 'q', 'r' o 's'. Por ejemplo si la región base mapeada se denomina *m*, las regiones hijo de *m* son de arriba y en el sentido del reloj: *mp*, *mq*, *mr* y *ms*, como se ilustra.

mp	mq
ms	mr

Cada sub región puede ser subdividida. Por ejemplo la región denominada *ms* puede ser sub dividida en mas sub regiones *msp*, *msq*, *msr* y mss, como se ilustra.

msp	msq
mss	msr

Como otro ejemplo la figura muestra el resultada de dividir las sub regiones hijo de llamada msr.

msrpp	msrpq	msrqp	msrqq
msrps	msrpr	msrqs	msrqr
msrsp	msrsq	msrrp	msrrq
msrss	msrsr	msrrs	msrrr

Los nombres de la sub regiones tienen la misma longitud de del nivel de zoom, dado que representan regiones del mismo tamaño. Las sub regiones en el mismo nivel de zoom que comparten un lado común se dicen vecinos.

Todo lo que esta fuera de la región base *m* no se mapea para cada nivel de zoom todas las sub regiones de *m* son mapeadas.

La tecnología de mapas Bubble's le provee al usuario provee al usuario una forma de navegar de una sub región a una sub región vecina en las direcciones arriba, abajo, izquierda y derecha. Su misión es la de ayudar a Bubble Inc. a encontrar la sub región vecina de una sub región dada. Esto es que dado el nombre de una sub región rectangular usted debe determinar los nombres de sus cuatro vecinos.

Entrada

La entrada contiene varios casos de prueba. La primera línea contiene un entero representando el número de casos de prueba. La primera línea contiene un entero *N* indicando el número de casos de prueba. Cada una de las siguientes *N* líneas representan un caso de prueba conteniendo el nombre la región Compuesta por *C*

caracteres (2 ≤ C ≤ 5000), la primera letra siempre es una letra 'm' seguida por una de las siguientes 'p', 'q', 'r' o 's'.

La entrada se debe leer de standard input (teclado).

Salida

Para cada caso en la entrada su programa debe producir una línea de salida, que contiene los nombres de las cuatro regiones de una región dada, en el orden de arriba abajo izquierda y derecha. Para vecinos que no esta en mapa debe escribir <none> en lugar de su nombre. Deje una línea en blanco entre dos nombres consecutivos.

La salida debe ser standard output(pantalla).

Ejemplo de entrada

```
2
mrspr
mps
```

Ejemplo de salida

```
mrspq mrssq mrsps mrsqs
mpp msp <none> mpr
```

Análisis y Solución

Por Jorge Terán

Analizando el problema

Para analizar el problema realizaremos una expansión de los datos de de los ejemplos. Comencemos con *mps*. La región *m* se divide en p, q, r y s:

```
mp   mq
ms   mr
```

Cada región a su vez se divide en cuatro regiones formando:

```
mpp   mpq   mqp   mqq
mps   mpr   mqs   mqr
msp   msq   mrp   mrq
mss   msr   mrs   mrr
```

Analizando la región mps se ve claramente que tiene un vecino arriba mpp, abajo *msp* a la izquierda no tiene porque esta fuera del área, y a la derecha *mpr*, lo que produce la respuesta al ejemplo planteado. Por ejemplo la región *msq* tiene cuatro vecinos.

Para esbozar una solución podemos desarrollar el caso de buscar el vecino superior. Los casos cuando estamos en una región *r* o una región *s* siempre se tiene un vecino superior dado que cada región tiene cuatro partes de las cuales *r* y *s* están abajo.

En los otros casos que son los sectores *p* y *q* puede ser que no tengan vecino arriba. Para esto debemos recorrer la cadena hacia atrás hasta llegar a una región que tenga un vecino superior. Si llegamos al final de la cadena sin lograr esto significa que no tiene un vecino.

En el código queda representado como:

```
public static void Arriba(int i) {
   if (i == 0) {
      vecinos[0] = 1;
      return;
   }
   switch (vecinos[i]) {
   case 'p':
      Arriba(i - 1);
      vecinos[i] = 's';
      break;
   case 'q':
      Arriba(i - 1);
      vecinos[i] = 'r';
      break;
   case 'r':
      vecinos[i] = 'q';
      break;
   case 's':
      vecinos[i] = 'p';
      break;
   }
}
```

Para poder trabajar la cadena de caracteres leídos como un vector se ha convertido esta a un vector de caracteres. Esto se hace

```
m = in.nextLine();
vecinos = m.toCharArray();
```

La lógica para las 3 regiones restantes es la misma.

Código Fuente en Java:

La solución completa del problema es la siguiente:

```
import java.util.*;

public class Maps {
   /**
    * J. Teran requieres jdk 1.5
    */
   public static char[] vecinos = new char[5000];

   public static void main(String[] args) {
      int n, i, l, l1;
      String m;
      Scanner in = new Scanner(System.in);
      m = in.nextLine();
      n = Integer.parseInt(m);
      for (i = 0; i < n; i++) {
         m = in.nextLine();
         vecinos = m.toCharArray();
         l1 = m.length() - 1;
         l = l1;
         Arriba(l);
         if (vecinos[0] == 1)
            System.out.print("<none> ");
         else {
            System.out.print(vecinos);
            System.out.print(" ");
         }
         l = l1;
         vecinos = m.toCharArray();
         Abajo(l);
         if (vecinos[0] == 1)
            System.out.print("<none> ");
         else {
            System.out.print(vecinos);
            System.out.print(" ");
         }
         vecinos = m.toCharArray();
         l = l1;
         Izquierda(l);
         if (vecinos[0] == 1)
            System.out.print("<none> ");
         else {
            System.out.print(vecinos);
            System.out.print(" ");
         }
```

```
      vecinos = m.toCharArray();
      l = l1;
      l = l1;
      Derecha(l);
      if (vecinos[0] == 1)
         System.out.println("<none>");
      else {
         System.out.println(vecinos);
      }
   }
}

public static void Arriba(int i) {
   if (i == 0) {
      vecinos[0] = 1;
      return;
   }
   switch (vecinos[i]) {
   case 'p':
      Arriba(i - 1);
      vecinos[i] = 's';
      break;
   case 'q':
      Arriba(i - 1);
      vecinos[i] = 'r';
      break;
   case 'r':
      vecinos[i] = 'q';
      break;
   case 's':
      vecinos[i] = 'p';
      break;
   }
}

public static void Abajo(int i) {
   if (i == 0) {
      vecinos[0] = 1;
      return;
   }
   switch (vecinos[i]) {
   case 'p':
      vecinos[i] = 's';
      break;
   case 'q':
      vecinos[i] = 'r';
      break;
   case 'r':
      Abajo(i - 1);
      vecinos[i] = 'q';
      break;
   case 's':
      Abajo(i - 1);
      vecinos[i] = 'p';
```

```
            break;
        }
    }

    public static void Derecha(int i) {
        if (i == 0) {
            vecinos[0] = 1;
            return;
        }
        switch (vecinos[i]) {
        case 'p':
            vecinos[i] = 'q';
            break;
        case 'q':
            Derecha(i - 1);
            vecinos[i] = 'p';
            break;
        case 'r':
            Derecha(i - 1);
            vecinos[i] = 's';
            break;
        case 's':
            vecinos[i] = 'r';
            break;
        }
    }

    public static void Izquierda(int i) {
        // System.out.print("i="+i);
        //System.out.println(vecinos);
        if (i == 0) {
            vecinos[0] = 1;
            return;
        }
        switch (vecinos[i]) {
        case 'p':
            Izquierda(i - 1);
            vecinos[i] = 'q';
            break;
        case 'q':
            vecinos[i] = 'p';
            break;
        case 'r':
            vecinos[i] = 's';
            break;
        case 's':
            Izquierda(i - 1);
            vecinos[i] = 'r';
            break;
        }
    }
}
```

Análisis y Solución 2

Por: Alberto Suxo

Analizando el Problema

Para entender y resolver el problema es necesario poner énfasis en el primer párrafo del planteamiento del problema. Este párrafo nos dice:

Bubble Inc. Desarrolla una nueva solución de mapeo. Asume que el área es superficie plana, y ha logrado representarlo en árboles-cuadrangulares (árboles con cuatro hojas).

Teniendo una región rectangular llamada ***x***, podemos dividirla por la mitad horizontal y verticalmente obteniendo cuatro sub-regiones rectangulares de igual tamaño. Estas sub-regiones son llamadas regiones hijas de x, y son llamadas: ***xp*** para superior-izquierdo, ***xq*** para superior-derecho, ***xr*** para inferior-derecho y ***xs*** para inferior-izquierdo, donde ***xc*** representa la concatenación de la cadena x con el carácter ***c***=’p’, ‘q’, ‘r’ o ‘s’. Por ejemplo, si tenemos una región llamada ***prs***, sus hijos serán ***prsp***, ***prsq***, ***prsr*** y ***prss***.

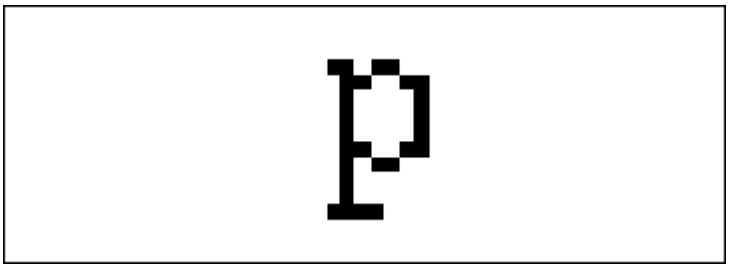
¿Qué es lo que nos pide?

En base a una cadena (que empiece con p) debemos hallar (desplegar en pantalla) a sus vecinos superior, inferior, izquierdo y derecho respectivamente, en caso de no existir vecino desplegar “<no-map>”.

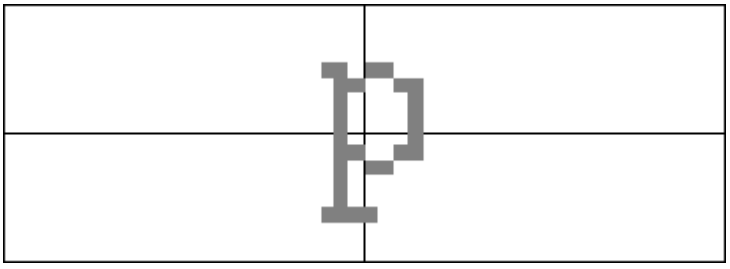
¿Cómo lo haremos?

Primero simularé lo que esta en el planteamiento del problema, con esto podremos comprender mejor la solución al problema.

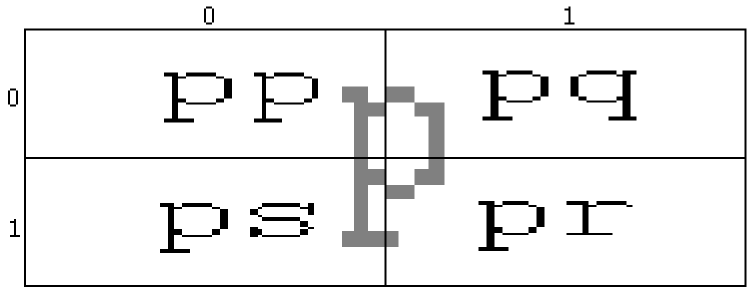
Tenemos un área (región) inicial llamada p.



La dividimos en cuatro sub-regiones de igual tamaño:

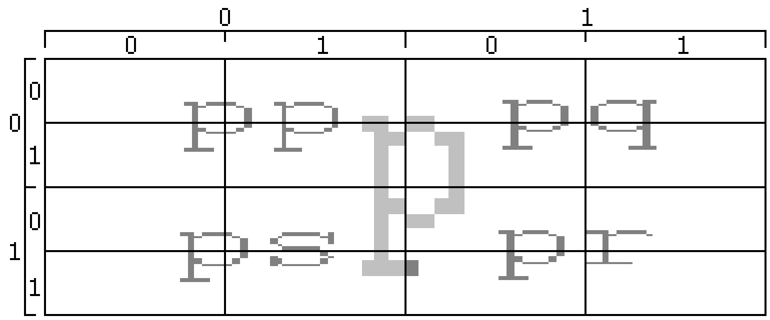


Y concatenamos con p, q, r y s en sentido de las manecillas del reloj respectivamente:



El resultado es un área (matriz) de 2x2.

Volvemos a dividirlo en cuatro regiones iguales cada sub-región:



Y concatenamos con p, q, r y s en sentido de las manecillas del reloj respectivamente:

		0		1	
		0	1	0	1
0	0	ppp	ppq	pqp	pqq
	1	pps	ppr	pqs	pqr
1	0	psp	psq	prp	prq
	1	pss	psr	prs	prr

El resultado es una matriz de 4x4:

Volvemos a realizar el proceso de división en cuatro partes iguales de cada sub-región, concatenamos y el resultado es:

			0				1			
			0		1		0		1	
			0	1	0	1	0	1	0	1
0	0	0	pppp	pppq	ppqp	ppqq	pqpp	pqpq	pqqp	pqqr
		1	ppps	pppr	ppqs	ppqr	pqps	pqpr	pqqs	pqqr
	1	0	ppsp	ppsq	pprp	pprq	pqsp	pqsq	pqrp	pqrq
		1	ppss	ppsr	pprs	pprr	pqss	pqsr	pqrs	pqrr
1	0	0	pspp	pspq	psqp	psqq	prpp	prpq	prqp	prqq
		1	psps	pspr	psqs	psqr	prps	prpr	prqs	prqr
	1	0	pssp	pssq	psrp	psrq	prsp	prsq	prrp	prrq
		1	psss	pssr	psrs	psrr	prss	prsr	prrs	prrr

El resultado es una matriz de 8x8.

Con este ejemplo ya podemos sacar nuestras conclusiones:

1. Si en tamaño de la cadena es k, la matriz resultante tendrá un tamaño de $2^{k-1}x2^{k-1}$
2. las coordenadas de columna y fila se duplican por cada carácter en la cadena.
3. Si mi cadena es pqsr sus coordenadas son columna 101= 5 y fila 011 = 3
4. Lógicamente, un proceso inverso puede transformar coordenadas x, y (columna, fila) en una cadena (nombre de sub-región).

Ejemplo:

En los casos de entrada se propone:

```
pps
```

en la imagen 5 podemos apreciar que sus coordenadas son (x, y) = $(00,01)_b$ = (0, 1)

y sus vecinos tienen las siguientes coordenadas:

arriba = (x, y-1) = (0, 0) =$(00,00)_b$ = ppp
abajo = (x,y+1) = (0, 2) = $(00,10)_b$ = psp
izquierda = (x-1,y) = (-1, 1) = fuera de la matriz = <no-map>
derecha = (x+1, y) = (1, 1) = $(01,01)_b$ = ppr
como resultado debemos desplegar:
ppp psp <no-map> ppr

Asumo que no es necesaria mayor explicación así que aquí va el código fuente en Java:

Código Fuente en Java:

```
/* Problema : Boubble Maps
 * Lenguaje : JAVA (version: 1.5 )
 * Por      : Alberto Suxo
 *************************/
import java.util.Scanner;

public class Maps {

   static int x, y, size;

   static void coordenadas(String cad) {
      int i;
      x = 0;
      y = 0;
      for (i = 0; i < size; i++) {
         x <<= 1;
         y <<= 1;
         switch (cad.charAt(i)) {
         case 'p': break;
         case 'q': x++; break;
         case 'r': x++; y++; break;
         case 's': y++; break;
         default: System.out.println("Invalid character");
         }
      }
   }
```

```
   static String cadena( int x, int y ) {
      char c[]={'p','s','q','r'};
      int dim, xx=x, yy=y;
      int i;
      String resp="";
      dim = 1<<(size-1);
      if( xx<0 || xx>=dim || yy<0 ||y>=dim ) {
         return "<no-map>";
      }
      for( i=size-1; i>=0; i-- ) {
         resp = c[((xx&1)<<1)+(yy&1)] + resp;
         xx >>= 1;
         yy >>= 1;
      }
      return "m" + resp;
   }

   public static void main(String args[]) {
      int n;
      String cad = null;
      Scanner in = new Scanner( System.in );
      n = in.nextInt();
      while ( 0 < n-- ) {
         cad = in.next();
         size = cad.length();
         coordenadas( cad );
         System.out.println(cadena(x, y - 1) + " " + cadena(x, y + 1)
                  + " " + cadena(x - 1, y) + " " + cadena(x + 1, y) );
      }
   }
}
```

Monstruo de las Galletas

La lectura de datos es de teclado. Los resultados se muestran por pantalla.

Todo el mundo necesita pasatiempos y Cristian no es la excepción. Él es fuera de serie porque es un aficionado a las galletas. Donde hay galletas, ahí estará Cristian. Lamentablemente, el no es demasiado bueno para hacer seguimiento a la entregas de las galletas. Tiene que comer por lo menos una galleta todos los días, y siempre que haya los suficientes sobrantes, el comerá **C** de ellos. Dado que él tiene **N** galletas sobrantes, por cuántos días el come al menos una galleta?

Entrada

La primera línea de entrada contiene un número único **T**, el número de casos de prueba ($0 < T \leq 100$).

Cada caso de prueba consiste en una línea que contiene dos números N y C, el número total de “galletas" que tiene y el número que come cada día. ($0 < N \leq 1000000000$), ($0 < C \leq 5000$).

La entrada termina con t = 0.

Salida

Para cada caso de prueba, imprima una línea que contiene un solo número, la cantidad de días en el que Cristian tiene todavía “galletas" para comer.

Ejemplo de Entrada

```
2
6 2
10 3
0
```

Ejemplo de Salida

```
3
4
```

Análisis y Solución

Por: Gabriel Rivera

Analizando el problema

Sabemos que el monstruo tiene que comer una cantidad de galletas por día, y si sobran galletas, come un día más.

Analizando la entrada, si tiene 6 galletas y tiene que comer 2 por día, entonces, 6 / 2 = 3 y no sobra nada, así que come 3 días. Si tiene 10 galletas y tiene que comer 3 por día, entonces 10 / 3 = 3 y 10 % 3 = 1, entonces, come 3 días, pero como sobraron galletas, come un día más, por lo que come 4 días.

Código Fuente en Java:

```
/* Problema : Monstruo de las Galletas
 * Lenguaje : Java JDK 1.6
 * Por      : Gabriel Rivera Safadi
 ********************************/

import java.util.Scanner;

class Monstruo {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		while (true) {
			int t = sc.nextInt();
			if (t == 0) {
				break;
			}
			for (int tt = 0; tt < t; tt++) {
				int n = sc.nextInt();
				int c = sc.nextInt();
				int res = n / c;
				res += n % c != 0 ? 1 : 0;
				System.out.println(res);
			}
		}
	}

}
```

Números Romanos

Escriba un programa que convierta un entero positivo en un número romano. Asuma que los números a convertir son menores a 3500. Las reglas para construir un número romano son las que siguen.

En el sistema de números romanos, i es el símbolo para 1, v para 5, x para 10, l para 50, c para 100, d para 500 y m para 1000. Los símbolos con un valor grande usualmente aparecen antes que los símbolos de temor valor. El valor de un número romano es, en general, la suma de los valores de los símbolos. Por ejemplo, ii es 2, viii es 8. Sin embargo, si un símbolo de menor valor aparece antes de un símbolo de mayor valor, el valor de los dos símbolos es la diferencia de los dos valores. Pro ejemplo, iv es 4, ix es 9, y lix es 59. Note que no hay cuatro símbolos consecutivos iguales. Por ejemplo, iv, pero no iiii, es el número 4. Los números romanos construidos de esta forma pueden no ser únicos. Por ejemplo, ambos mcmxc y mxm son validos para 1990. Aunque el número romano generado por su programa no debe necesariamente ser el más corto, nunca use vv para 10, ll para 100, dd para 1000, o vvv para 15, etc.

Entrada

La entrada consistirá en una serie de líneas, cada línea conteniendo un entero x. L entrada terminará cuando la línea tenga un 0.

Salida

Por cada número, imprima el número en decimal y en forma romana.

Ejemplo de entrada

```
3
8
172
4
1990
5
0
```

Ejemplo de Salida

```
3      iii
8      viii
172    clxxii
4      iv
1990   mcmxc
5      v
```

Nota: La salida esta en este formato:

```
         111
123456789012
|-----||----
1990   mcmxc
```

Análisis y Solución

Por: Alberto Suxo

El problema solicita traducir un número menor o igual a 3500 a número romano.

La solución clásica es, dado un número N, hacer un bucle mientras N sea mayor a cero, y dentro de este bucle tener un conjunto de if's anidados preguntando si el número N es mayor o igual a algún valor.

Ejemplo: Para N en un rango entre 1 y 35 pues es el siguiente código.

```
while(N>0)
ff(N>=10){print X, N=N-10}
else if(N=9){print IX, N=0}           //N=0 <=> N = N-9
     else if(N>=5){print V, N=N-5}
          else if(N=4){print IV, N=0} //N=0 <=> N = N-4
               else{ print I, N=N-1}
```

Y esto mismo se puede hacer para el problema planteado.

En lo personal, a mi me disgusta utilizar tantos if's, así que propongo la siguiente solución, que utiliza un solo if, esta solución puede ser planteada en clase para demostrar que siempre hay una forma más fácil de hacer las cosas.

Código Fuente en C:

```c
/* Problema : Numeros Romanos
 * Lenguaje : ANSI C (version 4.0)
 * Por      : Alberto Suxo
 ************************/
#include<stdio.h>

int main(){
  int  x, i;
  int  Vn[13]={ 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1 };
  char *Vc[13]={"m","cm","d","cd","c","xc","l","xl","x","ix","v","iv",
                "i"};
   while( 1 ){
      scanf( "%d", &x );
      if( x==0 )
         break;
      printf( "%-4d   ", x);
      i = 0;
      while( x>0 ){
         if( x>=Vn[i] ){
            printf( "%s", Vc[i] );
            x = x - Vn[i];
         }
         else
            i++;
      }
      printf( "\n" );
   }
   return 0;
}
```

Código Fuente en JAVA:

```
/* Problema : Numeros Romanos
 * Lenguaje : JAVA (version: 1.5 )
 * Por      : Alberto Suxo
 ************************/
import java.util.Scanner;

public class A{
   public static void main(String args[]){
      int  x, i;
      int  Vn[]={1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1};
      String Vc[]={"m","cm","d","cd","c","xc","l","xl","x","ix","v",
                   "iv","i"};
      Scanner in = new Scanner( System.in );
      while( true ){
         x = in.nextInt();
         if( x==0 )
            break;
         System.out.printf( "%-4d   ", x);
         i = 0;
         while( x>0 ){
            if( x>=Vn[i] ){
               System.out.print( Vc[i] );
               x = x - Vn[i];
            }
            else
               i++;
         }
         System.out.println();
      }
   }
}
```

Petya

Petya es bien conocida por sus famosas empanadas de repollo. El cumpleaños de Petya será muy pronto, y quiere invitar a la mayor cantidad de invitados posible. Pero quiere que todos para prueben la especialidad de la casa. Es por eso que necesita saber el número de las empanadas que se tendrá que cocinar con los ingredientes disponibles. Petya tiene **P** gramos de harina, **M** mililitros de leche y **C** gramos de repollo y cuenta con un montón de otros ingredientes. Petya sabe que necesita **K** gramos de harina, **R** mililitros de leche y **V** gramos de repollo para cocinar una empanada. Por favor, ayuda Petya a calcular el número máximo de empanadas que se pueden cocinar.

Input

La entrada contiene los números enteros P, M, C, K, R y V, separadas por espacios y/o saltos de línea (1 ≤ P, M, C, K, R, < V ≤ 10000). La entrada termina con P = M = C = K = R = V = 0.

Output

Imprima el número máximo de empanadas que Petya puede cocinar.

Ejemplo de entrada

```
3000 1000 500
30 15 60
0 0 0 0 0 0
```

Ejemplo de salida

```
8
```

Análisis y Solución

Por : Gabriel Rivera

Analizando el Problema

Se ve que lo único que hay que encontrar es el mínimo que lograríamos con cada cantidad de ingredientes, y ese es el máximo que podemos hacer con estos.

Analizando la entrada de ejemplo, tenemos lo siguiente:

Ingrediente	Existente	Necesario	Cantidad Resultado
Harina	3000	30	100
Leche	1000	15	66
Repollo	500	60	8

Vemos que con esa cantidad de harina podríamos obtener 100 empanadas, pero el repollo sería insuficiente para eso, por lo que la mayor cantidad de empanadas que podríamos obtener es el menor de esos resultados, en este caso, 8 empanadas.

Código Fuente en Java:

```
/* Problema : Petya
 * Lenguaje : Java JDK 1.6
 * Por      : Gabriel Rivera Safadi
 ********************************/

import java.util.Scanner;

class Petya {

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      while (true) {
         int p = sc.nextInt();
         int m = sc.nextInt();
         int c = sc.nextInt();
         int k = sc.nextInt();
         int r = sc.nextInt();
         int v = sc.nextInt();
         if (p == 0 && m == 0 && c == 0 && k == 0 && r == 0 && v == 0){
            break;
         }
         int pk = p / k;
         int mr = m / r;
         int cv = c / v;
         int min = pk < mr ? pk : mr;
         min = min < cv ? min : cv;
         System.out.println(min);
      }
   }

}
```

Romanos, ¿Cuantos son?

Muchas personas están familiarizadas con los números romanos relativamente pequeños. Los símbolos "i", "v", "x", "l", y "c" representan los valores decimales 1, 5, 10, 50, y 100 respectivamente. Para representar otros valores, estos símbolos, y múltiplos son necesarios, son concadenados, con el símbolo de menor valor escrito a la derecha. Por ejemplo, en número 3 es representado con "iii", y el valor 73 es representado con "lxxiii". La excepción a esta regla ocurre con los números con valores de 4 o 9, y con decenas de 40 o 90. Para estos casos, la representación de los números romanos son "iv" (4), "ix" (9), "xl" (40), y "xc" (90). Así que la representación romana para 24, 39, 44, 49, y 94 son "xxiv", "xxxix", "xliv", "xlix", y "xciv", respectivamente.

Los números de las páginas de los prefacios de muchos libros están numeradas con números romanos, comenzando con "i" para la primera página, y continuando la secuencia. Asumiendo que un libro tiene 10 o menos páginas en el prefacio. ¿Cuantos caracteres "i", "v", "x", "l", y "c" se requieren para numerar las páginas del prefacio?. Por ejemplo, un prefacio con cinco páginas usará los números romanos "i", "ii", "iii", "iv", y "v", significando que necesita 7 caracteres "i", y 2 caracteres "v".

Entrada

La entrada consiste en una secuencia de enteros en el rango de 1 a 100, que termina con un cero. Por cada uno de estos, exceptuando el cero, determine el número de caracteres diferentes que necesitará para numerar las paginas con número romanos.

Salida

Por cada entero en la entrada, escriba un línea que contiene el entero de entrada y el número de caracteres de cada tipo que son requeridos. El ejemplo mostrado abajo ilustrará el formato aceptable.

Sample Input

```
1
2
20
99
0
```

Sample Output

```
1: 1 i, 0 v, 0 x, 0 l, 0 c
2: 3 i, 0 v, 0 x, 0 l, 0 c
20: 28 i, 10 v, 14 x, 0 l, 0 c
99: 140 i, 50 v, 150 x, 50 l, 10 c
```

Análisis y Solución

Por: Alberto Suxo

En el planteamiento del problema se especifica que los números con los que trabajaremos son del 1 al 100, por lo que se nos hace más simple si para resolver el problema dividimos nuestro número en unidades y decenas (incluyendo al 100 entre las decenas), para lo que calcularemos el número de caracteres necesarios para las unidades y las decenas por separado.

Código Fuente en C:

```
/* Problema : Romanos ¿Cuánto Son?
 * Lenguaje : ANSI C (version: 4.0 )
 * Por      : Alberto Suxo
 ************************/
#include <stdio.h>

#define I 0
#define V 1
#define X 2
#define L 3
#define C 4

int R[5];

void unidades( int n ) {
   switch(n) {
      case 1: R[I]+=1; break;
      case 2: R[I]+=2; break;
      case 3: R[I]+=3; break;
      case 4: R[I]+=1; R[V]+=1; break;
      case 5: R[V]+=1; break;
      case 6: R[V]+=1; R[I]+=1; break;
      case 7: R[V]+=1; R[I]+=2; break;
      case 8: R[V]+=1; R[I]+=3; break;
      case 9: R[I]+=1; R[X]+=1; break;
   }
}

void decenas( int n ) {
   switch(n) {
      case 1: R[X]+=1; break;
      case 2: R[X]+=2; break;
      case 3: R[X]+=3; break;
      case 4: R[X]+=1; R[L]+=1; break;
      case 5: R[L]+=1; break;
      case 6: R[L]+=1; R[X]+=1; break;
      case 7: R[L]+=1; R[X]+=2; break;
      case 8: R[L]+=1; R[X]+=3; break;
      case 9: R[X]+=1; R[C]+=1; break;
      case 10: R[C]+=1; break;
   }
}
```

```c
int main() {
   int i, n;
   /*freopen("roman.in", "r", stdin);*/
   /*freopen("roman.sol", "w", stdout);*/

   while( 1 ) {
      scanf( "%d", &n );
      if( n == 0 )
        break;
      R[I] = R[V] = R[X] = R[L] = R[C] = 0;
      for( i=1; i<=n; i++ ) {
         unidades( i%10 );
         decenas( i/10 );
      }
      printf("%d: %d i, %d v, %d x, %d l, %d c\n",
               n, R[I], R[V], R[X], R[L], R[C] );
   }
   return 0;
}
```

PROBLEMAS CON CADENAS.

¿Fácil de Decir?

Un password seguro es algo delicado. Los usuarios prefieren passwords que sean fáciles de recordar (como *amigo*), pero este password puede ser inseguro. Algunos lugares usan un generador randómico de passwords (como *xvtpzyo*), pero los usuarios toman demasiado tiempo recordándolos y algunas veces lo escriben en una nota pegada en su computador. Una solución potencial es generar password “pronunciables” que sean relativamente seguros pero fáciles de recordar.

FnordCom está desarrollando un generador de passwords. Su trabajo en el departamento de control de calidad es probar el generador y asegurarse de que los passwords sean aceptables. Para ser aceptable, el password debe satisfacer estas tres reglas:

1. Debe contener al menos una vocal.
2. No debe tener tres vocales consecutivas o tres consonantes consecutivas.
3. No debe tener dos ocurrencias consecutivas de la misma letra, excepto por ‘ee’ o ‘oo’.

(Para el propósito de este problema, las vocales son 'a', 'e', 'i', 'o', y 'u'; todas las demás letras son consonantes.) Note que Estas reglas no son perfectas; habrán muchas palabras comunes/pronunciables que no son aceptables.

La entrada consiste en una o más potenciales passwords, uno por línea, seguidas por una línea conteniendo una palabra 'end' que señala el fin de la entrada. Cada password tiene como mínimo una y como máximo veinte letras de largo y esta formado por solo letras en minúscula. Por cada password, despliegue si es o no aceptable, usando el formato mostrado en el ejemplo de salida.

Ejemplo de entrada:

```
a
tv
ptoui
bontres
zoggax
wiinq
eep
houctuh
end
```

Ejemplo de salida:

```
<a> is acceptable.
<tv> is not acceptable.
<ptoui> is not acceptable.
<bontres> is not acceptable.
<zoggax> is not acceptable.
<wiinq> is not acceptable.
<eep> is acceptable.
<houctuh> is acceptable.
```

Análisis y Solución

Por Alberto Suxo

Este problema presenta tres simples reglas que se deben cumplir para que un password sea aceptable.

Primero es necesario identificar si un carácter es vocal o no, para lo cual utilizo la siguiente función que pregunta si el carácter es una vocal, y devuelve 1 (**true**) por verdad y 0 (**false**) por falso.

```
int isVowel( char ch ){
   if( ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u' )
      return 1;
   return 0;
}
```

También podemos utilizar la un MACRO para reemplazar esta función en C:

```
#define isVowel( ch ) ( ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u' )
```

El código es representado como función para simplificar la comprensión de su traducción en JAVA.

Ahora iremos validando cada una de las tres condiciones expuestas en el planteamiento del problema en su respectivo orden.

Debe contener al menos una vocal.- Basta con realizar un recorrido por toda la cadena (word), en cuanto encontremos una vocal se retorna 1 (**true**) y si se ha terminado de hacer el recorrido y no se ha encontrado ninguna vocal retornamos 0 (**false**).

```
int rule_1(){
   int i;
   for( i=0; i<len; i++ )
      if( isVowel( word[i] ) )
         return 1;
   return 0;
}
```

No debe tener tres vocales consecutivas o tres consonantes consecutivas.- Otra vez un recorrido por toda la cadena, pero esta vez utilizando dos contadores **v** y **c**, que se incrementan en cuanto se encuentra una vocal o consonante respectivamente, en cuanto alguno llegue a tres, la función termina con falso, y si logra terminar el recorrido sin problemas se da por cumplida la segunda regla.

```
int rule_2() {
   int i, v=0, c=0;
   for( i=0; i<len; i++ ) {
      if( isVowel( word[i] ) ) {
         v++; c=0;
      } else {
         c++; v=0;
      }
      if( v==3 || c==3 )
         return 0;
   }
```

```
   return 1;
}
```

No debe tener dos ocurrencias consecutivas de la misma letra, excepto por 'ee' o 'oo'.- Otro recorrido más, esta función, como las anteriores es muy explícita, la única diferencia es que empieza en el segundo elemento de la cadena y no así en el primero.

```
int rule_3() {
   int i;
   for( i=1; i<len; i++ ) {
      if( word[i]==word[i-1] && word[i]!='e' && word[i]!='o' )
         return 0;
   }
   return 1;
}
```

Bien, ahora solo resta leer cadenas, verificamos si cumplen las tres reglas e imprimir el resultado correspondiente, en caso de haber leído la cadena 'end', termina el programa.

Código Fuente en C:

```
/* Problema : ¿Fácil de Decir?
 * Lenguaje : ANSI C (version: 4.0)
 * Por      : Alberto Suxo
 ************************/

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

char word[25];

int len;

int isVowel( char ch ) {
   if( ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u' )
      return 1;
   return 0;
}

int rule_1() {
   int i;
   for( i=0; i<len; i++ )
      if( isVowel( word[i] ) )
         return 1;
   return 0;
}

int rule_2() {
   int i, v=0, c=0;
   for( i=0; i<len; i++ ) {
      if( isVowel( word[i] ) ) {
         v++;  c=0;
      }else{
         c++;  v=0;
      }
```

```
      if( v==3 || c==3 )
         return 0;
   }
   return 1;
}

int rule_3() {
   int i;
   for( i=1; i<len; i++ ) {
      if( word[i]==word[i-1] && word[i]!='e' && word[i]!='o' )
         return 0;
   }
   return 1;
}

int main() {
   while(1) {
      scanf( "%s", word);
      if( strcmp( word, "end" )==0 )
         break;
      len = strlen( word );
      if( rule_1() && rule_2() && rule_3() )
         printf( "<%s> is acceptable.\n", word );
      else
         printf( "<%s> is not acceptable.\n", word );
   }
   return 0;
}
```

Código Fuente en JAVA:

```
/* Problema :¿Fácil de Decir?
 * Lenguaje : JAVA (version: 1.5)
 * Por      : Alberto Suxo
 *************************/
import java.util.Scanner;

public class D {
   static char word[];
   static int len;

   static boolean isVowel( char ch ) {
      if( ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u' )
         return true;
      return false;
   }

   static boolean rule_1() {
      for( int i=0; i<len; i++ )
         if( isVowel( word[i] ) )
            return true;
      return false;
   }
```

```
   static boolean rule_2() {
      int i, v=0, c=0;
      for( i=0; i<len; i++ ) {
         if( isVowel( word[i] ) ) {
            v++; c=0;
         }else{
            c++; v=0;
         }
         if( v==3 || c==3 )
            return false;
      }
      return true;
   }

   static boolean rule_3() {
      for( int i=1; i<len; i++ ) {
         if( word[i]==word[i-1] && word[i]!='e' && word[i]!='o' )
            return false;
      }
      return true;
   }

   public static void main( String args[] ) {
      String line;
      Scanner in = new Scanner( System.in );
      while( true ) {
         line = in.next();
         if( line.equals("end") )
            break;
         word = line.toCharArray();
         len = line.length();
         if( rule_1() && rule_2() && rule_3() )
            System.out.println( "<" + line+ "> is acceptable." );
         else
            System.out.println( "<" + line+ "> is not acceptable." );
      }
   }
}
```

¿Es posible hacerlo todo con una única función?.

Claro, existen muchas formas de hacer las cosas.

```
int rules_1_2_3() {
   int i, v=0, c=0, vocs=0;
   for( i=0; i<len; i++ ) {
      if( word[i]=='a' || word[i]=='e' || word[i]== 'i' ||
          word[i]=='o' || word[i]=='u' ){
         v++;  c=0;  vocs++;
      } else {
         c++;  v=0;
      }
      if( v==3 || c==3 )
         return 0;
      if( word[i]==word[i+1] && word[i]!='e' && word[i]!='o' )
         return 0;
```

```
    }
    return vocs;
}
```

Esta función es más complicada, y más difícil de entender, pero es un poco más rápida.

Haciendo Fácil SQL

SQL (Structured Query Language, Lenguaje Estructurado de Consulta) es un lenguaje hecho para manipular los datos contenidos dentro de una Base de Datos. Cada tabla en una Base de Datos tiene un DDL (Data Definition Language, Lenguaje de Definición de Datos) y un DML (Data Management Language, Lenguaje de Manipulación de Datos).

Para hacer la Manipulación de Datos más fácil, queremos crear la sentencia SELECT, usando la Definición de Datos de cada tabla, que usa la sentencia CREATE TABLE.

Por ejemplo, tenemos la tabla: CREATE TABLE table1 (field1 type1, field2 type2) Y para obtener su contenido usamos: SELECT field1, field2 FROM table1

Tu tarea es dada una sentencia CREATETABLE, construir la sentencia SELECT.

Input

La entrada tiene varios casos de prueba. Cada línea contiene una sentencia CREATE TABLE. El fin de la entrada es representada por un #.

Output

Para cada sentencia CREATE TABLE, imprimir la sentencia SELECT

Ejemplo de Entrada

```
CREATE TABLE table1 (field1 type1, field2 type2)
CREATE TABLE table2 (field3 type3, field4 type4, field5 type5, field6
type6)
#
```

Ejemplo de Salida

```
SELECT field1, field2 FROM table1
SELECT field3, field4, field5, field6 FROM table2
```

Análisis y Solución

Por: Gabriel Rivera

Se tiene la siguiente cadena:

```
CREATE TABLE table1 (field1 type1, field2 type2)
```

De donde nos interesa obtener el nombre de la tabla, el cual se encuentra antes de la apertura de paréntesis y después de la palabra TABLE, para lo cual obtenemos la cadena y eliminamos espacios, con eso obtendríamos el nombre de la tabla.

Luego necesitamos los nombres de los campos, los cuales están dentro de los paréntesis, separados por comas. Para esto, separamos todo lo que esté dentro de los paréntesis por las comas y tendríamos:

```
field1 type1      field2 type2
```

Ahora, obtenemos todo lo que esté antes del primer espacio, que indica el nombre del campo, es decir:

```
field1      field2
```

Finalmente armamos la cadena nueva con estos datos, que sería:

```
SELECT field1, field2 FROM table1
```

Código Fuente en Java:

```
import java.util.Scanner;
import java.util.StringTokenizer;
class Sql {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      while (true) {
         String str = sc.nextLine();
         if (str.equals("#"))
            break;
         String res = "SELECT ";
         String tabla = str.substring(str.indexOf("TABLE") + 5,
                                      str.indexOf("(")).trim();
         String campos = str.substring(str.indexOf("(") + 1,
                                       str.indexOf(")"));
         StringTokenizer st = new StringTokenizer(campos, ",");
         while (st.hasMoreTokens()) {
            String campo = st.nextToken().trim();
            campo = campo.substring(0, campo.indexOf(" "));
            if (res.length() > "SELECT ".length())
               res = res + ", " + campo;
            else
               res = res + campo;
         }
         res = res + " FROM " + tabla;
         System.out.println(res);
      }
   }
}
```

Instruens Fabulam

Instruens Fabulam es la manera de *dibujar un cuadro* (o tabla) en idioma Latino. Esto es lo que debes hacer para este problema.

La entrada consiste en una o más descripciones de tablas, seguidas por una línea cuyo primer carácter es '*', que señala el final de la entrada. Cada descripción empieza con una línea de encabezado que contiene uno o más caracteres que definen el número y el alineamiento de de las columnas de la tabla. Los caracteres del encabezado son '<','=' o '>' que son las justificaciones izquierda, central y derecha de cada columna. Después del encabezado hay al menos dos y a lo sumo 21 líneas de datos que contienen las entradas de cada fila. Cada línea de datos consiste en una o más entradas (no vacías) separadas por un ampersand ('&'), donde el número de entradas es igual al número de columnas definidas en el encabezado. La primera línea contiene los títulos de las columnas, y las líneas de datos restantes contienen las entradas del cuerpo de la tabla. Los espacios pueden aparecer dentro de una entrada, pero nunca al principio ni al final de la entrada. Los caracteres '<', '=', '>', '&', y '*' no aparecerán en la entrada excepto en los lugares indicados arriba.

Por cada descripción de tabla, despliegue la tabla usando el formato exacto mostrado en el ejemplo. Note que:

- El ancho total de la tabla no excederá los 79 caracteres (sin contar el fin-de-línea).
- Los guiones ('-') son usados para dibujar líneas horizontales, no ('_'). El signo de arroba ('@') aparece en cada esquina. El signo de suma ('+') aparece en una intersección entre la línea que separa el título y el cuerpo de la tabla.
- Las entradas de una columna estas separadas por el carácter ('|') por exactamente un espacio.
- Si una entrada centreada no es exactamente centreada en una columna, el espacio extra debe ir a la derecha de la entrada.

Ejemplo de Entrada:

```
<>=>
TITLE&VERSION&OPERATING SYSTEM&PRICE
Slug Farm&2.0&FreeBSD&49.99
Figs of Doom&1.7&Linux&9.98
Smiley Goes to Happy Town&11.0&Windows&129.25
Wheelbarrow Motocross&1.0&BeOS&34.97
>
What is the answer?
42
<>
Tweedledum&Tweedledee
"Knock, knock."&"Who's there?"
"Boo."&"Boo who?"
"Don't cry, it's only me."&(groan)
*
```

Ejemplo de Salida:

```
@-------------------------------------------------------------------@
| TITLE                      | VERSION | OPERATING SYSTEM |  PRICE |
|----------------------------+---------+------------------+--------|
| Slug Farm                  |     2.0 |     FreeBSD      |  49.99 |
| Figs of Doom               |     1.7 |      Linux       |   9.98 |
| Smiley Goes to Happy Town |    11.0 |     Windows      | 129.25 |
| Wheelbarrow Motocross      |     1.0 |       BeOS       |  34.97 |
@-------------------------------------------------------------------@
@---------------------@
| What is the answer? |
|---------------------|
|                  42 |
@---------------------@
@-----------------------------------------------@
| Tweedledum                 |     Tweedledee |
|----------------------------+----------------|
| "Knock, knock."            | "Who's there?" |
| "Boo."                     |     "Boo who?" |
| "Don't cry, it's only me." |        (groan) |
@-----------------------------------------------@
```

Análisis y Solución

Por: Alberto Suxo

Este problema no requiere mayor explicación, basta con ver el ejemplo de entrada y salida para saber de que se trata, y claro, al leer el planteamiento del problema, se tiene la seguridad que éste es un problema de formato de salida.

También nos aclara que el tamaño de la tabla jamás excederá los 79 caracteres. Si todas nuestras columnas tuvieran solo un caracter, entonces tendríamos como máximo 20 columnas (en realidad 19), así:

```
@-------------------------------------------------------------------------------@<81
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
```

Y también sabemos que tendremos de 2 a 21 filas en nuestra tabla. Así que, en conclusión, necesitamos una tabla de cadenas de 21 filas y 20 columnas para almacenar nuestros campos.

Código Fuente en C:

```c
/* Problema : Instruens Fabulam
 * Lenguaje : ANSI C (version: 4.0)
 * Por      : Alberto Suxo
 ************************/

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

char Mat[25][20][80], dir[100];
int size[20], cols;

void chars( char ch, int length ) {
   int i;
   for( i=0; i<length; i++ )
      putchar( ch );
}

void print( char ch1, char ch2 ) {
   int i;
   putchar( ch1 );
   for( i=0; i<cols; i++ ){
      if( i )
         putchar( ch2 );
      chars( '-', size[i]+2 );
   }
   printf( "%c\n", ch1 );
}

void printline( int row ) {
   int le, ri, wrd, i;
   putchar( '|' );
   for( i=0; i<cols; i++ ){
      if( i ) putchar( '|' );
      wrd = strlen( Mat[row][i] );
      switch( dir[i] ) {
```

```
            case '<':le = 1; ri = size[i]-wrd+1; break;
            case '>':le = size[i]-wrd+1; ri = 1; break;
            case '=':le = (size[i]-wrd)/2 +1; ri = size[i]+2 - wrd-le;
                     break;
         }
         chars( ' ', le );
         printf( "%s", Mat[row][i] );
         chars( ' ', ri );
      }
      printf( "%c\n", '|' );
   }

   int main() {
      char line[100], *cad;
      int col, row, sz, i, j;
      gets( line );
      while( 1 ) {
         if( line[0]=='*' )
           break;
         strcpy( dir, line );
         cols = strlen( line );
         for( i=0; i<cols; i++ )
            size[i] = 0;
         row = 0;
         while( 1 ) {
            gets( line );
            if( line[0]=='<' || line[0]=='>' || line[0]=='='
                || line[0]=='*' )
               break;
            for( cad = strtok(line,"&"), col=0;
                 cad!=NULL; cad = strtok(NULL,"&" ), col++ ) {
               strcpy( Mat[row][col], cad );
               sz = strlen( cad );
               if( sz > size[col] )
                  size[col] = sz;
            }
            row++;
         }
         print( '@', '-' );
         printline( 0);
         print( '|', '+' );
         for( j=1; j<row; j++ ) {
            printline( j );
         }
         print( '@', '-' );
      }
      return 0;
   }
```

Código Fuente en JAVA:

```
/* Problema : Instruens Fabulam
 * Lenguaje : JAVA (version: 1.5)
 * Por      : Alberto Suxo
 *************************/

import java.util.Scanner;
import java.util.StringTokenizer;

public class Fab {

   static String[][] Mat = new String[25][20];
   static String dir;
   static int[] size;
   static int cols;

   static void chars( char ch, int length ) {
      for( int i=0; i<length; i++ )
         System.out.print( ch );
   }

   static void print( char ch1, char ch2 ) {
      System.out.print( ch1 );
      for(int i=0; i<cols; i++){
         if( i!=0 )
            System.out.print( ch2 );
         chars('-', size[i]+2 );
      }
      System.out.println( ch1 );
   }

   static void printline( int row ) {
      int le, ri, wrd, i;
      System.out.print( '|' );
      for(i=0; i<cols; i++){
         if( i!=0 ) System.out.print( '|' );
         wrd = Mat[row][i].length();
         switch( dir.charAt(i) ){
            case '<': le = 1; ri = size[i]-wrd+1; break;
            case '>': le = size[i]-wrd+1; ri = 1; break;
            default : le = (size[i]-wrd)/2 +1; ri = size[i]+2 - wrd-le;
                      break;
         }
         chars(' ', le );
         System.out.print( Mat[row][i] );
         chars(' ', ri );
      }
      System.out.println( '|' );
   }
```

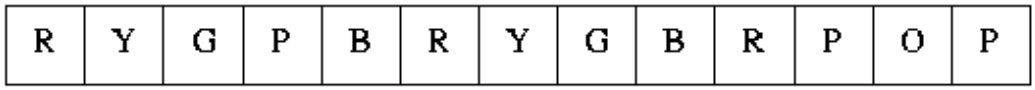
```
   public static void main( String args[] ) {
      String line, cad;
      int col, row, sz, i, j;
      char ch;
      Scanner in = new Scanner( System.in );
      line = in.nextLine();
      while( true ) {
         if( line.equals("*") )
            break;
         dir = line;
         cols = line.length();
         size = new int[cols];
         row=0;
         while( true ) {
            line = in.nextLine();
            ch = line.charAt(0);
            if( ch=='<' || ch=='>' || ch=='=' || ch=='*' )
               break;
            StringTokenizer st = new StringTokenizer(line,"&");
            for (col=0; st.hasMoreTokens(); col++) {
               cad = st.nextToken();
               Mat[row][col] = cad;
               sz = cad.length();
               if (sz > size[col])
                  size[col] = sz;
            }
            row++;
         }
         print('@','-');
         printline(0);
         print('|','+');
         for(j=1; j<row; j++) {
            printline(j);
         }
         print('@','-');
      }
   }
}
```

PROBLEMAS DE SIMULACIÓN.

Colorville

Un simple juego de niños usa un tablero que es una secuencia de cuadrados coloreados. Cada jugador tiene una pieza de juego. Los jugadores alternan turnos, sacando cartas que tienen cada una uno o dos cuadrados coloreados del mismo color. Los jugadores mueven su pieza hacia adelante en el tablero hacia el siguiente cuadrado que haga pareja con el color de la carta, o hacia adelante hasta el segundo cuadrado que haga pareja con el color de la carta que contiene dos cuadrados coloreados, o hacia adelante hasta el último cuadrado en el tablero si no hay un cuadrado con el que emparejar siguiendo la descripción anterior. Un jugador gana si su pieza está en el último cuadrado del tablero. Es posible que no exista ganador después de sacar todas las cartas.

En este problema los colores se representan las letras mayúsculas A-Z, a continuación se presenta un ejemplo.



Considere el siguiente deck de cartas: R, B, GG, Y, P, B, P, RR

Para 3 jugadores, el juego procede como sigue:

```
Jugador 1 saca R,  se mueve al 1er cuadrado
Jugador 2 saca B,  se mueve al 5to cuadrado
Jugador 3 saca GG, se mueve al 8vo cuadrado
Jugador 1 saca Y,  se mueve al 2do cuadrado
Jugador 2 saca P,  se mueve al 11vo cuadrado
Jugador 3 saca B,  se mueve al 9no cuadrado
Jugador 1 saca P,  se mueve al 4to cuadrado
Jugador 2 saca RR, Gano! (no hay R's al frente de esta pieza así que va
hasta el último cuadrado).
```

Usando la misma tabla y el mismo deck de cartas, pero con 2 jugadores, el jugador 1 gana después de 7 cartas. Con 4 jugadores, no hay ganador después de utilizar todas las 8 cartas.

La entrada consiste en información de uno o más juegos. Cada juego comienza con una línea conteniendo el número de jugadores (1-4), el número de cuadrados en el tablero (1-79), y el número de cartas en el deck (1-200). Seguido por una línea de caracteres que representan los cuadrados coloreados del tablero. Seguidos por las cartas en el deck, uno el cada línea. Las Cartas pueden tener una letra o dos de las mismas letras. El final de la entrada esta señalado con una línea que tiene 0 para el número de jugadores – los otros valores son indiferentes.

Por cada juego, la salida es el jugador ganador y el número total de cartas usadas, o el número de cartas en el deck, como se muestra en el ejemplo de salida. Siempre use el plural "cards".

Ejemplo de entrada

```
2 13 8
RYGPBRYGBRPOP
R
B
GG
Y
P
B
P
RR
2 6 5
RYGRYB
R
YY
G
G
B
3 9 6
QQQQQQQQQ
Q
QQ
Q
Q
QQ
Q
0 6 0
```

Ejemplo de salida

```
Player 1 won after 7 cards.
Player 2 won after 4 cards.
No player won after 6 cards.
```

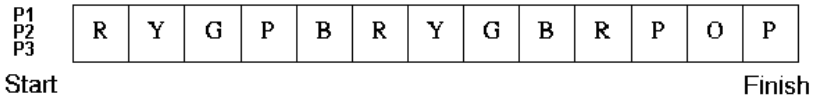
Análisis y Solución

Por: Alberto Suxo

El primer párrafo del planteamiento del problema describe claramente en que consiste el mismo.

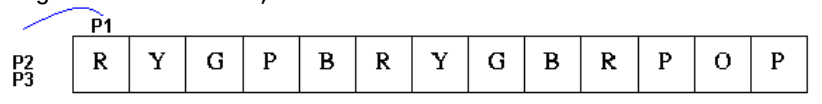
Para quien no entendió en qué consiste este problema, bastará con ver mejor el ejemplo propuesto en el problema.

Este es nuestro tablero, y al inicio se encuentran nuestros tres jugadores (P1, P2 y P3)

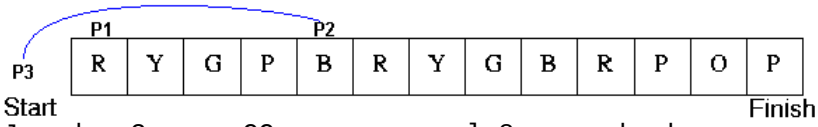


También sabemos que las cartas de nuestro deck saldrán en el siguiente orden: R, B, GG, Y, P, B, P, RR.

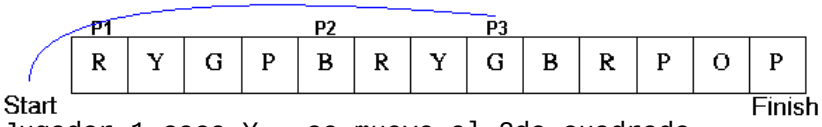
```
Jugador 1 saca R,  se mueve al 1er cuadrado
```




```
Jugador 2 saca B,  se mueve al 5to cuadrado
```

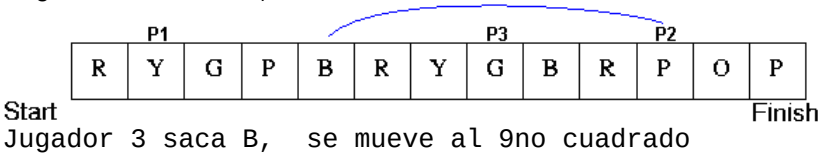



```
Jugador 3 saca GG, se mueve al 8vo cuadrado
```

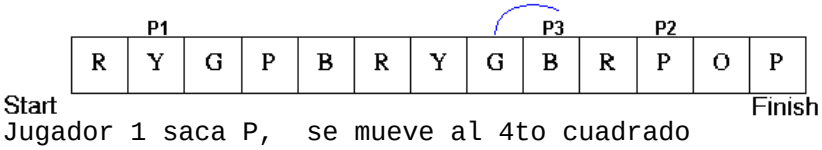



```
Jugador 1 saca Y,  se mueve al 2do cuadrado
```

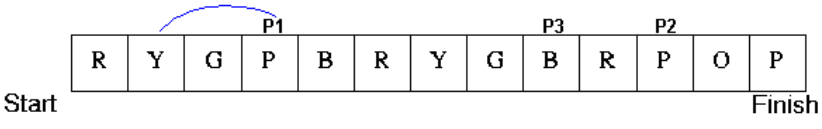

```
Jugador 2 saca P,  se mueve al 11vo cuadrado
```



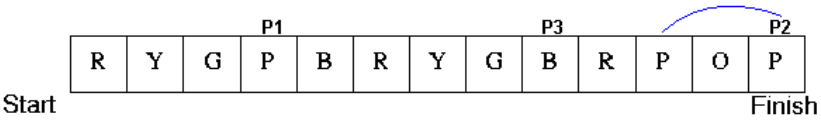

```
Jugador 3 saca B,  se mueve al 9no cuadrado
```




```
Jugador 1 saca P,  se mueve al 4to cuadrado
```



Jugador 2 saca RR, Gano! (no hay R's al frente de esta pieza así que va hasta el último cuadrado).



Bueno, con la simulación gráfica del ejemplo del problema ya se entiende todo perfectamente.

Código Fuente en C:

```
/* Problema : Colorville
 * Lenguaje : ANSI C (version: 4.0)
 * Por      : Alberto Suxo
 ************************/
#include<stdio.h>

char board[100];

int hasNext( int ps, char ch ) {
   int i;
   for( i=ps+1; board[i]!='\0'; i++ ) {
      if( board[i]==ch )
         return i;
   }
   return -1;
}

int main(){
   int players, size, cards;
   char card[5];
   int pos[4];
   int i, j, win, player;
   while( 1 ){
      scanf( "%d %d %d\n", &players, &size, &cards );
      if( players==0 )
         break;
      scanf( "%s", board );
      win = 0;
      pos[0] = pos[1] = pos[2] = pos[3] = -1;
      for(i=0; i<cards; i++) {
         scanf( "%s", card );
         if( !win ){
            player = i % players;
            for( j=0; card[j]!='\0' && !win; j++ ) {
               pos[player] = hasNext( pos[player], card[j] );
               if( pos[player]<0 || pos[player]==(size-1) ) {
                  win = 1;
                  printf( "Player %d won after %d cards.\n",
                          player+1, i+1);
               }
            }
         }
      }
```

```
      if( !win )
         printf( "No player won after %d cards.\n", cards );
   }
   return 0;
}
```

Código Fuente en JAVA:

```
/* Problema : Colorville
 * Lenguaje : JAVA (version: 1.5)
 * Por      : Alberto Suxo
 ************************/
import java.util.Scanner;

public class C{

   public static void main( String args[] ){
      int players, size, cards;
      String board, card;
      int[] pos = new int[4];
      int i, j, player;
      boolean win;

      Scanner in = new Scanner( System.in );
      while( true ){
         players = in.nextInt();
         size = in.nextInt();
         cards = in.nextInt();
         if( players==0 )
            break;
         board = in.next();
         win = false;
         pos[0] = pos[1] = pos[2] = pos[3] = -1;
         for( i=0; i<cards; i++ ){
            card = in.next();
            if( !win ){
               player = i % players;
               for( j=0; j<card.length() && !win; j++ ){
                  pos[player]=board.indexOf(card.charAt(j),
                                            pos[player]+1 );
                  if( pos[player]<0 || pos[player]==(size-1) ){
                     win = true;
                     System.out.println( "Player " + (player+1)
                                         + " won after "+(i+1)+" cards.");
                  }
               }
            }
         }
         if( !win )
            System.out.println( "No player won after " + cards
                                + " cards.");
      }
   }
}
```

Suma Máxima

Tienes una secuencia de $A[1], A[2], \ldots, A[N]$ ($0 \le A[i] \le 10^8, 2 \le n \le 10^5$). Hay dos tipos de operaciones y que se definen como sigue:

Update:

Esta se indicará en la entrada por una 'U' seguida por un espacio y luego dos números enteros i y x.

U i x, con $1 \le i \le N$, y $0 \le x \le 10^8$.

Esta operación establece el valor de A[i] a x.

Query:

Esta se indicará en la entrada por una 'Q' seguida de un espacio único y dos enteros i y j.

Q x y , con $1 \le x < y \le N$.

Debes encontrar i y j tal que $x \le i$, $j \le y$ e $i \ne j$, de manera que la suma de A[i] + A[j] se maximiza. Imprimir la suma de A[i] + A[j].

Entrada

La primera línea de entrada consta de un entero **N** que representa la longitud de la secuencia. La siguiente línea consiste en **N** enteros separados por un espacio A[i]. La siguiente línea contiene un entero Q, Q ≤ 10^5, que representan el número de operaciones. Siguen Q líneas que contienen las operaciones.

La entrada termina con N = 0.

Salida

Imprimir la suma máxima que se mencionó anteriormente, en una línea distinta, para cada consulta.

Ejemplo de entrada

```
5
1 2 3 4 5
6
Q 2 4
Q 2 5
U 1 6
Q 1 5
U 1 7
Q 1 5
0
```

Ejemplo de salida

```
7
9
11
12
```

Análisis y Solución

Por: Gabriel Rivera

Analizando el Problema

Simplemente tenemos un vector, el cual se actualiza con la sentencia U y se consulta la suma máxima entre dos elementos mediante Q dando rangos.

Código Fuente en JAVA:

```
/* Problema : Suma Maxima
 * Lenguaje : Java JDK 1.6
 * Por      : Gabriel Rivera Safadi
 ********************************/

import java.util.Scanner;

class SumaMaxima {

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      while (true) {
         int n = sc.nextInt();
         if (n == 0) {
            break;
         }
         int a[] = new int[n];
         for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
         }
         int q = sc.nextInt();
         for (int i = 0; i < q; i++) {
            String str = sc.next();
            if (str.equals("Q")) {
               int ini = sc.nextInt() - 1;
               int fin = sc.nextInt() - 1;
               int max = -1;
               for (int j = ini; j < fin; j++) {
                  for (int k = j + 1; k <= fin; k++) {
                     max = a[j] + a[k] > max ? a[j] + a[k] : max;
                  }
               }
               System.out.println(max);
            } else {
               a[sc.nextInt() - 1] = sc.nextInt();
            }
         }
      }
   }

}
```

PROBLEMAS CON ESTRUCTURAS DE DATOS.

Amalgamación de Palabras

En millones de periódicos en Estados Unidos, hay un juego de palabras llamado, confusión. El objetivo de este juego es resolver el acertijo, pero para encontrar las letras que aparecen en la respuesta es necesario descifrar cuatro palabras. Tu tarea es escribir un programa que descifre las palabras.

La entrada contiene cuatro partes:

1. Un diccionario, que consiste en al menos una y a lo sumo 100 palabras, uno por línea.
2. Una línea conteniendo XXXXXX, que señala en final del diccionario.
3. Una o más palabras revueltas que debe descifrar, cada una en una línea; y
4. Otra línea conteniendo XXXXXX, que señala el final de la entrada.

Todas las palabras, las de diccionario y las revueltas, están solo en minúsculas del alfabeto en ingles y son como mínimo una y a lo sumo seis caracteres de largo. (Note que la centinela XXXXXX contiene X mayúsculas) No es necesario que el diccionario este ordenado, pero cada palabra en el diccionario es único.

Por cada palabra revuelta en la entrada, despliegue una lista alfabética de todas las palabras formadas por las letras reacomodadas de las palabras revueltas. Cada palabra de esta lista debe aparecer una en cada línea. Si la lista esta vacía (o sea, que no se pueden formar palabras en el diccionario), despliegue la línea "NOT A VALID WORD". En otro caso, despliegue una línea con seis asteriscos señalando el final de la lista.

Ejemplo de Entrada

```
tarp
given
score
refund
only
trap
work
earn
course
pepper
part
XXXXXX
resco
nfudre
aptr
sett
oresuc
XXXXXX
```

Ejemplo de Salida

```
score
******
refund
******
part
tarp
trap
******
NOT A VALID WORD
******
course
******
```

Análisis y Solución

Por: Alberto Suxo

Este es un clásico problema de estructuras de datos, particularmente mapas o diccionarios.

Primero, al leer cada palabra del diccionario, haremos una copia de dicha palabra a la que llamaremos clave, esta clave debe tener a sus caracteres ordenados.

Por ejemplo

Clave	Palabra
aprt	tarp
eginv	given
ceors	score
aprt	trap
aprt	part

Almacenamos todos estos datos en nuestro mapa o diccionario. Tendremos algo así:

Clave	Palabra
aprt	tarp
aprt	trap
aprt	part
ceors	score
eginv	given

Ahora leemos las palabras revueltas, a las que les haremos lo mismo, copiamos la cadena, la ordenamos y la llamamos clave. Ahora buscamos todas coincidencias de nuestra clave en el mapa o diccionario y las desplegamos ordenadas alfabéticamente. ¿Simple no?, pues si, es simple si se conoce las estructuras de datos básicas. Pero si no se las conoce, la codificación del problema se hace un poco larga. Para poder comprender mejor la solución a este problema, la presentaré en CPP (usando STL) y en C.

Código Fuente en C++:

```
/* Problema : Amalgamacion de Palabras
 * Por      : Alberto Suxo
 ****************************/
#include<iostream>
#include<map>
#include<vector>
using namespace std;

int main() {
   multimap <string, string> M;
   string str, str2;
   int cnt, i;
   //freopen("word.in", "r", stdin);
   while( 1 ) {
      cin >> str;
      if( str=="XXXXXX" )
         break;
```

```
      str2 = str;
      sort( str2.begin(), str2.end() );
      M.insert( pair<string,string>( str2, str ) );
   }
   while( 1 ) {
      cin >> str;
      if( str=="XXXXXX" )
         break;
      sort( str.begin(), str.end() );
      cnt = M.count( str );
      if( cnt>0 ) {
         vector<string> V;
         multimap<string,string>::const_iterator iter = M.find( str );
         for( i=0; i<cnt; i++ ) {
            V.push_back( iter->second );
            iter++;
         }
         sort( V.begin(), V.end() );
         for( i=0; i<cnt; i++ )
            cout<< V[i] << endl;
      }
      else
         cout << "NOT A VALID WORD" << endl;
      cout << "******" << endl;
   }
   return 0;
}
```

Código Fuente en C:

La versión en C puede parecer complicada, pero es básicamente una copia del anterior código, donde almacenaremos las claves y palabras en nuestro propio diccionario, una vez almacenados todas las palabras, ordenamos el diccionario primero por clave y luego por palabra. Luego leemos las palabras revueltas, hallamos su clave, hacemos una búsqueda binaria en nuestro diccionario y de existir las desplegamos tal cual van apareciendo (porqué ya que están ordenadas apropiadamente).

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct{
   char key[7];
   char word[7];
} dictionary;
dictionary Dic[101];

int fword( const void *a, const void *b ) {
   char *A = (char *)a;
   char *B = (char *)b;
   return( *A - *B );
}
int fdic( const void *a, const void *b ) {
   dictionary *A = (dictionary *)a;
   dictionary *B = (dictionary *)b;
   int cmp = strcmp( A->key, B->key );
```

```
   if(cmp==0)
      return( strcmp( A->word, B->word ) );
   else
      return cmp;
}

int search( char *key, int n ) {
   int i=0, f=n-1, c, cmp;
   while( i<=f ) {
      c = (i+f)/2;
      cmp=strcmp( key, Dic[c].key );
      if(cmp==0) {
         while( c>0 && strcmp(key, Dic[c-1].key)==0 )
            c--;
         return c;
      } else {
         if( cmp<0 )
            f=c-1;
         else
            i=c+1;
      }
   }
   return -1;
}

int main() {
   int n = 0;
   char key[7];
   int i, ps;
   //freopen("word.in", "r", stdin);
   while( 1 ) {
      scanf( "%s", Dic[n].word );
      if( Dic[n].word[0]=='X' )
         break;
      strcpy( Dic[n].key,Dic[n].word);
      qsort( Dic[n].key, strlen(Dic[n].key), sizeof(char), fword );
      n++;
   }
   qsort( Dic, n, sizeof(Dic[0]), fdic );
   while( 1 ) {
      scanf( "%s", key );
      if( key[0]=='X' )
         break;
      qsort( key, strlen(key), sizeof(char), fword );
      ps = search( key, n );
      if( ps>=0 )
         for(  ; ps<n&& strcmp( Dic[ps].key, key )==0; ps++ )
            puts( Dic[ps].word );
      else
         puts( "NOT A VALID WORD" );
      puts( "******" );
   }
   return 0;
}
```

Concatenación de Lenguajes

Un lenguaje es un conjunto de cadenas. Y la concatenación de dos lenguajes es el conjunto de todas las cadenas que son formadas concatenado las cadenas del segundo lenguaje al final de las cadenas del primer lenguaje.

Por ejemplo, si tenemos dos lenguajes A y B de modo que:

A = {cat, dog, mouse}
B = {rat, bat}

La concatenación de A y B sería:

C = {catrat, catbat, dograt, dogbat, mouserat, mousebat}

Dados dos lenguajes, tu tarea es solo contar el número de cadenas en la concatenación de los dos lenguajes.

Entrada

Pueden haber múltiples casos de prueba. La primera línea del archivo de entrada contiene el número de casos de prueba, T ($1 \leq T \leq 25$). Luego siguen T casos de prueba. La primera línea de cada caso de prueba contiene dos enteros, M y N ($M,N < 1500$), el número de cadenas en cada lenguaje. Luego las siguientes M líneas contienen las cadenas del primer lenguaje. Las N líneas siguientes te dan las cadenas del Segundo lenguaje. Puedes asumir que las cadenas están formadas solamente por letras minúsculas (‘a’ a ‘z’), y que son de menos de 10 caracteres de largo y que cada cadena es presentada en una línea sin espacios por delante o por detrás. Las cadenas en los lenguajes de entrada no deben ser ordenadas y no habrán cadenas duplicadas.

Salida

Para cada caso de prueba necesitas imprimir una línea de salida. La salida para cada caso de prueba comienza con el número consecutivo del caso de prueba, seguido por el número de cadenas en la concatenación del segundo lenguaje después del primer lenguaje.

Ejemplo de Entrada

```
2
3 2
cat
dog
mouse
rat
bat
1 1
abc
cab
```

Ejemplo de Salida

```
Case 1: 6
Case 2: 1
```

Análisis y Solución

Por: Gabriel Rivera

Analizando el Problema

Se necesita realizar la concatenación de todas las palabras del primer lenguaje con todas del segundo lenguaje, pero en el caso de tener repetidas, no se deberían tomar en cuenta. Por ejemplo:

```
A: “a”, “aaa”
B: “aa”, “aaaa”
```

La concatenación sería:

```
C: “aaa”, “aaaaa”, “aaaaaaa”
```

Porque se produciría 2 veces la cadena “aaaaa”, por esto:

```
“a” + “aaaa” = “aaa” + “aa” = “aaaaa”
```

Entonces, el tamaño sería todas las combinaciones, excepto las repetidas, para lo cual se puede ir almacenando en una estructura de datos de tipo Conjunto (Set) y al final, solo se cuenta el tamaño del conjunto resultante.

Código Fuente en JAVA:

```java
/* Concatenacion de Lenguajes
 * @author Gabriel Rivera Safadi
 */
import java.util.HashSet;
import java.util.Iterator;
import java.util.Scanner;
import java.util.Set;
import java.util.StringTokenizer;

class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int t = Integer.parseInt(sc.nextLine());

        for (int tt = 1; tt <= t; tt++) {
            StringTokenizer st = new StringTokenizer(sc.nextLine());
            int m = Integer.parseInt(st.nextToken());
            int n = Integer.parseInt(st.nextToken());
            Set<String> s1 = new HashSet<String>();
            for (int i = 0; i < m; i++) {
                s1.add(sc.nextLine());
            }
            Set<String> s2 = new HashSet<String>();
            for (int i = 0; i < n; i++) {
                s2.add(sc.nextLine());
            }
            Set<String> s3 = new HashSet<String>();
            Iterator<String> i1 = s1.iterator();
```

```
            while (i1.hasNext()) {
                Iterator<String> i2 = s2.iterator();
                String str1 = i1.next();
                while (i2.hasNext()) {
                    s3.add(str1 + i2.next());
                }
            }
            System.out.println("Case " + tt + ": " + s3.size());
        }
    }

}
```

Extraño sueldo del Jefe

Como es bien conocido, últimamente se esta realizando ajustes en las escalas de salarios en muchas instituciones debido a regulaciones gubernamentales.

Por este motivo la empresa ACME ha decidido que cada trabajador debe ganar Bs. 1 y cada jefe debe ganar Bs. 1 más que su subalterno que más gana.

La compañía tiene una jerarquía muy estricta en la cual cada empleado tiene exactamente un jefe directo, con la excepción del jefe máximo, que reporta directamente a los accionistas. Los trabajadores son aquellos empleados que no son jefes de otros empleados. Los que no son trabajadores son jefes.

Tome en cuenta que un jefe puede tener trabajadores y jefes como subalternos al mismo tiempo.

Dada la jerarquía de la empresa se desea conocer cual será el salario del jefe máximo.

Entrada

Existen varios casos de prueba. Los datos de prueba están dados en dos líneas exactamente. La primera línea contiene un entero N ($1 \leq N \leq 10^5$), que indica el número de empleados de la compañía. Cada empleado se identifica por un número entero entre 1 y N. El jefe máximo se identifica por el número 0. La segunda línea contiene una lista de enteros separados por un solo espacio. El entero B_i en la posición **i** en esta lista indica la identificación (comenzando en **1**) del jefe directo del empleado **i** ($0 \leq B_i \leq i-1$).

La última línea de los casos de prueba es una línea que contiene un cero.

Salida

Para cada caso de prueba escriba una línea contiene un solo entero de con el salario que ganaría el jefe máximo.

Ejemplo de Entrada

```
14
0 0 1 1 2 2 2 5 7 5 7 5 7 13
0
```

Ejemplo de Salida

```
5
```

Análisis y Solución

Por: Jorge Terán

En este problema nos indican que la estructura de la empresa es un árbol n-ario. Como cada jefe esta en un nivel superior y gana 1 más que su empleado que más gana, es equivalente a hallar la profundidad de un árbol.

Hay dos estrategias una construir su propio árbol y otra utilizar las clases disponibles en el lenguaje.

Primero construyamos nuestro propio árbol.

```
ArrayList[] arbol = new ArrayList[1];
int[] totalHijos = new int[1];
Scanner br = new Scanner(System.in);
int casoP=0;
int raiz = 0;
// carga de datos
String s;
s = br.readLine();
while ((!s.equals("0"))) {
      StringTokenizer linea = new StringTokenizer(s);
      // cantidad de casos
      int cantNodos = Integer.parseInt(linea.nextToken());
      maximo = Integer.MIN_VALUE;
      int contador=1;
      arbol = new ArrayList[cantNodos + 1];
      for (int i = 0; i < arbol.length; i++) {
            arbol[i] = new ArrayList<Integer>();
      }
      totalHijos = new int[cantNodos + 1];
      raiz = 0;
      s = br.readLine();
      StringTokenizer datos = new StringTokenizer(s);

      for (int i = 1; i <= cantNodos; i++) {
            // para cada caso
            int papa = Integer.parseInt(datos.nextToken());
            int hijo = i;
            totalHijos[papa]++;  // asi se cantidad de hijos
            int pos = totalHijos[papa];
            arbol[papa].add(hijo);
      }
```

Con esto ya tenemos un árbol n-ario, ahora hay que recorrer el mismo buscando la profundidad máxima.

```
public static void revisar(ArrayList[] unArbol, int[] topes, int
unNodo, int contador) {
      int cantHijos = topes[unNodo];
      if (cantHijos == 0) {
            return;
      }
      int hijo=0;
      for (int i=0;i < cantHijos; i++){
```

```
            hijo=Integer.parseInt(unArbol[unNodo].get(i).toString());
                  revisar(unArbol,topes, hijo, ++contador);
            maximo=Math.max(maximo,contador);
            contador--;
      }

      return;
```

La respuesta es el valor máximo.

Código Fuente en JAVA:

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

public class Salario {

   public static void revisar(ArrayList[] unArbol, int[] topes, int
unNodo,int contador) {
      //System.out.println("nodo a procesar "+unNodo+" Nro.Hijos "+
      //       topes[unNodo]);
      //System.out.println("contador = "+ contador);
      int cantHijos = topes[unNodo];
      if (cantHijos == 0) {
         return;
      }
      int hijo=0;
      //for (int i=0;i < cantHijos; i++){
      //    hijo=Integer.parseInt(unArbol[unNodo].get(i).toString());
      //    System.out.print("hijo = "+hijo +" ");
      //}
      //System.out.println();
      for (int i=0;i < cantHijos; i++){
         hijo=Integer.parseInt(unArbol[unNodo].get(i).toString());
         //System.out.println("hijo = "+hijo);
         //if (topes[hijo]>0)
            revisar(unArbol,topes, hijo, ++contador);
         maximo=Math.max(maximo,contador);
         //System.out.println("maximo = "+maximo);
         contador--;
      }

      return;
   }

   static int maximo;

   public static void main(String args[]) throws Exception {
      ArrayList[] arbol = new ArrayList[1];
      int[] totalHijos = new int[1];
      BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      int casoP=0;
      int raiz = 0;
      // carga de datos
```

```
      String s;
      s = br.readLine();
      while ((!s.equals("0"))) {
//System.out.println("caso P= "+(++casoP));
         StringTokenizer linea = new StringTokenizer(s);
         // cantidad de casos
         int cantNodos = Integer.parseInt(linea.nextToken());
//System.out.println("CantNodos = "+cantNodos);
         maximo = Integer.MIN_VALUE;
         int contador=1;
         arbol = new ArrayList[cantNodos + 1];
         for (int i = 0; i < arbol.length; i++) {
            arbol[i] = new ArrayList<Integer>();
         }
         totalHijos = new int[cantNodos + 1];
         raiz = 0;
         s = br.readLine();
         StringTokenizer datos = new StringTokenizer(s);

         for (int i = 1; i <= cantNodos; i++) {
            // para cada caso
            int papa = Integer.parseInt(datos.nextToken());
            int hijo = i;
            totalHijos[papa]++;  // asi se cantidad de hijos
            int pos = totalHijos[papa];
            arbol[papa].add(hijo);

         }
         // MOSTRAR EL ARBOL QUE LEIMOS

         //System.out.println("ARBOL");
         //for (int i = 0; i <= cantNodos; i++) {
         //   System.out.print(" Padre "+ i+" Tiene "+ totalHijos[i]+
         //             " Hijos ");
         //for (int j = 0; j < totalHijos[i]; j++) {
         //   System.out.print(arbol[i].get(j)+" ");
         //}
         //System.out.println(" ");
         //}

         // FIN MOSTRAR
         int aux = 0;
         revisar(arbol, totalHijos, raiz, contador);
         System.out.println(maximo);
         s = br.readLine();
      }
   }
}
```

Análisis y Solución2

Por: Alberto Suxo

Este es un problema que claramente se muestra como un problema de árboles, también puede resolverse utilizando programación dinámica, sin embargo, si analizamos un poquito más sobre la estructura del árbol que se genera, y, a sabiendas de que lo que nos importa al final es el número de niveles que tendrá el árbol, podemos, según se vallan leyendo los nodos, al nodo leído le sumamos uno al valor de su padre. Y luego buscamos al mayor valor.

En pero, si lo pensamos un poquito más, no es necesario reproducir la estructura del árbol, todo lo podemos calcular directamente sobre un simple vector.

Como los empleados son identificados desde el 1 en adelante, usaremos la posición 0 (cero) del vector como nuestro valor inicial (la posición del jefe máximo) y desde esa posición iremos calculando el resto.

```
14
0 0 1 1 2 2 2 5 7 5 7 5 7 13
```

Son 14 empleados, al vector `V[0] = 1; //nivel jefe máximo.` Ahora, el empleado 1 es empleado de jefe máximo (posición 0), `V[empleado] = V[jefe] + 1` o sea que `V[1] = V[0] + 1`, y para todos los demás será lo mismo.

Código Fuente en C:

```c
#include<stdio.h>

#define MAX 100000L

int main() {
	long N, empleado, jefe, maximo;
	long V[MAX+1];

	while ( 1 ) {
		scanf( "%ld", &N );
		if( !N )
			break;
		V[0] = 1; //jefe maximo.
		maximo = V[0];
		for( empleado=1; empleado<=N; empleado++ ) {
			scanf("%ld", &jefe);
			V[empleado] = V[jefe] + 1;
			if( V[empleado] > maximo )
				maximo = V[empleado];
		}
		printf("%ld\n", maximo );
	}
	return 0;
}
```

Hojas Caídas

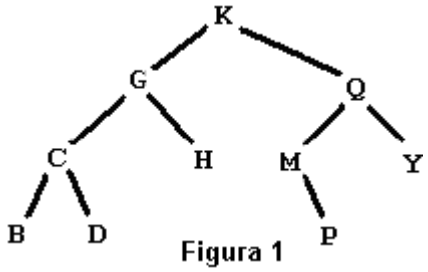


Figura 1

La Figura 1 muestra la representación gráfica de un árbol binario de letras. Lo familiarizados con los árboles binarios pueden saltarse la definición de árbol binario de letras, hojas de un árbol binario, y búsqueda en un árbol binario de letras, e ir directo al problema.

Definición.

Un *árbol binario de letras* puede ser una de dos cosas:

1. Puede estar vacía.
2. Puede tener un nodo raíz. Un nodo tiene una letra como dato y hace referencia a subárboles izquierdo y derecho. Los subárboles izquierdo y derecho son también árboles binarios de letras.

En la representación gráfica de un árbol binario de letras:

1. Un árbol vacío es omitido completamente.
2. Cada nodo esta indicado por
    - Su dato letra,
    - Un segmento de línea abajo a la izquierda hacia su subárbol izquierdo, si el subárbol izquierdo no está vacío,
    - Un segmento de línea abajo a la derecha hacia su subárbol derecho, si el subárbol derecho no esta vacío.

Una *hoja* en un árbol binario es un nodo donde ambos subárboles están vacíos. En el ejemplo en la Figura 1, tiene cinco nodos con datos B, D, H, P, y Y.

El recorrido preorder de un árbol de letras satisface las propiedades:

1. Si el árbol esta vacío, entonces el recorrido preorder está vacío.
2. Si el árbol no esta vacío, entonces el recorrido preorder consiste en lo siguiente, en orden:
    - El dato del nodo raíz,
    - El recorrido preorder del subárbol izquierdo del nodo raíz,
    - El recorrido preorder del subárbol derecho del nodo raíz.

El recorrido preorder del árbol de la Figura 1 es KGCBDHQMPY.

Un árbol como el de la Figura 1 es también un árbol binario de búsqueda de letras. Un árbol binario de búsqueda de letras es un árbol de letras en el cual cada nodo satisface:

1. Los datos raíz vienen después en el alfabeto que todos los datos en los nodos en el subárbol izquierdo.

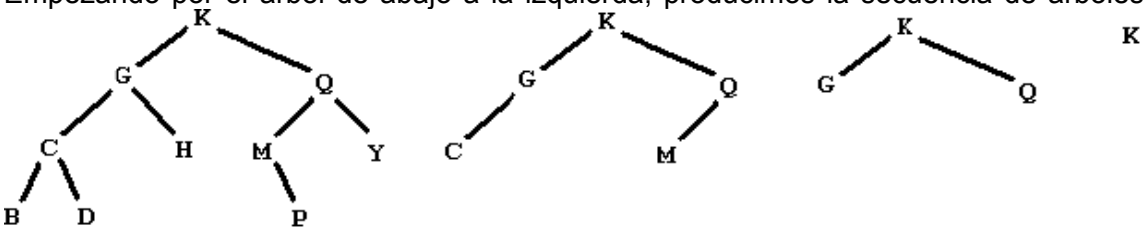
2. Los datos raíz vienen antes en el alfabeto que todos los datos en los nodos en el subárbol derecho.

El problema:

Considere la siguiente secuencia de operaciones en un árbol binario de búsqueda de letras:

- Borrar las hojas y listar los datos removidos
- Repetir este proceso hasta que el árbol este vacío.

Empezando por el árbol de abajo a la izquierda, producimos la secuencia de árboles



mostrados, y hasta que el árbol este vacío removiendo las hojas de datos

BDHPY
CM
GQ
K

Tu problema es empezar con tales secuencias de líneas de hojas de un árbol binario de búsqueda de letras y desplegar el recorrido preorder del árbol.

La entrada contiene uno o más sets de datos. Cada set de datos es una secuencia de uno o más líneas con letras mayúsculas. Las líneas contienen las hojas removidas del árbol binario de búsqueda de la forma descrita anteriormente. Las letras en una línea están listados en orden alfabético. Los sets de datos están separados por una línea que contiene un asterisco (‘*’). El último set de datos está seguido por un signo de dólar (‘$’). No hay espacios en blanco ni líneas vacías en la entrada.

Por cada set de datos de entrada, hay un único árbol binario de búsqueda que puede ser producido con la secuencia de hojas. La salida es una línea que contiene solo el recorrido preorder del árbol, sin blancos.

Ejemplo de entrada

```
BDHPY
CM
GQ
K
*
AC
B
$
```

Ejemplo de salida

```
KGCBDHQMPY
BAC
```

Análisis y Solución

Por: Alberto Suxo

Este es un clásico problema de estructuras de datos (árboles), la mayor parte del problema se hace referencia a los árboles binarios ordenados de caracteres.

En resumen, dado un árbol de caracteres ordenado, se van retirando todas la hojas del mismo, este proceso se repite hasta terminar vaciando el árbol. La información que se nos otorgará son los conjuntos de hojas que se van retirando en el orden en que se van retirando.

Ejemplo

BDHPY
CM
GQ
K

La salida de nuestro programa debe ser una cadena que exprese el recorrido preorder del árbol

Para esto es lógico pensar que debemos reconstruir el árbol, esta tarea es relativamente simple.

¿Cómo?

Pues el problema dice que es un árbol ordenado, así que lo que tenemos que hacer es leer todas las cadenas de cada set de datos, empezando desde la última línea hacia la primera insertaremos las letras en nuestro árbol (inserción ordenada), y luego lo imprimimos en preorder.

Demos un vistazo al código que presento, debo aclarar que no estoy utilizando un árbol de la forma correcta, en realidad estoy utilizando un vector que simula ser un árbol.

```
struct tree{
   char l, r;
} T['Z'+1];
```

Este vector T tiene 91 posiciones (T[0], T[1],.., T[64], T[65], ... ,T[90]), pero a mi solo me interesa las posiciones desde 65 al 90 (T[65],... ,T[90]) = (T[‘A’],... ,T[‘Z’]), como se puede ver, para mi, el dato (letra) es la posición en el vector, como cada elemento de este vector es una estructura que contiene dos caracteres l y r que representan left (izquierdo) y right (derecho) respectivamente, y estos caracteres apuntan a las posiciones de su subárbol izquierdo y derecho respectivamente, y en caso de no existir un subárbol pues tendrán el valor 0 (que sería nuestro NULL).

```
void insert( char rt, char ch );
```

Esta función recursiva se encarga de insertar el carácter ch en orden alfabético ascendente, realizando un recorrido empezando por la raíz rt, el algoritmo es bastante específico, así que no es necesario explicarlo con mayor detenimiento.

Ahora está la función find(), en esta función leemos de forma recursiva cada uno de los sets de datos, esta función termina su lectura recursiva cuando encuentra un ‘*’ o ‘$’ (en base a estos caracteres identifica si todavía hay más sets de datos de entrada),

luego identifica cuál es la raíz del árbol, y posteriormente va insertando todos los caracteres de las líneas leídas, (como es una función recursiva que primero lee las líneas de entrada, pues queda claro que empieza a procesar dichas líneas empezando por la última hacia la primera). Adjunto comentarios por cada operación que realiza, estos comentarios ayudarán a la comprensión de dicha función.

```
void find(){
   char line[27];
   int i;
   gets( line );
   if( line[0]=='*' || line[0]=='$' ){ /* Si termino el set de datos */
      root = 0;                        /* aún no existe raíz         */
      if( line[0]=='*' ) /* Si el set de datos terminó con * signi- */
         hasNext = 1;    /* fica que todavía hay más sets de datos  */
   }else{
      find();
      if( root ){                  /* Si hay  raíz, insertar todos    */
         for( i=0; line[i]; i++ ){/*los datos de la línea en el árbol*/
            insert( root, line[i] ); /* con raíz root                */
         }
      }else{                 /* Si no hay raiz, pues la raiz será */
         root = line[0];     /* el carácter de la última línea.   */
      }
   }
}
```

Por último está la función print(), que se encarga de la impresión en preorder.

```
void print( char rt ){
   if( rt ){
      putchar( rt );
      print( T[rt].l );
      print( T[rt].r );
   }
}
```

Que mientras exista un subárbol imprimirá su raíz, su subárbol izquierdo y su subárbol derecho.

Código Fuente en C:

```
/* Problema : Falling Leaves
 * Lenguaje : ANSI C (version: 4.0)
 * Por      : Alberto Suxo
 ************************/
#include<stdio.h>
#include<memory.h>

char ch;
char root;
int hasNext;

struct tree{
   char l, r;
} T['Z'+1];
```

```
void insert( char rt, char ch ){
   if( ch<rt ){
      if( T[rt].l )
         insert( T[rt].l, ch );
      else
         T[rt].l = ch;
   }else{
      if( T[rt].r )
         insert( T[rt].r, ch );
      else
         T[rt].r = ch;
   }
}

void find(){
   char line[27];
   int i;
   gets( line );
   if( line[0]=='*' || line[0]=='$' ){
      root = 0;
      if( line[0]=='*' )
         hasNext = 1;
   }else{
      find();
      if( root ){
         for( i=0; line[i]; i++ ){
            insert( root, line[i] );
         }
      }else{
         root = line[0];
      }
   }
}

void print( char rt ){
   if( rt ){
      putchar( rt );
      print( T[rt].l );
      print( T[rt].r );
   }
}

int main(){
   do{
      memset( T, 0, sizeof( T ) );
      hasNext = 0;
      find();
      print( root );
      printf( "\n" );
   }while( hasNext );
   return 0;
}
```

PROBLEMAS CON TEORÍA DE NÚMEROS Y MATEMÁTICA GRAL.

A Desenroscar

La ***Criptografia*** es un método utilizado en la comunicación secreta que transforma mensajes (el texto plano o **plaintext**) en su forma oculta (el texto cifrado o **ciphertext**) de modo que nadie viendo el ciphertext vea el plaintext excepto el receptor deseado. Transformar el plaintext en su ciphertext es **encriptación**; transformar el ciphertext en su plaintext es **desencriptación**. ***Twisting*** es un simple método de encriptación que requiere que tanto el emisor como e receptor tengan la clave secreta *k*, que es un número positivo.

El método twisting usa cuatro arrays: *plaintext* y *ciphertext* que son arrays de caracteres, y *plaincode* y *ciphercode* son arrays de enteros. Todos los arrays son de tamaño n, donde n es el tamaño del mensaje que debe ser encriptado. Los arrays empiezan en cero, por lo que los elementos están numerados desde 0 a *n* - 1. Para este problema, todos los mensajes estarán en minúsculas, el punto, y la barra baja '_' (representa un espacio).

El mensaje a ser encriptado es puesto en el *plaintext*. Dada una clave *k*, en método de encriptación trabajo como sigue: primero se convierte los caracteres del *plaintext* a códigos enteros en *plaincode* de a cuerdo a la siguiente regla: '_' = 0, 'a' = 1, 'b' = 2, ..., 'z' = 26, y '.' = 27. Luego, se convierte a cada código de *plaincode* en un código encriptado en *ciphercode* de acuerdo a la siguiente fórmula: para todo *i* desde 0 a *n* - 1,

ciphercode[*i*] = (*plaincode*[*ki* mod *n*] - *i*) mod 28.

(Aquí *x* mod *y* es el resto positivo de cuando *x* es dividido por *y*. Por ejemplo, 3 mod 7 = 3, 22 mod 8 = 6, and -1 mod 28 = 27. Puede usar el operador '%' en C o el operador 'mod' en Pascal para calcularlo, y si es necesario adicionar si el resultado es negativo.) Finalmente, convertir los códigos en *ciphercode* luego a letras en *ciphertext* de acuerdo a la regla anterior. El mensaje final estará en *ciphertext*. Para el mensaje cat usando la llave 5 produce lo siguiente:

Array	0	1	2
plaintext	'c'	'a'	't'
plaincode	3	1	20
ciphercode	3	19	27
ciphertext	'c'	's'	'.'

Tu trabajo es escribir un programa que des-encripte el mensaje, esto es, convertir el ciphertext en el plaintext original dada una llave k. Por ejemplo, dada una llave 5 y el ciphertext 'cs.', tu programa debe desplegar el plaintext 'cat'.

La entrada contiene uno o más casos de prueba, seguida de una línea que contenta a un número 0 que señala en fin de la entrada. Cada caso de prueba es una línea que consiste en la llave k, un espacio, y el mensaje que contiene como mínimo uno y máximo 70 caracteres. La llave k es un entero positivo no mayor a 300. Para cada caso de prueba, despliegue el mensaje desencriptado uno por línea.

Nota: Debe asumir que el mensaje desencriptado siempre produce un único resultado. (Para ello necesitarás de conocimiento básico de teoría de números y algebra, este será el caso que provee el máximo común divisor de la llave k y el tamaño n es 1, que es para todos los casos de prueba).

Ejemplo de Entrada	Ejemplo de Salida
5 cs. 101 thqqxw.lui.qswer 3 b_ylxmhzjsys.virpbkr 0	cat this_is_a_secret beware._dogs_barking

Análisis y Solución

Por: Alberto Suxo

Una vez entendido el proceso de encriptación sabemos que convertir de plaintext a plaincode y viceversa en muy simple.

Empecemos a resolver el problema, primero declararemos variables globales

```
int k;
char line[80]; /* cadena de entrada y luego de salida.*/
int ciphercode[80];
int plaincode[80];
```

Ahora convertiremos el ciphertext (line) en su respectivo ciphercode, con esta función, también aprovecharemos para hallar el tamaño de la cadena.

```
int toCipherCode() {
  int i;
  for( i=0; line[i]; i++ ) {
    if( line[i]=='_' )
      ciphercode[i] = 0;
    else
      if( line[i]=='.' )
        ciphercode[i] = 27;
      else
        ciphercode[i] = line[i]-'a'+1;
  }
  return i;
}
```

Ya tenemos el ciphercode que es generado con la siguiente fórmula.

$ciphercode[i] = (plaincode[ki \bmod n] - i) \bmod 28.$

La parte más complicada de este problema es invertir la fórmula anterior de tal forma que podamos hallar el plaincode requerido. Lo haremos despejando plaincode de la siguiente forma (aplicando un poco de aritmética modular):

$plaincode[ki \bmod n] = (ciphercode[i] + i) \bmod 28.$

Esto, para todo elemento del vector ciphercode de esta forma:

```
void untwist( int n ) {
  int i;
  for( i=0; i<n; i++ ) {
    plaincode[ (k*i)%n ] = ( ciphercode[i]+i )%28;
  }
}
```

Ahora convertimos la plaincode a plaintext (line) para su posterior impresión.

```
void toLine( int n ) {
  int i;
  for( i=0; i<n; i++ ) {
    if( plaincode[i]==0 )
      line[i] = '_';
    else
      if( plaincode[i]==27 )
        line[i] = '.';
      else
        line[i] = plaincode[i]-1+'a';
  }
}
```

Y listo, ya podemos imprimir line, que contiene la cadena desencriptada.

Código Fuente en C:

```
/* Problema : A Desenroscar -Do the Untwist
 * Lenguaje : ANSI C
 * Por      : Alberto Suxo
 ****************************/

#include<stdio.h>

int k;
char line[80];
int ciphercode[80];
int plaincode[80];


int toCipherCode() {
  int i;
  for( i=0; line[i]; i++ ) {
    if( line[i]=='_' )
      ciphercode[i] = 0;
    else
      if( line[i]=='.' )
        ciphercode[i] = 27;
      else
        ciphercode[i] = line[i]-'a'+1;
  }
  return i;
}


void untwist( int n ) {
  int i;
  for( i=0; i<n; i++ ) {
    plaincode[(k*i)%n] = (ciphercode[i]+i)%28;
  }
}
```

```
void toLine( int n ) {
  int i;
  for( i=0; i<n; i++ ) {
    if( plaincode[i]==0 )
      line[i] = '_';
    else
      if( plaincode[i]==27 )
        line[i] = '.';
      else
        line[i] = plaincode[i]-1+'a';
  }
}

int main() {
  int n;

  while( 1 ) {
    scanf("%d", &k);
    if( !k )
      break;
    scanf( "%s", line );
    n = toCipherCode();
    untwist( n );
    toLine( n );
    printf( "%s\n", line );
  }
  return 0;
}
```

Auto-Números (Self Numbers)

En 1949 el matemático Indio DR. Kaprekar descubrió una clase números llamados self-numbers (auto-números). Para algún entero positivo n, define a d(n) como n mas la suma de los dígitos de n. (La d significa digit-adition, termino elegido por Kaprekar). Poro ejemplo, d(75) = 75+7+5=87. Dado un entero positivo n que es un punto inicial, usted puede construir la secuencia creciente infinita de enteros n, d(n), d(d(n)), d(d(d(n))), ... Por ejemplo, si empieza con 33, el siguiente es 33 + 3 + 3 = 39, el siguiente es 39 + 3 + 9 = 51, el siguiente es 51 + 5 + 1 = 57, y así, puede generar la siguiente secuencia.

33, 39, 51, 57, 69, 84, 96, 111, 114, 120, 123, 129, 141, ...

El número n es llamado generador de d(n). En la anterior secuencia 33 es un generador de 39, 39 es un generador de 51, 51 es un generador de 57 y así sucesivamente. Algunos números tienen más de un generador, por ejemplo 101 tiene dos generadores, 91 y 100. Un número sin generadores es un self-number. Hay trece self-numbers menores a 100: 1, 3, 5, 7, 9, 20, 31, 42, 53, 64, 75, 86, y 97.

Escriba un programa que muestre todos los self-number positivos menores a 10000 en orden creciente, uno por línea.

Salida

```
1
3
5
7
9
20
31
42
53
64
 |
 |       <-- un montón de números
 |
9903
9914
9925
9927
9938
9949
9960
9971
9982
9993
```

Análisis y Solución

Por Alberto Suxo

Repasemos, en el problema, lo que solicita es encontrar todos los número x para los cuales no exista un n talque d(n) = x.

La solución a este problema se simplifica cuando hacemos una pequeña prueba.

d(0)=0, d(1)=2, d(2)=4, d(3)=6, d(4)=8, d(5)=10, d(6)=12, d(7)=14, d(8)=16, d(9)=18, d(10)=11, d(11)=13, d(12)=15, d(13)=17, d(14)=19 y d(15)=21, bien, hasta este momento (n=15), ya tenemos algunos self-numbers, pero para que sea más claro distinguirlos usaremos un vector, que, inicialmente tendrá a todos sus valores iguales a 1 (que para nosotros significará 1 = es self-number). Conforme vallamos calculando los valores de d(n), iremos poniendo a nuestro vector V[d(n)] = 0, donde 0 significará "no es self-number", si hacemos este proceso para los números del 1 al 15 tendremos el siguiente resultado:

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

Como puede ver, ya tenemos la seguridad que desde el 1 al 15 existen los siguientes self-numbers: 1, 3, 5, 7 y 9.

Note que, no podemos asegurar que hasta este momento que 20 es un self-number, puesto que no tenemos la seguridad de que entre d(16) hasta d(19) alguno de estos genere o no al número 20, por lo que no lo consideramos hasta que lleguemos a procesar dicho rango.

Bien, como necesitamos encontrar los self-numbers del 1 al 10000, es lógico suponer que tendremos que procesar hasta d(9999), como d(9999)=10035, sabemos ahora que nuestro vector debe tener como mínimo 10035 elementos, caso contrario corremos el riesgo de desbordar la memoria y obtener un error en tiempo de ejecución.

Ahora, para facilitarnos un poco la codificación, creemos una función que haga la adición de dígitos (d()).

```
int d(int n) {
   int s = n;
   while( n>0 ) {
      s += n%10;
      n /= 10;
   }
   return s;
}
```

Es una función básica que se utiliza casi siempre en lo que es la introducción a la programación.

Cuando, declaramos un vector en Java, este es automáticamente inicializado con los valores por defecto, (para el caso de un vector de booleans, lo inicializará con valores false), pero en C y C++ ese no es el caso, por lo que tendremos que hacer la inicialización nosotros.

```
   for( i=0; i<MAX; i++)
      V[i] = 1;
```
Luego haremos el siguiente recorrido
```
   for( i=0; i<MAX; i++)
      V[ d(i) ] = 0;
```

Con esto, ya tenemos identificados a todos lo self-numbers del 1 hasta MAX, ahora solo tenemos que desplegar a aquellas posiciones que tengan el valor 1 como dato.

Código Fuente en C:

```
/* Problema : Auto-Numeros (Selt Numbers)
 * Lenguaje : ANSI C
 * Por      : Alberto Suxo
 * Nota     : pre-calcular
 *************************/

#include<stdio.h>

#define MAX 10000

int d( int n ) {
  int s = n;
  while( n>0 ) {
    s += n % 10;
    n /= 10;
  }
  return s;
}

int main() {
  unsigned char V[ MAX+40 ];
  int i;
  for( i=0; i<MAX; i++ )
    V[i] = 1;
  for( i=0; i<MAX; i++ )
    V[ d(i) ] = 0;
  for( i=0; i<MAX; i++ )
    if( V[i] )
      printf( "%d\n", i );
  return 0;
}
```

Código Fuente en JAVA:

Para la solución en Java, recordemos que todo dato de tipo boolean tiene por defecto el valor false, por lo que la condición será invertida en java para no tener que inicializar el vector y así ahorrar un poco de tiempo.

```
/* Problema : Auto-Numeros (Selt Numbers)
 * Lenguaje : JDK 1.5
 * Por      : Alberto Suxo
 * Nota     : pre-calcular
 *************************/
```

```java
class Self {
   static int MAX = 10000;

   static int d( int n ) {
      int s = n;
      while( n>0 ) {
         s += n % 10;
         n /= 10;
      }
      return s;
   }

   public static void main( String args[] ) {
      boolean V[] = new boolean[ MAX+40 ];
      int i;
      for( i=0; i<MAX; i++ )
         V[ d(i) ] = true;
      for( i=0; i<MAX; i++ )
         if( !V[i] )
            System.out.println( i );
   }
}
```

Coloréame menos

Una reducción de colores es un mapeo de un conjunto de colores discontinuos a otros pocos. La solución a este problema requiere que mejore el mapeo en colores de veinticuatro bits RGB estándar. La entrada consiste en un conjunto de dieciséis valores de color RGB, y un conjunto de colores RGB arbitrarios a ser mapeados con su color más cercano del conjunto. Para nuestro propósito, un color RGB es definido como un triple ordenado (R, G, B) donde cada valor es un entero desde 0 a 255. La distancia entre dos colores está definida como la distancia euclidiana entre dos puntos tridimensionales. Que es, dados dos puntos (R_1,G_1,B_1) y (R_2,G_2,B_2), la distancia D está dada por la ecuación:

$$D = \sqrt{(R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2} \ .$$

Entrada

La entrada es una lista de colores RGB, un color por línea, especificando tres enteros desde 0 a 255 delimitados por un espacio. Los primeros dieciséis colores son los colores con los que los demás colores deben ser mapeados. La entrada termina cuando la línea contiene tres valores -1.

Salida

Por cada color para mapear, despliegue el color y el color más cercano del conjunto.

Ejemplo de Entrada

```
0 0 0
255 255 255
0 0 1
1 1 1
128 0 0
0 128 0
128 128 0
0 0 128
126 168 9
35 86 34
133 41 193
128 0 128
0 128 128
128 128 128
255 0 0
0 1 0
0 0 0
255 255 255
253 254 255
77 79 134
81 218 0
-1 –1 -1
```

Ejemplo de Salida

```
(0,0,0) maps to (0,0,0)
(255,255,255) maps to (255,255,255)
(253,254,255) maps to (255,255,255)
(77,79,134) maps to (128,128,128)
(81,218,0) maps to (126,168,9)
```

Análisis y Solución

Por: Alberto Suxo

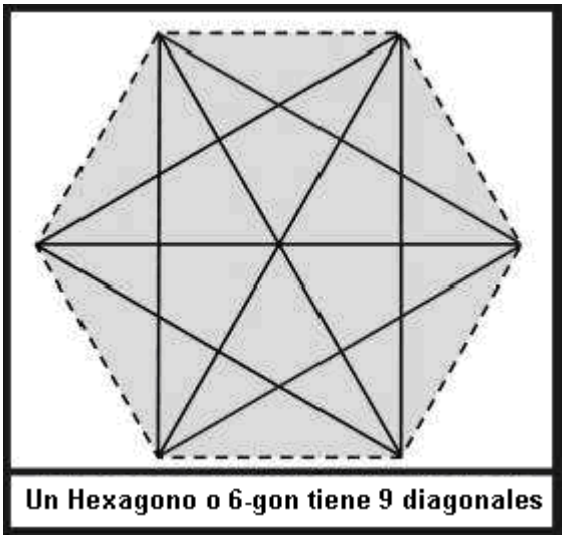
Este problema es muy simple, solo hay que leer los primero 16 colores (Rojo, Verde, Azul), los almacenamos en un vector o una estructura de datos. Luego, para cada uno de los siguientes colores, buscaremos cual de los primero 16 colores está más cerca de él, y lo desplegamos. Como puede ver, en el planteamiento del problema se presenta la fórmula para determinar la distancia entre dos punto por lo cual este problema no representa la más mínima dificultad.

Código Fuente en C:

```
/* Problema : Coloreame Menos
 * Lenguaje : ANSI C
 * Por      : Alberto Suxo.
 ************************/
#include<stdio.h>
#include<math.h>

double C[16][3];

double dist(int cl, int rr, int gg, int bb){
  double r = C[cl][0]-rr;
  double g = C[cl][1]-gg;
  double b = C[cl][2]-bb;
  return sqrt(r*r+g*g+b*b);
}

int main() {
  int I, cl;
  double r, g, b, min, d;
  /*freopen("colors.in", "r", stdin);*/
  /*freopen("colors.sol", "w", stdout);*/
  for(i=0; i<16; i++)
    scanf("%lf %lf %lf", &C[i][0], &C[i][1], &C[i][2]);
  while( 1 ) {
    scanf("%lf %lf %lf", &r, &g, &b);
    if( r<0 && g<0 && b<0 )
      break;
    min=500.0;
    for(i=0; i<16; i++) {
      d = dist(i, r, g, b);
      if( d<min ) {
        min = d;
        cl = i;
      }
    }
    printf("(%.0lf,%.0lf,%.0lf) maps to (%.0lf,%.0lf,%.0lf)\n",
                  r,    g,    b,    C[cl][0], C[cl][1],C[cl][2]);
  }
  return 0;
}
```

Diagonales

El número de diagonales de un n-gon no es menor que N. ¿Cuál es me valor mínimo posible de n?



Un Hexagono o 6-gon tiene 9 diagonales

Entrada

La entrada contiene menos de 1001 líneas de entrada. Cada línea contiene un entero positivo N ($N \leq 10^{15}$) que indica el número mínimo posible de diagonales. La entrada termina con una línea que contiene un cero. Esta línea no será procesada.

Salida

Por cada línea de entrada produzca una línea de salida, que contenga un número de serie, y sea el valor mínimo posible para n (Número de lados).

Ejemplo de Entrada

```
10
100
1000
0
```

Ejemplo de Salida

```
Case 1: 7
Case 2: 16
Case 3: 47
```

Análisis y Solución

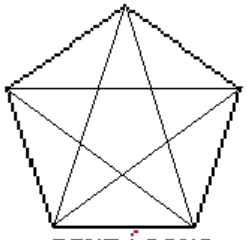
Por: Alberto Suxo

El primer párrafo dice:

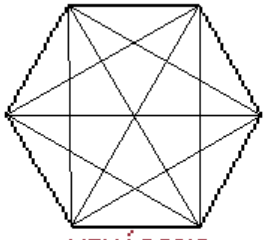
El número de diagonales de un n-gon no es menor que N. ¿Cuál es el mínimo valor posible de n?

Primero aclara que N es mayor o igual que n (N ≥ n) y que en base a algún N, debemos encontrar n.

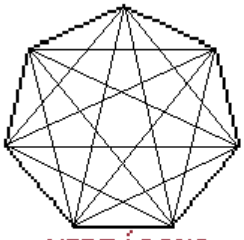
Para resolver este problema hay tener en cuenta lo siguiente:



PENTÁGONO REGULAR



HEXÁGONO REGULAR

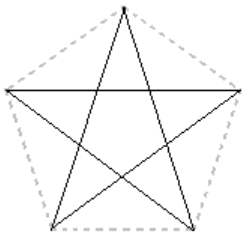


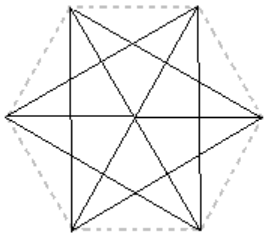
HEPTÁGONO REGULAR

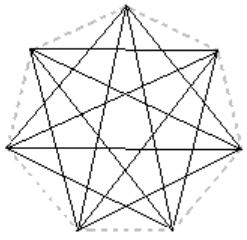
Para el pentágono (5-gon) Si suponemos que cada punta es una persona, que las personas llegan una detrás de otra, y cada persona que llega saluda a los anteriores, entonces, la 1ra no saluda a nadie, la 2da saluda a 1 persona, la 3ra saluda a 2 personas, la 4ta saluda a 3 personas y la 5ta saluda a 4 personas, el total de saludos es $N = 0+1+2+3+4$ que es: $N = \frac{n\cdot(n-1)}{2}$, esta formula es válida para todo n-gon.

Pero como sólo queremos las diagonales (interiores) de cada figura, pues simplemente le restamos n.

$N = \frac{n\cdot(n-1)}{2} - n$ o si prefieren $N = \frac{n^2 - 3n}{2}$







Y ya tenemos la mitad del trabajo realizado.

La diagonales de las figuras son:

n (n-gon)	5	6	7	8
N (Diagonales)	5	9	14	44

Esto lo debemos interpretar como:

Para n = 6 (6-gon) puede contener 6, 7, 8 y 9 diagonales.

Para n = 7 (7-gon) puede contener 10, 11, 12, 13 y 14 diagonales.

Y así sucesivamente. Ahora despejemos n

$$2N = n^2 - 3n$$
$$n^2 - 3n - 2N = 0$$
$$n = \frac{3 \pm \sqrt{9 + 8N}}{2}$$

Pero como $\forall N > 0 : \sqrt{9 + 8N} > 3$ y como n no puede ser negativo, entonces:

$$n = \frac{3 + \sqrt{9 + 8N}}{2}$$

Pero esta fórmula debe ser ligeramente modificada (N por N-1) para satisfacer nuestras necesidades, este cambio será claramente justificado en los resultados de la siguiente tabla:

N (Diagonales)	n (n-gon)	$n = \frac{3 + \sqrt{9 + 8N}}{2}$		n-1	$n = \frac{3 + \sqrt{9 + 8(N-1)}}{2}$	
5	5	5	5	4	4,70156212	4
6	6	5,27491722	5	5	5	5
7	6	5,53112887	5	5	5,27491722	5
8	6	5,77200187	5	5	5,53112887	5
9	6	6	6	5	5,77200187	5
10	7	6,21699057	6	6	6	6
11	7	6,42442890	6	6	6,21699057	6
12	7	6,62347538	6	6	6,42442890	6
13	7	6,81507291	6	6	6,62347538	6
14	7	7	6	6	6,81507291	6
15	8	7,17,890835	7	7	7	7

En la tabla en cada columna significa:

Col 1: En número N con en cual debemos trabajar.
Col 2: El valor de n que debemos encontrar.
Col 3: El valor de n hallado con la formula.
Col 4: El resultado de la Col 3 con los decimales truncados.
Col 5: El valor de n-1
Col 6: El resultado de n hallado con la fórmula modificada (N por N-1)
Col 7: El resultado de la Col 6 con los decimales truncados.

En las columnas 4 y 7 consideramos los resultados con los decimales truncado, esto es justificable, puesto que en C, C++ y JAVA, cuando asignamos un valor punto flotante a una variable de tipo entero pues los decimales se truncan (no se redondean).

Bien, podemos concluir que el resultado para este problema lo hallamos con la siguiente formula:

$$n = \frac{3 + \sqrt{9 + 8(N-1)}}{2} + 1$$

Código Fuente en C:

```c
/* Problema : Diagonales
 * Lenguaje : ANSI C (version: 4.0)
 * Por      : Alberto Suxo
 *************************/
#include<stdio.h>
#include<math.h>

int main(){
   long long N, n;
   long C=1;
   while( 1 ){
      scanf("%lld", &N);
      if( !N )
         break;
      n = (3 + sqrt(9.0+8.0*(N-1)))/2;
      printf("Case %ld: %lld\n", C++, n+1);
   }
   return 0;
}
```

Código Fuente en JAVA:

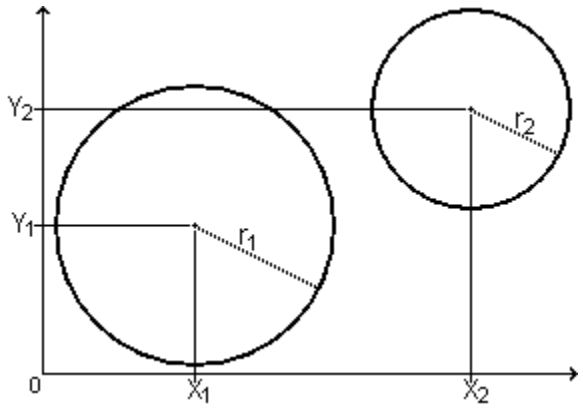
```java
/* Problema : Diagonales
 * Lenguaje : JAVA (version: 1.5)
 * Por      : Alberto Suxo
 *************************/

import java.util.Scanner;

public class D{
   public static void main( String args[] ){
      long N, n;
      long C=1;
      Scanner in = new Scanner( System.in );
      while( true ){
         N = in.nextLong();
         if( N==0 )
            break;
         n = (long)(3 + Math.sqrt( 9.0+8.0*(N-1) ) )/2;
         System.out.println("Case "+ C +": " + (n+1) );
         C++;
      }
   }
}
```

Dime Si Intersectan

Hace una semana, a un curso de niños les dieron un trabajo práctico.

Este trabajo consistía en determinar si un par de circunferencias se intersectan o no.

A cada niño, se le dio los siguientes datos: las coordenadas iniciales (x_1; y_1) y el radio r_1 de la primera circunferencia, y, las coordenadas iniciales (x_2; y_2) y el radio r_2 de la segunda circunferencia.



Cada uno de los niños ha resuelto su trabajo, sin embargo, es el profesor el que no quiere cometer error al momento de calificarlos, por lo que te pide ayuda para saber si los niños han hecho bien su trabajo.

Entrada

La entrada consiste en un set de datos, que, en cada línea tendrá 6 enteros positivos x_1, y_1, r_1, x_2, y_2 y r_2 separados por un espacio en blanco. Con las coordenadas ($0 \leq x_1$; y_1; x_2; $y_2 \leq 10000$) y los radios ($0 \leq r_1$; $r_2 \leq 1000$)

El set de datos terminará con una línea con 6 ceros: 0 0 0 0 0 0.

Salida

Por cada línea de entrada, usted debe desplegar “SI" si las dos circunferencias se intersectan y “NO" en caso contrario.

Ejemplo de Entrada

```
10 10 20 20 20 10
10 10 20 40 40 10
7062 2479 833 6611 3926 744
0 0 0 0 0 0
```

Ejemplo de Salida

```
SI
NO
SI
```

Análisis y Solución

Por: Alberto Suxo

Este un problema simple, lo único que debemos saber es la distancia entre dos puntos (las coordenadas x1,y1 y x2,y2), si esta distancia es menor a la suma de los radios de las circunferencias significa que se intersectan, caso contrario, no se intersectan.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} = \sqrt{\Delta x^2 + \Delta y^2}$$

Código Fuente en C:

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

inline double dist(long dx, long dy) {
   return sqrt(dx*dx+dy*dy);
}
int main() {
   long x1, y1, r1, x2, y2, r2;

   while( 1 ) {
      scanf( "%ld %ld %ld %ld %ld %ld",
               &x1, &y1, &r1, &x2, &y2, &r2);
      if( x1+y1+r1+x2+y2+r2==0 )
         break;
      if( dist(x2-x1,y2-y1)<(double)(r1+r2) )
         printf("SI\n");
      else
         printf("NO\n");
   }
   return 0;
}
```

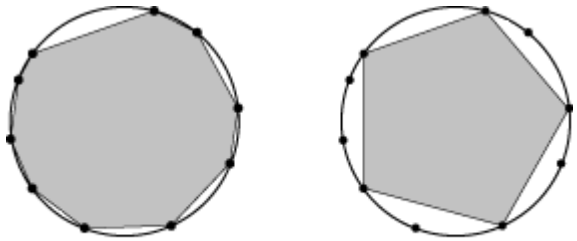
Encogiendo Polígonos

Un polígono se dice ser *inscrito* en un círculo cuando todos los vértices yacen en ese círculo.

En este problema te darán un polígono inscrito en un círculo, y debes determinar el mínimo número de vértices que deben ser eliminados para transformar el polígono dado en un *polígono regular*, esto es, un polígono que sea equiangular (todos los ángulos son congruentes) y equilateral (todos los lados tienen la misma longitud).

Cuando elimine un vértice v del polígono, primero eliminas el vértice y los lados conectan a este a sus vértices adyacentes w_1 y w_2, y entonces creas un nuevo lado conectando a w_1 y w_2.

La figura (a) de abajo ilustra un polígono inscrito en un círculo, con diez vértices, y la figura (b) muestra un pentágono (polígono regular con cinco lados) formado eliminando cinco vértices del polígono en (a).



En este problema, consideramos que cualquier polígono debe tener mínimo tres lados.

Entrada

La entrada contiene varios casos de prueba. La primera línea de un caso de prueba contiene un entero N indicando el número de vértices del polígono inscrito ($3 \leq N \leq 10^4$). La segunda línea contiene N enteros X_i separados por un solo espacio ($1 \leq X_i \leq 10^3$, para $0 \leq i \leq N-1$).

Cada X_i representa la longitud del arco definido en el círculo inscrito, de forma horaria, por vértice i y vértice (i+1) mod N. Recuerda que un *arco* es un segmento de la circunferencia del círculo; no lo confundas con *cuerda*, que es el segmento de una línea cuyos inicio y fin yacen en el círculo.

El fin de la entrada está indicado por una línea conteniendo solo un cero.

Salida

Para cada caso de prueba en la entrada, tu programa debe imprimir una sola línea, conteniendo el mínimo número de vértices que deben ser eliminados de dicho polígono para formar un polígono regular. Si no es posible formar un polígono regular, la línea debe contener solo el valor -1.

Ejemplo de Entrada

```
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
```

Ejemplo de Salida

```
0
2
-1
5
```

Análisis y Solución

Por: Jorge Terán

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 ayudar a resolver el problema:

- E 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 circulo separados por **arcLen**, esto 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 construirlo quitando i vértices
      }
   }
   isPos = false; //no es posible construirlo

}
```

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 este numero de lados tenemos una solución.

Codigo Fuente en JAVA:

```
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");
      }
   }
}
```

El Hotel con Habitaciones Infinitas

La ciudad de HaluaRuti tiene un extraño hotel con habitaciones infinitas. Los grupos que llegan a ese hotel siguen las siguientes reglas:

1. Al mismo tiempo, solo miembros de un grupo pueden rentar el hotel.
2. Cada grupo llega en la mañana de un día y salen al anochecer de otro día.
3. Otro grupo llega en la mañana siguiente después de que un grupo ha abandonado el hotel.
4. Una característica muy importante de un grupo que llega es que tiene un miembro más que el grupo anterior a menos que sea el primer grupo. Usted tendrá el número de miembros del grupo inicial.
5. Un grupo con n miembros se queda por n días en el hotel. Por ejemplo, si un grupo de cuatro miembros llega el 1ro de Agosto en la mañana, este se irá del hotel el 4 de Agosto por la noche y el siguiente grupo de cinco miembros llegará el 5 de Agosto en la mañana y se irá en 5 días y así sucesivamente.

Dado un tamaño de grupo inicial usted debe encontrar el tamaño del grupo que se encuentra en el hotel en un día específico.

Entrada

La entrada contiene números enteros **S** ($1 \leq \mathbf{S} \leq 10000$) y **D** ($1 \leq \mathbf{D} < 10^{15}$) en cada línea. **S** denota el tamaño inicial del grupo y **D** denota el día en para el cual debe encontrar el tamaño del grupo que está en el hotel, **D**-ésimo día (empezando desde 1). Todos los enteros de entrada y salida son menores a 10^{15}. Un tamaño de grupo **S** significa que en el primer día un grupo de **S** miembros llegó al hotel y se quedará por **S** días, entonces llegará un grupo de **S + 1** miembros de acuerdo a las reglas descritas previamente.

Salida

Por cada línea de entrada, imprima en una línea el tamaño del grupo que esta en el hotel en el D-ésimo día.

Ejemplo de entrada

```
1 6
3 10
3 14
```

Ejemplo de salida

```
3
5
6
```

Análisis y Solución

Por: Alberto Suxo

El inciso **5)** es bastante explícito, utilicemos los siguientes ejemplos:

```
3 10 => 5
3 14 => 6
         10  14
         Y   Y
33344445555566666677777777888888889
```

Esta secuencia sigue la misma conducta que la siguiente serie:

```
122333444455555666666777777888888
```

Para el primer ejemplo: 3 14 => 5 donde S=3 , D=14 y el resultado es k=6

Tenemos que 3+4+5 < D <= 3+4+5+**6**

$$\frac{k\cdot(k-1)}{2}-\frac{S\cdot(S-1)}{2}<D\le k+\frac{k\cdot(k-1)}{2}-\frac{S\cdot(S-1)}{2}$$

Por mi conveniencia, trabajaré con esta parte:

$$D\le k+\frac{k\cdot(k-1)}{2}-\frac{S\cdot(S-1)}{2}$$

Podemos apreciar que tengo, S, D y k, ahora debemos despejar k.

$$2D\le 2k+k\cdot(k-1)-S\cdot(S-1)$$
$$0\le 2k+k^2-k-S\cdot(S-1)$$
$$k^2+2k-k-S\cdot(S-1)-2D\ge 0$$
$$k^2+k-\left[2D+S\cdot(S-1)\right]\ge 0$$
$$k\ge\frac{-1\pm\sqrt{1+4\cdot\left[2D+S\cdot(S-1)\right]}}{2}$$
$$k\ge\sqrt{\frac{1}{4}+\left[2D+S\cdot(S-1)\right]}-\frac{1}{2}$$

También utilizaremos la función ceil(k), que nos devuelve el entero más pequeño que no sea menor que k.

Código Fuente en C:

```c
/* Problema : El Hotel con Habitaciones Infinitas
 * Lenguaje : ANSI C (version: 4.0 )
 * Por      : Alberto Suxo
 ************************/

#include<stdio.h>
#include<math.h>

int main(){
   double S, D, k;
   while( scanf( "%lf %lf", &S, &D ) !=EOF ){
      k = sqrt( 2.0*D+S*(S-1.0)+0.25 ) -0.5;
      printf( "%.0lf\n", ceil(k) );
   }
   return 0;
}
```

Código Fuente en Java:

```java
/* Problema : El Hotel con Habitaciones Infinitas
 * Lenguaje : JAVA (version: 1.5 )
 * Por      : Alberto Suxo
 ************************/

import java.util.Scanner;

public class A{
   public static void main( String args[] ){
      long S, D;
      double k;
      Scanner in = new Scanner( System.in );
      while( in.hasNext() ){
         S = in.nextLong();
         D = in.nextLong();
         k = Math.sqrt( 2.0*D+S*(S-1.0)+0.25 ) -0.5;
         System.out.println( (int)Math.ceil(k) );
      }
   }
}
```

¿En que base esta?

En la notación posicional, nosotros conocemos que la posición de un dígito indica el peso de ese dígito con respecto al valor de un número. Por ejemplo, el número 362 en base 10, conocemos que 2 tiene un peso 10^2, 6 el peso 10^1, y 2 el peso 10^0, produce el valor 3 x 100 + 6 x 10 + 2 x 1, o 300+60+2. el mismo mecanismo es usado para números expresados en otras bases. Mientras que la mayoría de las personas asumen que los números que usan cada día están en base 10, nosotros sabemos que podrían estar en otras bases. Por ejemplo, el número 362 en base 9 o base 14 representan números totalmente diferentes al valor de 362 en base 10.

Para este problema tu programa de presentar con una secuencia de pares de enteros. Llamaremos a los miembros de un par X y Y. Tu programa debe determinar la base mínima para X y la base mínima para Y (probablemente diferente a la de X) tal que X y Y representen el mismo valor.

Considere, por ejemplo, los enteros 12 y 5. Ciertamente no son iguales en base 10, Pero, ¿si 12 esta en base 3 y 5 en base 6? 12 base 3 = 1 x 3 + 2 x 1 = 5 en base 10, y 5 en base 6 es 5 en base 10. Así que 12 y 5 son iguales si usted selecciona las bases correctas para cada uno de ellos.

Entrada

En cada línea de entrada habrá un par de enteros X y Y, separados por uno o más espacios en blanco; pueden aparecer espacios al principio y/o al final de cada línea. La base asociada a X y Y están entre 1 y 36(inclusive), y como fue notado arriba, no necesariamente son la misma para X y Y. La representación de los dígitos 0 a 9 son de la usual decimal, los caracteres en mayúscula desde la A hasta la Z representan los valores desde 10 hasta 35 respectivamente.

Salida

Por cada par de enteros en la entrada, despliegue un mensaje similar al mostrado en los ejemplo de salida abajo. Claro, si dos enteros no son iguales en ninguna base, muestre un mensaje como se muestra en los ejemplos de salida abajo.

Ejemplo de Entrada	**Ejemplo de Salida**
`12   5`	`12 (base 3) = 5 (base 6)`
`    10     A`	`10 (base 10) = A (base 11)`
`12 34`	`12 (base 17) = 34 (base 5)`
`  123   456`	`123 is not equal to 456 in any base 2..36`
`  1    2`	`1 is not equal to 2 in any base 2..36`
`  10   2`	`10 (base 2) = 2 (base 3)`

Análisis y Solución

Por: Alberto Suxo

Otro problema de conversión de bases numéricas, mismas que Java hace automáticamente, este problema, parece ser un juego de la balanza, es decir, intentamos equilibrar los lados de la balanza incrementando gradualmente las bases de los datos. El incremento se hace hasta que los dos lados queden iguales en decimal o hasta que alguna de las bases sea mayor a 36 (que es nuestro límite).

Para resolver este problema de forma simple, primero tenemos que conocer que hace el método parseLong del objeto Long, pues simple, en el api de Java podemos encontrarlo, y se presenta así:

```
public static long parseLong(String s) throws NumberFormatException;
```

y

```
public static long parseLong(String s,
                             int radix) throws NumberFormatException;
```

Este segundo es el que nos importa por ahora, lo que hace es, data una cadena s en base radix, la convierte a base 10.

Por ejemplo

```
Long.parseLong("A", 16 ) = 10
```

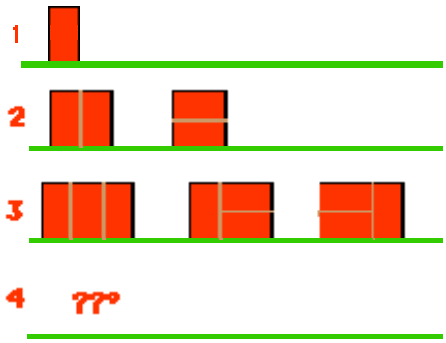
Código Fuente en JAVA:

```
/* Problema : ¿En Que Base Esta?
 * Lenguaje : JAVA (version: 1.5 )
 * Por      : Alberto Suxo
 ************************/
import java.util.Scanner;

class Base {

  static int baseOf( String str ) {
    char ch='1';
    for( int i=0; i<str.length(); i++)
      if( str.charAt(i)>ch )
        ch = str.charAt( i );
      return Integer.parseInt( ""+ch, 36 ) + 1;
  }
  static void solve( String A, String B ) {
    int baseA = baseOf( A );
    int baseB = baseOf( B );
    long valueA, valueB;
    do {
      valueA = Long.parseLong( A, baseA );
      valueB = Long.parseLong( B, baseB );
      if( valueA == valueB ) {
        System.out.println( A + " (base "+baseA+") = " +
                            B + " (base "+baseB+")");
        return;
      }
```

```
      if( valueA < valueB )
        baseA ++;
      else
        baseB ++;
    } while( baseA<37 && baseB<37 );
    System.out.println( A + " is not equal to " +
                        B + " in any base 2..36");
  }

  public static void main( String args[] ) {
    Scanner in = new Scanner( System.in );
    String A, B;
    while( in.hasNext() ) {
      A = in.next();
      B = in.next();
      solve( A, B );
    }
  }
}
```

Formas del Muro de Ladrillos

Si necesitáramos construir un muro de ladrillos con ladrillos que tienen e largo igual a dos veces su alto, y nuestro muro debe ser de un alto de dos unidades, podemos hacer el muro de un número de formas, dependiendo de cuan largo lo necesitemos. En la figura, uno observa que:



- Hay solo una forma para un muro de una unidad de largo - poniendo un ladrillo en su fin.
- Hay dos formas para un muro de largo 2: los dos ladrillos de lado puestos uno sobre el otro y la otra forma es con los dos ladrillos de pie uno a lado del otro.
- Hay tres formas para un muro de largo 3.

Cuantas formas puedes encontrar para un muro de largo 4? Y, para un muro de largo 5?

Problema

Tu trabajo es escribir un programa que dado un largo de muro, determine cuantas formas pueden haber para un muro de este largo.

Entrada

Tu programa recibirá una secuencia de enteros positivos, uno por línea, cada uno representa el largo del muro. El valor máximo de tamaño de muro es 50. La entrada termina con un 0.

Salida

Por cada largo de muro dado en la entrada, tu programa debe lanzar el número correspondiente de formas en que se puede armar el muro en una línea separada.

Ejemplo de Entrada	Ejemplo de Salida
1 2 3 0	1 2 3

Análisis y Solución

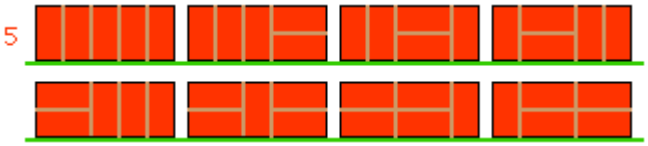
Por: Alberto Suxo

Este problema no es nada complicado, pero para comprenderlo mejor hagamos unas pequeñas pruebas, gracias a la gráfica provista en el planteamiento del problema conocemos las respuestas para cuando el muro de ladrillos tiene longitud 1, 2 y 3, ahora hagamos pruebas para muros de tamaño 4 y 5.

Para longitud 4:

Podemos ver que hay 5 formas de ordenar los ladrillos para formar un muro de longitud 4.

Ahora para longitud 5:



Hay 8 formar de ordenar los ladrillos para formar un muro de longitud 5.

Hasta ahora lo que tengo es lo siguiente:

Longitud del muro	Número de Formas
1	1
2	2
3	3
4	5
5	8

Esta serie es claramente la serie fibonacci, con lo que ya tengo la seguridad de que para cuando el muro sea de longitud 6 habrán 13 formas de ordenar los ladrillos.

A modo de práctica sugiero al lector hacer la prueba de forma gráfica para muros de longitud 6 y 7 (21 formas), ya que es mejor tener la seguridad de muestra conclusión.

Ahora otro factor importante a tener en cuenta al resolver este problema es el máximo valor que tendrá la longitud del muro. Afortunadamente, la longitud de muro tendrá como máximo 50 unidades.

Código Fuente en C.

```c
#include<stdio.h>

int main() {
   long long V[51];
   int n;
   V[1] = 1;
   V[2] = 2;
   for( n=3; n<51; n++ )
      V[n] = V[n-1] + V[n-2];
   //freopen("brick.in", "r", stdin);
   //freopen("brick.out", "w", stdout);
   while( 1 )
   {
      scanf("%d", &n );
      if( !n )
         break;
      printf("%lld\n", V[n] );
   }
   return 0;
}
```

Código Fuente en Java.

```java
import java.util.Scanner;

class Brick {
   public static void main( String args[] )
   {
      Scanner in = new Scanner( System.in );
      long V[] = new long[51];
      int n;
      V[1] = 1;
      V[2] = 2;
      for(n=3; n<51; n++)
         V[n] = V[n-1] + V[n-2];
      while( true ){
         n = in.nextInt();
         if( n==0 )
            break;
         System.out.println( V[n] );
      }
   }
}
```

Número de la Suerte

Hemos observado que muchos concursos de programación tiene un problema llamado “Número de la suerte". Así que decidimos tener uno también.

Definimos una secuencia de suerte, a la secuencia infinita de todos los enteros, en orden ascendente, que puede representarse como cualquier potencia entera positiva de 5 (es decir 5^k, donde k es un entero positivo) o como una suma de distintas potencias enteras positivas de 5 (es decir, $5^{a1} + 5^{a2} + 5^{a3} + \ldots$, donde a1, a2, a3, … son enteros positivos distintos). Todos los números en la secuencia de suerte se llaman números de la suerte. Los primeros números de la suerte son pocos 5, 25, 30, 125, 130, 150,… Dado n su tarea es encontrar el n-ésimo número de la suerte.

Entrada

La primera línea de entrada contiene un número entero t, t ≤ 200, que representa el número de casos.

A continuación, las t líneas siguen, cada uno con un entero n, 1 ≤ n ≤ 8000. La entrada termina con t = 0.

Salida

Para cada caso de prueba imprima el n-ésimo número de la suerte en una línea independiente. Las respuestas van a caber en un entero de 32-bits con signo.

Ejemplo de Entrada

```
4
1
2
3
9
0
```

Ejemplo de Salida

```
5
25
30
630
```

Análisis y Solución

Por: Gabriel Rivera

Analizando el problema

Como se necesitan las potencias de 5 y sus sumas, entonces se va calculando cada potencia de 5 y almacenándola, por ejemplo, todas las sumas posibles con las potencias 1, 2 y 3 sería de la siguiente manera:

Calculamos la primera potencia y la almacenamos y tendríamos: 5

Luego, calculamos la segunda potencia, y hallamos la suma con los anteriores datos y tendríamos almacenados: 5, 25, 30

Ahora, calculamos la tercera potencia y la sumamos con las anteriores y tendríamos: 5, 25, 30, 125, 130, 150, 155

Finalmente para tener la serie en orden, hacemos una ordenación de estos datos y así obtenemos la serie.

Código Fuente en JAVA:

```
/* Problema : Numero de la Suerte
 * Lenguaje : Java JDK 1.6
 * Por      : Gabriel Rivera Safadi
 ********************************/
import java.util.Arrays;
import java.util.Scanner;
class NumeroSuerte {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		int lim = 8000;
		int lpow = 13;
		int v[] = new int[lim];
		int pos = 0;
		for (int i = 1; i <= lpow && pos < lim; i++) {
			v[pos++] = (int)Math.pow(5, i);
			int lpos = pos - 1;
			for (int j = 0; j < lpos && pos < lim; j++) {
				v[pos++] = v[lpos] + v[j];
			}
		}
		Arrays.sort(v);
		while (true) {
			int t = sc.nextInt();
			if (t == 0) {
				break;
			}
			for (int tt = 0; tt < t; tt++) {
				System.out.println(v[sc.nextInt() - 1]);
			}
		}
	}
}
```

Números Casi Primos

Los números casi primos son número no-primos que son divisibles por solo un número primo. En este problema tu trabajo es escribir un programa que encuentre la cantidad de número casi primos dentro de cierto rango.

Entrada

La primera línea de la entrada contiene un entero N (N ≤ 600) que indica cuantos sets de datos siguen. Cada una de las siguientes N líneas es un set de entrada. Cada set contiene dos número enteros *low* y *high* (0 < *low* ≤ *high* < 10^{12}).

Salida

Por cada línea de entrada excepto la primera usted debe producir una línea de salida. Esta línea contendrá un entero, que indica cuantos números casi primos hay dentro del rango(incluyendo) *low . . . high*.

Ejemplo de entrada

```
3
1 10
1 20
1 5
```

Ejemplo de salida

```
3
4
1
```

Análisis y Solución

Por: Alberto

El primer párrafo dice:

Los Números Casi Primos son número no primos, que son divisibles por un único número primo. En este problema tu trabajo es escribir un programa que encuentre el número de número casi primos dentro de cierto rango.

Estos son los primeros 15 números casi primos:

```
4 8 9 16 25 27 32 49 64 81 121 125 128 169 243
```

Por ejemplo 16 es el resultado de la multiplicación de 2*2*2*2, por lo que es un número válido para nuestro propósito.

Un valor no válido es el 10, puesto que es resultado de la multiplicación de 2*5, que son dos números primos diferentes.

Una vez comprendido esto, sabremos que para un número primo p, los números casi primos que de él surjan serán $p^2, p^3, ..., p^k$ donde $p^k < 10^{12}$

Conociendo nuestro límite 10^{12} sabemos que el número primo más grande no será mayor a 10^6, por lo que el rango de los números primos con los que trabajaremos esta entre 2 y 10^6. Hay 78498 primos y el mayor es 999983.

Por cada uno de estos números primos, generaremos los números casi primos y uniremos estos para tener un único conjunto de números casi primos.

¿Al unir estos conjuntos podrán existir números repetidos?

No, como estos números son el resultado de alguna exponencial de un número primo, pues es lógico afirmar que no existen x,y>1 que multiplicados nos den un primo.

Para resolver este problema utilizaré tablas precalculadas, criba de Eratostenes, ordenación y búsqueda binaria.

Primero: Criba de Eratostenes para encontrar todos los número primos desde 2 hasta 106.

```
for( i=2; i<1000; i++){
   if(P[i]==false){
      for( j=i+i; j<1000000; j+=i ){
         P[(int)j] = true;
      }
   }
}
```

Segundo: En base a mi criba, generaré todas las potencias de los números primos, almacenados en el vector A, uno detrás de otro.

```
k=0;
for(i=2; i<1000000; i++){
   if( P[i]==false ){
      m=(long)i;
      for(j=m*m; j<LIMIT; j*=m){
```

```
            A[k++] = j;
         }
      }
   }
```

Tercero: Ordenar el vector A.

```
   Arrays.sort(A);
```

Cuarto: Para todos los pares de datos, utilizando "Arrays.binarySearch(A, dato)", identificaré la posición de los datos low y high. Y el resultado no es más que "highPs-lowPs+1".

Es necesario aclarar que "Arrays.binarySearch(Vector, dato)" funciona de la siguiente manera, si encuentra el dato dentro del Vector retorna la posición donde se encuentra dicho dato, caso contrario retorna –(posición de inserción) -1.

En caso de no encontrar low y/o high, pues nos da el lugar donde deberían ser insertados, y para poder utilizar nuestra fórmula sin complicaciones hay que modificar este valor negativo de la siguiente forma:

```
Para low:  lowPs  = (-1)*(lowPs+1);
Para high: highPs = (-1)*(highPs+2);
```

Código Fuente en JAVA:

```
/* Problema : Numero Casi Primos
 * Lenguaje : JAVA (version: 1.5)
 * Por      : Alberto Suxo
 ************************/
import java.util.Arrays;
import java.util.Scanner;

public class E {
   static long[] A = new long[80070];

   static void generate() {
      boolean[] P = new boolean[1000000];
      long LIMIT = (long)Math.pow(10.0, 12);
      int i, k;
      long j, m;
      for( i=2; i<1000; i++) {
         if(P[i]==false){
            for( j=i+i; j<1000000; j+=i )
               P[(int)j] = true;
         }
      }
      K = 0;
      for( i=2; i<1000000; i++ ) {
         if( P[i]==false ) {
            m = (long)i;
            for( j=m*m; j<LIMIT; j*=m )
               A[k++] = j;
         }
      }
      Arrays.sort(A);
   }
```

```
   public static void main( String args[] ) {
      Scanner in = new Scanner( System.in );
      int N;
      long low, high;
      int lowPs, highPs;
      generate();
      N = in.nextInt();
      while( (N--)>0) {
         low  = in.nextLong();
         high = in.nextLong();
         lowPs = Arrays.binarySearch(A, low);
         if( lowPs<0 )
            lowPs = (-1)*(lowPs+1);
         highPs = Arrays.binarySearch(A, high);
         if( highPs<0 )
            highPs = (-1)*(highPs+2);
         System.out.println( highPs – lowPs + 1 );
      }
   }
}
```

Pares de Ruth-Aaron

Este nombre fue dado por Carl Pomerance en honor a Babe Ruth y Hank Aaron estrellas de las ligas del baseball estadounidense. El récord que tenia Ruth era de 714 entradas, que superada por Aron el 8 de abril de 1974, con un total de 715 entradas. Uno de los estudiantes de Pomerance advirtió que la suma de los factores primos de 714 y 715 eran iguales. Veamos:

Si factorizamos ambos números obtenemos las siguientes descomposiciones:

$$714 = 2 \times 3 \times 7 \times 17$$
$$715 = 5 \times 11 \times 13$$

Si nos fijamos en las sumas de ambas factorizaciones tenemos que:

$$2 + 3 + 7 + 17 = 5 + 11 + 13 = 29$$

A los números que satisfacen esta propiedad, es decir, a los pares consecutivos cuya descomposición en factores primos tienen la misma suma, Pomerance les llamó pares de *Ruth-Aaron*. Y claro está, los ordenadores son fantásticos. Pomerance descubrió que entre los números menores a 20.000 hay 26 pares de *Ruth-Aaron*. El mayor en este rango lo forman el 18.490 y el 18.491.

Si consideramos solo los factores primos diferentes, los primeros pares de *Ruth-Aaron* son:

(5, 6), (24, 25), (49, 50), (77, 78), (104, 105), (153, 154), (369, 370), (492, 493), (714, 715), (1682, 1683) y (2107, 2108)

Entrada

La entrada consiste de varios casos de prueba. Cada caso de prueba contiene dos enteros a,b (2 <= a,b <= 10^7). La entrada termina cuando a=b=0.

Salida

Para cada caso de prueba su programa debe mostrar en la salida los pares de Ruth-Aron, entre a y b.

Ejemplo de Entrada

```
300 1000
5000 7000
0 0
```

Ejemplo de Salida

```
369 370
492 493
714 715
5405 5406
6556 6557
6811 6812
```

Análisis y Solución

Por: Jorge Terán

Para resolver este problema construiremos una criba de Eratostenes.

```
for (i = 2; i <= n; i++)
   if (p[i] == 0) {
      for (j = i ; j <= n; j = j + i) {
         p[j] += i;
         //pcont[j]++;
      }
   }
```

En este algoritmo de criba en lugar de marcar con 1 o 0 hemos sumado el índice que estamos marcando.

En esta criba nos queda la suma de los divisores primos sin repetición del número.

Luego podemos recorrer la criba y ver si dos números consecutivos son iguales, estos serian los números de Ruth Aron.

```
for (i = a; i <= b; i++) {
   if (p[i] == p[i+1]) {
      System.out.println(i +" "+  (i+1));
   }
}
```

Código Fuente en Java

```
/**
 * Solucion final al problema Ruth Aron con
 * factores con factores NO repetidos
 *
 * @author Jorge Teran
 **/
import java.util.Scanner;

public class RuthAronNORepetido {

   public static int n = 10000000;
   public static int[] p = new int[n + 1];
   //public static int[] pcont = new int[n + 1];

   public static void main(String[] args) {

      int i = 2, j = 0, a, b;
      // construccion de la criba
      for (i = 2; i <= n; i++) {
         if (p[i] == 0) {
            for (j = i ; j <= n; j = j + i) {
               p[j] += i;
               //pcont[j]++;
            }
         }
      }
```

```
        Scanner in = new Scanner(System.in);
        while (in.hasNext()) {
           a = in.nextInt();
           b = in.nextInt();
           if((a+b)==0) break;
           for (i = a; i <= b; i++) {
              if (p[i] == p[i+1]) {
                 System.out.println(i +" "+  (i+1));
              }
           }
        }
     }
}
```

Raíz Digital Prima

La *raíz digital* de un número es hallado adicionando todos lo dígitos en un número. Si el número resultante tiene más de un dígito, el proceso es repetido hasta temer un simple dígito.

Tu trabajo en este problema es calcular una variación de la raíz digital – una *raíz digital prima*. El proceso de adición descrito arriba para cuando solo queda un dígito, pero podemos para en el número original, o en cualquier número intermedio (formado por la adición) que sea número primo. Si el proceso continúa y el resultado es un dígito que no es primo, entonces el número original no tiene raíz digital prima.

Un entero *mayor que* uno es llamado número primo si tiene solo dos divisores, el uno y si mismo.

- Por ejemplo, los primeros seis primos son 2, 3, 5, 7, 11, y 13.
- El número 6 tiene cuatro divisores: 6, 3, 2, y 1. Por eso 6 *no* es primo.
- Advertencia: en número 1 *no* es primo.

EJEMPLOS DE RAIZ DIGITAL PRIMA

1 Este no es un número primo, así que 1 no tiene raíz digital prima.
3 Este es un número primo, así que la raíz digital prima de 3 es 3.
4 Este no es un número primo, así que 4 no tiene raíz digital prima.
11 Este es un número primo, así que la raíz digital prima de 11 es 11.
642 Este no es un número primo, así que sumando 6 + 4 + 2 = 12. Este no es un número primo, así que sumando 1 + 2 = 3. Este si es un número primo, así que la raíz digital prima de 642 es 3.
128 Este no es un número primo, así que sumando 1 + 2 + 8 = 11. Este es un número primo, así que la raíz digital prima de 128 es 11.
886 Este no es un número primo, así que sumando 8 + 8 + 6 = 22. Este no es un número primo, así que sumando 2 + 2 = 4. Este no es un número primo, así que 886 no tiene raíz digital prima.

Entrada

La entrada contendrá un entero en cada línea en el rango de 0 a 999999 inclusive. El fin de la entrada se indica con el valor 0.

Salida

Si el número ingresado tiene raíz digital prima, entonces se debe desplegar el valor original y el valor de la raíz digital prima, caso contrario se despliega el valor original seguido por la palabra “none”, los valores deben estar alineados con una justificación derecha de 7 espacios, como se muestra en el ejemplo de salida.

Ejemplo de entrada

```
1
3
4
11
642
128
886
0
```

Ejemplo de salida

```
      1    none
      3       3
      4    none
     11      11
    642       3
    128      11
    886    none
```

```
Nota: la salida tiene el siguiente formato:
         111111
123456789012345
      4    none
    642       3
```

Análisis y Solución

Por: Alberto Suxo

Como la descripción lo indica, el problema consiste en verificar si un número es primo, de no serlo, sumar todos los dígitos de dicho número y repetir el proceso hasta encontrar un primo o hasta que el número solo tenga un dígito.

Como podemos ver, sólo se requiere de dos funciones que por lo general se utilizan en introducción a la programación, o sea, que este problema se clasifica dentro de los más fáciles.

Pare resolver este problema solo necesitamos recordar como sumar los dígitos de un número, y cómo verificar si un número es primo o no.

En la función `int prime( long N );` veremos que hay una serie de condicionales, pues esta función parte de los siguientes principios:

1.- Un número menor a dos no es primo.
2.- El único primo par es el dos.
3.- Cualquier otro par no es primo.
4.- Basta con encontrar un divisor de N entre 3 hasta $\sqrt{N}$ para asegurar que no es primo.

Código Fuente en C:

```
/* Problema : Raiz Digital Prima
 * Lenguaje : ANSI C (version: 4.0 )
 * Por      : Alberto Suxo
 ************************/

#include<stdio.h>
#include<math.h>

int prime( long N ) {
   int i, root;
   if ( N<2 )  return 0;
   if ( N==2 )  return 1;
   if ( !(N&1) )  return 0;
   root = (long)sqrt( N );
   for( i=3; i<=root; i+=2 )
      if ( N%i == 0 )
         return 0;
   return 1;
}

long SumDig( long Ns ){
   long s = 0;
   long X = Ns;
   while( X>0 ){
      s = s + (X%10);
      X = X/10;
   }
   return s;
}
```

```
int main(){
   long N, N2;
   int pr;
   scanf( "%ld", &N );
   while( N>0 ){
      pr = 0;
      N2 = N;
      pr = prime( N2 );
      while( N2>9 && !pr ){
         N2 = SumDig ( N2 );
         pr = prime ( N2 );
      }
      if( pr )
         printf( "%7ld %7ld\n", N, N2 );
      else
         printf( "%7ld    none\n", N );
      scanf( "%ld", &N );
   }
   return 0;
}
```

Código Fuente en JAVA:

```
/* Problema : Raiz Digital Prima
 * Lenguaje : JAVA (version: 1.5 )
 * Por      : Alberto Suxo
 *************************/

import java.util.Scanner;

public class E{

   static boolean prime( long N ) {
      int i, root;
      if ( N<2 )  return false;
      if ( N==2 )  return true;
      if ( (N&1)==0 )  return false;
      root = (int)Math.sqrt( N );
      for( i=3; i<=root; i+=2 )
         if ( N%i == 0 )
            return false;
         return true;
  }

   static long SumDig( long Ns ){
      long s = 0;
      long X = Ns;
      while( X>0 ){
         s = s + (X%10);
         X = X/10;
      }
      return s;
   }
```

```
   public static void main( String args[] ){
      long N, N2;
      boolean pr;
      Scanner in = new Scanner( System.in );
      N = in.nextLong();
      while( N>0 ){
         pr = false;
         N2 = N;
         pr = prime( N2 );
         while( N2>9 && !pr ){
            N2 = SumDig ( N2 );
            pr = prime ( N2 );
         }
         if( pr )
            System.out.printf( "%7d %7d\n", N, N2 );
         else
            System.out.printf( "%7d    none\n", N );
         N = in.nextLong();
      }
   }
}
```

Regreso a la Física de Secundaria

Una partícula tiene una velocidad y aceleración inicial. Si la velocidad después de cierto tiempo es v, entonces cual es el desplazamiento en el doble del tiempo?

Entrada

La entrada contendrá dos enteros el cada línea. Cada línea es un set de entrada. Estos dos enteros denotan los valores de v ($-100 \leq v \leq 100$) y t ($0 \leq t \leq 200$) (t es el tiempo que la partícula tiene la velocidad v)

Salida

Por cada línea de entrada imprima un entero en una línea que denote el desplazamiento en el doble del tiempo.

Ejemplo de entrada

```
0 0
5 12
```

Ejemplo de salida

```
0
120
```

Análisis y Solución

Por: Alberto Suxo

Pues bien, este problema no requiere mayor explicación:

El primer párrafo dice:

Una particular tiene una velocidad y aceleración inicial. Si la velocidad después de cierto tiempo es v entonces cual es el desplazamiento en el doble de tiempo?.

La fórmula para el desplazamiento es: $x = v_0 \cdot t + \frac{1}{2} \cdot a \cdot t^2$

Pero como no tenemos la aceleración asumimos que a=0, y como nos pide la distancia en el doble del tiempo pues tendremos: $x = 2 \cdot v_0 \cdot t$

Código Fuente en C:

```c
/* Problema : Regreso a la Fisica de Secundaria
 * Lenguaje : ANSI C (versión: 4.0 )
 * Por      : Alberto Suxo
 *************************/
#include<stdio.h>

int main(){
   long Vo, t;
   while( scanf( "%ld %ld", &Vo, &t )!=EOF )
      printf( "%ld\n", 2*Vo*t );
   return 0;
}
```

Código Fuente en JAVA:

```java
/* Problema : Regreso a la Fisica de Secundaria
 * Lenguaje : JAVA (versión: 4.0 )
 * Por      : Alberto Suxo
 *************************/
import java.util.Scanner;

public class B{
   public static void main( String args[] ){
      int Vo, t;
      Scanner in = new Scanner( System.in );
      while( in.hasNext() ){
         Vo = in.nextInt();
         t  = in.nextInt();
         System.out.println( 2*Vo*t );
      }
   }
}
```

Nota: No deberían existir distancias negativas, pero para este problema en particular, es valido cuando la distancia resultante es negativa.

Transmisores

En una red inalámbrica con múltiples transmisores transmitiendo en la misma frecuencia, es un requisito que las señales no se sobrepongan, o por lo menos no entren en conflicto. Una forma de solucionar esto es restringir el área de cobertura del transmisor. En este caso, se usa un transmisor blindado que solo emite su señal en un semicírculo.

Un transmisor T se encuentra en alguna área plana de unos 1,000 metros cuadrados. Transmitiendo en un área semicircular de radio r. El transmisor puede girar cualquier cantidad, pero no moverse. Dados N puntos dentro del área, calcule el máximo número de puntos a los que la señal les llega simultáneamente.

La Figura 1 muestra los mismos puntos con dos diferentes rotaciones del transmisor.

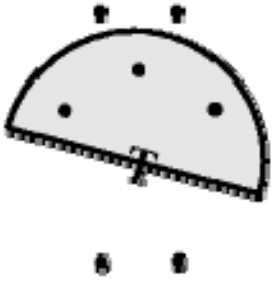


Figure 1a

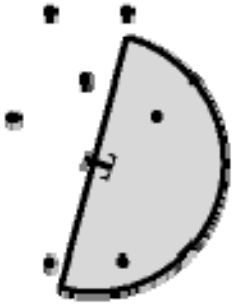


Figure 1b

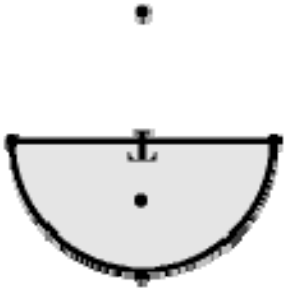


Figure 2

Todas las coordenadas de entrara son enteros (0 a 1000). El radio es un número real positivo mayor a cero.

All input coordinates are integers (0-1000). The radius is a positive real number greater than 0. Los puntos en el límite del semicírculo se consideran dentro del semicírculo. Hay de 1 a 150 puntos para examinar por cada transmisor. No hay puntos en la misma posición que el transmisor.

Entrada

La entrada consiste en la información de uno o más problemas de transmisores. Cada problema comienza con una línea que contiene las coordenadas (x, y) del transmisor seguidos por el radio de transmisión r. La siguiente línea contiene el número de puntos N dentro del área, seguido por N sets de coordenadas, un set en cada línea. El final de la entrada esta señalado por una línea que contiene un radio negativo; los valores de (x, y) estarán presentes pero serán indeterminados. Las Figuras 1 y 2 representan los datos del primer y segundo ejemplo de entrada presentados abajo en diferentes escalas. La figura 1a y 2 muestran la rotación de transmisor con la cobertura máxima.

Salida

Por cada transmisor, la salida contiene una única línea con el número máximo de puntos contenidas en el semicírculo.

Ejemplo de Entrada

```
25 25 3.5
7
25 28
23 27
27 27
24 23
26 23
24 29
26 29
350 200 2.0
5
350 202
350 199
350 198
348 200
352 200
995 995 10.0
4
1000 1000
999 998
990 992
1000 999
100 100 -2.5
```

Ejemplo de Salida

```
3
4
4
```

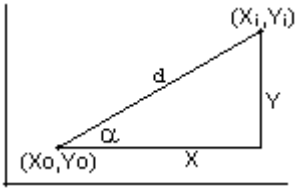
Análisis y Solución

Por: Alberto Suxo

Para resolver este problema hay que analizarlo un poco.

Primero, imaginemos a las coordenadas del transmisor (x_0, y_0) como las coordenadas centrales de nuestra área.

Y a todos los demás puntos los calcularemos con respecto a origen del transmisor.



Donde: $x = x_i - x_0$, $y = y_i + y_0$ y $d = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2} = \sqrt{x^2 + y^2}$

Bueno, una vez calculada la distancia entre el transmisor y un punto cualquiera, vemos, si este punto se encuentra dentro del radio de transmisión, y esto es simple teniendo la distancia, si la distancia es menor o igual al radio, el punto esta dentro del radio de transmisión, caso contrario, este punto debe ser desechado.

Ahora bien, esta solución (la que les presento), si bien no es la mejor, es una muy simple, primero, ordenaremos todos los puntos, en base a sus ángulos (todo en radianes), y contamos la mayor cantidad de puntos que se encuentren en un rango de π (media circunferencia), para esto, necesitamos sus ángulos, y lo haremos con la siguiente fórmula.

$$\cos(\alpha) = \frac{base}{hipo} = \frac{x}{d}$$

La distancia d jamás será cero, ya que, claramente se dijo que no existe ningún punto que se encuentre en la misma posición del transmisor.

Ahora, despejamos el ángulo: $\alpha = \cos^{-1}\left(\frac{x}{d}\right)$

Por último, esta ecuación es correcta cuando y es positivo, pero cuando y es negativo el ángulo esta en dirección inversa, y lo resolvemos así: $2\pi - \alpha$

Código Fuente en C:

```
/* Problema : Transmisores
 * Lenguaje : ANSI C
 * Por      : Alberto Suxo.
 ************************/
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
```

```c
int s_f( const void *a, const void *b ) {
  double A=*(double *)a, B=*(double *)b;
  if( A==B )
    return 0;
  else
    if( A>B )
      return 1;
  return -1;
}

int main() {
  double xo, yo, r;
  int n, i, j, k, max;
  double xi, yi, x, y, d, V[151], fin;
  while( 1 ) {
    scanf( "%lf %lf %lf", &xo, &yo, &r );
    if( r<0 )
      break;
    scanf( "%d", &n );
    i = 0;
    while( n-- ) {
      scanf( "%lf %lf", &xi, &yi );
      x = xi - xo;
      y = yi - yo;
      d = sqrt( x*x + y*y );
      if( d<=r ) {
        V[i] = acos( x/d );
        if( y<0 )
          V[i] = 2*M_PI - V[i];
        i++;
      }
    }
    V[i] = 15.0;
    if( i ) {
      qsort( (void *)V, i, sizeof(double), s_f );
      max = 1;
      for( j=0; j<i; j++ ) {
        fin = V[j] + M_PI;
        k = j;
        while( V[++k]<=fin );
        if( k-j>max )
          max = k - j;
      }
      printf( "%d\n", max );
    }
    else
      printf( "0\n" );
  }
  return 0;
}
```

Triángulos Isósceles

Un triángulo dado puede ser equilátero (tres lados de la misma longitud), escaleno (tres lados de diferente longitud) o isósceles (dos lados de la misma longitud y el tercer lado de una longitud diferente). Es conocido el hecho de que los puntos con todas coordenadas no pueden ser vértices de un triángulo equilátero.

Dado un conjunto de diferentes puntos con coordenadas enteras en un plano XY, de tal modo que no hay tres puntos en el conjunto que estén en la misma línea. Tu trabajo es calcular cuántas de las posibles elecciones de tres puntos son vértices de un triángulo isósceles.

Entrada

Existen varios casos de prueba. Cada caso de prueba está dado en varias líneas. La primera línea de cada caso de prueba contiene un entero N indicando el número de puntos en el conjunto ($3 \leq N \leq 1000$). Cada una de las siguientes N líneas, describe un diferente punto del conjunto, usando dos enteros X e Y separados por un simple espacio ($1 \leq X, Y \leq 106$); estos valores representan las coordenadas del punto en el plano XY. Puedes asumir que en cada caso de prueba no existen dos puntos en la misma ubicación y no hay tres puntos que sea colineales.

El último caso de prueba es seguido de una línea que contiene un cero.

Salida

Para cada caso de prueba, mostrar una sola línea con un solo entero, indicando el número de subconjuntos de tres puntos que son vértices de un triángulo isósceles.

Ejemplo de Entrada

```
5
1 2
2 1
2 2
1 1
1000 1000000
6
1000 1000
996 1003
996 997
1003 996
1003 1004
992 1000
0
```

Ejemplo de Salida

```
4
10
```

Análisis y Solución

Por: Gabriel Rivera

Primeramente es necesario saber la distancia entre dos puntos, calculada usando la fórmula:

Math.sqrt(Math.pow(x1 – x2, 2) + Math.pow(y1 – y2, 2))

Primeramente se almacenan los puntos en una matriz:

X	1	2	2	1	1000
Y	2	1	2	1	1000000

Con esto, podemos almacenar las distancias en una matriz, donde se tenga la distancia del punto i al punto j.

	(1,2)	(2,1)	(2,2)	(1,1)	(1000,1000000)
(1,2)	0	1.41421356…	1	1	999998.4990013..
(2,1)	1.4142135623…	0	1	1	999999.4980023..
(2,2)	1	1	0	1.41421356…	999998.4980028..
(1,1)	1	1	1.41421356…	0	999999.4990008..
(1000, 1000000)	999998.4990013…	999999.4980023..	999998.4980028..	999999.4990008..	0

Luego ordenamos la matriz, para que queden juntos los puntos que tengan la misma distancia.

0	1	1	1.41421356…	999998.4990013..
0	1	1	1.41421356…	999999.4980023..
0	1	1	1.41421356…	999998.4980028..
0	1	1	1.41421356…	999999.4990008..
0	999998.498002..	999998.4990013..	999999.4980023..	999999.4990008..

Con esto podemos ver que si un punto tiene la misma distancia con otros dos, puede formar un triángulo isósceles, y ahora solo queda contarlos.

0	1	1	1.41421356…	999998.4990013..
0	1	1	1.41421356…	999999.4980023..
0	1	1	1.41421356…	999998.4980028..
0	1	1	1.41421356…	999999.4990008..
0	999998.498002…	999998.499001…	999999.4980023..	999999.4990008..

Con esto vemos que son 4 triángulos isósceles que se forman con estos puntos.

Viendo con el segundo ejemplo, surge otro caso, qué pasa si más de 2 puntos tienen la misma distancia entre ellos.

0	5	5	5	5	8
0	5	5	6	7.0710678118654755	9.899494936611665
0	5	5	6	7.0710678118654755	9.899494936611665
0	5	7.0710678118654755	8	9.899494936611665	11.704699910719626
0	5	7.0710678118654755	8	9.899494936611665	11.704699910719626
0	5	5	8	11.704699910719626	11.704699910719626

Para los que están juntos, podemos ver que si son 2 puntos se forma 1 triángulo, si son 3 puntos se forman 3 triángulos, y si son 4 puntos se forman 6 triángulos. Una serie 1, 3, 6, 10, 15,…

Código Fuente en JAVA:

```
import java.util.Arrays;
import java.util.Scanner;

class IsoscelesGabo {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (true) {
            int n = sc.nextInt();
            if (n == 0)
                break;
            //Almacenar los Puntos
            int pts[][] = new int[n][2];
            for (int i = 0; i < n; i++) {
                pts[i][0] = sc.nextInt();
                pts[i][1] = sc.nextInt();
            }
            //Calcular las Distancias
            double d[][] = new double[n][n];
            for (int i = 0; i < n - 1; i++) {
                for (int j = i + 1; j < n; j++) {
                    //Calcular la Distancia entre 2 puntos
                    d[i][j] = Math.sqrt(Math.pow(pts[i][0] - pts[j][0],
2) + Math.pow(pts[i][1] - pts[j][1], 2));
                    //Copiar la distancia, porque d(p1,p2) = d(p2,p1)
                    d[j][i] = d[i][j];
                }
            }
            int res = 0;
            for (int k = 0; k < n; k++) {
                //Ordenar una fila de la Matriz de Distancias
                Arrays.sort(d[k]);
                //Acumulativo para la serie
                int cumul = 0;
                for (int i = 0; i < n - 1; i++) {
                    //Si son igules
                    if (d[k][i] == d[k][i + 1]) {
                        res++;
                        res += cumul;
                        //El numero de la serie aumenta
                        cumul++;
                    } else {
                        //La serie vuelve a cero
                        cumul = 0;
                    }
                }
            }
            System.out.println(res);
        }
    }

}
```

¡Un Problema Fácil!

¿Has oído el hecho “La base de todo número normal en nuestro sistema es 10”?. Prosupuesto, Yo no hablo de sistema numérico de Stern Brockot. Este problema no tiene nada que ver con este hecho pero quizá tenga alguna similitud.

Debes determinar la base N para un número entero R y deberás garantizar que R es divisible por (N-1). Debes imprimir el menor valor posible para N. El rango de N es $2 \leq N \leq 62$ y los símbolos para una base 62 son (0..9 y A..Z y a..z). Similarmente, los símbolo para una base 61 son (0..9 y A..Z y a..y) así sucesivamente.

Entrada

Cada línea de la entrada contiene un entero (como se define en matemática) en cualquier base (2..62). Debes determinar cual es la menor base posible que satisface la condición. No hay valores inválidos en la entrada. El tamaño más grande del archivo de entrada es de 32KB.

Salida

Si un número en estas condiciones es imposible imprima la línea ‘such number is impossible!’. Por cada línea de entrada debe haber una sola línea de salida. La salida debe estas en un sistema numérico decimal.

Ejemplo de entrada

```
3
5
A
```

Ejemplo de salida

```
4
6
11
```

Análisis y Solución

Por: Alberto Suxo

El escritor del problema da por hecho lo siguiente:

Un número R en base N es divisible por (N-1) si y solo si la suma de sus dígitos es divisible por (N-1)

Ejemplo:

6893064_{10} es múltiplo de 9 (765986*9=6893064) y la suma de sus dígitos es también múltiplo de 9.(6+8+9+3+0+6+4=36).

Una vez comprendido el hecho en el cual se apoya el problema, pues es simple resolverlo.

1. No nos importa si el número es positivo o negativo (+ o -) puesto que da lo mismo. (Ejemplo: 36 y -36 ambos son divisibles entre 9)
2. Sumamos todos los dígitos (en decimal) de nuestro número R.
3. Hallamos el valor del dígito más alto (x)
4. Verificamos si el resultado de suma es divisible por algún valor desde x hasta 62, de existir, imprimimos dicho valor, de no existir imprimimos "such number is impossible!"

Una consideración importante es:

El tamaño más grande del archivo de entrada puede ser de 32KB, esto quiere decir que, el caso más extremo sería cuando el número R ocupara esos 32KB (32768 Bytes), es decir, que tenga 32768 caracteres, y si los dígitos de este número fueran ‘z’ (el dígito más grande), esto significaría que tendríamos 32768 zetas o 32768 valores 62, y si sumamos los dígitos la variable tendría que soportar 32768 x 62 = 2031616 ≤ este valor entra tranquilamente en una variable de tipo long en C y el int en JAVA.

Código Fuente en C:

```
/* Problema : ¡Un Problema Facil!
 * Lenguaje : ANSI C (versión: 4.0 )
 * Por      : Alberto Suxo
 ************************/

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

long count( char ch ){
   if( isdigit(ch) )
      return ch-'0';
   if( isupper(ch) )
      return ch-'A'+10;
   if( islower(ch) )
      return ch-'a'+36;
   return 0;
}
```

```c
int main(){
   char ch;
   long sum, d, big;

   while( scanf( "%c", &ch )==1 ){
      if( ch=='\n' )
         continue;
      sum = 0;
      big = 1;
      while( ch!='\n' ){
         d = count( ch );
         sum += d;
         if( d > big )
            big = d;
         scanf( "%c", &ch );
      }
      for( ; big<62; big++ )
         if( (sum%big)==0 ){
            printf( "%ld\n", big+1 );
            break;
         }
      if( big==62 ) puts( "such number is impossible!" );
   }
   return 0;
}
```

Una Función Inductivamente-Definida

Considere la función f que esta inductivamente definida para enteros positivos, como sigue:

$$f(1) = 1 \quad (1)$$
$$f(2n) = n \quad (2)$$
$$f(2n+1) = f(n) + f(n+1) \quad (3)$$

Dado un valor entero positivo **n** (mayor o igual a 1), encontrar el valor de $f(n)$.

Entrada

La entrada consiste en una secuencia en valores enteros positivos para **n** seguidos por -1. Los enteros son precedidos y/o seguidos por espacios en blanco (blancos, tabs, y fin de líneas).

Salida

Por cada entero positivo **n**, desplegar el valor de **n** y el valor de $f(n)$. Use la forma mostrara en el ejemplo de salida, y ponga líneas en blanco entre las salidas de cada valor **n**.

Ejemplo de entrada

```
2     53

            153
-1
```

Ejemplo de salida

```
f(2) = 1

f(53) = 27

f(153) = 77
```

Análisis y Solución

Por: Alberto Suxo

Este problema se presenta como uno recursivo:

$$f(1) = 1 \qquad (1)$$
$$f(2n) = n \qquad (2)$$
$$f(2n+1) = f(n) + f(n+1) \qquad (3)$$

Realizando los cambios necesarios podremos comprender mejor esta función.

$2n = k \Rightarrow n = \frac{k}{2}$ reemplazando en (2) tendremos: $f(k) = \frac{k}{2}$

$2n+1 = k \Rightarrow n = \frac{k-1}{2}$ reemplazando en: (3) tendremos $f(k) = f\left(\frac{k-1}{2}\right) + f\left(\frac{k+1}{2}\right)$

$$f(k) = \begin{cases} 1 & k = 1 \\ \frac{k}{2} & k\ es\ par \\ f\left(\frac{k-1}{2}\right) + f\left(\frac{k+1}{2}\right) & k\ es\ impar \end{cases}$$

Para resolver el problema basta con transcribir esta función recursiva.

Código Fuente en C:

```
/* Problema : Una Funcion Inductivamente Definida
 * Lenguaje : ANSI C (version: 4.0 )
 * Por      : Alberto Suxo
 ************************/

#include<stdio.h>

int f( int n ){
   if( n==1 )
      return 1;
   if( n%2==0 )
      return n/2;
   else
      return f( (n-1)/2 ) + f( (n+1)/2 );
}

int main(){
   int n;

   while( 1 ){
      scanf( "%d", &n );
      if( n==-1 )
         break;
```

```
      printf( "f(%d) = %d\n\n", n, f( n ) );
   }
   return 0;
}
```

Código Fuente en JAVA:

```
/* Problema : Una Funcion Inductivamente Definida
 * Lenguaje : JAVA (version: 1.5 )
 * Por      : Alberto Suxo
 *************************/

import java.util.Scanner;

public class B{

   static int f( int n ){
      if( n==1 )
         return 1;
      if( n%2==0 )
         return n/2;
      else
         return f( (n-1)/2 ) + f( (n+1)/2 );
   }

   public static void main( String args[] ){
      int n;
      Scanner in = new Scanner( System.in );

      while( true ){
         n = in.nextInt();
         if( n==-1 )
            break;
         System.out.println( "f(" + n + ") = " + f( n ) );
         System.out.println();
      }
   }
}
```

Nota: después de ejecutar el programa para un rango de datos desde 0 hasta 1000 y verificando las salidas, podemos ver que la solución también se logra con (n+1) div 2.

Es decir, remplacemos las siguientes líneas:

En C: `printf( "f(%d) = %d\n\n", n, f( n ) );`
Por: `printf( "f(%d) = %d\n\n", n, (n+1)/2 );`
En JAVA: `System.out.println( "f(" + n + ") = " + f( n ) );`
Por: `System.out.println( "f(" + n + ") = " + (int)((n+1)/2) );`

¿Que pasa cuando n es par o impar?

Bueno, es una división entera, así que los decimales simplemente se truncan.

PROBLEMAS DE PROGRAMACIÓN DINÁMICA.

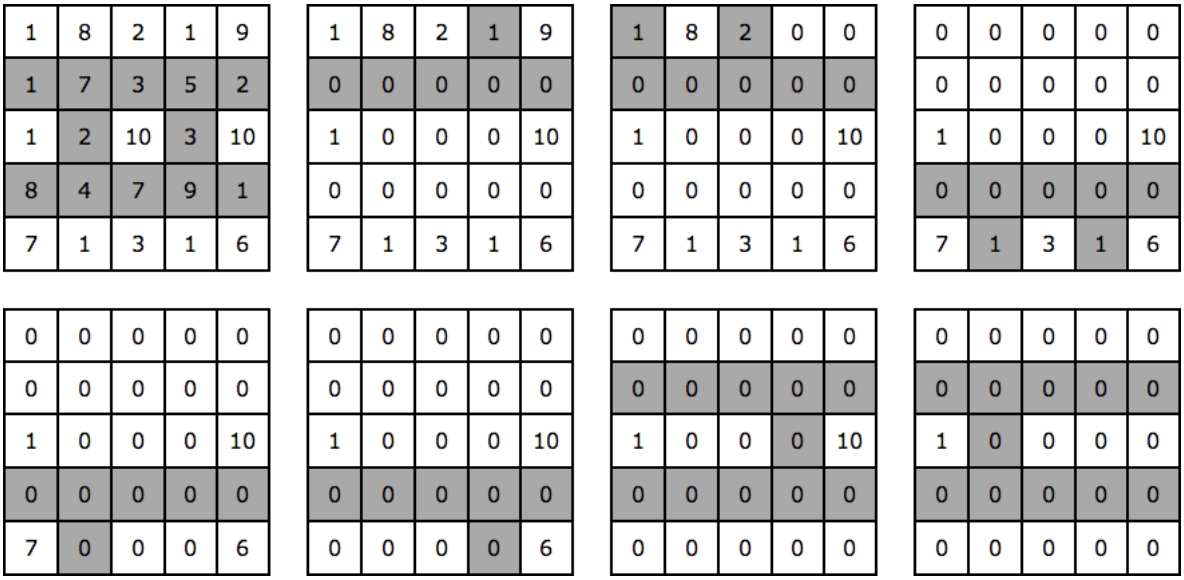
Dulce

El pequeño Charlie es un niño adicto a los dulces. Él es un suscriptor de la revista Todos los Dulces y fue seleccionado para participar en el Concurso Internacional de Elección de Dulces.

En este concurso un número aleatorio de cajas conteniendo dulces están puestas en M filas con N columnas cada una (así que, existe un total de M x N cajas). Cada caja tiene un número indicando cuantos dulces contiene.

El concursante puede elegir una caja (solo una) y obtener todos los dulces que contiene. Pero existe una trampa (siempre hay una trampa): cuando se elige una caja, todas las cajas de las filas inmediatas de arriba y abajo son vaciadas, y también las filas de la izquierda y derecha de la caja elegida. Los concursantes siguen recogiendo una caja hasta que no queden dulces.

La figura de abajo ilustra esto, paso a paso. Cada celda representa una caja y el número de dulces que ésta contiene. En el siguiente paso, la caja seleccionada es encerrada en un círculo y son sombreadas las celdas que representan las cajas que serán vaciadas. Después de ocho pasos, el juego se termina y Charlie recogió 10 + 9 + 8 + 3 + 7 + 6 + 10 + 1 = 54 dulces.



Para pequeños valores de M y N, Charlie puede fácilmente encontrar el máximo número de dulces que puede recoger, pero cuando los números son realmente grandes, él se encuentra completamente perdido. ¿Puedes ayudar a Charlie a maximizar el número de dulces que el puede recoger?

Entrada

La entrada contiene varios casos de prueba. La primera línea de un caso de prueba contiene dos enteros positivos M y N (1 <= M x N <= 105), separados por un solo espacio, indicando el número de filas y columnas respectivamente. Cada una de las siguientes M líneas contiene N enteros separados por un solo espacio, cada una representa el número inicial de dulces en la caja correspondiente. Cada caja tendrá inicialmente mínimo 1 y máximo 103 dulces.

El fin de la entrada es indicado por una línea que contiene dos ceros separados por un solo espacio.

Salida

Para cada caso de prueba en la entrada, tu programa debe imprimir una sola línea, conteniendo un solo valor, el entero indicando el máximo número de dulces que Charlie puede recoger.

Ejemplo de Entrada

```
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
```

Ejemplo de Salida

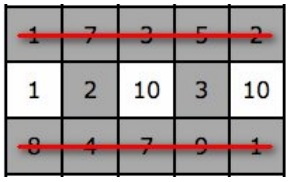
```
54
40
17
```

Análisis y Solución

Por: Jorge Terán

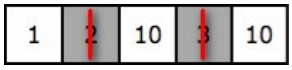
Es posible ver que es el mismo problema para cada una de las filas, que si se tomaran todas las filas a la vez. Ya que al elegir una fila, descartamos sus filas adyacentes.

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



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.



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 $E : e_1, e_2, e_3 ... e_n$, de donde deseamos buscar la respuesta para la subsecuencia $e_1 ... e_n$, 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 vacas 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 esto guardamos las soluciones en un vector `sol[]` de tamaño igual al número de filas más dos vacas.

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

```
int[] sol = new int[M + 2];
for (int i = 0; i < M; i++) {
   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`.

Código Fuente en JAVA:

```
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]);
         }
      }
   }
```

Si realizamos los mismos cálculos el momento de leer podemos crear una versión mucho más simple.

Esta versión se presenta codificada en lenguaje c.

Código Fuente en C:

```
#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);
}
```

Subsecuencias de ADN

Tomás, un científico computacional que trabaja con secuencias de ADN, necesita calcular las más largas subsecuencias comunes de pares de cadenas dados. Considerar un alfabeto Σ de letras y una palabra $w = a_1 a_2 ... a_r$, donde $a_i \in \Sigma$, para $i = 1,2,...,r$. Una subsecuencia de w es una palabra $x = a_{i1} a_{i2} ... a_{is}$ de tal forma que $1 \le i_1 < i_2 < ... < i_s \le r$. La subsecuencia x es un segmento de w si $i_{j+1} = i_j + 1$, para $j = 1,2,...,s-1$. Por ejemplo la palabra ove es un segmento de la palabra lovely, como la palabra loly es una subsecuencia de lovely, pero no un segmento.

Una palabra es una subsecuencia común de dos palabras w_1 y w_2 si es una subsecuencia de cada una de las dos palabras. La mayor subsecuencia común de w_1 y w_2 es una subsecuencia común de w_1 y w_2 teniendo la mayor longitud posible. Por ejemplo, considerando las palabras w_1 = lovxxelyxxxxx y w_2 = xxxxxxxlovely. Las palabras w_3=lovely y w_4 = xxxxxxx, la última de longitud 7, son ambas sobsecuencias comunes de w_1 y w_2. De hecho, w_4 es su mayor subsecuencia común. Nótese que la palabra vacía, de longitud cero, es siempre una subsecuencia común, pero no necesariamente la mayor.

En el caso de Tomás, existe un requerimiento extra: la subsecuencia debe estar formada desde los segmentos comunes teniendo longitud K o mayor. Por ejemplo, si Tomás decide que K = 3, entonces él considera lovely para ser una subsecuencia común aceptable de lovxxelyxxxxx y xxxxxxxlovely, no obstante xxxxxxx, la cual tiene longitud 7 y es también una subsecuencia común, no es aceptable. ¿Puedes ayudar a Tomás?

Entrada

La entrada contiene varios casos de prueba. La primera línea de un caso de prueba contiene un entero K representando la mínima longitud de un segmento común, donde 1 ≤ K ≤ 100. Las siguientes dos líneas contienen cada una, una cadena en minúsculas de un alfabeto regular de 26 letras. La longitud l de cada cadena satisface la desigualdad 1 ≤ l ≤ 103. No existen espacios en ninguna línea de la entrada. El final de la entrada es indicado por una línea conteniendo un cero.

Salida

Para cada caso de prueba en la entrada, tu programa debe imprimir una sola línea, conteniendo la longitud de la mayor subsecuencia formada por segmentos consecutivos de longitud al menos K de ambas cadenas. Si no existe dicha subsecuencia común de longitud mayor a cero, entonces se debe imprimir 0.

Ejemplo de Entrada

```
3
lovxxelyxxxxx
xxxxxxxlovely
1
lovxxelyxxxxx
xxxxxxxlovely
3
lovxxxelxyxxxx
xxxlovelyxxxxxxx
4
lovxxxelyxxx
xxxxxxlovely
0
```

Ejemplo de Salida

```
6
7
10
0
```

Análisis y Solución

Por: Jorge Terán

Analisis del Problema

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 Bioinforámtica busca la subsecuencia más 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 más 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, nótese 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 más, 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.

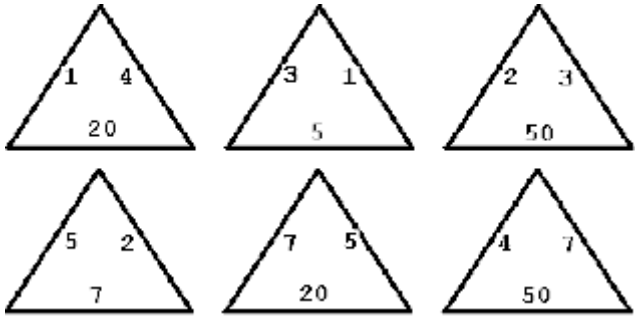
Este problema se puede considerar como de gran complejidad si uno no esta familiarizado con técnicas de programación dinámica.

Código Fuente en JAVA:

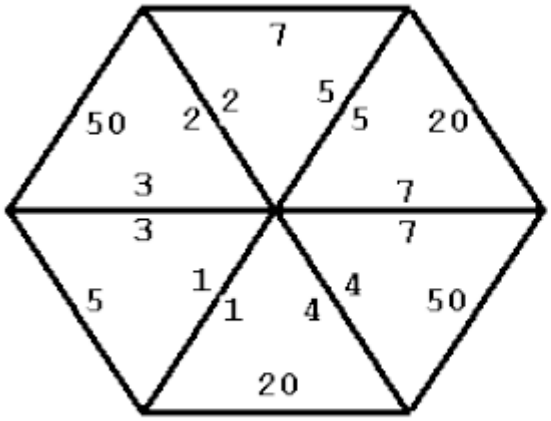
```
import java.util.Scanner;
import static java.lang.Math.max;
class sequence {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String first, second;
        int k;
        while(true){
            k = in.nextInt();
            if(k == 0) break;
            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];
    }
}
```

PROBLEMAS CON BACKTRACKING.

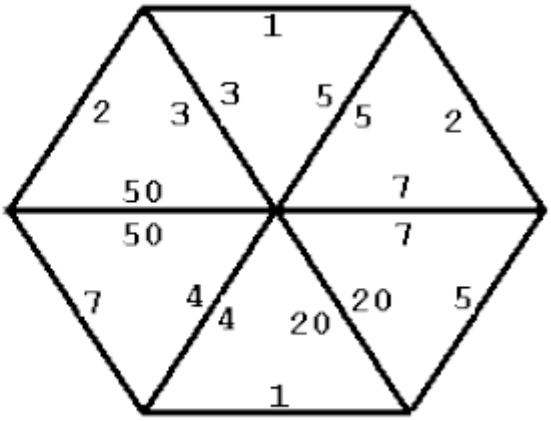
El Juego de Triángulos



En un juego de triángulos usted comienza con seis triángulos numerados en cada eje, como en el ejemplo de arriba. Puede cambiar y rotar los triángulos para formar un hexágono, pero el hexágono es válido solo cuando los ejes comunes entre dos triángulos tienen el mismo número. No se puede voltear ningún triángulo. A continuación se muestran dos hexágonos válidos formados con los seis triángulos anteriores.



20+5+50+7+20+50 = 152



1+7+2+1+2+5 = 18

La puntuación de un hexágono válido es la suma de los números de los seis ejes externos.

Su problema es encontrar la puntuación más alta que se puede lograr con seis triángulos en particular.

La entrada contendrá uno o más sets de datos. Cada set de datos es una secuencia de seis líneas con tres enteros del 1 al 100 separados por un espacio en blanco en cada línea. Cada línea contiene los números en un triángulo en sentido del reloj. Los sets de datos están separados por una línea conteniendo un asterisco. El último set de datos es seguido por una línea que contiene un signo de dólar.

Por cada set datos de entrada, la salida es una línea conteniendo solo la palabra “none” si no hay ningún hexágono válido, o la puntuación más alta si hay un hexágono válido.

Ejemplo de Entrada

```
1 4 20
3 1 5
50 2 3
5 2 7
7 5 20
4 7 50
*
10 1 20
20 2 30
30 3 40
40 4 50
50 5 60
60 6 10
*
10 1 20
20 2 30
30 3 40
40 4 50
50 5 60
10 6 60
$
```

Ejemplo de salida:

```
152
21
none
```

Análisis y Solución

Por: Alberto Suxo

La descripción del problema es bastante simple y concisa, así que estoy seguro que no hay dudas con respecto a las características de problema ni del resultado que se nos pide.

La solución que presento a continuación utiliza la recursividad para resolver el problema, realizando las combinaciones posibles para formar todos los hexágonos válidos, al encontrar un hexágono suma los valores de los vértices externos y los almacena en una variable global (si es mayor a dicha variable), una vez terminados todos los recorridos válidos posibles, pregunta si la variable global almacenó algún valor, de ser verdad, lo imprime, caso contrario imprime el mensaje “none”.

Es cierto que los algoritmos recursivos son lentos, pero en este caso en particular, representa una solución eficiente y veloz.

La función recursiva, realiza un recorrido de todos los triángulos girándolos en sentido del reloj, una vez encontrado un triángulo válido, lo marca como utilizado y vuelve a buscar otro triángulo, cuando termina formando el hexágono suma los vértices exteriores y cuando ya no es posible utilizar los triángulos restantes, pues desmarca el último marcado y busca desde el siguiente.

Código Fuente en C:

```
/* Problema : The Triangle Game
 * Lenguaje : ANSI C (version: 4.0)
 * Por      : Alberto Suxo
 ************************/
#include<stdio.h>
#define P(x) (x)%3

int max, k;
struct triangle{
   int d[3];
   int sw;
} T[6];

void find( int dt, int lvl, int sum ) {
   int i, j;
   if ( lvl==6 ){
      if( (dt==T[0].d[k]) && (sum>max) )
         max = sum;
   }else{
      for( i=1; i<6; i++ )
         if( T[i].sw==1 )
            for( j=0; j<3; j++)
               if( dt == T[i].d[j] ) {
                  T[i].sw = 0;
                  find( T[i].d[P(j+1)], lvl+1, sum+T[i].d[P(j+2)] );
                  T[i].sw = 1;
               }
   }
}
```

```c
int main(){
   char ch;
   int i;
   do{
      for( i=0; i<6; i++ ){
         scanf( "%d %d %d\n", &T[i].d[0], &T[i].d[1], &T[i].d[2] );
         T[i].sw = 1;
      }
      scanf( "%c\n", &ch );
      max = -1;
      /*Los tres posibles giros del primer triangulo*/
      for( k=0; k<3; k++ )
         find( T[0].d[P(k+1)], 1, T[0].d[P(k+2)] );
      if( max<0 )
         printf( "none\n" );
      else
         printf( "%d\n", max );
   }while( ch=='*' );
   return 0;
}
```

Código Fuente en JAVA:

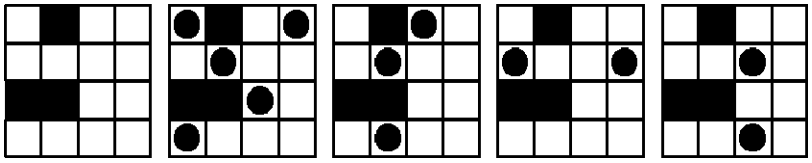
```java
/* Problem A: The Triangle Game
 * Lenguaje : JAVA (version: 1.5)
 * Por      : Alberto Suxo
 *************************/
import java.util.Scanner;

public class A {
   static int max, k;
   static int P(int x) {
      return x % 3;
   }
   /*struct triangle{  int d[3];  int sw;} T[6];*/
   static int[][] T = new int[6][4];

   static void find(int dt, int lvl, int sum) {
      int i, j;
      if ( lvl == 6 ) {
         if ( (dt == T[0][k]) && (sum > max) )
            max = sum;
      } else {
         for ( i = 1; i < 6; i++ )
            if ( T[i][3] == 1 )
               for ( j = 0; j < 3; j++ )
                  if ( dt == T[i][j] ) {
                     T[i][3] = 0;
                     find( T[i][P(j + 1)], lvl+1, sum + T[i][P(j+2)] );
                     T[i][3] = 1;
                  }
      }
   }
```

```
   public static void main( String args[] ) {
      String ch;
      int i;
      Scanner in = new Scanner( System.in );
      do {
         for ( i = 0; i < 6; i++ ) {
            T[i][0] = in.nextInt();
            T[i][1] = in.nextInt();
            T[i][2] = in.nextInt();
            T[i][3] = 1;
         }
         ch = in.next();
         max = -1;
            /* Los tres posibles giros del primer triangulo */
         for ( k = 0; k < 3; k++ )
            find( T[0][P(k + 1)], 1, T[0][P(k + 2)] );
         if ( max < 0 )
            System.out.println( "none" );
         else
            System.out.println( max );
      } while ( ch.equals("*") );
   }
}
```

Torres que no se Ven

En ajedrez, la torre es una pieza que puede moverse cualquier cantidad de cuadros en dirección horizontal o vertical. En este problema consideremos un pequeño tablero de ajedrez, (a lo sumo 4x4) el cual tiene barreras por las cuales las torres no pueden atravesar. El objetivo es poner la mayor cantidad de torres posible de tal forma que dos torres no puedan capturarse entre ellas. Una configuración legal de poner las torres es donde dos torres que estén en la misma fila o columna estén separadas por alguna barrera u obstáculo.

La siguiente imagen muestra cinco dibujos del mismo tablero. El primero es un tablero vació. La segunda y tercera muestran configuraciones legales, y la cuarta y quinta muestran configuraciones ilegales. Para este tablero, el máximo número de torres en una configuración legal es 5; el segundo dibujo muestra una forma de entra solución, pero hay más formas.



Tu trabajo es escribir un programa que, dada una descripción de tablero, calcule el número máximo de torres que pueden ser puestos en el tablero en una configuración legal.

La entrada contiene una o más descripciones de tableros, seguidas por una línea con el número 0 que señala el fin de la entrada. Cada descripción de tablero empieza con una línea con un número entero positivo n que es el tamaño del tablero n que a lo sumo será 4. Las siguientes n líneas describen una fila del tablero, con un '.' Que indica un espacio y 'X' que indica el obstáculo. No hay espacios en la entrada.

Para cada descripción de tablero, despliegue una línea conteniendo el máximo número de torres que pueden ser puestos en el tablero en una configuración legal.

Ejemplo de Entrada

```
4
.X..
....
XX..
....
2
XX
.X
3
.X.
X.X
.X.
3
...
.XX
.XX
4
```

Ejemplo de Salida.

```
5
1
5
2
4
```

```
....
....
....
....
0
```

Análisis y Solución

Por: Alberto Suxo

Este problema se resuelve por backtracking, pero como ha de suponer el lector, al ser las matrices de a lo sumo 4 x 4, también puede ser resuelto por fuerza bruta, presento a continuación una solución que trabaja con backtracking.

La idea es relativamente simple (explicarlo es el problema), bien, supongo que la explicación la haremos de forma recursiva:

Primero, se hace un recorrido de la matriz, una vez se encuentre una zona segura se llama a la función recursiva “f_rec”

La función “f_rec” continúa con el recorrido ya iniciado, una vez que encuentra una zona segura, realiza una copia del tablero en el cual marca todos los posibles recorridos de ataque de la torres, o sea, marca tanto vertical como horizontalmente desde el punto donde se encuentre la torre. Luego llama nuevamente a la función recursiva “f_rec” con el nuevo tablero (la copia realizada).

Esta función recursiva termina cuando termina el recorrido en el tablero. El número de llamadas recursivas representa también el número de torres que pueden ser apostadas en el tablero. Por lo que al final de todos los recorridos se despliega el máximo de los resultantes.

Sugiero al lector, realizar una pequeña prueba de escritorio del programa para comprenderlo mejor (no llevará mucho tiempo, ni muchas hojas ;-b)

Código Fuente en C:

```
/* Problema : Torres que no se Ven
 * Por      : Alberto Suxo
 * Nota     : Bute Force
 ************************/

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

```
int n;
char line[5];
int board[5][5];

int f_rec(int f, int c, int O[5][5], int cnt) {
  int y=f, x=c, i, max=cnt, cnt2;
  int D[5][5];
  while( 1 ) {
    if( x>=n ) {
      y++; x=0;
    }
    if( y>=n )
      return max;
    if( O[y][x] ) {
      memcpy( D, O, sizeof(board) );
      i = x;
      while( D[y][i] )
        D[y][i++]=0;
      i = y+1;
      while( D[i][x] )
        D[i++][x]=0;
      cnt2 = f_rec( y, x+1, D, cnt+1 );
      if( cnt2>max )
        max = cnt2;
    }
    x++;
  }
}

int main() {
  int f, c, max, cnt;
  //freopen("rook.in","r", stdin);
  while( 1 ) {
    scanf( "%d", &n );
    if ( !n )
      break;
    memset( board, 0, sizeof(board) );
    for(f=0; f<n; f++) {
      scanf("%s", line);
      for(c=0; c<n; c++)
        board[f][c] = line[c]=='X'?0:1;
    }
    max = 0;
    for( f=0; f<n; f++ )
      for( c=0; c<n; c++ ) {
        cnt = f_rec( f, c, board, 0 );
        if( cnt>max )
          max = cnt;
      }
    printf( "%d\n", max );
  }
  return 0;
}
```

PROBLEMAS CON GRAFOS.

La Rana Saltadora

La rana vive en un pantano en forma de rejilla de forma rectangular, compuesta de celdas de igual tamaño, algunas de las cuales que son secas, y algunas que son lugares con agua. La Rana vive en una celda seca y puede saltar de una celda seca a otra celda seca en sus paseos por el pantano.

La rana desea visitar a su enamorado que vive en una celda seca en el mismo pantano. Sin embargo la rana es un poco floja y desea gastar el mínimo de energía en sus saltos rumbo a la casa de su cortejo. La Rana conoce cuanta energía gasta en cada uno de sus saltos.

Para cada salto, la rana siempre utiliza la siguiente figura para determinar cuales son los posibles destinos desde la celta donde esta (la celda marca con F), y la correspondiente energía en calorías gastada en el salto.

Ninguna otra celda es alcanzable desde la posición corriente de la rana con un solo salto.

7 6 5 6 7

6 3 2 3 6

5 2 F 2 5

6 3 2 3 6

7 6 5 6 7

Su tarea es la de determinar la cantidad de energía mínima que la rana debe gastar para llegar de su casa a la casa de su enamorado.

Entrada

La entrada contiene varios casos de prueba. La primera línea de un caso de prueba contiene dos enteros, *C* y *R*, indicando el numero de columnas y filas del pantano ($1 \leq C;R \leq 1000$). La segunda línea de los casos de prueba contiene cuatro enteros C_f , R_f , C_t, y R_t, donde (R_f ;C_f) especifican la posición de la casa de la rana y (R_t;C_t) especifican la casa de enamorada donde ($1 \leq C_f$;$C_t \leq C$ y $1 \leq R_f$;$R_t \leq R$). La tercera línea de los casos de prueba contiene un numero entero W ($0 \leq W \leq 1000$) indicando los lugares donde hay agua en el pantano. Cada una de las siguientes W líneas contienen cuatro enteros C_1, R_1, C_2, y R_2 ($1 \leq C_1 \leq C_2 \leq C$ y $1 \leq R_1 \leq R_2 \leq R$) describiendo un lugar acuoso rectangular que abarca coordenadas de las celdas cuyas coordenadas (x; y) son tales que $C_1 \leq x \leq C_2$ y $R_1 \leq y \leq R_2$. El final de los datos se especifica con C = R = 0.

La entrada se debe leer de standard input (teclado).

Salida

Para cada caso de prueba en la entrada, su programa debe producir una línea de salida, conteniendo el mínimo de calorías consumidas por la rana para llegar desde su casa a la casa de su enamorada.

Si no existe un camino su programa debe imprimir impossible.

La salida debe ser standard output(pantalla).

Ejemplo de entrada

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

Ejemplo de salida

```
14
impossible
12
```

Análisis y Solución

Por: Jorge Terán

Analizando el problema

Tomemos como ejemplo el primer caso de los datos de ejemplo. Se indica que se tiene un pantano de 4 por 4 que quedaría representado por la siguiente matriz:

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	(2,4)
(3,1)	(3,2)	(3,3)	(3,4)
(4,1)	(4,2)	(4,3)	(4,4)

La segunda línea indica los lugares de donde parte la rana y a donde debe llegar que son las celdas (1,1) y (4,2). En nuestra matriz serian:

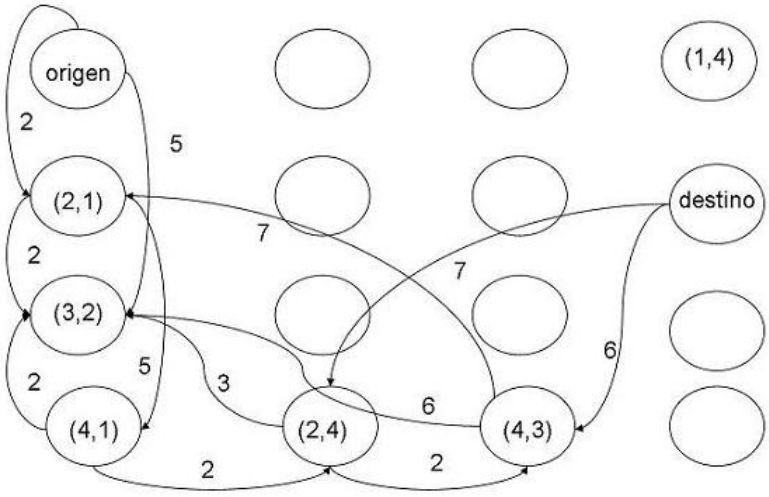
origen	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	destino
(3,1)	(3,2)	(3,3)	(3,4)
(4,1)	(4,2)	(4,3)	(4,4)

Seguidamente se especifican dos regiones de agua la primera (1,2) y (3,3) y la segunda (3,4) y (4,4). En nuestra matriz marcamos las regiones acuosas:

origen	1-	-1	(1,4)
(2,1)	-1	-1	destino
(3,1)	-1	-1	-1
(4,1)	(4,2)	(4,3)	-1

Los posibles lugares donde se puede ir del origen con la tabla de saltos son las celdas (1,2) y (1,3), dado que las otras posiciones están cubiertas de agua. De la posición (1,2) se puede ir a las posiciones (1,3), (1,4) y (4,3).

Continuando con todas las celdas podemos construir el siguiente grafo que representa los posibles saltos de la rana.



Ahora se puede implementar una solución para resolver el problema. Este tipo de soluciones viene caracterizado por el algoritmo de Dijkstra.

Implementando una solución

Para implementar una solución primeramente es necesario definir una matriz que represente el grafo. Colocaremos en esta matriz las áreas acuosas en -1 y en 1 las áreas secas.

Como de la posición actual de la rana se puede uno mover dos lugares hacia arriba, abajo, izquierda y derecha crearemos una matriz con _las y columnas adicionales para evitar controlar si estamos dentro o fuera del pantano. Estas filas las representaremos con -2. El código java para leer los datos de entrada seria el siguiente:

```
public static final int agua = -1;
public static final int libre = 1;
public static final int fueraPantano = -2;
public static final int enteroMaximo = 999999;
public static final int energia[][] = { { 7, 6, 5, 6, 7 },
   { 6, 3, 2, 3, 6 }, { 5, 2, 0, 2, 5 }, { 6, 3, 2, 3, 6 },
   { 7, 6, 5, 6, 7 } };
public static void main(String[] args) {
   int c = 0, r = 0, cs = 0, rs = 0, ct = 0, rt = 0, b;
   int c1, r1, c2, r2;
   int i, j, k;
   int[][] pantano = null;
   int[][] costo = null;
   Scanner in = new Scanner(System.in);
   c = in.nextInt();// leer las dimensiones del pantano
   r = in.nextInt();
   while (c > 0) {
      // crear el pantano y matriz de costos
      pantano = new int[r + 4][c + 4];
      costo = new int[r + 4][c + 4];
      // indicar que la fila 0 y columa 0
      // estan fuera del pantano
      for (i = 0; i < c + 4; i++)
         pantano[0][i] = pantano[1][i] = fueraPantano;
      for (i = 0; i < r + 4; i++)
         pantano[i][0] = pantano[i][1] = fueraPantano;
      for (i = 2; i < c + 4; i++)
         pantano[r + 2][i] = pantano[r + 3][i] = fueraPantano;
      for (i = 2; i < r + 4; i++)
         pantano[i][c + 2] = pantano[i][c + 3] = fueraPantano;
      // Marcar las celdas del pantano como libres
      // y los costos como un entero grande
      for (i = 2; i < r + 2; i++) {
         for (j = 2; j < c + 2; j++) {
            pantano[i][j] = libre;
            costo[i][j] = enteroMaximo;
         }
      }
      cs = in.nextInt();// leer el origen y el destino
      rs = in.nextInt();
      ct = in.nextInt();
```

```
            rt = in.nextInt();
            // leer el numero de zonas acuosas
            b = in.nextInt();
            for (i = 0; i < b; i++) {
               c1 = in.nextInt();// leer las cordenadas de la region
               r1 = in.nextInt();
               c2 = in.nextInt();
               r2 = in.nextInt();
               c1 += 1;
               c2 += 1;
               r1 += 1;
               r2 += 1;
               for (k = r1; k <= r2; k++) {
                  for (j = c1; j <= c2; j++) {
                     pantano[k][j] = agua;
                  }
               }
            }
            cs++;
            rs++;
            ct++;
            rt++;
            // ver(pantano,r, c);
            // ver(costo,r, c);
            dijkstra(pantano, costo, rs, cs, rt, ct);
            if (costo[rt][ct] < enteroMaximo)
               System.out.println(costo[rt][ct]);
            else
               System.out.println("Impossible");
            c = in.nextInt();
            r = in.nextInt();
         }
      }
   }
```

Este código arma una matriz con dos filas al contorno marcadas con -2 que representa la región fuera del pantano. La matriz referente al ejemplo queda como sigue:

-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	1	-1	-1	1	-2	-2
-2	-2	1	-1	-1	1	-2	-2
-2	-2	1	-1	-1	-1	-2	-2
-2	-2	1	1	1	-1	-2	-2
-2	-2	-2	-2	-2	-2	-2	-2
-2	-2	-2	-2	-2	-2	-2	-2

Para el procesamiento del algoritmo de Dijkstra se comienza del origen y se ve a que lugares se puede ir, si se mejora el costo se guarda en una estructura de datos y se anota en la matriz de costos, luego se toma un valor de los guardados y se repite el proceso.

Codificando esto queda como sigue:

```
public static void dijkstra(
   int[][] pantano, int[][] costo,
   int rs, int cs,int rt, int ct) {
   int rv, cv;
   int i, j;
   Nodo filcol;
   PriorityQueue<Nodo> cp = new PriorityQueue<Nodo>();
   costo[rs][cs] = 0;
   rv = rs;
   cv = cs;
   cp.add(new Nodo(0, rs, cs));
   while (!cp.isEmpty()) {
      filcol = cp.remove();
      rv = filcol.fila;
      cv = filcol.col;
      for (i = -2; i < 3; i++) {
         for (j = -2; j < 3; j++) {
            if (pantano[rv + i][cv + j] == libre) {
               if (costo[rv + i][cv + j] >
               (costo[rv][cv] + energia[i + 2][j + 2])) {
                  costo[rv + i][cv + j] = costo[rv][cv]
                                          + energia[i + 2][j + 2];
                  cp.add(new Nodo(costo[rv + i][cv + j],
                        rv + i, cv + j));
               }
            }
         }
      }
   }
}
```

Ahora nos queda escoger una estructura de de datos apropiada. Siempre es deseable comenzar por el nodo que tenga el costo mínimo, dado que de esta forma es posible que se reduzca el tiempo de proceso. Por esto considere apropiado utilizar una estructura de cola de prioridad donde una vez insertados los valores se obtiene al extraer un valor el de menor costo.

En java se implementa definiendo un objeto de clase PriorityQueue, en el programa se utilizo:

```
PriorityQueue<Nodo> cp = new PriorityQueue<Nodo>();
```

La clase nodo se definió como sigue:

```
class Nodo implements Comparable<Nodo> {
   int costo, fila, col;
   public Nodo(int costo, int fila, int col) {
      this.costo = costo;
      this.fila = fila;
      this.col = col;
   }
   public int compareTo(Nodo other) {
      return costo - other.costo;
   }
}
```

Encontrando al Prof. Miguel ...

Pienso que algún día podré encontrar al Profesor Miguel, quien me ha permitido organizar varios concursos. Pero en realidad he fallado en todas mis oportunidades. En el último con la ayuda de un mago he logrado encontrarme con él en la mágica Ciudad de la Esperanza. La ciudad de la esperanza tiene muchas calles. Algunas son bi-direccionales y otros son unidireccionales. Otra característica importante de estas calles es que algunas son para personas menores de treinta años, y el resto son para los otros. Esto es para dar a los menores libertad en sus actividades. Cada calle tiene un cierto tamaño. Dada una descripción de tal ciudad y nuestras posiciones iniciales, debes encontrar el lugar más adecuado donde nos podemos encontrar. El lugar más apropiado es el lugar en donde nuestros esfuerzos para llegar combinados sea el mínimo. Debes asumir que yo tengo 25 años y el Prof. Miguel mas de 40.

Primer encuentro después de cinco años de colaboración (Shanghai, 2005)

Entrada

La entrada contiene varias descripciones de ciudades. Cada descripción de ciudad empieza con un entero N, el cual indica cuantas calles hay. Las siguientes N líneas tienen las descripciones de N calles. La descripción de cada calle consiste en cuatro letras mayúsculas y un entero. La primera letra puede ser 'Y' (indica que es una calle para jóvenes "young"), o 'M' (indica que es una calle para gente de 30 o mas años). La segunda letra puede ser 'U' (indica que la calle es unidireccional) o 'B' (indica que la calle es bidireccional). Las tercera y cuarta letras, X y Y (letras mayúsculas) indican los lugares llamados X y Y de la ciudad que son conectados (in caso de unidireccional significa que es de un solo sentido de x hacia Y) y el último entero no negativo C indica la energía requerida para caminar a través de la calle. Si ambos estamos en el mismo lugar nos encontraremos uno al otro con cero de costo de todas formas. Cada valor de energía es menor que 500.

Después de la descripción de la ciudad, la última línea de cada entrada contiene dos nombre de lugares, que son las posiciones iniciales de mi y del Prof. Miguel respectivamente.

Un valor cero para N indica el fin de entrada.

Salida

Por cada ser de entrada, imprima el mínimo costo de energía y el lugar, que es más adecuado para encontrarnos. Si hay más de un lugar para encontrarnos imprima todos en orden lexicografito en la misma línea, separados por un espacio en blanco. Si no existe tal lugar donde encontrarnos entonces imprima la línea 'You will never meet.'

Ejemplo de Entrada

```
4
Y U A B 4
Y U C A 1
M U D B 6
M B C D 2
A D
2
Y U A B 10
M U C D 20
A D
0
```

Ejemplo de Salida

```
10 B
You will never meet.
```

Análisis y Solución

Por: Alberto Suxo

Encontrando al profesor Miguel es un clásico problema de recorrido de grafos, para resolver este problema yo utilizaré el algoritmo Floyd, (no es la única forma, también pueden hacerlo de otra forma, ejemplo Dijkstra que sería mucho más rápido).

En resumen, el problema consiste en encontrar el punto de reunión entre Shahriar (autor del problema) y el profesor Miguel que requiera del menor esfuerzo posible, en caso de existir varios, pues desplegarlos en orden alfabético, y por último, en caso de no existir dicho lugar pues imprimir “You will never meet.”.

La entrada del problema especifica si es un camino para Shahriar o para Miguel, si es Unidireccional o bi-direccional, origen, destino y la energía requerida.

Y luego, las calles en donde inician Shahriar y Miguel.

La energía requerida es menor a 500, en el caso de tener que ir de A a Z pasando por todas las demás letras, la energía total seria de 500*25 que es 12500, cualquier valor superior a este será nuestro “INFINITO” que en mi caso en particular será 16843009 que en binario es 00000001000000010000000100000001, este valor es apropiado porque yo utilizo la función memset(M, 1, sizeof(M)), que básicamente llena todos los bytes de la matriz M con el valor 1.

Esta matriz M es tridimensional (pero la utilizo como si fueran dos matrices bidimensionales)

```
long M[2][26][26];
```

Después de haber llenado la matriz con el valor de INFINITO, procedemos a llenar la matriz con los costos de cada camino.

```
for( i=0; i<N; i++ ){
   scanf( "%c %c %c %c %ld\n", &P, &D, &X, &Y, &C );
   if( P=='Y' ) ps=0;
   else         ps=1;
   if( D=='U' )
      M[ps][PS(X)][PS(Y)] = C;
   else
      M[ps][PS(X)][PS(Y)] = M[ps][PS(Y)][PS(X)] = C;
}
```

Aquí, cuando P=’Y’ asignamos a ps = 0 que significa que es un camino para Shariar, caso contrario asignamos a ps = 1 que es un camino para el Prof. Miguel. De la misma forma Cuando D=’U’ es unidireccional (de X a Y), caso contrario bidireccional (de X a Y y de Y a X).

Es lógico suponer que de A hasta A el costo de energía debe ser cero, por lo que lo plasmaremos en nuestra matriz.

```
for( i=0; i<26; i++ )
   M[0][i][i] = M[1][i][i] = 0;
```

El trabajo de identificar los caminos más cortos en nuestro grafo lo dejamos al algoritmo Floyd, donde claramente se aprecia la diferencia entre los caminos de Shahriar M[**0**][u][v] y de Miguel M[**1**][u][v].

```
      for( k=0; k<26; k++ )
         for( u=0; u<26; u++ )
            for( v=0; v<26; v++ ){
               M[0][u][v] = MIN( M[0][u][k]+M[0][k][v], M[0][u][v] );
               M[1][u][v] = MIN( M[1][u][k]+M[1][k][v], M[1][u][v] );
            }
```

Al ingresar los orígenes de Shahriar X y Miguel Y, pues simplemente sumo los resultados de la fila X de la matriz de Shahriar con los resultados de la fila Y de la matriz de Miguel en un vector Auxiliar. Si en dicho vector no apareciera un valor menor al INFINITO, es obvio que nunca se encuentran, de existir un valor menor, despliego en orden alfabético todas las posiciones donde el resultado de la suma sea igual a dicho mínimo.

Código Fuente en C:

```
/* Problema : Encontrando al Prof. Miguel ...
 * Lenguaje : ANSI C (version: 4.0)
 * Por      : Alberto Suxo
 *************************/

#include<stdio.h>
#include<memory.h>

#define MIN(a,b) a<b?a:b
#define PS(ch) ch-'A'
#define INFINITE 16843009

int main() {
   int N;
   char P, D, X, Y;
   long C;
   int i, ps, k, u, v;
   long M[2][26][26], S[26], min;

   while( 1 ){
      scanf( "%d\n", &N );
      if( N==0 )
         break;
      memset( M, 1, sizeof(M) ); /*Llena toda la matriz con INFINITE*/

      for( i=0; i<N; i++ ){
         scanf( "%c %c %c %c %ld\n", &P, &D, &X, &Y, &C );
         if( P=='Y' ) ps=0;
         else         ps=1;
         if( D=='U' )
            M[ps][PS(X)][PS(Y)] = C;
         else
            M[ps][PS(X)][PS(Y)] = M[ps][PS(Y)][PS(X)] = C;
      }

      /*por si algun bromista pone: ? ? A A x*/
      for( i=0; i<26; i++ )
         M[0][i][i] = M[1][i][i] = 0;
```

```
      /*Floyd;*/
      for( k=0; k<26; k++ )
         for( u=0; u<26; u++ )
            for( v=0; v<26; v++ ){
               M[0][u][v] = MIN( M[0][u][k]+M[0][k][v], M[0][u][v] );
               M[1][u][v] = MIN( M[1][u][k]+M[1][k][v], M[1][u][v] );
            }

      scanf( "%c %c\n", &X, &Y );
      min = INFINITE;

      for( i=0; i<26; i++ ){
         S[i] = M[0][PS(X)][i] + M[1][PS(Y)][i];
         min = MIN( S[i], min );
      }

      if( min==INFINITE ){
         printf( "You will never meet.\n" );
      }else{
         printf( "%ld", min );
         for( i=0; i<26; i++ )
           if( S[i]==min )
              printf( " %c", i+'A' );
         printf( "\n" );
      }
   }
   return 0;
}
```

Viaje Sin Escalas

David odia esperar en las señales de alto, peligro y avance mientras conduce. Para minimizar esa perdida, él prepara mapas de varias regiones por las que conduce frecuentemente, y calcula el tiempo promedio (en segundos) en cada una de las intersecciones des esas regiones. Él quiere encontrar las rutas entre dos puntos específicos de esas regiones que minimicen el retraso en intersecciones (sin importar la distancia que recorra para evitar las demoras), y solicita tu ayuda para este trabajo.

Entrada

Por cada región, David te da un mapa. En este mapa primero identifica el número de intersecciones, NI. Las regiones jamás tienen más de 10 intersecciones. Las intersecciones de cada región son numeradas secuencialmente, empezando con el número uno (1). Por cada intersección, en turno, la entrada especifica el número de calles que salen de la intersección, y por cada una, el número de intersección a la que llega, y el tiempo que tarda, en segundos, que David calculó para esa intersección. Después de los datos de la última intersección en la región, aparecerá un par de números asociados a las intersecciones que David tiene como inicio y fin de su ruta. La entrara consiste en una secuencia de mapas seguidos por un simple cero (0).

Salida

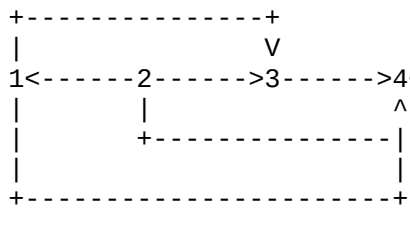
Por cada región, en orden, imprima una línea que contenga el número de región (que también es secuencial empezando en 1), una lista de números de intersección que David encontrará en la ruta de menor desperdicio de tiempo, y el tiempo promedio que tardará en atravesarlo. Un formato aceptable es mostrado en el ejemplo de abajo, pero otros estilos de salida son aceptables.

Notas

1. Siempre habrá una única ruta de retraso mínimo en cada región.
2. Una calle de la intersección I a la intersección J es de un solo sentido. Para representar una calle de doble sentido de I a J, el mapa debe incluir esa ruta desde la intersección J a la intersección I.
3. Jamás habrá más de una ruta directa de la intersección I a la intersección J.

Ejemplo

Suponga que David quiere ir de la intersección 2 a la intersección 4 de la región que se muestra en el siguiente mapa:



Desde	Hasta	Tarda
1	3	3
1	4	6
2	1	2
2	3	7
2	5	6
3	4	5
5	4	7

La entrada y la salida de este ejemplo se muestran en el primer caso de ejemplo de entrada y el resultado esperado en el Ejemplo de salida.

Ejemplo de Entrada

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

2
1   2 5
1   1 6
1 2

7
4   2 5   3 13
    4 8   5 18
2   3 7   6 14
1   6 6
2   3 5   5 9
3   6 2   7 9
    4 6
1   7 2
0
1 7

0
```

Ejemplo de Salida

```
Case 1: Path = 2 1 4; 8 second delay
Case 2: Path = 1 2; 5 second delay
Case 3: Path = 1 2 3 6 7; 20 second delay
```

Análisis y Solución

Por: Alberto Suxo

Este es, un clásico problema de grafos, la descripción es muy simple y bastante comprensible, no deja lugar a duda con respecto al algoritmo que necesitamos para resolver (Dijkstra).

Código Fuente en C:

```c
/* Problema : Viaje Sin Escalas
 * Lenguaje : ANSI C
 * Por      : Alberto Suxo.
 ************************/

#include <stdio.h>

#define INFINITE 100000L

int main() {

  int n, c, origen, destino, tiempo, i, j, k, caso=0;
  long M[11][11];
  long d[11], s[11], p[11], cm[11];
  long min;

  /* freopen("nonstop.in", "r", stdin);   */
  /* freopen("nonstop.out", "w", stdout); */

  while( 1 ) {
    scanf( "%d", &n );
    if( !n )
      break;

    /* Matriz en Infinito */
    for( i=1; i<=n; i++ ) {
      for( j=1; j<=n; j++ ) {
        M[i][j] = INFINITE;
      }
    }

    /* Leedo Datos */
    for( origen=1; origen<=n; origen++ ) {
      scanf( "%d", &c );
      for( i=1; i<=c; i++ ) {
        scanf( "%d %d", &destino, &tiempo );
        M[origen][destino] = tiempo;
      }
    }

    /* Diagonal principal en 0's. */
    for( i=1; i<=n; i++ )
      M[i][i] = 0;
    scanf( "%d %d", &origen, &destino );
```

```
    /* Dijkstra (inicialización) */
    for( i=1; i<=n; i++ ) {
      if( i==origen ) {
        d[i] = 0;
        s[i] = 1;
        p[i] = INFINITE;
      } else {
        d[i] = M[origen][i];
        s[i] = 0;
        p[i] = origen;
      }
    }

    /* Dijkstra; */
    for( i=1; i<n; i++ ) {
      min = INFINITE;
      for( j=1; j<=n; j++ ) {
        if( s[j]==0 && d[j]<min ) {
          min = d[j];
          k = j;
        }
      }
      s[k] = 1;
      for( j=1; j<=n; j++ ) {
        if( s[j]==0 && d[j]>(d[k]+M[k][j]) ) {
          d[j] = d[k] + M[k][j];
          p[j] = k;
        }
      }
    }

    /* Buscar camino; */
    i = 0;
    cm[i] = destino;
    while( cm[i]!=origen ) {
      i++;
      cm[i] = p[cm[i-1]];
    }

    /* Imprimir Solucion */
    printf( "Case %d: Path =", ++caso );
    for( ; i>=0; i-- )
      printf( " %d", cm[i] );
    printf( "; %ld second delay\n", d[destino] );

  }
  return 0;
}
```