Cómo resolver los problemas de ProgramaMe

Facultad de Informática
Universidad Complutense de Madrid

I.E.S. Antonio de Nebrija (Móstoles)

http://www.programa-me.com

Cómo resolver los problemas de ProgramaMe

1 ¿Por dónde empezar?

Cuando comienza el concurso se da a los participantes el enunciado de un número determinado de problemas (entre 8 y 10) de distinta dificultad. La competición consiste en intentar resolver el mayor número de problemas en el menor tiempo posible. Ganará el equipo que tenga más problemas resueltos correctamente cuando termine el tiempo.

Es importante destacar que en caso de empate (dos o más equipos han resuelto el mismo número de problemas) se utiliza *el tiempo total* para desempatar (también aquí hay que sumar el tiempo por las penalizaciones de envíos incorrectos). Eso se traduce en que es preferible resolver los problemas *cuanto antes*.

Teniendo en cuenta esto, los equipos experimentados en concursos de este tipo suelen comenzar *repartiéndose los enunciados* de los ejercicios para leerselos rápidamente y evaluar de forma rápida cuáles son más fáciles y cuáles son más difíciles. Tras este periodo inicial de lectura, y una vez que se tienen más o menos ordenados los problemas por orden de dificultad, se empieza por el más fácil. De esta forma se pretende tener ejercicios resueltos lo antes posible para, en caso de empate, conseguir la victoria.

2 ¿Qué hacer durante el concurso? (Consejos para las convocatorias presenciales)

Durante los concursos presenciales sólo hay un ordenador por equipo, por lo que sólo una persona puede estar tecleando. Es habitual, no obstante, que al menos dos personas estén pensando y programando el ejercicio en cuestión, para evitar errores.

Mientras tanto, el otro miembro del equipo puede dedicarse a *escribir manualmente casos de prueba*. En general, las soluciones planteadas *deberían probarse con más casos* de los mostrados como ejemplo en los enunciados. La creación de casos de prueba no es una tarea trivial, pero es muy importante si se quieren evitar envíos incorrectos. Un problema que da el resultado esperado para el ejemplo del enunciado *no* necesariamente es una solución correcta. Un miembro del equipo puede dedicarse a plantear casos de prueba adicionales (y sus soluciones, obtenidas manualmente) que busquen los límites en la entrada, por ejemplo, o las esquinas más oscuras de la solución.

Si se considera que la escritura de casos de prueba no es necesaria, otra forma de aprovechar el tiempo es que el tercer integrante del equipo vaya pensando en cómo

resolver otro problema. Según el caso, podría incluso escribir en papel el código fuente, para reducir el tiempo necesario para probarlo una vez que el ordenador "quede libre".

Tened en cuenta que, tras enviar la solución de un problema, la respuesta de los jueces puede tardar algún tiempo. Mientras tanto el ordenador queda libre y esa tercera persona que estaba pensando en el otro ejercicio puede comenzar a resolverlo (con la ayuda de otro compañero).

Es muy normal durante el concurso tener dos o tres problemas "abiertos" a medio programar. Por ejemplo, si la respuesta de los jueces es negativa y hay algún error, uno de los miembros del equipo puede dedicarse a intentar encontrar el fallo (sobre el papel) mientras los otros dos siguen adelante con la programación de otro ejercicio.

3 ¿Qué hacer durante el concurso? (Consejos para las convocatorias on-line)

En las convocatorias on-line del concurso, la limitación de un ordenador por equipo *no existe*. Por tanto, la estrategia más directa para optimizar el tiempo consiste en dividirse los problemas, y que cada miembro del equipo trate de solucionar aquellos de los que se haya hecho cargo.

Dado que, como en los concursos presenciales, se utiliza como criterio de desempate en la clasificación el tiempo total necesario para resolver los problemas, es interesante resolver los problemas lo antes posible. Por tanto, una distribución de los problemas entre los participantes debería intentar que todos reciban al menos un problema fácil, para resolverlos en paralelo y entregarlos antes.

No obstante, el reparto de los problemas deberá adaptarse a la idiosincrasia de cada equipo, y a las habilidades concretas de sus integrantes. Por ejemplo, si sólo uno de los participantes conoce el concepto de recursión y se identifica que en el concurso hay algún problema de este tipo, debería ser asignado desde el primer momento a quién más probabilidades tiene de saber resolverlo.

Por último, aunque cada participante podría estar usando su propio ordenador, eso no significa que *tenga que hacerlo siempre*. La colaboración entre los miembros de un equipo a la hora de resolver problemas y, sobre todo, buscar errores puede ser de una gran ayuda. Un pequeño retraso en la resolución de un problema para ayudar a un compañero de equipo puede resultar muy beneficioso al final.

4 Cómo son los problemas

Como se describe más adelante, los problemas a resolver durante el concurso, son siempre aplicaciones de consola, recibiendo los datos de ejecución a través de la entrada estándar, y enviando los resultados a la salida estándar.

A nivel de estructuras de programación, los problemas pueden incluir, entre otras cosas, tipos básicos y compuestos, expresiones, saltos condicionales, bucles, estructuras de datos típicas y recursión. Se intenta que ninguno de los lenguajes permitidos en el

concurso aporte una ventaja significativa en el tiempo de resolución de los problemas que se plantean.

El espíritu de las ediciones presenciales y on-line de ProgramaMe es diferente, y por tanto también lo son sus problemas.

Si se revisan los cuadernillos de los problemas de las convocatorias *presenciales* anteriores, se puede comprobar que muchos de los enunciados están ambientados, de modo que se dota al ejercicio de un contexto que le concede cierta gracia. Debajo de esa ambientación, hay un problema de programación de diversa naturaleza.

Algunos ejercicios centran su dificultad en los aspectos meramente técnicos de la programación (por ejemplo, recorridos de arrays bidimensionales). Otros, sin embargo, tienen soluciones muy sencillas *de programar*, pero que necesitan una cierta reflexión para llegar al método de resolverlos. Estos ejercicios suelen ser idóneos para que los miembros del equipos que no tienen acceso al ordenador aprovechen el tiempo avanzando en ellos. Naturalmente, hay también ejercicios que caen en ambos grupos, que suelen ser los más difíciles.

Debido a sus características particulares, las ediciones on-line del concurso tienden a tener más problemas sencillos de programación rápida. Se persigue más bien comprobar las destrezas de programación rápida que la disección analítica de los enunciados.

Nota: es importante resaltar el hecho de que el juez automático exige que todos los programas acaben con éxito (resultado de ejecución 0). Esto supone que en las soluciones programadas en C o C++ es importante acabar la función `main` con un `return 0`. En otro caso, el juez dará como veredicto un error de ejecución.

5 Entrada de datos

Todos los problemas utilizan el mismo esquema: dado un caso de entrada hay que escribir algo sobre él.

Para que se pueda probar con certeza que el programa funciona, éste tendrá que ser probado con numerosos casos de entrada, y dar la respuesta correcta para todos ellos. Para hacerlo, hay tres alternativas o "estilos de entrada":

1. Al principio de la ejecución, el programa recibe *el número de casos de prueba* que se utilizan.

2. El programa va leyendo casos de prueba hasta que se encuentra con un caso de prueba *especial*.

3. El programa va leyendo casos de prueba hasta que se alcanza el final de la entrada (no quedan más datos).

Dependiendo de si es una u otra alternativa el esquema general del programa será distinto. Como consejo recomendamos que se utilice una función/método que se encargue de resolver un caso de prueba y que sea llamado desde el programa principal tantas veces como sea necesario.

Como ejemplo, una solución en C++ de un problema que comienza con el número de casos de prueba (primer tipo) podría tener el siguiente esquema:

```
#include <iostream>
using namespace std;

// Resuelve un caso de prueba, leyendo de la entrada la
// configuracion, y escribiendo la respuesta.
void casoDePrueba() {
    // ...
}

int main() {

    int numCasos;
    cin >> numCasos;

    for (int i = 0; i < numCasos; i++)
        casoDePrueba();

    return 0;
}
```

Si la entrada del problema en vez de empezar con el número de casos de prueba termina con un *caso de prueba especial*, podemos utilizar el siguiente:

```
#include <iostream>
using namespace std;

// Resuelve un caso de prueba, leyendo de la entrada la
// configuracion, y escribiendo la respuesta. Si el caso
// de prueba leido es el que marca el final de la ejecucion,
// la funcion devuelve false para indicar al main que hay
// que terminar.
bool casoDePrueba() {
    // ...
    // Leemos de la entrada
    cin >> ...

    if (caso_especial)
        return false;

    // ... resolvemos ...

    return true;
```

```
}

int main() {

    bool seguir;

    do {
        seguir = casoDePrueba();
    } while (seguir);

    // Tambien se puede resumir en una unica linea:
    // while (casoDePrueba()) ;
}
```

La última posibilidad en la que el final no viene marcado con un caso de prueba especial sino con la terminación de la entrada utiliza un esquema similar al anterior. La única diferencia consiste en que la función `casoDePrueba` comprueba si se ha llegado al final para devolver `false`.

En las siguientes secciones, planteamos un hipotético problema con los tres estilos de entrada, y lo resolvemos en los tres lenguajes de programación admitidos en el concusro.

6 Ejemplo de ejercicio con casos de prueba limitados

Imaginemos que nos piden un problema tan simple como éste[1]:

Entrada

La primera línea de la entrada contendrá el número de casos de prueba que el programa debe leer. A continuación vendrá uno detraś de otro todos esos casos. Cada uno de ellos consiste en una única línea con un número entero.

Salida

Para cada caso de prueba el programa escribirá `PAR` si el caso de prueba es un número par y escribirá `IMPAR` si el número es impar.

Las soluciones en C, C++ y Java aparecen a continuación. Como se ve, en todas ellas se sigue el mismo esquema descrito en la sección anterior.

```
// SOLUCION EN C
#include <stdio.h>

// Resuelve un caso de prueba, leyendo de la entrada la
// configuracion, y escribiendo la respuesta.
```

[1]Los problemas del concurso, especialmente en su edición presencial, serán más complicados. Éste lo utilizamos como ejemplo para ver cómo sería el esquema de la solución.

```
void casoDePrueba() {
   int num;

   scanf("%d\n", &num);

   if ((num % 2) == 0)
      printf("PAR\n");
   else
      printf("IMPAR\n");
}

int main() {

   int numCasos;
   int i;
   scanf("%d\n", &numCasos);

   for (i = 0; i < numCasos; i++)
      casoDePrueba();

  return 0;
}
```

```
// SOLUCION EN C++
#include <iostream>
using namespace std;

// Resuelve un caso de prueba, leyendo de la entrada la
// configuracion, y escribiendo la respuesta.
void casoDePrueba() {
   int num;

   cin >> num;

   if ((num % 2) == 0)
      cout << "PAR\n";
   else
      cout << "IMPAR\n";

}

int main() {
```

```
    int numCasos;
    cin >> numCasos;

    for (int i = 0; i < numCasos; i++)
       casoDePrueba();

    return 0;
}
```

```
// SOLUCION EN Java
class solution {

     static java.util.Scanner in;

     public static void casoDePrueba() {
        int n;
        n = in.nextInt();

        if ((n % 2) == 0)
           System.out.println("PAR");
        else
           System.out.println("IMPAR");
     }

     public static void main(String args[]) {

          in = new java.util.Scanner(System.in);

          int numCasos;
          numCasos = in.nextInt();
          for (int i = 0; i < numCasos; i++)
             casoDePrueba();
     }
}
```

7 Ejemplo de ejercicio con casos de prueba ilimitados acotados por caso de prueba especial

Los jueces podrían haber puesto el mismo problema pero cambiando el formato de los casos de entrada. En ese caso el enunciado tendría la forma siguiente:

Entrada

La entrada consistirá en un número indeterminado de casos de prueba. Cada caso de prueba consistirá en un número entero. Los casos de prueba terminarán con el número `-1`, que marcará el final de la entrada y que no será procesado.

Salida

Para cada caso de prueba el programa escribirá `PAR` si el caso de prueba es un número par y escribirá `IMPAR` si el número es impar.

Las soluciones en C, C++ y Java aparecen a continuación. Como se ve, en todas ellas se sigue el mismo esquema descrito más arriba en el documento.

```c
// SOLUCION EN C
#include <stdio.h>

int casoDePrueba() {
    int num;

    scanf("%d\n", &num);

    if (num == -1)
        // Marca de fin de entrada
        return 0;

    if ((num % 2) == 0)
        printf("PAR\n");
    else
        printf("IMPAR\n");

    return 1;
}

int main() {

    while (casoDePrueba())
        ;

    return 0;
}
```

```cpp
// SOLUCION EN C++
#include <iostream>
using namespace std;
```

```
bool casoDePrueba() {
   int num;

   cin >> num;

   if (num == -1)
      return false;

   if ((num % 2) == 0)
      cout << "PAR\n";
   else
      cout << "IMPAR\n";

   return true;
}

int main() {

   while (casoDePrueba())
      ;

   return 0;
}
```

```
// SOLUCION EN Java
class solution {

    static java.util.Scanner in;

    public static boolean casoDePrueba() {
       int n;
       n = in.nextInt();

       if (n == -1)
          return false;

       if ((n % 2) == 0)
          System.out.println("PAR");
       else
          System.out.println("IMPAR");

       return true;
    }
```

```
    public static void main(String args[]) {

        in = new java.util.Scanner(System.in);

        while (casoDePrueba())
            ;

    }
}
```

8 Ejemplo de ejercicio con casos de prueba ilimitados

Podríamos tener el mismo problema pero en el que la entrada no termina con un caso especial:

Entrada

La entrada consistirá en un número indeterminado de casos de prueba. Cada caso de prueba consistirá en un número entero.

Salida

Para cada caso de prueba el programa escribirá `PAR` si el caso de prueba es un número par y escribirá `IMPAR` si el número es impar.

Las soluciones en C, C++ y Java aparecen a continuación. Como se ve, en todas ellas se sigue el mismo esquema descrito más arriba en el documento.

```
// SOLUCION EN C
#include <stdio.h>

int casoDePrueba() {
    int num;

    scanf("%d", &num);
    if (feof(stdin))
      return 0;

    // Tambien valido:
    // if (scanf("%d", &num) != 1))
    //    return 0;

    if ((num % 2) == 0)
        printf("PAR\n");
```

```
    else
        printf("IMPAR\n");

    return 1;
}

int main() {

    while (casoDePrueba())
        ;

    return 0;

}
```

```
// SOLUCION EN C++
#include <iostream>
using namespace std;

bool casoDePrueba() {
    int num;

    cin >> num;

    if (!cin)  // Fin de la entrada
        return false;

    if ((num % 2) == 0)
        cout << "PAR\n";
    else
        cout << "IMPAR\n";

    return true;
}

int main() {

    while (casoDePrueba())
        ;

    return 0;

}
```

```
// SOLUCION EN Java
class solution {

    static java.util.Scanner in;

    public static boolean casoDePrueba() {
       int n;

       if (!in.hasNext())
          return false;

       n = in.nextInt();

       if ((n % 2) == 0)
          System.out.println("PAR");
       else
          System.out.println("IMPAR");

       return true;
    }

    public static void main(String args[]) {

        in = new java.util.Scanner(System.in);

        while (casoDePrueba())
           ;
    }
}
```