Cómo resolver los problemas para el juez automático[1]

Alberto Verdejo.
Adaptación para FP2 de Sonia Estévez e Isabel Pita

1. Introducción

Todos los problemas son aplicaciones de consola que leen los datos por la entrada estándar y escriben los resultados por la salida estándar. La entrada siempre se va a ajustar a lo que describa el enunciado. De igual forma, la salida debe ajustarse exactamente a lo que indique cada enunciado.

Por cada problema, el juez dispone de un conjunto de casos de prueba. Cuando recibe el código de una supuesta solución, lo compila y lo ejecuta, enviándole por la entrada estándar esos casos de prueba. El programa escribirá en la salida estándar los resultados, que serán comparados por el juez con los resultados correctos, especificados por la profesora. El veredicto emitido dependerá del resultado de esa comparación.

Es importante resaltar el hecho de que el juez automático exige que todos los programas acaben con éxito (resultado de ejecución 0). Por lo tanto, es importante acabar la función "`main`" con un "`return 0`". En otro caso, el juez dará como veredicto un error de ejecución.

Todos los enunciados contienen un ejemplo de entrada y un ejemplo de salida, que se muestran para ilustrar el problema y que se separan entre sí por claridad y simplicidad. Pero las soluciones no tienen que leer toda la entrada, procesarla y luego escribir toda la salida. Más bien al contrario. Hacerlo así requeriría que el programa utilizara más memoria para guardar toda la entrada antes de empezar a hacer algo útil. Más abajo aparecen los esquemas de programa que deben seguirse.

Los programas deben probarse antes de ser enviados al juez automático, compilándolos y ejecutándolos en local. Y no solamente con los ejemplos sencillos que aparecen en el enunciado (que sirven para aclarar el problema más que para

[1] Este documento es una adaptación de la documentación de ayuda que aparece en las webs www.aceptaelreto.com y www.programa-me.com.

comprobar la corrección de la solución) sino con otros desarrollados para intentar descubrir los errores que pueda haber en la solución.

Todos los ficheros enviados al juez deben contener un comentario con:

```
// Autor/a: Nombre y apellidos
// email:
// Compilador y S.O. utilizado
// Nombre del problema
// Comentario general sobre la solución
```

El juez es online, por lo tanto, se debe abrir un navegador y teclear la siguiente url fp.fdi.ucm.es.

A continuación, se debe introducir el usuario y contraseña que os haya entregado la profesora.

2. Esquemas de programas

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, este 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 (centinela).
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 algo distinto. Pero en cualquier caso todos deben estar bien estructurados y comentados.

1. Casos de prueba limitados

Para el primer tipo de entrada, el esquema debe ser el siguiente:

```
// Autor/a: Nombre y apellidos
// email:
// Compilador y S.O. utilizado
// Nombre del problema
// Comentario general sobre la solución

#include <iostream>
#include <fstream>
#include <...>
#include "..."  // propios o los de las estructuras de datos de clase

// función que resuelve el problema
// comentario sobre el coste, O(f(N)), donde N es ...
Solucion resolver(Datos datos) {
   ...
}

// resuelve un caso de prueba, leyendo de la entrada la
// configuración, y escribiendo la respuesta
void resuelveCaso() {

   // leer los datos de la entrada

   Solucion sol = resolver(datos);

   // escribir solución
}

int main() {

       int numCasos;
       std::cin >> numCasos;
       for (int i = 0; i < numCasos; ++i)
              resuelveCaso();
       return 0;
}
```

La función "main" simplemente tiene el bucle que recorre todos los casos. Para cada caso, se llama a la función "resuelveCaso" que lee los datos asociados a un caso de prueba, lo resuelve y escribe la solución. Para resolver un caso se utiliza la función "resuelve" que recibe los datos concretos del problema, ya construidos a partir de la entrada, y los procesa.

2. Casos de prueba ilimitados acotados por caso de prueba especial (centinela)

Con el segundo tipo de entrada, la estructuración en funciones es la misma. Solamente cambia el bucle en la función "main" y cómo se detecta la condición de terminación en la función "resuelveCaso":

```
// Resuelve un caso de prueba, leyendo de la entrada la
// configuración, y escribiendo la respuesta
bool resuelveCaso() {
	// leer los datos de la entrada
	if (caso especial)
		return false;
	Solucion sol = resolver(datos);
	// escribir sol
	return true;
}
int main() {
	while (resuelveCaso());
	return 0;
}
```

3. Casos de prueba ilimitados

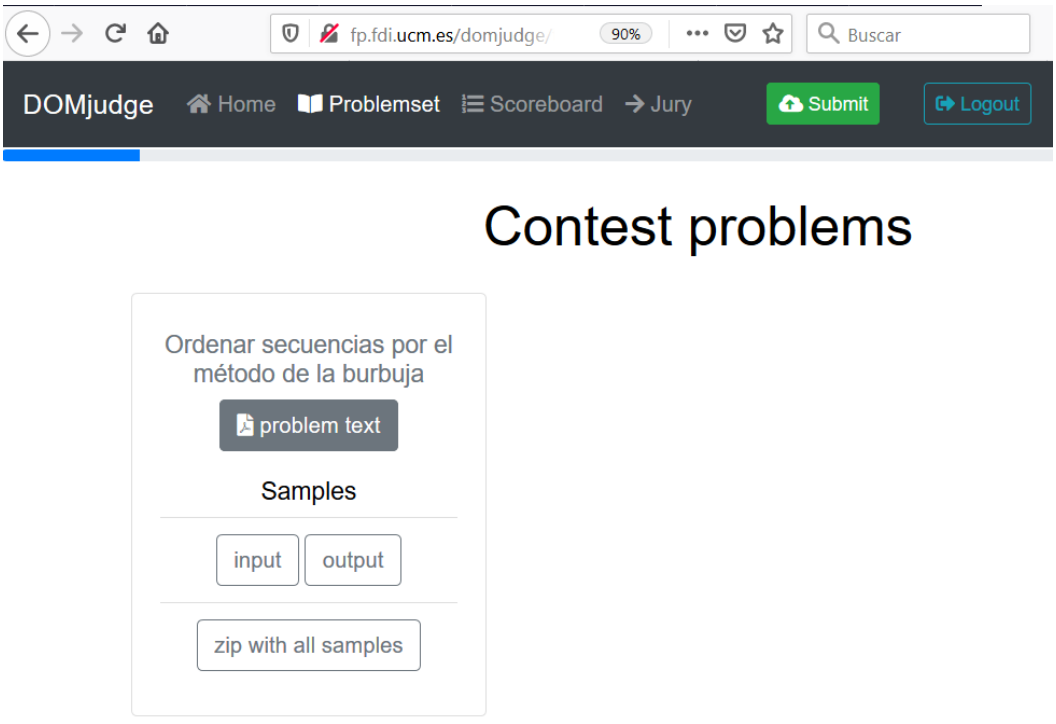
Solamente cambia la forma de detectar la terminación de los casos de prueba:

```
// Resuelve un caso de prueba, leyendo de la entrada la
// configuración, y escribiendo la respuesta
bool resuelveCaso() {

	// leer los datos de la entrada
	if (!std::cin) // fin de la entrada
		return false;

	Solucion sol = resolver(datos);

	// escribir sol

	return true;
}
int main() {
	while (resuelveCaso());
	return 0;
}
```

Probar los programas antes de enviarlos al juez

Las aplicaciones que se envían al juez leen los datos de la entrada estándar y escriben los resultados por la salida estándar. Es muy conveniente leer los datos de un fichero durante la fase de prueba local. De esta forma no es necesario introducir los datos por teclado en cada ejecución, sino que automáticamente el programa lee el fichero. Los ficheros de entrada y salida del juez los podemos descargar de la siguiente forma.

1) Accedemos al juez (fp.fdi.ucm.es)
2) Seleccionamos Problemset
3) Descargamos los ficheros input y output. Pueden existir varios casos de prueba.



Para probar el código en local, sin tener que modificar el código al subirlo al juez, basta con introducir las siguientes instrucciones en el programa. Estas instrucciones redireccionan a la entrada estándar el contenido de un fichero cuando se está ejecutando en local. El efecto es un programa que cuando se ejecute en el juez leerá los datos de la entrada estándar (como el juez requiere)

pero cuando se ejecute localmente leerá los datos del fichero que se indica en las instrucciones. Ojo, el fichero de entrada debe ser el que nos hemos descargado del juez y el fichero de salida se crea en cada ejecución. Con estas modificaciones el programa principal se adapta como se indica a continuación.

```
#include <fstream>

int main() {
	// ajustes para que cin extraiga directamente de un fichero
#ifndef DOMJUDGE
	std::ifstream in("datos.in");
	auto cinbuf = std::cin.rdbuf(in.rdbuf());
	std::ofstream out("datos.out");
	auto coutbuf = std::cout.rdbuf(out.rdbuf());
#endif

	// Main que corresponda según el apartado anterior

	// para dejar todo como estaba al principio
#ifndef DOMJUDGE
	std::cin.rdbuf(cinbuf);
	std::cout.rdbuf(coutbuf);
	system("PAUSE");
#endif
	return 0;
}
```

Ejemplo 1

El ejercicio de ordenar secuencias de números por el método de ordenación por inserción tiene una "Entrada" que dice: La primera línea dice el número de casos de prueba. A continuación, cada línea contiene un caso de prueba y cada caso de prueba contiene una secuencia de números que finaliza con el centinela 0. Por lo tanto, es un caso de pruebas limitados (4 casos).

```
4
12 3 20 31 43 23 123 0
9 8 7 6 5 4 3 2 1 0
0
1 2 3 0
```

Hacemos el siguiente programa

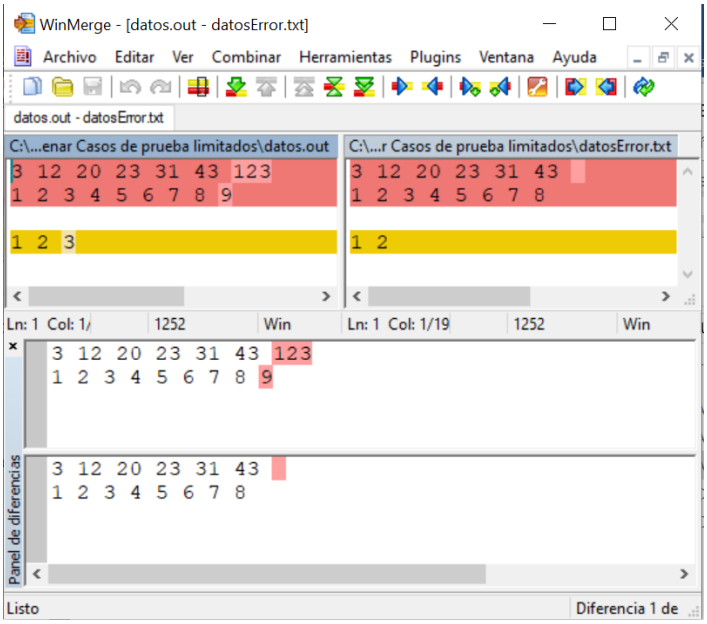
```
// Autor/a: Sonia Estévez Martín
// email: soesteve@ucm.es
// Compilador y S.O. utilizado: VS profesional 2013
// Nombre del problema: Ordenar por el método de la burbuja
// Comentario general sobre la solución: Datos de entrada limitados

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

const int N = 100;
typedef int tLista[N];

// función que resuelve el problema de ordenar los "total" elementos de la "lista"
void resolver(tLista lista, int total) {

// Introducir el código que corresponda
```

```
}

// Resuelve un caso de prueba, leyendo de la entrada que es una secuencia de números que
// finaliza cuando se encuentra al centinela “0”
// Se escribe la secuencia de números separados por un espacio

void resuelveCaso() {
	// leer la secuencia de números que finaliza con el centinela “0”
	// y se almacenan en un array llamado “lista”

	tLista lista;
	int num, i = 0;

	cin >> num;
	while (num != 0){
		lista[i] = num;
		i++;
		cin >> num;
	}

	// Es necesario pasar el número de elementos que contiene la lista
	resolver(lista, i);

	// Escribo la secuencia de números ordenados, separados por un espacio en blanco
	for (int j = 0; j < i; j++){
		cout << lista[j] << " ";
	}
	cout << endl;
}



int main() {
	// ajustes para que cin extraiga directamente de un fichero
#ifndef DOMJUDGE
	std::ifstream in("datos.in");
	auto cinbuf = std::cin.rdbuf(in.rdbuf());
	std::ofstream out("datos.out");
	auto coutbuf = std::cout.rdbuf(out.rdbuf());
#endif

	// caso de pruebas limitados (4 casos).
	int numCasos;
	std::cin >> numCasos;
	for (int i = 0; i < numCasos; ++i)
		resuelveCaso();

	// para dejar todo como estaba al principio
#ifndef DOMJUDGE
	std::cin.rdbuf(cinbuf);
	std::cout.rdbuf(coutbuf);
	system("PAUSE");
#endif
	return 0;
}
```

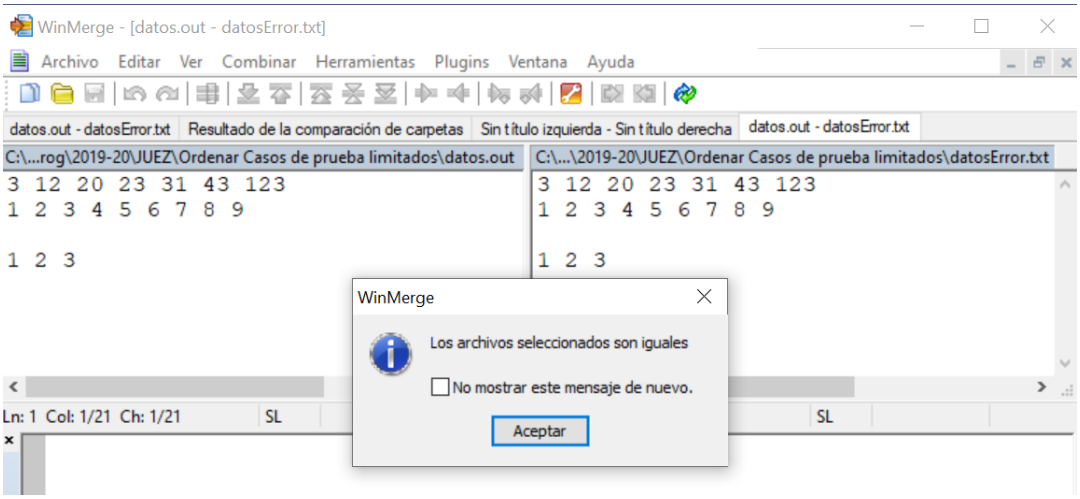
Compruebo en el fichero de salida si coincide con los datos del ejemplo.

```
3 12 20 23 31 43 123
1 2 3 4 5 6 7 8 9

1 2 3
```

En algunos casos es sencillo comprobar la salida. Pero en otros casos no es tan fácil. Por esto recomendamos usar algún programa de comparación de ficheros. Como por ejemplo WinMerge.
Supongamos que el fichero del juez es el anterior y el fichero que se acaba de generar en la prueba local es el mismo, pero falta el último elemento de cada fila. La comparación sería.

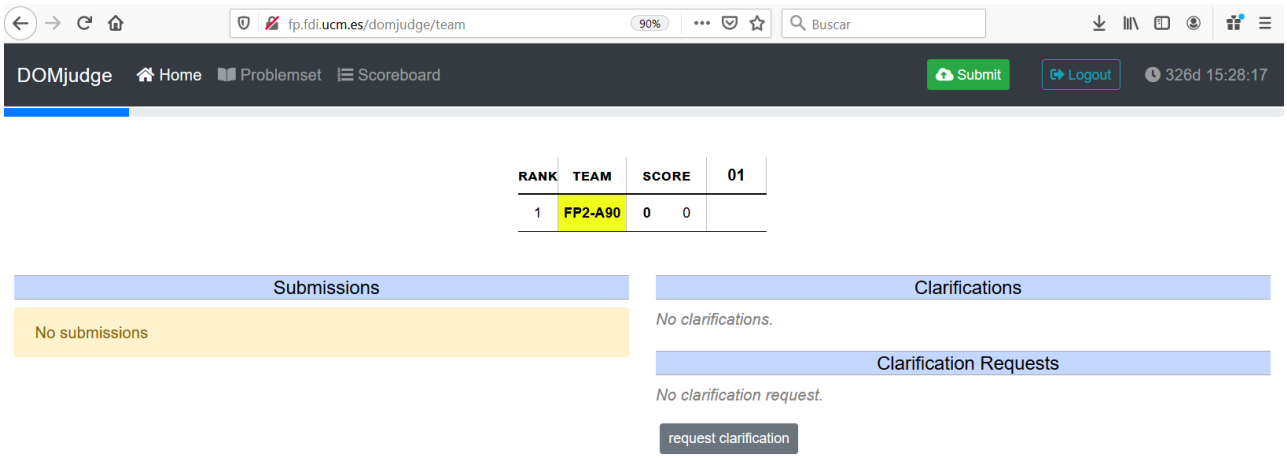


Cuando coindicen el fichero de salida de la prueba local y el fichero de salida del juez,

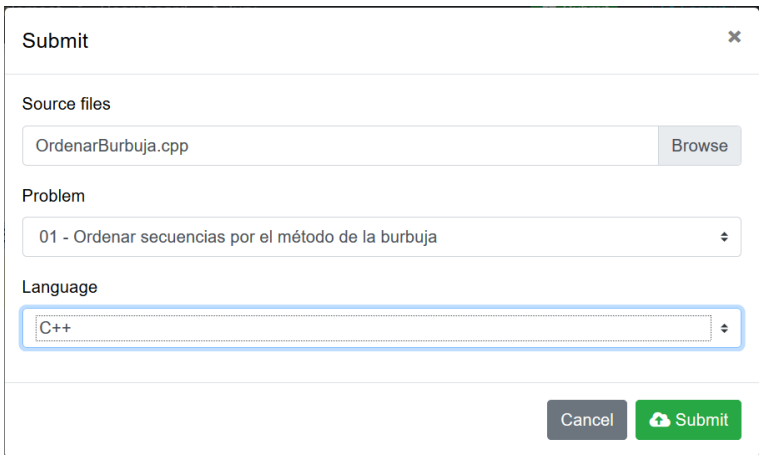


envío mi código al juez fp.fdi.ucm.es. de la siguiente forma.

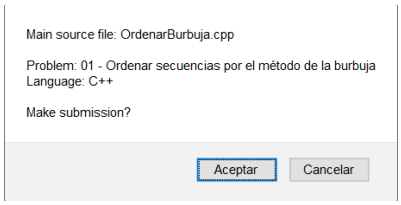
Accedo con el usuario y contraseña que me ha entregado la profesora y me aparece una ventana como



Después pulsar el botón Submit nos aparece la siguiente ventana.



Selecciono problema y lenguaje.
Pulso Submit



Acepto y espero el veredicto del juez.