

# El Concurso Mundial de Programación de la ACM

Alfredo Paz-Valderrama <sup>1</sup>

<sup>1</sup>Sociedad Peruana de Computación

12 de octubre de 2008

# Contenidos

- 1 Antecedentes y Contexto
- 2 El ACM-ICPC
  - Objetivo
  - Descripción
  - Los problemas
- 3 Problema H: He is offside!
  - Enunciado
  - Solución
- 4 Conclusión
  - Fotos
  - Resumen

# ACM

## Association for Computing Machinery

- Es la sociedad de computación más antigua del mundo (1947).
- Auspicia eventos relacionados al area de computación en todo el mundo.
- Cualquier Universidad puede contar con un “capítulo” local.
- El portal brinda acceso a una biblioteca virtual de millones de artículos.
- Publicar en un evento ACM, debería ser algo deseable para cualquier persona dedicada a la investigación en computación.



# ACM-ICPC

## Objetivo

El ACM *International Collegiate Programming Contest* (ICPC) brinda a los estudiantes universitarios, una oportunidad para mejorar sus conocimientos y habilidades en la resolución de problemas computacionales, además de la posibilidad de compararse con otros estudiantes de todo el mundo.



**acm** International Collegiate  
Programming Contest

**IBM** | event  
sponsor

# ACM-ICPC

## Descripción

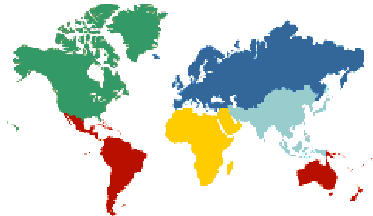
- El ACM-ICPC tiene dos etapas:
  - **La final** , el 21 de Abril del 2009 en el *The Royal Institute of Technology*, Estocolmo-Suecia.
  - **Las eliminatorias regionales** , el 15 de Noviembre de manera simultánea en todo sudamerica.
- Es el Concurso Mundial de Programación más prestigioso (desde 1977).
- Es un evento ACM auspiciado por IBM.
- Cualquier Universidad o Facultad puede inscribir equipos en el concurso.
- El concurso de es exclusivo para Facultades de computación.

# ACM-ICPC

## Organización-Mundial

- El concurso divide al planeta en 6 regiones:
  - 1 Norteamérica
  - 2 Europa y la República Rusa
  - 3 Asia
  - 4 África y el Medio Oriente
  - 5 El Pacífico Sur
  - 6 Latinoamérica

Regional Contest Info Finder...



# ACM-ICPC

## Organización-Regional

La región de latinoamérica tiene 4 sub-regiones:

- 1 Mexico y America Central
- 2 Sudamérica/Brasil
- 3 Sudamérica/Norte.
- 4 Sudamérica/Sur: Argentina, Bolivia, Chile, Paraguay, Perú y Uruguay.



# ACM-ICPC

Región: Sudamérica/Sur

Existen 4 sedes, que coordinan la realización simultanea de las eliminatorias, para este 15 de Noviembre.

- Argentina
- Bolivia
- Chile
- Peru

# ACM-ICPC

## Los equipos

- Cada Equipo está conformado por 3 estudiantes.
- Los equipos pueden representar a la Universidad o al Departamento al que pertenecen.
- Se recomiendan que al menos uno de los integrantes tenga conocimientos de inglés y al menos puedan leer un texto con la ayuda de un diccionario.
- No todo el equipo tiene que estar conformado por personas de computación, puede ser conveniente incluir a matemáticos, electrónicos o estudiantes de ciencias.
- El equipo necesita un entrenador (coach) que será su delegado para el concurso.
- Típicamente el entrenador es el profesor del curso de Algoritmos.

# ACM-ICPC

## El concurso

- El concurso dura 5 horas.
- Los Coach pueden observar el desarrollo del concurso en ambientes especiales.
- Cada equipo dispone de una sola computadora.
- Los equipos tienen posibilidad de imprimir sus programas.
- Durante el concurso, hasta una hora antes de su finalización, se dan informes preliminares sobre el avance del concurso. Estos informes generales, pueden ser conocidos por los mismos participantes.
- El informe final recién se hace publico en la cena de gala, en la clausura del evento.

# Los Problemas

## Descripción

- Se proponen de 5 a 9 problemas de tipo algorítmico, a resolver en 5 horas.
- Cada problema tiene una descripción además de un ejemplo de las entradas (*INPUT*) y salidas (*OUTPUT*) esperadas.
- Al menos dos de ellos son resolubles por alguien de primer año.
- Otros dos por alguien de segundo o tercer año (Ya llevó el curso de algoritmos <http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/CourseHome/index.htm>).
- El resto de problemas hacen la diferencia.

# Los Problemas

## Las Soluciones

- Una solución consiste del código fuente de un programa que cumple, exactamente, con el formato de la salida pedido.
- Los jurados sólo evalúan las salidas producidas por el programa, al ingresarle un archivo de pruebas.
- El jurado entonces, proporciona un escueto veredicto:
  - Problema Correcto.
  - Tiempo Excedido.
  - Error en el formato.
  - Error en la compilación.
  - Error en el formato de la respuesta.

## Criterios de calificación

- Existen dos criterios, en orden de prioridad:
  - 1 El que resuelve más problemas.
  - 2 El que lo hizo en menos tiempo.
- Si un equipo se equivoca en la solución de un problema, se le castiga agregándole una cantidad de tiempo, al tiempo en que obtuvo la respuesta correcta.
- El castigo no tiene efecto si nunca se entrega el programa correcto.

## Contexto

### Problema H: He is offside!

Hemisphere Network es la cadena de televisión más grande en Tumbolia, un pequeño país localizado al este de Sudamerica (o sur de América del Este). El deporte más popular en Tumbolia es el futbol, por supuesto; muchos juegos son transmitidos en señal abierta, cada semana en Tumbolia.

## El centro del problema

La cadena recibe muchos pedidos para repetir jugadas dudosas; usualmente, esto ocurre cuando un jugador es encontrado en posición adelantada (*offside*). Un jugador atacante está en *offside*, si él está más cerca a la línea de meta que el segundo último oponente. Un jugador no está en *offside* si:

- él está en línea con el segundo último oponente o
- él está en línea con al menos dos oponentes.

## La conclusión

A través del uso de la tecnología de gráficos en computadora, Hemisphere Network puede tomar una imagen del campo y determinar la distancia de los jugadores a la línea de méta, pero ellos aún necesitan un programa que dadas estas distancias, decida si un jugador está en *offside*.

## Entrada

El archivo de entrada contiene varios casos de prueba. La primera línea de cada caso de prueba contiene dos enteros  $A$  y  $D$  separados por un sólo espacio indicando, respectivamente, el número de atacantes y defensores involucrados en el juego ( $d \leq A, D \leq 11$ ). La siguiente línea contiene  $A$  enteros  $B_i$  separados por espacios en blanco, indicando las distancias de los atacantes a la línea de meta ( $1 \leq B_i \leq 10^4$ ). La siguiente línea contiene  $D$  enteros  $C_j$  separados por espacios en blanco, indicando las distancias de los defensores a la línea de meta ( $1 \leq C_j \leq 10^4$ ). El final de la entrada es indicado por  $A = D = 0$ .

*La entrada debería ser leída de la entrada estandar*

# La Salida

Para cada paso de prueba en la entrada imprimir una línea conteniendo un solo caracter: “Y” (mayúscula) si hay un atacante en *offside*, y “N” (mayúscula) en otro caso.  
*La salida debe ser escrita en la salida estandar.*

| Ejemplo de Entrada | Ejemplo de Salida |
|--------------------|-------------------|
| 2 3                | N                 |
| 500 700            | Y                 |
| 700 500 500        | N                 |
| 2 2                |                   |
| 200 400            |                   |
| 200 1000           |                   |
| 3 4                |                   |
| 530 510 490        |                   |
| 480 470 50 310     |                   |
| 0 0                |                   |

## Iniciando la solución del problema

Como el tiempo es importante, mientras vamos pensando en la solución un integrante del equipo puede ir escribiendo el esqueleto básico, para leer las entradas:

```
class he{
    public static void main(String [] args){
        Scanner sc = new Scanner(System.in);
        int A = sc.nextInt();
        int D = sc.nextInt();
        while(A != 0 && D != 0){
            //Aqui va el resto del codigo
            A = sc.nextInt();
            D = sc.nextInt();
        }
    }
}
```

## Leyendo los casos de prueba

- Se deben leer  $A$  valores y luego  $D$  valores

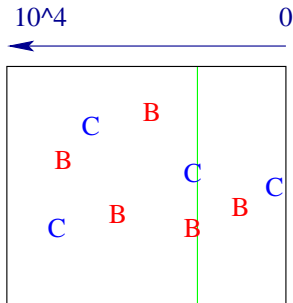
```
while (A != 0 && D != 0){  
    for (int i=0; i<A; i++)  
        int B = sc.nextInt();  
    for (int j=0; j<D; j++)  
        int C = sc.nextInt();  
    A = sc.nextInt();  
    D = sc.nextInt();  
}
```

- Parece que nos conviene almacenar cada uno de los  $B_i$  y  $C_j$ .
- Entonces, podemos usar un `ArrayList` o un simple arreglo de enteros, `ArrayList` suele ser más poderoso.

```
while(A != 0 && D != 0){  
    ArrayList B = new ArrayList();  
    for(int i=0; i<A; i++)  
        B.add(sc.nextInt());  
  
    ArrayList C = new ArrayList();  
    for(int j=0; j<D; j++)  
        C.add(sc.nextInt());  
  
    A = sc.nextInt();  
    D = sc.nextInt();  
}
```

## Pensando en la Solución

- Contamos con 2 conjuntos de números  $B$  atacantes y  $C$  defensores.
- Tenemos que saber si todos los miembros de  $B$  tienen números mayores o iguales que al menos dos integrantes de  $C$ .



## Fuerza Bruta

- Hay que verificar que cada uno de los integrantes de  $B$  tiene números mayores o igual que al menos dos integrantes de  $C$ .
- También se pueden buscar dos integrantes de  $C$  que tengan números menores que cualquier integrante de  $B$ .
- Esta solución costará  $\theta(n^2)$ , porque por tendremos que compara a cada integrante de un grupo con un integrante del otro grupo.
- La solución será difícil de implementar porque bastará encontrar un jugador en posición adelantada para responder “Si”, luego tendríamos un problema: ¿Cómo se detendría el ciclo, con un *break*?
- Sugerencias ...

## Usando la lógica matemática

- Lo que parecía un problema puede ser la solución:  
Basta que un jugador esté en posición adelantada para responder “Si”.
- Hay que encontrar a los dos jugadores del conjunto ( $C$ ) con las menores distancias a la línea de meta.
- Sería conveniente tener este conjunto ordenado, luego bastaría comparar sólo al segundo menor.
- Si el conjunto de atacantes ( $B$ ) está ordenado sería mejor, porque sólo tendríamos que comparar al menor de todos ellos  $B_{menor}$ . Si  $B_{menor}$  está adelantado se responde “Si” y si él no lo está, ninguno más podrá estarlo, porque  $B_{menor}$  es el menor de todos, no hay nadie más cerca a la línea de meta que él.

## Pensando en la implementación

- ¿La API de java ofrece las funcionalidades que necesitamos?
- Es posible ordenar un *ArrayList*, pero realmente ¿no necesitamos tanto!
- Existe una clase *Arrays* que trabaja sobre arreglos simples y tiene un método *sort*, que ordena arreglos.
- Este método está sobrecargado y también permite ordenar enteros. *¡Hay que usar arreglos!*
- El elemento menor de *B* estará en la posición 0.
- El segundo menor de *C* estará en la posición 1.

```
while (A != 0 && D != 0){  
    int [] B = new int [A];  
    for (int i=0; i<A; i++)  
        B[i] = sc.nextInt();  
  
    int [] C = new int [D];  
    for (int j=0; j<D; j++)  
        C[j] = sc.nextInt();  
  
    Arrays.sort(B);  
    Arrays.sort(C);  
  
    if (B[0]<C[1]) System.out.print("S\n");  
    else System.out.print("N\n");  
    A = sc.nextInt();  
    D = sc.nextInt();  
}
```

## Un error!!!

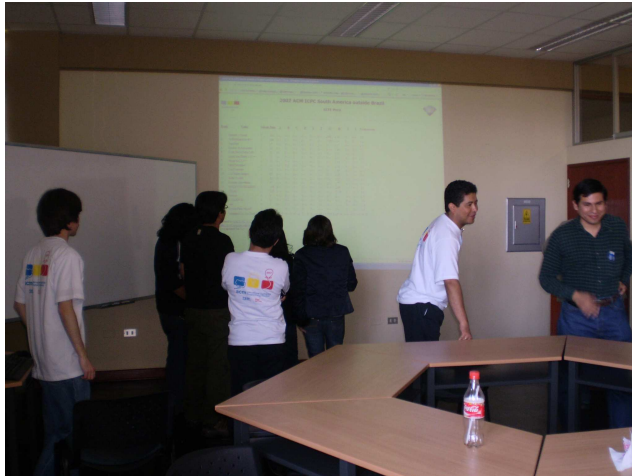
- Enviamos la solución a los jueces y nos han respondido “NO: Error en el formato”.
- En nuestra emoción por haber resuelto el problema tan rápido, hemos cometido un pequeño error.
- El enunciado pedía que escribamos “Y” y “N”, en mayúsculas; y nosotros escribimos “S” y “N”.
- Qué error!!!! ahora nos castigarán con unos minutos más.

## El costo algorítmico

- La solución planteada tiene costo  $\theta(n \log n)$ , por el paso de ordenamiento.
- Sin embargo, no se necesita conocer el orden de todos los elementos, sólo necesitamos conocer al  $B$  menor y al segundo  $C$  menor.
- Estos valores pueden ser calculados directamente de la lectura de los datos.
- El costo de esta solución será sólo de  $\theta(n)$ .
- Los jueces podrían haber puesto un límite de tiempo muy bajo y el costo de la ordenación habría hecho que nos digan *tiempo excedido!*

```
while (A != 0 && D != 0){  
    int minB = fMinC = sMinC = Integer.MAX_VALUE;  
    for (int i=0; i<A; i++){  
        int B = sc.nextInt();  
        minB = Math.min(minB, B);  
    }  
    for (int j=0; j<D; j++){  
        int C = sc.nextInt();  
        if ( C < fMinC ){  
            sMinC = fMinC; fMinC = C;  
        } else sMinC = Math.min(sMinC,C);  
    }  
    if (minB < sMinC) System.out.println("Y");  
    else System.out.println("N");  
    A = sc.nextInt(); D = sc.nextInt();  
}
```

# Coach - Lima



# Concurso - Bolivia



# Volante - Bolivia

**acm** International Collegiate Programming Contest  
Competencia Sudamericana de Programación

*"... la competencia de programación más antigua, más grande, y más prestigiosa del mundo"*

El ACM - ICPC permite a los estudiantes de ciencias de la computación de todos los niveles representar a **BOLIVIA** en la Competencia Sudamericana de Programación que es clasificatoria a la Final Mundial.

**Final Sudamericana:** La Paz, 10 de Noviembre de 2007.

Te invitamos a representar a tu Instituto, Tecnológico o Universidad y a **BOLIVIA** en las dos fases de la Competencia Sudamericana del 2008.

Para más información sobre las competencias del 2007 y 2008 visita: <http://www.icpc-bolivia.edu.bo/>

ORACLE SIMI Partner Advantage C-ISO Networking Academy STC Science & Technology Center S.R.L. <http://cti.nur.edu>

# Concurso - Chile



## Cena de Gala - Chile



## Concurso - Argentina



# Pizarra de Jueces - Argentina



## Participación Peruana en Resumen

| Año   | Participantes     | Prob Perú       | Prob Región    |
|-------|-------------------|-----------------|----------------|
| 1999: | 1 Universidad,    | 1/6 Problemas,  | 4/6 Resueltos  |
| 2001: | 1 Universidad,    | 1/6 Problemas,  | 4/6 Resueltos  |
| 2002: | 1 Universidad,    | 3/6 Problemas!, | 5/6 Resueltos  |
| 2003: | 2 Universidades,  | 1/8 Problemas,  | 5/8 Resueltos  |
| 2004: | 4 Universidades,  | 2/8 Problemas,  | 5/8 Resueltos  |
| 2005: | 9 Universidades!, | 4/8 Problemas,  | 7/8 Resueltos  |
| 2006: | 6 Universidades,  | 3/9 Problemas,  | 9/9 Resueltos  |
| 2007: | 6 Universidades,  | 4/10 Problemas, | 8/10 Resueltos |