Ejemplos de Soluciones del Torneo Suamericano ICPC 2006

Lic. Jorge Teran M.Sc.
email: teranj@acm.org
Universidad Mayor de San Andrés
La Paz - Bolivia

April 22, 2007

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 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 and 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 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.

Input

La entrada contiene varios casos de prueba. La primera linea contiene un entero representando el numero de casos de prueba. La Primera linea contiene un entero N indicando el numero de

casos de prueba. Cada una de las siguientes N lineas 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).

Output

Para cada caso en la entrada su programa debe producir una linea 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 linea en blanco entre dos nombres consecutivos.

La salida debe ser standard output(pantalla).

Ejemplo de la entrada	Salida para el ejemplo de entrada
2 mrspr mps	mrspq mrssq mrsps mrsqs mpp msp <none> mpr

Analizando el problema

Para analizar el problema realizaremos una expasión de los datos de de los ejemplos. Comencemos con *mps*. La region m se divide en `p,q,r` y `s`: Cada región a su ves se divide en cuatro

mp	mq
ms	mr

regiones formando: Analizando la región *mps* se ve claramente que tiene un vecino arriba *mpp*,

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

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

```
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

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

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 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 linea 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 linea 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 rna 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 linea 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 lineas 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 linea 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	**Salida para el 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`
	`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 linea 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:

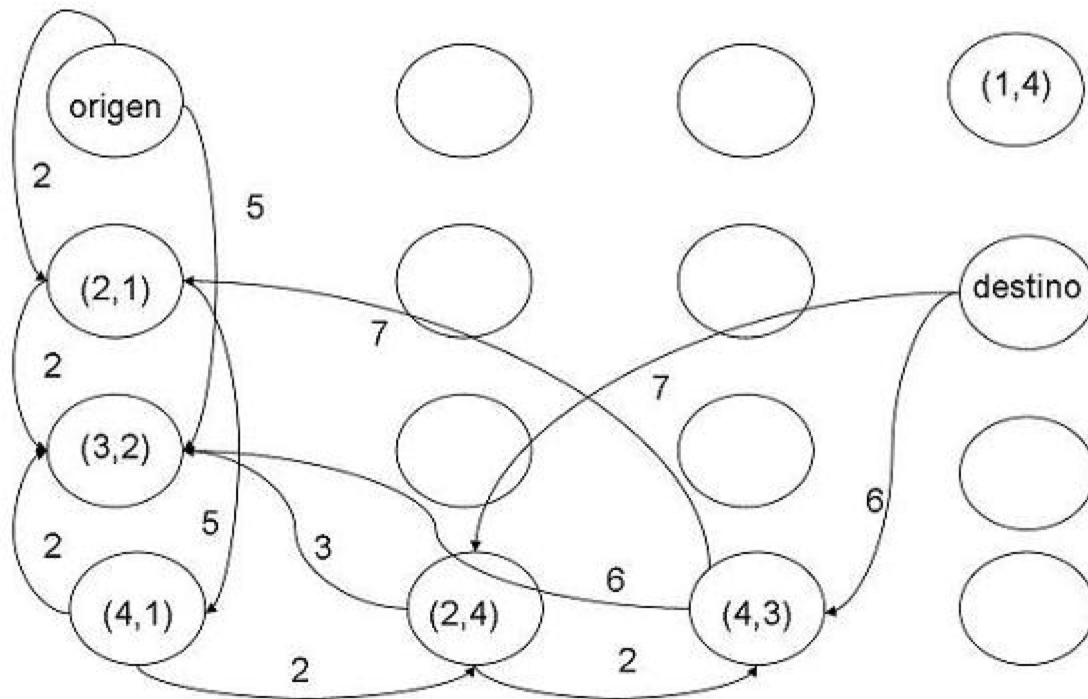
origen	(1,2)	(1,3)	(1,4)
(2,1)	(2,2)	(2,3)	destino
(3,1)	(3,2)	(3,3)	(3,4)
(4,1)	(4,2)	(4,3)	(4,4)

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 filas y columnas adicionales para evitar controlar si estamos dentro o fuera del pantano. Estas filas las representaremos con -2. El código java para leer los datos de entrada seria el siguiente:

```
public static final int agua = -1;

public static final int libre = 1;

public static final int fueraPantano = -2;

public static final int enteroMaximo = 999999;

public static final int energia[][] = { { 7, 6, 5, 6, 7 },
```

```
        { 6, 3, 2, 3, 6 }, { 5, 2, 0, 2, 5 }, { 6, 3, 2, 3, 6 },
        { 7, 6, 5, 6, 7 } };

    public static void main(String[] args) {
        int c = 0, r = 0, cs = 0, rs = 0, ct = 0, rt = 0, b;
        int c1, r1, c2, r2;
        int i, j, k;
        int[][] pantano = null;
        int[][] costo = null;
        Scanner in = new Scanner(System.in);

        // 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 caminos de máximo esfuerzo

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 $C = 6yK = 49$).

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 debe ser leída como entrada estándar.

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 escrita como salida estándar.

La salida debe ser standard output(pantalla).

Sample input	**Output for the sample input**
5 4 6 6 2 3 4 3 4 6 5 2 3 6 5 4 5 2 6 2 3 6 4 4 3 4 3 2 1 2 1 4 4 3 2 1 4 3 0 0 0	1 1 2 3 4

Analizando el problema

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

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

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

Para realizar esta cuenta se puede utilizar el siguiente código

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

Implementando una solución

La solucion 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 el en lugar de números se extraen letras.
- Determinar si hay dos números que hayan salido la misma cantidad de veces.

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 resolvera 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 numero 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 colocadasen 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 linea de la prueba consiste en un entero N que representa el numero de juegos ($1 \leq N \leq 100$). La segunda linea 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 numero 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	**Salida para el 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	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 umero par puede escribirce como $2n$ y todo numero impar como $2n + 1$. de donde se puede fácilmente deducir la tabla anterior.

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

Contando los casos tenemos:

	Pares	Impares
Miaría	2	1
Juan	1	2

El mínimo numero 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++;
     }
        }
```

```
        System.out.println(Juan>Maria?Juan-Maria:Maria-Juan);
    }
}
```