Introducción a las competencias de programación

Fidel I. Schaposnik M.
Universidad Nacional de La Plata

fidel.s@gmail.com

13 y 14 de mayo de 2014

Parte I

¿Qué ...?

¿Por qué ...?

¿Qué son las competencias de programación?

Para estudiantes secundarios

International Olympiads in Informatics

www.ioinformatics.org

A nivel universitario

ACM International Collegiate Programming Contest

icpc.baylor.edu

Abiertas a todo público

Google CodeJam

code.google.com/codejam

Facebook HackerCup

www.facebook.com/hackercup

TopCoder

www.topcoder.com/tc

¿Qué son las competencias de programación? ACM - ICPC

¿En qué consiste?

- 3 participantes por equipo;
- 1 computadora;
- $\sim$ 10 problemas;
- 5 horas de competencia.

¿Qué son las competencias de programación? ACM - ICPC

¿En qué consiste?

- 3 participantes por equipo;
- 1 computadora;
- $\sim$ 10 problemas;
- 5 horas de competencia.

NO se puede

- conectarse a internet;
- usar material electrónico alguno;
- intentar hacer trampa de cualquier tipo :-)

SÍ se puede

- consultar material impreso;
- utilizar librerías estándar de C++ ó Java.

¿Qué son las competencias de programación? ACM - ICPC

¿Quién gana?

- El equipo que resolvió más problemas;
- Si hay empate, el equipo que lo hizo más rápido.

¿Qué son las competencias de programación? ACM - ICPC

¿Quién gana?

- El equipo que resolvió más problemas;
- Si hay empate, el equipo que lo hizo más rápido.

Hay 3 etapas

- **Local**: compiten los equipos de una misma universidad para seleccionar a los que van a ir a la etapa regional;
- **Regional**: compiten los equipos de una (sub)región para seleccionar a los que van a ir a la final;
- **World Finals**: compite a lo sumo un equipo por universidad.

¿Qué es un “problema”?

Un problema consiste en

- Una historia en la que se describe el problema a resolver.
- La “entrada” explica cómo nos describen la situación.
- La “salida” explica cómo presentamos nuestra respuesta.
- Algunos ejemplos con instancias particulares del problema para entender mejor la situación.

¿Qué es un "problema"?

Un problema consiste en

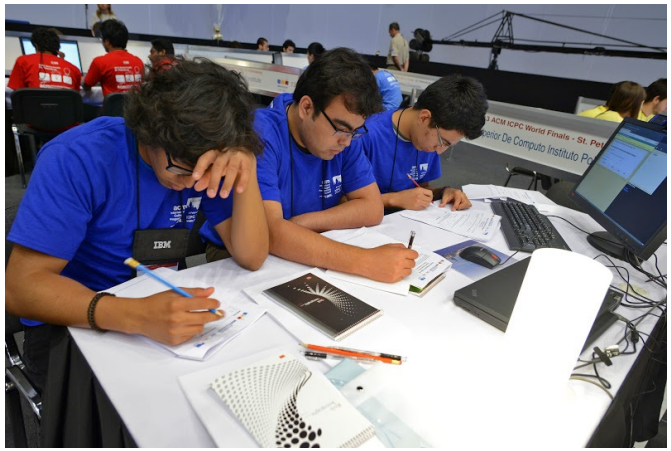
- Una historia en la que se describe el problema a resolver.
- La "entrada" explica cómo nos describen la situación.
- La "salida" explica cómo presentamos nuestra respuesta.
- Algunos ejemplos con instancias particulares del problema para entender mejor la situación.

¿Qué significa resolverlo?

- Diseñar un algoritmo capaz de encontrar la respuesta buscada.
- Lograr que el algoritmo sea suficientemente rápido para el tamaño de los problemas que se nos van a presentar.
- Implementar el algoritmo (¡sin errores!) en un solo archivo de código fuente.

¿Por qué participar en competencias de programación?

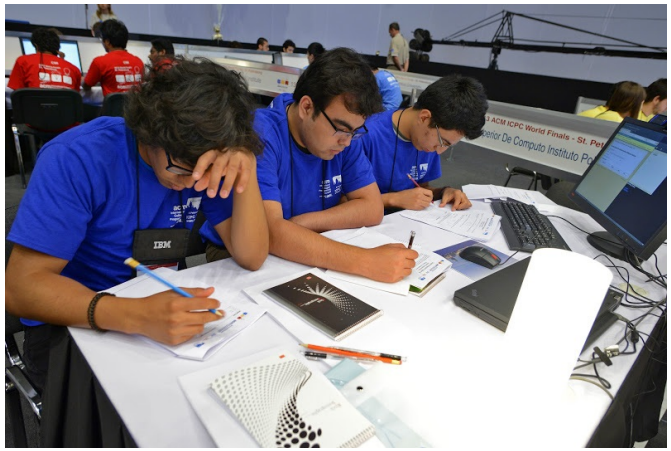
For fun...



$\Longleftarrow$ ¡gente divirtiéndose!

¿Por qué participar en competencias de programación?

For fun...



⟸ ¡gente divirtiéndose!

Y también para:

- tener una (¿otra?) razón para juntarse con amigos;
- tener una buena excusa para no estudiar;
- tomarse vacaciones (¡pagas!) con amigos;
- ...

¿Por qué participar en competencias de programación?

... & Profit

Se desarrollan

- habilidades para programar rápida y eficientemente;
- intuición práctica de conceptos "académicos" ($\mathcal{O}$, divide-and-conquer, programación dinámica, ...);
- capacidad de trabajar en equipo;
- capacidad de trabajar bajo presión;
- imaginación y pensamiento creativo para resolver problemas en forma no-ortodoxa.

¿Por qué participar en competencias de programación?

Google $\Longrightarrow$

Hi Fidel,

I'm a talent scout on the Engineering Staffing Team at Google. I came across your details online through your participation in various coding competitions and feel that you could be exactly the sort of person we are looking for to work as part of our engineering team. I understand that you've been in contact with colleagues of mine previously and that the timing has never been right so I wanted to circle back to see if now might be a better time for you to explore job opportunities with Google?

I look forward to hearing back from you!

Best regards,

Kevin Grice
Google
kevingrice@google.com
Technical Sourcer

Hi Fidel,

I'll keep this short & sweet since I'm sure you get several messages every week. Your reply is greatly appreciated!

I came across your name on a variety of programming websites, and was impressed by your interest and passion for coding. I am curious: would you be interested in applying those skills to challenged at Facebook? Might you be interested in [confidentially] learning about things going on at Facebook that are relevant to your background?

$\Longleftarrow$ Facebook

Let me know, either way!

Cheers!
Nicole

¿Por qué participar en competencias de programación?

Encuesta rápida ($\sim$ 60 respuestas):

- Cantidad media de participaciones en una regional: 2.68
- Cantidad media de participaciones en una final: 0.60

¿Por qué participar en competencias de programación?

Encuesta rápida ($\sim$ 60 respuestas):

- Cantidad media de participaciones en una regional: 2.68
- Cantidad media de participaciones en una final: 0.60
- ¿Ha sido contactado alguna vez por Facebook o Google acerca de oportunidades laborales? 59%
- ¿Ha realizado una pasantía en Facebook o Google? 22%
- ¿Trabaja o trabajó alguna vez en Facebook o Google? 17%

Parte II

¿Cómo ...?

¿Cómo es un problema?

Historia: Tenemos una fila de soldados formada de izquierda a derecha. Nos van a indicar grupos de soldados contiguos que mueren durante una batalla, y queremos saber cuál es el primer soldado vivo a la izquierda y a la derecha de cada grupo.

Entrada:

- Una línea con el número de soldados ($1 \leq S \leq 10^5$) y el número de reportes ($1 \leq R \leq S$).
- R líneas de la forma I D con $1 \leq I \leq D \leq S$, indicando que los soldados desde el I hasta el D han muerto.

Salida: Por cada reporte, escribimos una línea con el número del primer soldado vivo a la izquierda del soldado I, y el número del primer soldado vivo a la derecha del soldado D (o un '*' en caso de que alguno de estos números no exista).

¿Cómo es un problema? Ejemplo desarrollado

Uno de los ejemplos:

- 10 4
  - Tenemos $S = 10$ y $R = 4$
  - La fila de soldados es 1 2 3 4 5 6 7 8 9 10

¿Cómo es un problema? Ejemplo desarrollado

Uno de los ejemplos:

- 10 4
  - Tenemos $S = 10$ y $R = 4$
  - La fila de soldados es 1 2 3 4 5 6 7 8 9 10
- 2 5
  - Ahora es 1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 7 8 9 10
  - Respondemos 1 6

¿Cómo es un problema? Ejemplo desarrollado

Uno de los ejemplos:

- 10 4
  - Tenemos $S = 10$ y $R = 4$
  - La fila de soldados es 1 2 3 4 5 6 7 8 9 10
- 2 5
  - Ahora es 1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 7 8 9 10
  - Respondemos 1 6
- 6 9
  - Ahora es 1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ 10
  - Respondemos 1 10

¿Cómo es un problema? Ejemplo desarrollado

Uno de los ejemplos:

- 10 4
  - Tenemos $S = 10$ y $R = 4$
  - La fila de soldados es 1 2 3 4 5 6 7 8 9 10
- 2 5
  - Ahora es 1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 7 8 9 10
  - Respondemos 1 6
- 6 9
  - Ahora es 1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ 10
  - Respondemos 1 10
- 1 1
  - Ahora es ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ 10
  - Respondemos $*$ 10

¿Cómo es un problema? Ejemplo desarrollado

Uno de los ejemplos:

- 10 4
  - Tenemos $S = 10$ y $R = 4$
  - La fila de soldados es 1 2 3 4 5 6 7 8 9 10
- 2 5
  - Ahora es 1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 7 8 9 10
  - Respondemos 1 6
- 6 9
  - Ahora es 1 ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ 10
  - Respondemos 1 10
- 1 1
  - Ahora es ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ 10
  - Respondemos $*$ 10
- 10 10
  - Ahora es ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~
  - Respondemos $*$ $*$

¿Cómo nos evalúan?

Enviamos nuestro código fuente a los jueces:

- Compilan nuestro programa:
  - Si no compila, "Compilation Error".
- Lo ejecutan con los casos de prueba **secretos** como entrada:
  - si tarda demasiado (> 2 segundos), "Time Limit Exceeded";
  - si la salida no es **exactamente** la correcta, "Wrong Answer";
  - si la salida es idéntica a la que ellos tienen, "Accepted".
- Podemos reintentar tantas veces como queramos hasta obtener Accepted.
- Sólo los problemas correctamente resueltos dan puntos (y tiempo de penalidad).

¿Cómo es una solución? Algoritmo I

Primera idea:

- Usamos un arreglo de números enteros $v[S]$ para representar a los soldados:
  - $v[i] == 1 \iff$ el i-ésimo soldado está vivo.
  - $v[i] == 0 \iff$ el i-ésimo soldado está muerto.

- Al comenzar, ponemos $v[i] = 1$ para $i = 1, 2, \ldots, S$.
- Al recibir cada nuevo reporte:
  - ponemos $v[i] = 0$ para $i = I, I + 1, \ldots, D - 1, D$
  - buscamos el primer $v[i] == 1$ con $i = I - 1, I - 2, \ldots$
  - buscamos el primer $v[i] == 1$ con $i = D + 1, D + 2, \ldots$

¿Cuánto tarda este algoritmo? (No hace falta hacer un análisis demasiado riguroso.)

Algoritmo I (análisis)

Un juez perverso podría probar el siguiente caso con $S = 100000$ y $R = 100000$:

- El primer reporte es 1 1
- El segundo reporte es 2 2
- El siguiente reporte es 3 3
- ...
- El último reporte es 100000 100000

Algoritmo I (análisis)

Un juez perverso podría probar el siguiente caso con $S = 100000$ y $R = 100000$:

- El primer reporte es `1 1`
- El segundo reporte es `2 2`
- El siguiente reporte es `3 3`
- ...
- El último reporte es `100000 100000`

Ya sabemos cuáles deberían ser las respuestas:

- Para el primer reporte, `* 2`
- Para el segundo reporte, `* 3`
- ...
- Para el anteúltimo reporte, `* 100000`
- Para el último reporte, `* *`

Algoritmo I (análisis)

¿Cuántas operaciones de lectura o escritura de una posición en $v[S]$ tenemos que hacer con nuestro algoritmo?

- Para el primer reporte son ~ 1.
- Para el segundo reporte, unas ~ 2.
- Para el tercer reporte, unas ~ 3.
- ...
- Para el último reporte, ya son unas ~ 100000.

¡En total, son unas $1 + 2 + \ldots 100000 \sim 10^{10}$ operaciones!

Algoritmo I (análisis)

¿Cuántas operaciones de lectura o escritura de una posición en $v[S]$ tenemos que hacer con nuestro algoritmo?

- Para el primer reporte son ~ 1.
- Para el segundo reporte, unas ~ 2.
- Para el tercer reporte, unas ~ 3.
- ...
- Para el último reporte, ya son unas ~ 100000.

¡En total, son unas $1 + 2 + \ldots 100000 \sim 10^{10}$ operaciones!

Una computadora personal como las usadas en las competencias puede realizar aproximadamente 10^8 operaciones por segundo, luego nuestro algoritmo tardaría casi 2 minutos, lo que seguramente dará Time Limit Exceeded...

¡$\mathcal{O}(RS)$ con $R, S \leq 10^5$ es demasiado lento!

¿Cómo es una solución? Algoritmo II

Una idea mejor:

- Usamos 2 arreglos de números enteros, izq[S] y der[S]:
  - izq[i] es el primer soldado vivo a la izquierda del i-ésimo.
  - der[i] es el primer soldado vivo a la derecha del i-ésimo.
- Al comenzar, ponemos:
  - izq[i] $= i - 1$ para $i = 1, \ldots, S$.
  - der[i] $= i + 1$ para $i = 1, \ldots, S$.
- Al recibir cada nuevo reporte:
  - Llamamos $a =$ izq[I] y $b =$ der[D].
  - Respondemos "a b".
  - Actualizamos los arreglos para mantener nuestro invariante:

$$\text{izq}[b] = a \qquad \text{der}[a] = b$$

¿Cómo funciona este algoritmo? ¿Cuánto tarda?

Algoritmo II (análisis)

Por cada reporte debemos hacer muy pocas operaciones:

- Leer dos posiciones en los arreglos.
- Actualizar dos posiciones en los arreglos.

Algoritmo II (análisis)

Por cada reporte debemos hacer muy pocas operaciones:

- Leer dos posiciones en los arreglos.
- Actualizar dos posiciones en los arreglos.

¡Como el ńumero máximo de reportes es $R = 10^5$, ni el juez más perverso podría obligarnos a hacer más de $\sim 4 \times 10^5$ operaciones!

$$\text{¡}\mathcal{O}(R) \Longrightarrow \text{Accepted!}$$

Algoritmo II (código)

```cpp
#include <cstdio>
using namespace std;

int main() {
    int S, R, I, D, izq[100100], der[100100], i, a, b;

    while (scanf("%d %d", &S, &R) && (S!=0 || R!=0)) {
        for (i=1; i<=S; i++) izq[i] = i-1;
        for (i=1; i<=S; i++) der[i] = i+1;
        for (i=0; i<R; i++) {
            scanf("%d %d", &I, &D);
            a = izq[I]; b = der[D];
            izq[b] = a; der[a] = b;

            if (a == 0) printf("* ");
            else printf("%d ", a);
            if (b == S+1) printf("*\n");
            else printf("%d\n", b);
        }
        printf("-\n");
    }
    return 0;
}
```

¿Qué aprendimos de este ejemplo?

En general:

- La solución más simple de un problema puede (¿suele?) no ser la que necesitamos.
- Los límites de las variables que describen al problema nos indican cuán eficiente debe ser nuestra solución.
- La entrada siempre se ajusta a las especificaciones.
- Si nuestro algoritmo falla con algún caso particular, podemos suponer que el juez lo sabe y no lo va a dejar pasar.

¿Qué aprendimos de este ejemplo?

En general:

- La solución más simple de un problema puede (¿suele?) no ser la que necesitamos.
- Los límites de las variables que describen al problema nos indican cuán eficiente debe ser nuestra solución.
- La entrada siempre se ajusta a las especificaciones.
- Si nuestro algoritmo falla con algún caso particular, podemos suponer que el juez lo sabe y no lo va a dejar pasar.

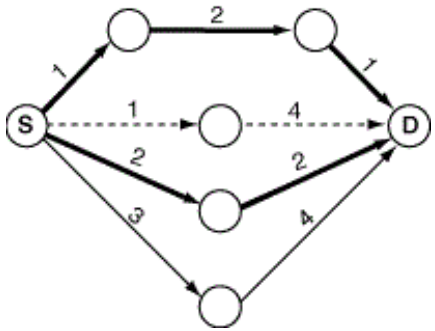
En particular:

- La entrada se lee del *standard input* (`cin`/`scanf`).
- La salida se imprime al *standard output* (`cout`/`printf`).
- Se nos puede pedir resolver varios casos de prueba hasta alcanzar una señal determinada (aquí era $S = 0$ y $R = 0$).

Otros tipos de problemas

Problemas de grafos

- Dados dos árboles de N y M nodos respectivamente, hallar el tamaño del árbol formado al unirlos con una arista tomada con probabilidad uniforme entre todas las posibles en $\mathcal{O}(N + M)$
- Dado un grafo de N nodos y E aristas, hallar el camino mínimo entre un par de nodos fijo que no usa ninguna arista que pertenezca a algún camino mínimo entre ellos en $\mathcal{O}(N \cdot E)$

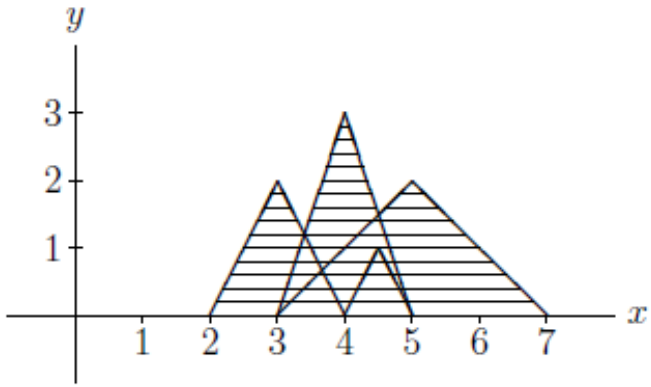


- ...

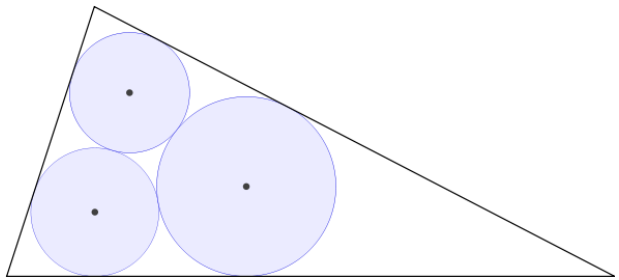
Otros tipos de problemas

Problemas de geometría computacional

- Hallar el área sombreada en $\mathcal{O}(N^2)$



- Hallar las posiciones de los centros de los círculos



- ...

Otros tipos de problemas

- Programación dinámica
- Algoritmos sobre cadenas
- Estructuras de datos
- Matemática
- Teoría de juegos
- ...

¿Cómo ganar una competencia de programación ...

... en tres simples pasos? $\Longrightarrow$

¿Cómo ganar una competencia de programación ...

... en tres simples pasos? $\Longrightarrow$

1. Ser ruso
2. Competir
3. Ganar

¿Cómo ganar una competencia de programación ...

... en tres simples pasos? $\Longrightarrow$

1. Ser ruso
2. Competir
3. Ganar

Método alternativo:

1. Conseguir dos compañeros de equipo rusos
2. No molestar durante 5 horas
3. Ganar

Practicar, practicar, practicar

Jueces online

- **ACM ICPC Live Archive** (icpcarchive.ecs.baylor.edu)
  Competencias regionales y mundiales pasadas
- **Caribbean Online Judge** (coj.uci.cu)
  Problemas de diversas fuentes traducidos al castellano
- **CodeForces** (codeforces.com)
  Problemas y competencias online
- **SPOJ** (www.spoj.com), **URI** (urionlinejudge.com.br), ...
  Muchos problemas de todo tipo

¿Y ahora ...?

Parte III

Para empezar a competir...

1. Crear una cuenta en un juez online y empezar a resolver problemas;
2. Formar un equipo y practicar juntos hasta tener una estrategia clara;
3. Participar de la regional sudamericana (noviembre 2014).

Parte III

Para empezar a competir...

1. Crear una cuenta en un juez online y empezar a resolver problemas;
2. Formar un equipo y practicar juntos hasta tener una estrategia clara;
3. Participar de la regional sudamericana (noviembre 2014).

Para progresar más rápido...

- Preguntar mucho para no trabarse (e.g. en latinoamerica-icpc@googlegroups.com);
- Leer código de soluciones para problemas que no nos salen;
- Participar de campamentos o escuelas específicas...

Training Camp Argentina 2014

La 5^{ta} edición:

- Facultad de Ciencias Exactas y Naturales - Universidad de Buenos Aires
- 28 de julio al 8 de agosto de 2014
- Charlas teóricas, simulación de pruebas, análisis y resolución de problemas
- **Inscripciones abiertas hasta el viernes 16 de mayo**
- Sponsors:

¡Gracias!