PROBLEMAS RESUELTOS

Jorge Terán Pomier
Docente de programación
teranj@acm.org

Alberto Suxo Riveros
Docente de programación
albertosuxo@hotmail.com
alberto_suxo@acm.org

La Paz – Bolivia, Noviembre 2007

Indice

PROBLEMAS RESUELTOS

Impares o Pares

Tipos de archivos aceptados: odd.c, odd.cpp, odd.java

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.

Input

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).

Output
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
```

Salida para el ejemplo de Entrada
```
0
3
```

Analizando el problema

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.

Implementando una solución

La solución completa del problema es la siguiente:

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

public class Odds {
   /**
    * J. Teran requieres jdk 1.5
    */
   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++;
         }
      }
      if ( Juan > Maria )
         System.out.println(Juan – Maria);
      else
         System.out.println(Maria - Juan);
   }
}
```

Lotería de fin de semana

Tipos de archivos aceptados: lottery.c, lottery.cpp, lottery.java

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.

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

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.

La salida debe ser standard output(pantalla).

Ejemplo de entrada
```
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
```

Ejemplo de salida
```
1
1 2 3 4
```

Analizando el problema

Si revisamos el problema vemos que lo que 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]++;
```

Implementando una solución

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();
         }
      }
}
```

Ejercicios

- Modificar el programa para imprimir los números que más han salido.
- Modificar el programa para el caso en que en lugar de números se extraen letras.
- Determinar si hay dos números que hayan salido la misma cantidad de veces.

Bubble Mapas

Tipos de archivos aceptados: maps.c, maps.cpp, maps.java

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 ($2 \leq C \leq 5000$), 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
```

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.

Implementando una solución

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;
        }
    }
}
```

Ejercicios

- Al subir en uno el nivel del zoom, determinar a que región pertenece.
- Determinar número de niveles de zoom con los que se llega a la región.
- Escribir las regiones que son diagonales a la región dada.

Solución 2.

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.

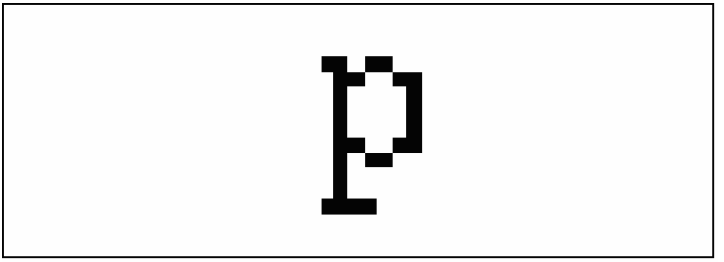
¿Qué es lo que nos pide?

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>”.

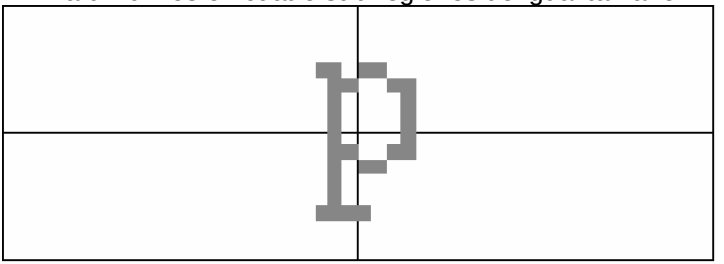
¿Cómo lo haremos?

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

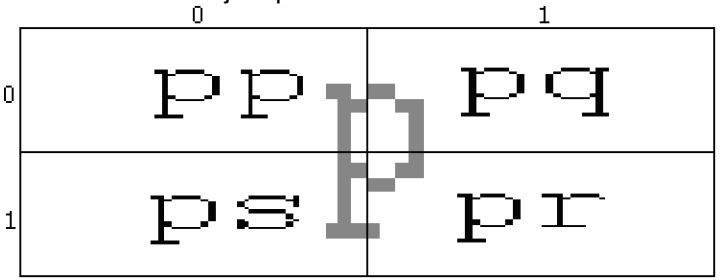
Tenemos un área (región) inicial llamada p.



La dividimos en cuatro sub-regiones de igual tamaño:

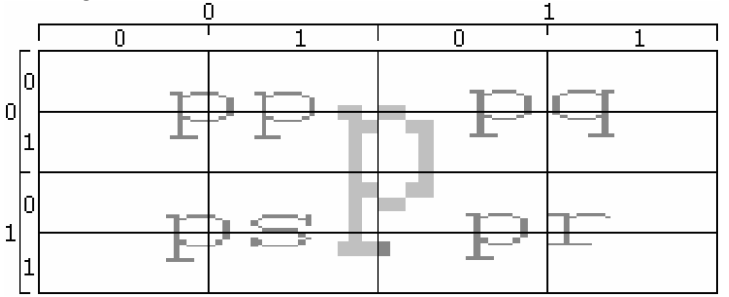


Y concatenamos con p, q, r y s en sentido de las manecillas del reloj respectivamente:

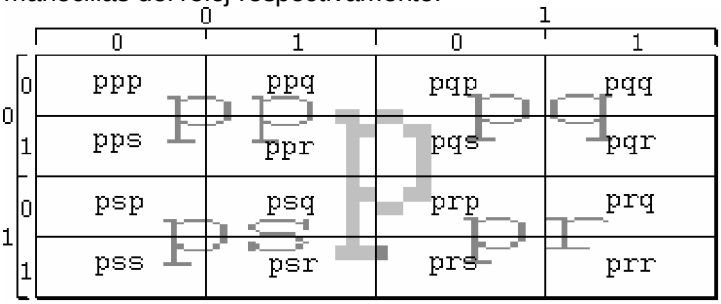


El resultado es un área (matriz) de 2x2.

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



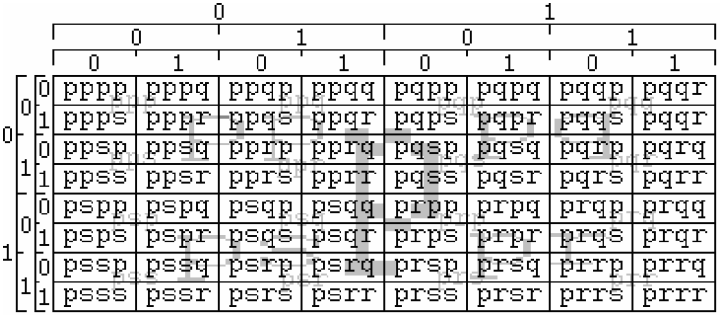
Y concatenamos con p, q, r y s en sentido de las manecillas del reloj respectivamente:



		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 propias 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: mpps, en una de las imágenes 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) =$(00,00)_b$ = mppp
abajo = (x, y+1) = (0, 2) = $(00,10)_b$ = mpsp
izquierda = (x-1, y) = (-1, 1) = fuera de la matriz = <no-map>
derecha = (x+1, y) = (1, 1) = $(01,01)_b$ = mppr

Y como resultado debemos desplegar:
`mppp mpsp <no-map> mppr`

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

```
/* Problem  : Bubble Maps
 * Language : JAVA (version: 1.5 )
 * By       : 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) );
      }
   }
}
```

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. La 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
```

Solución:

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

Y esto mismo se puede hacer para el problema planteado.

En lo personal (Alberto), me disgusta utilizar tantos `if`'s anidados, así que propongo la siguiente solución que utiliza un solo `if` dentro del `while`. La clave de este algoritmo está en que recorre un vector Vn en donde están

almacenados los posibles valores de **x** (N en el ejemplo de arriba), en caso de ser `x>=Vn[i]` imprime la cadena correspondiente y decrementa `x` en `Vn[i]`, caso contrario, incrementa la variable `i` para que apunte al siguiente elemento del vector `Vn`.

Esta solución puede ser planteada en clase para demostrar que siempre hay una forma más fácil y rápida de hacer las cosas.

Código fuente en C y JAVA:

```
/* Problem  : Numeros Romanos
 * Language : ANSI C (version 4.0)
 * By       : 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;
}
```

```java
/* Problem  : Numeros Romanos
 * Language : JAVA (version: 1.5 )
 * By       : 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();
      }
   }
}
```

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
```

Solución

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 y JAVA:

```
/* Problem  : Cara o Cruz
 * Language : ANSI C (version: 4.0 )
 * By       : 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;
}

/* Problem  : Cara o Cruz
 * Language : JAVA (version: 1.5 )
 * By       : 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" );
      }
   }
}
```

¿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.
```

Solución

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 y traducción a JAVA:

```
/* Problem  : Easier Done Than Said?
 * Language : ANSI C (version: 4.0)
 * By       : 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;
}

/* Problem  : Easy Done Than Said?
 * Language : JAVA (version: 1.5)
 * By       : 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.

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) |
@---------------------------------------------@
```

Solución

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).

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 y su traducción en JAVA:

```
/* Problem  : Instruens Fabulam
 * Language : ANSI C (version: 4.0)
 * By       : 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;
}


/* Problem  : Instruens Fabulam
 * Language : JAVA (version: 1.5)
 * By       : Alberto Suxo
 *************************/

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

public class Fabulam{

   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('@','-');
         }
      }
   }
```

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.

R	Y	G	P	B	R	Y	G	B	R	P	O	P

Start Finish

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

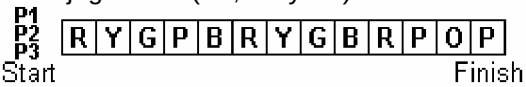
```
Player 1 won after 7 cards.
Player 2 won after 4 cards.
No player won after 6 cards.
```

Solución

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.

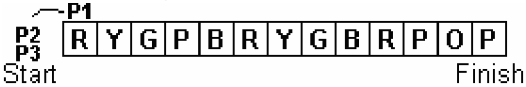
Este es nuestro tablero, y al inicio se encuentran nuestros tres jugadores (P1, P2 y P3)



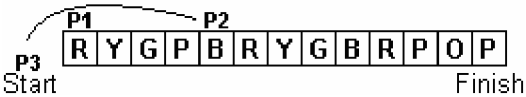
También sabemos que las cartas de nuestro deck saldrán en el siguiente orden: R, B, GG, Y, P, B, P, RR

Simulemos el juego de manera gráfica:

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

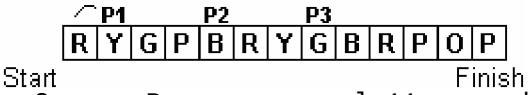



```
Jugador 2 saca B, se mueve al 5to cuadrado
```

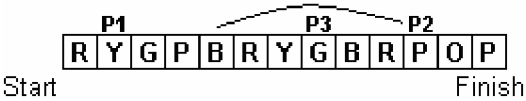



```
Jugador 3 saca GG, se mueve al 8vo cuadrado
```


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

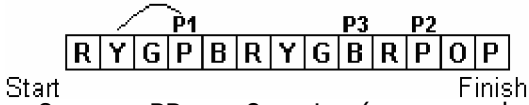



```
Jugador 2 saca P, se mueve al 11vo cuadrado
```



Jugador 3 saca B, se mueve al 9no cuadrado

Jugador 1 saca P, se mueve al 4to cuadrado



Jugador 2 saca RR, ¡Gano! (no mas 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 y JAVA:

```c
/* Problem  : Colorville
 * Language : ANSI C (version: 4.0)
 * By       : 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;
}

/* Problem  : Colorville
 * Language : JAVA (version: 1.5)
 * By       : 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.");
      }
   }
}
```

Falling Leaves

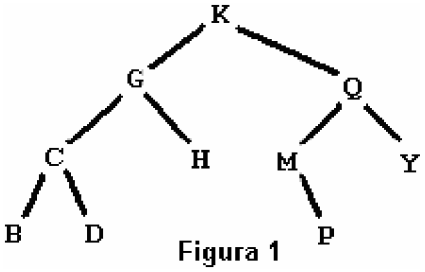


Figura 1

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.

PROBLEMAS RESUELTOS

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:

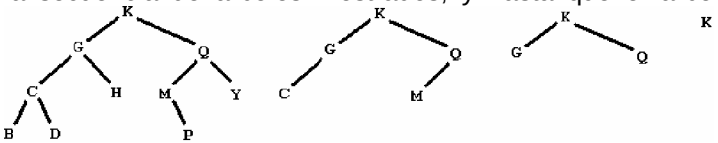
1. Los datos raíz vienen después en el alfabeto que todos los datos en los nodos en el subárbol izquierdo.
2. Los datos raíz vienen antes en el alfabeto que todos los datos en los nodos en el subárbol derecho.

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
```

Solución:

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 terminó con * hay más sets */
         hasNext = 1;      /* de datos.*/
   }else{
      find();
      if( root ){
    /* Si hay raíz, inserta todos los datos
de la línea en el árbol con raíz root */
         for( i=0; line[i]; i++ ){
            insert( root, line[i] );
         }
      }else{ /* Si no hay raiz, la raiz será
la letra de la última línea */
         root = line[0];
      }
   }
}
```

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:

```
/* Problem  : Falling Leaves
 * Language : ANSI C (version: 4.0)
 * By       : 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;
}
```

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
```

Solución

Este problema se presenta como uno recursivo por la siguiente función:

$$f(1) = 1 \qquad (1)$$

$$f(2n) = n \qquad (2)$$

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

Realizando los cambios necesarios podremos comprender mejor esta función.

Empecemos con: $2n = k \Rightarrow n = \frac{k}{2}$, reemplazándolo en (2) tendremos: $f(k) = \frac{k}{2}$

Y también con: $2n+1 = k \Rightarrow n = \frac{k-1}{2}$, reemplazándolo en: (3) tendremos $f(k) = f\left(\frac{k-1}{2}\right) + f\left(\frac{k+1}{2}\right)$

Por lo que nuestra función quedaría traducida de la siguiente forma:

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

Para resolver el problema basta con transcribir esta función recursiva.

Código fuente en C y JAVA:

```
/* Problem  : Una Funcion Inductivamente
Definida
```

```
 * Language : ANSI C (version: 4.0 )
 * By       : 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;
}

/* Problem  : Una Funcion Inductivamente
Definida
 * Language : JAVA (version: 1.5 )
 * By       : 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.

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 *mayo 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
```

Solución

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.

En la función `long SumDig( long Ns );` solo se suman los dígitos del número Ns. También se debe ter muy en cuenta que el resultado de SumDig (X) es igual a X cuando X<=9, si no se considerara este caso no enfrentaríamos a un bucle sin fin.

Código fuente en C y Java:

```
/* Problem  : Raiz Digital Prima
 * Language : ANSI C (version: 4.0 )
 * By       : 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, 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;
}
```

```
/* Problem  : Raiz Digital Prima
 * Language : JAVA (version: 1.5 )
 * By       : 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, 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();
         }
      }
   }
```

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:

a) Al mismo tiempo, solo miembros de un grupo pueden rentar el hotel.
b) Cada grupo llega en la mañana de un día y salen al anochecer de otro día.
c) Otro grupo llega en la mañana siguiente después de que un grupo ha abandonado el hotel.
d) 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.
e) 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
```

Solución

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

```
3 10 => 5
3 14 => 6
          10  14
          |   |
333444455555666666777777888888889
```

Esta serie sigue la misma secuencia que la siguiente serie: `122333444455555666666777777888888`

Para el segundo ejemplo: 3 14 => 6 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.

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

$$0\le 2k+k^2-k-S\cdot(S-1)$$

$$k^2+2k-k-S\cdot(S-1)-2D\ge 0$$

$$k^2+k-\left[2D+S\cdot(S-1)\right]\ge 0$$

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

$$k \geq \sqrt{\frac{1}{4} + [2D + S \cdot (S-1)]} - \frac{1}{2}$$

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

Código fuente en C y Java:

```c
/* Problem :Hotel con Habitaciones Infinitas
 * Language : ANSI C (version: 4.0 )
 * By       : 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;
}
```

```java
/* Problem :Hotel con Habitaciones Infinitas
 * Language : JAVA (version: 1.5 )
 * By       : Alberto Suxo
 *************************/
import java.util.Scanner;

public class Hotel{
   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));
      }
   }
}
```

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
```

Solución

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:

$$x = 2 \cdot v_0 \cdot t$$

Bueno, aquí va el código fuente en C y JAVA:

```c
/* Problem  : Regreso a Fisica de Secundaria
 * Language : ANSI C (versión: 4.0 )
 * By       : 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;
}
```

```java
/* Problem  : Regreso a Fisica de Secundaria
 * Language : JAVA (versión: 4.0 )
 * By       : 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.

¡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

Solución

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:

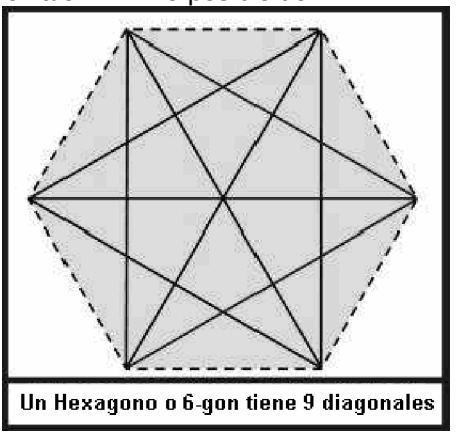
El tamaño más grande del archivo de entrada puede ser de 32KB, esto quiere decir que, el caso más extremo sería cuando el número R ocupara esos 32KB (32768 Bytes), es decir, que tenga 32768 caracteres, y si los dígitos de este número fueran ‘z’ (el dígito más grande), esto significaría que tendríamos 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.

Bueno, aquí va el código fuente en C:

```c
/* Problem  : ¡Un Problema Facil!
 * Language : ANSI C (versión: 4.0 )
 * By       : 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;
}

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;
}
```

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
```

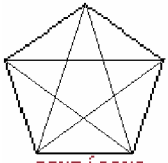
Solución

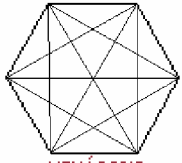
El primer párrafo dice:

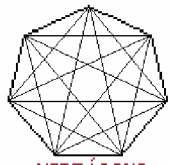
El número de diagonales de un n-gon no es menor que N. ¿Cuál es el mínimo valor posible de n?

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:



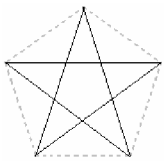


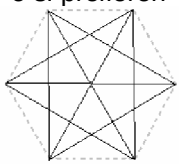


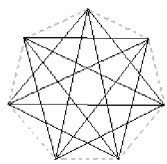
Para el pentágono (5-gon) Si suponemos que cada punta es una persona, que las personas llegan una detrás de otra, y cada persona que llega saluda a los anteriores, entonces, la 1ra no saluda a nadie, la 2da saluda a 1 persona, la 3ra saluda a 2 personas, la 4ta saluda a 3 personas y la 5ta saluda a 4 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.

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







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

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

Bueno, aquí va el código fuente en C y JAVA:

```
/* Problem  : Diagonales
 * Language : ANSI C (version: 4.0)
 * By       : 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;
}
```

```
/* Problem  : Diagonales
 * Language : JAVA (version: 1.5)
 * By       : 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++;
      }
   }
}
```

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
```

Solución:

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 busqueda binaria.

Primero: Criba de eratostenes para encontrar todos los número primos desde 2 hasta 10^6.

```
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:

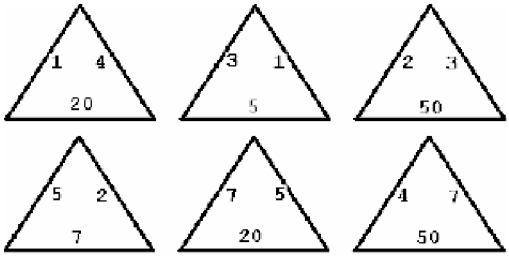
```
Para low:  lowPs  = (-1)*(lowPs+1);
Para high: highPs = (-1)*(highPs+2);
```

Código fuente en JAVA:

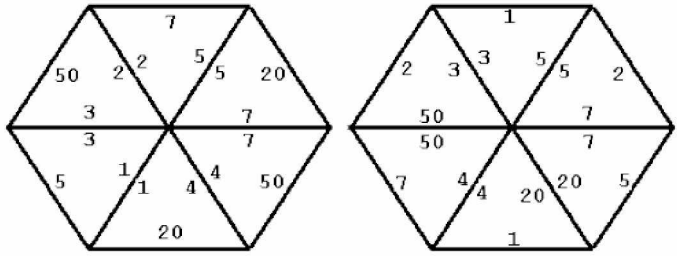
```
/* Problem  : Numero Casi Primos
 * Language : JAVA (version: 1.5)
 * By       : Alberto Suxo
 ************************/
import java.util.Arrays;
import java.util.Scanner;

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);
      }
   }
}
```

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.



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
```

Solución

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 y JAVA:

```
/* Problem A: The Triangle Game
 * Language : ANSI C (version: 4.0)
 * By       : 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;
                  }
      }
}

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;
      for( k=0; k<3; k++ )
        /*Los tres posibles giros del
          primer triangulo*/
         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;
}

/* Problem A: The Triangle Game
 * Language : JAVA (version: 1.5)
 * By       : Alberto Suxo
 *************************/
```

```
import java.util.Scanner;

public class A {

   static int max, k;

   static int P(int x) {
      return x % 3;
   }

   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;
            for ( k = 0; k < 3; k++ )
               /* Los tres posibles giros del
                  primer triangulo */
               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("*") );
   }
}
```

La Rana Saltadora

Tipos de archivos aceptados: frog.c, frog.cpp, frog.java

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
```

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)

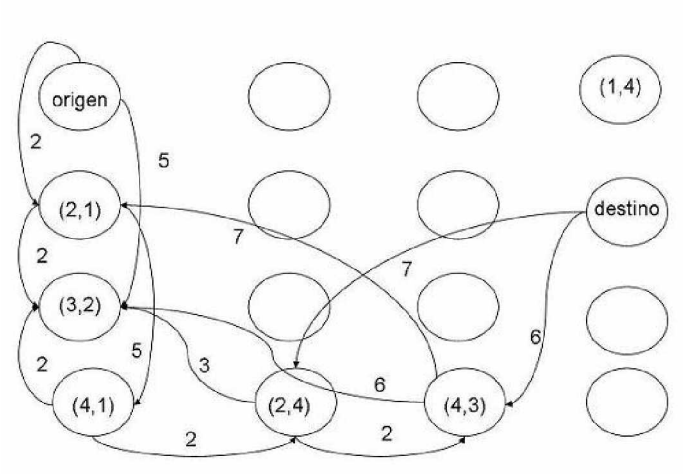
Seguidamente se especifican dos regiones de agua la primera (1,2) y (3,3) y la segunda (3,4) y (4,4). En nuestra matriz marcamos las regiones acuosas:

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;
      int 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);
      // leer las dimensiones del pantano
      c = in.nextInt();
      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;
            }
         }
         // leer el origen y el destino
         cs = in.nextInt();
         rs = in.nextInt();
         ct = in.nextInt();
         rt = in.nextInt();
         // leer el numero de zonas acuosas
         b = in.nextInt();
         for (i = 0; i < b; i++) {
         // leer las cordenadas de la region
            c1 = in.nextInt();
            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;
   }
}
```

Ejercicios

1. Probar la solución presentada con diferentes tipos de estructuras de datos, pilas, vector, etc.
2. Modificar el programa para decir cuantos caminos diferentes existen, que den el mismo recorrido mínimo.
3. Modificar el programa para listar todos los recorridos mínimos.
4. Hallar el camino de máximo esfuerzo.

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.
```

Solución

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 00000001 00000001 00000001 00000001, este valor es apropiado porque yo utilizo la función memset(M, 1, sizeof(M)), que llena todos los bytes de la matriz M con el valor 1.

Utilizo dos MACROS:

```
#define MIN(a,b) a<b?a:b
```

Este encuentra el menor de números.

```
#define PS(ch) ch-'A'
```

Este me retorna la posición de un caracter en mi matriz en donde A es 0, B es 1, C es 2, etc.

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.

A continuación el código fuente en C:

```
/* Problem H: Meeting Prof. Miguel ...
 * Language : ANSI C (version: 4.0)
 * By       : 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;
      /*Llena toda la matriz con INFINITE*/
      memset( M, 1, sizeof(M) );
      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;
      }
      for( i=0; i<26; i++ )
         /*por si alguien : ? ? A A ?*/
         M[0][i][i] = M[1][i][i] = 0;
```

```
     /*Algoritmo 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;
}
```