Ejercicios Interactivos en Juez Evaluador de Problemas

Máster en Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid

Curso 2015/2016

Convocatoria de Junio

Nota: 8

Realizado por: Jéssica Martín Jabón

Dirigido por:

Marco Antonio Gómez Martín y Pedro Pablo Gómez Martín

Agradecimientos

Cuando terminé la Ingeniería Técnica en Informática de Gestión decidí embarcarme en el Grado de Ingeniería de Software, pensando que sería el último título, pero al finalizarlo me di cuenta que la realización de un Máster podría ser una experiencia enriquecedora y productiva.

Ahora puedo decir que sí, que ha merecido la pena el esfuerzo que le he dedicado a mi formación y una prueba de ello es este proyecto. En algunos momentos ha sido duro, pero tal vez por eso sea tan gratificante. No obstante, en este camino no he estado sola, tengo que agradecer a todas esas personas que son su apoyo han hecho posible no sólo que alcanzara mi meta sino que el camino fuera más fácil.

En primer lugar quiero agradecer a mis directores de proyecto, Marco y Pedro, que no sólo me han guiado en este proyecto sino que también lo hicieron en el trabajo de fin de grado.

No puedo dejar de dar las gracias a mi familia y en especial a mis padres, que pese a que ellos no han participado en este trabajo si han tenido que lidiar con mi estrés. Por último, dar las gracias a mi novio, por ayudarme y animarme en los momentos que más lo necesitaba.

Resumen

Los jueces online de problemas no son nuevos y como todo, tienden a mejorarse y a renovarse. Es en este punto donde cobran sentido los ejercicios interactivos, ejercicios en los que la solución a un problema se obtiene mediante la comunicación con el propio juez. Este trabajo trata de implantar y adaptar a un juez, diseñado en sus inicios para ejercicios tradicionales, los ejercicios interactivos.

Esta nueva funcionalidad no sólo está pensada para que los usuarios habituales tengan nuevos retos, sino que los alumnos que utilizan esta herramienta como apoyo en asignaturas de algoritmia tengan la posibilidad de reforzar sus conocimientos a través de este tipo de ejercicios, ya que con ellos se pueden plantear problemas que no tienen sentido con los ejercicios tradicionales.

Palabras Clave

Juez Online

Juez Evaluador

Ejercicios interactivos

Desafíos de programación

Auto aprendizaje

Abstract

The online judges are not new, and they, try to improve and renew. It is at this point where to appear the interactive exercises, in which the solution to a problem is obtained by communicating with the judge. This paper attempts to implement and adapt to a judge, designed in the beginning to traditional exercises, interactive exercises.

This new functionality is not only for regular users to have new challenges, also, students using this tool as support in subjects algorithmic have the opportunity to strengthen their knowledge through these exercises, because with them they it's possible to pose problems that make no sense with traditional exercises.

Key Words

Online judge

Evaluator judge

Interactive exercises

Programming Challenges

Auto learning

Índice

1 Introducción *1*

Motivación 1

Objetivos 1

Repercusión 2

Plan de trabajo 2

Estructura del documento 2

2 Introduction *3*

Motivation 3

Goals 3

Repercussion 4

Work plan 4

Document Structure 4

3 Estado del arte *5*

Juez online evaluador de problemas 5

Ejecución juez online 6

Ejercicios normales 7

Enunciado 7

Problemas 9

Ejercicios interactivos 10

Problemas técnicos de los ejercicios interactivos 11

Enunciado 11

Jueces interactivos 11

UVa Online Judge 12

SPOJ (Sphere Online Judge) 15

CodeForces 17

ACM International Collegiate Programming Contest 19

4 Estado inicial *23*

ACEPTA EL RETO 23

Problemas 24

Enunciado 25

Ranking 27

Estadísticas de un problema 27

Estructura de un problema 29

Generadores 31

Tipos de entrada 31

Estructura del proyecto **33**

Solución **33**

VEREDICTOS **34**

Accepted (AC): Aceptado 35
Presentation error (PE): Error de presentación 35
Wrong Answer (WA): Respuesta Incorrecta 35
Compilation error (CE): error de compilación. 35
Run-time error (RTE): Error durante la ejecución 35
Time limit exceeded (TLE): Tiempo límite superado 35
Memory limit exceded (MLE): Límite de memoria superado 35
Output limit exceeded (OLE): Límite de salida superado 35
Restricted function (RF): Función restringida 36

Visión general de un juez **36**

5 Desarrollo del proyecto ***37***

Arquitectura **37**

Implementación **39**

Enunciado **42**

Solución **44**

Juez **46**

VEREDICTOS **47**

Time Limit Exceeded (TLE) 47
Protocol Error (EP) 47
Judge Error (JKO) 48

6 Conclusiones ***49***

Valoración personal **49**

Trabajo futuro **50**

7 Conclusions ***51***

Personal Ratings **51**

Future work **52**

Bibliografía ***53***

1 Introducción

En este trabajo vamos a ver cómo son y cómo funcionan los ejercicios interactivos dentro de un juez online evaluador de programas. Puesto que esta temática no tiene por qué ser conocida para todo el mundo, primero se explicará que es un juez online, para qué sirven y que ventajas aporta.

La idea principal de estas aplicaciones es que de manera autónoma, un usuario pueda resolver un problema de programación y saber prácticamente al instante si su solución es válida o no. De esta forma, es posible entrenar conocimientos de algoritmia de manera autodidacta.

En mi caso, el proyecto que he realizado va a estar dentro de un conjunto de herramientas, que engloban la librería acrex y es parte del juez online Acepta el Reto, juez desarrollado en la Universidad Complutense de Madrid y que ya conocía porque mi trabajo de fin de grado fue sobre este juez.

Como comentaba anteriormente, existen multitud de jueces online en todo el mundo, el hecho de introducir los ejercicios interactivos supone un nuevo reto al usuario habitual. Debido a que la solución de los problemas sigue una estructura diferente y debe tener siempre en cuenta que para encontrar la solución al problema es necesario comunicarse con el juez. Por lo que la forma de plantear las soluciones es diferente a como se hacía antes. Esto resulta muy interesante, porque permite crear otros tipos de problemas que no tenía sentido proponer a través de los ejercicios tradicionales.

Motivación

Este proyecto pretende ser una ampliación del juez evaluador de problemas Acepta el Reto. Éste ya se utiliza en alguna de las asignaturas de algoritmia que se imparten en las titulaciones de Informática de la Universidad Complutense de Madrid.

Como he mencionado, esta aplicación ya era conocida para mí porque el proyecto que realicé como trabajo de fin de grado fue sobre ella. Este fue uno de los motivos que me hicieron decantarme por este proyecto. Para mí suponía un reto formar parte de una ampliación que supondría una nueva forma de resolver problemas abriendo así un nuevo abanico de posibilidades, tanto en el ámbito de los jueces de problemas como en el ámbito docente, pues podría afianzarse aún más, como herramienta de apoyo en las aulas.

Objetivos

El principal objetivo de este proyecto era realizar un desarrollo que permitiera que el juez fuera capaz de dar veredictos a los ejercicios interactivos. Estos ejercicios no sólo son útiles para aquellas personas interesadas en la algoritmia, sino que puede convertirse en una herramienta básica para los profesores que imparten asignaturas de estructura de datos y algoritmia,

porque les permiten proponer problemas a los alumnos para que los resuelvan, y gracias a que las soluciones se evalúan al instante le permiten detectar al alumno errores, que de otra forma sería mucho más complicado detectar por ellos o simplemente necesitaría la supervisión del profesor.

Repercusión

La realización de este proyecto tiene una gran repercusión desde todos los puntos de vista.

Si pensamos en la visión de los usuarios finales, vemos como con este tipo de problemas, se enfrentan a nuevos retos, nuevas formas de pensar y nuevas formas de resolver un problema, ya que ahora tendrán que encontrar la solución a raíz de las preguntas que hagan a un juez y de las respuestas que obtengan del mismo.

Si lo vemos desde el punto de vista de los profesores, que utilizan esta herramienta como apoyo para los estudiantes, van a poder verse beneficiados ya que ahora van a poder proponer más tipos de ejercicios, lo que le ayudará a tener un refuerzo para una parte más amplia del temario.

Desde el punto de vista de la aplicación también habrá beneficios, si es posible resolver más tipos de problemas esto atraerá a nuevos usuarios, lo que potenciará el aumento de envíos y la visibilidad en el ámbito de los jueces online.

Plan de trabajo

Para realizar este trabajo se ha optado por una estrategia, con reuniones semanales en las que se iban realizando pequeños incrementos. En esas reuniones no sólo se mostraba lo que se había hecho durante toda la semana, sino que también comentaba las dificultades que había tenido y se acordaba el incremento de la semana siguiente.

Estructura del documento

Este trabajo tiene dos partes diferenciadas, una es la que va orientada a los programadores y otra la que va destinada al usuario final.

Programadores: en los apartados de arquitectura e implementación se ahondará de manera más técnica en que consiste la implementación de los ejercicios interactivos, así como la arquitectura inicial de la que se disponía y el resultado después de la adaptación.

Usuario final: en los apartados centrados en el uso de la aplicación se verá la manera en que un usuario puede resolver los ejercicios interactivos, así como las ventajas que ofrecen este tipo de problemas.

2 Introduction

In this paper we will see how interactive exercises are and how work within an online program evaluator judge. Since this concept may not be known to everyone, first I will explain what an online judge is and their advantages.

The main idea of these applications is that autonomously, a user can solve a programming problem and know at the time if his solution is valid or not. Thus, it is possible to train knowledge about algorithms.

In my case, the project I have done will be within a set of tools, which include the ACREX library and this is the core of the judge online Acepta el Reto, which was developed at the Complutense University of Madrid and I already knew because I made my final degree project about this judge too.

As I said above, there are many online judges around the world, the fact to introduce interactive exercises is a new challenge to the regular user. Because the solution of problems follow a different structure and users must always be aware that they have to communicate with the judge to find the solution to the problem. So the approach to the solutions is different than it was before. This is very interesting because it allows you to create other types of problems that made no sense to propose with traditional exercises.

Motivation

This project aims to be an extension of the evaluator judge Acepta el Reto, which is already used in some of the subjects about algorithms taught in Computing degrees at the Complutense University of Madrid.

As I mentioned, this application was already known to me because my final degree project was about it. This was one of the reasons whereby I chose this project. For me is a challenge to be part of an increase that would mean a new way of solving problems and opens a whole new range of possibilities, both in terms of the judges of problems and in teaching because it could consolidate like a support tool in classrooms.

Goals

The main objective of this project was to develop a feature that would allow the judge to give verdicts to interactive exercises. These exercises are not only useful for those interested in algorithms, but it can become a basic tool for teachers who teach subjects of data structures and algorithms, as it allows them to propose problems for students to solve, and thanks that the solutions are evaluated instantly they are allowed to detect the errors they have made, what would be much more difficult to detect otherwise or simply they would need the supervision of the teacher.

Repercussion

The realization of this project has a great impact from all points of view.

If we think of the vision of end users, we see that with this kind of problems, they are facing new challenges, new ways of thinking and new ways of solving a problem, as now they have to find the solution through questions made to a judge and the responses that the judge responds.

If we see it from the point of view of teachers, who use this tool as a support for students, they will be able to be benefited as they will now be able to propose more types of exercises, which will help students to have a reinforcement for the subjects.

From the point of view of the application, it will also be beneficial to have the ability of solving more types of problems, since it is possible that this feature will attract new users, which will boost shipments and increased visibility in the field of online judges.

Work plan

In order to perform this work, I have opted for a flexible strategy, generating small increments reviewed with weekly meetings. In these meetings I did not only show what I had done throughout the week, also I commented on the difficulties that I had and the work for the next week was decided too.

Document Structure

This work has two distinct parts, the first one is oriented to programmers and the other one is intended for the end user.

Programmers: in architecture and implementation sections, I will go deeper in a technical way to explain the implementation of interactive exercises as well as the initial architecture which were available for me and the result after the adaptation done.

User: in sections focused on the use of the application I will expose the way a user can solve interactive exercises, as well as the advantages of such problems.

3 Estado del arte

Juez online evaluador de problemas

Para poder comprender este trabajo es importante conocer ciertos términos como qué es un juez online.

Obviamente este tipo de sistema está orientado a personas con conocimientos de programación, y sirve para practicar y coger destreza en este campo o incluso para entretenerse, si programar en uno de nuestros hobbies.

Aunque no sólo sirve para reforzar conocimientos en el caso de que estés estudiando, sino que también sirve de entrenamiento a aquellas personas que participan en concursos de programación. De hecho, algunos de los jueces online organizan sus propios concursos.

Un juez online es un sistema que permite a los usuarios practicar sus conocimientos de programación, mediante la resolución de problemas propuestos en dicha aplicación. La idea es sencilla, se trata de subir un código que resuelva un problema y el juez será capaz de compilarlo, ejecutarlo y dar un veredicto. El más esperado es el AC (Accepted) lo que supondría que la solución enviada es correcta.

Generalmente estos sistemas están disponibles en páginas web, las cuales tienen varios apartados o secciones. Las más comunes son, lista de problemas, página de envío de soluciones y ranking, aunque por supuesto cada uno puede tener otras secciones, como por ejemplo, foros o estadísticas.

En la sección de la lista de los problemas se encuentran todos los problemas que ese juez tiene disponible. En la mayoría de los casos suelen estar agrupados por temáticas o por volúmenes para localizarlos fácilmente, aunque también en muchas ocasiones es posible realizar búsquedas. Si los problemas se agrupan por temáticas generalmente suelen ser por el tipo de algoritmo o técnica de programación por el que el problema se debe resolver. Si por ejemplo, vemos una temática que se llama “Recursión”, significa que los problemas propuestos serán normalmente resueltos mediante recursividad. Esto puede resultar muy útil a los usuarios, ya que pueden buscar problemas en función de qué quieren practicar.

Los rankings son tablas que, por ejemplo, suelen mostrar los usuarios que más ejercicios han resuelto. Es una manera de tener una competición abierta con todos los usuarios participando a la vez.

Una sección muy importante es la del envío del problema, como he comentado anteriormente. El código que se envía al juez debe ser una solución a un problema determinado. En esta página es habitual que haya un editor de texto, en el cual se puede escribir el código o simplemente pegarlo

si ya lo teníamos implementado en otro sitio. En algunos jueces también es posible adjuntar un fichero con el código, obviamente la extensión de dicho fichero debe identificar al lenguaje de programación con el que se ha desarrollado, si ha sido en java, .java, si ha sido en C++, .cpp… Una vez que el código está subido o puesto en el editor, es necesario seleccionar el lenguaje de programación. Generalmente se hace mediante la selección en un combo, el motivo por el que es necesario seleccionarlo es porque los jueces suelen soportar más de un lenguaje de programación, lo que quiere decir que los problemas pueden solucionarse con varios lenguajes.

Después de eso ya puedes enviar el código y esperar el veredicto del juez, para verlo, dependiendo de cada juez online se podrá ver en desde diferentes sitios. Los sitios más habituales pueden ser en una sección de problemas enviados por el usuario o tal vez, una vez enviado el problema te redirija a una página donde se ven todos los envíos que el juez tiene pendientes, porque no hay que olvidar que el juez es un sistema y si está ocupado compilando y ejecutando una solución y le llega otro envío, éste deberá esperar a que el juez se libere, por este motivo, es posible que cuando un usuario envíe un problema, tenga que esperar un lapso de tiempo hasta que aparezca el veredicto.

En cualquier caso el formato de la tabla en el que se ve el veredicto suele ser similar en todos los jueces online, contiene varias columnas tales como: el identificador del problema, el título del problema, el veredicto, el lenguaje de programación que se ha utilizado para resolver el ejercicio y el tiempo que ha tardado en ejecutarse, aunque es posible que en algunos jueces haya más columnas como por ejemplo, la memoria usada.

Una vez que el problema se envía el juez lo compila, si el código no compila el juez muestra el veredicto de error de compilación (CE) si el código compila, entonces procede a ejecutarlo, una vez ejecutado procederá a dar un veredicto. En el capítulo Estado inicial profundizaremos más sobre los veredictos.

Ejecución juez online

Para entender cómo el juez es capaz de dar un veredicto, es necesario entender el concepto de problema. Es posible pensar que un problema es lo mismo que un enunciado, pero no es así. Un problema contiene, al menos, un enunciado, unos casos de prueba (.in) y las respuestas a esos casos (.out).

El fichero de entrada (.in) almacena los datos de entrada para los que se va a probar la solución. A continuación, veremos un ejemplo de un fichero .in. Vemos que hay cuatro líneas, una por cada caso de prueba.

```
100 -10
-10 -100
-10 100
100 -1000
```

El fichero de salida (.out) es el que va a almacenar el resultado de la ejecución. Existe un fichero de salida que almacena los resultados de referencia y otro para la solución del usuario.

```
SI
NO
SI
NO
```

Una vez que el juez ha generado el .out de la solución del usuario, se procede a la comparación del .out del usuario con el .out oficial, para determinar si la solución obtenida con el código del usuario es correcta y así poder dar un veredicto.

El funcionamiento básico de un juez online es el siguiente, la solución que envía un usuario corresponde a un problema concreto, a continuación el juez la compila y la ejecuta con los casos de prueba correspondientes a ese problema. Las respuestas que genere se almacenarán en un fichero de manera transparente al usuario. Es transparente porque el usuario para dar la solución lo hace por la salida estándar y no escribiendo en un fichero.

En el caso de que la ejecución de la solución haya finalizado correctamente, se procederá a comparar estos resultados con los del .out asociado al problema.

Ejercicios normales

Ahora que ya hemos visto a grandes rasgos lo que es y cómo funciona un juez online es importante saber, qué nos puede aportar.

El primer tipo de ejercicios que vamos a tratar son los ejercicios normales. Los he denominado así porque son el tipo más común que se utiliza en los jueces online. Es así porque estos ejercicios abarcan diferentes técnicas de programación y diferentes complejidades.

Enunciado

Para entender mejor estos problemas pensemos en un caso concreto, imaginemos que alguien está aprendiendo a programar y empieza a resolver problemas de complejidad baja, por ejemplo, esa persona quiere diferenciar entre números pares e impares. En el enunciado que vamos a ver a continuación se plantea justo el ejemplo que acabo de proponer.

Como podemos ver, el objetivo es averiguar si un número es par o impar, pero realmente la solución que debemos devolver es izquierda y derecha. Esto se debe a que el contexto en el que se ambienta el problema relaciona los números impares con las viviendas situadas a la izquierda y los pares con las situadas a la derecha. Realmente deben hacer lo mismo que con el ejercicio que yo propuse, pero de esta manera al usuario se le cuenta una historia que hace que el problema les atraiga más y quieran resolverlo. Este

es un problema muy sencillo, pero plasma perfectamente el aspecto de un ejercicio normal.

Si nos fijamos en la estructura del enunciado[1], tiene una descripción del problema, una explicación de la entrada que va a recibir la solución, una explicación de la salida que debe devolver y unos ejemplos de cómo sería la entrada y la salida.

Por supuesto que se pueden proponer problemas de complejidad más avanzada, no olvidemos que este modelo de juez online se puede utilizar en concursos como el SWERC[2], cuyos participantes deben ser estudiantes universitarios de la rama de informática. Esto significa que puede haber ejercicios en los que se necesiten utilizar estructuras de datos, como grafos o que haya que aplicar metodologías como vuelta atrás.

[1] https://www.aceptaelreto.com/problem/statement.php?id=217
[2] https://icpc.baylor.edu/regionals/finder/swerc-2016

Problema número 217

¿Qué lado de la calle?

Tiempo máximo: 1,000 s Memoria máxima: 4096 KiB

http://www.aceptaelreto.com/problem/statement.php?id=217

Cuando necesitamos llegar a una dirección específica, es útil conocer la manera en la que las casas se numeran dentro de una calle para no perder demasiado tiempo buscando. Dependiendo de las reglas urbanísticas de cada lugar, esa numeración seguirá unas reglas u otras. En Japón, por ejemplo, los números se asignan por orden de construcción de los edificios, lo que no es algo particularmente amigable para un despistado turista.

En la mayoría de las ciudades y pueblos de Europa, sin embargo, se utiliza un mecanismo que facilita la búsqueda, aunque dificulta los cambios si se construyen o derrumban edificios. En concreto, se elige uno de los extremos como el *inicio de la calle*, utilizándose normalmente como criterio aquél que esté más cerca de un punto significativo de la ciudad (el centro urbano, el ayuntamiento, un río, el mar...). Una vez hecho eso, las viviendas que quedan al lado izquierdo (respecto al punto inicial) reciben números impares consecutivos, y las que quedan al lado derecho números pares consecutivos. Dependiendo del tamaño de cada edificio, a menudo ocurre que los números se descompensan y hay portales que tienen enfrente otros con números muy diferentes.

Entrada

El programa recibirá, por la entrada estándar, un conjunto de casos de prueba, cada uno en una línea. Cada caso de prueba estará compuesto de un número de vivienda. Se garantiza que nunca será mayor que 1.000.

El último caso, que no deberá procesarse, será el número `0`.

Salida

Para cada caso de prueba el programa escribirá "`IZQUIERDA`" (sin las comillas) si la vivienda está situada a la izquierda de la calle, y "`DERECHA`" si está a la derecha.

Entrada de ejemplo

```
3
10
41
0
```

Salida de ejemplo

```
IZQUIERDA
DERECHA
IZQUIERDA
```

Autores: Pedro Pablo Gómez Martín y Marco Antonio Gómez Martín.

Problemas

Es cierto que hay multitud de posibilidades a la hora de desarrollar un enunciado para un juez online. De hecho podemos pensar que todos los problemas de programación pueden ser evaluados con este tipo de sistema, pero eso no es del todo cierto.

Pensamos en un problema en el cual el usuario deba encontrar si un número está en un array de números, por ejemplo, el número que el usuario debe encontrar es el 3 y el array de números tiene una longitud de 10 elementos (1, 3, 5, 7, 8, 11, 15, 16, 20, 23) y sabemos que los números están ordenados

en orden creciente. Pero no sólo queremos saber si está ese número, sino que lo haga de una manera eficiente.

Es obvio que lo que queremos es que el problema se resuelva mediante una búsqueda binaria, sin embargo, si recordamos cómo funcionan los jueces, vemos que algo falla. Como hemos descrito anteriormente la solución del usuario lee una entrada, es decir, en este caso leería los 10 números, esto significa que mientras lee la entrada puede ir viendo en qué posición se encuentra el número. Como vemos, un problema cuya intención era realizar la búsqueda binaria, se ha traducido en una simple lectura por la entrada estándar.

La razón por la que el juez no puede saber si el problema se ha resuelto mediante una búsqueda binaria es porque al leer la entrada la complejidad del problema es lineal, por lo que aunque se resolviera por búsqueda binaria que es de complejidad logarítmica el resultado queda enmascarado por la complejidad de la lectura.

Ejercicios interactivos

Acabamos de ver que los ejercicios normales son poco útiles en determinados casos. Para solventar esta carencia, han aparecido los ejercicios interactivos.

Su nombre se debe a que la solución del usuario debe interactuar con el juez para encontrar la solución al problema. La manera en la que se comunican es mediante preguntas, la solución realiza una pregunta al juez y éste la responde, con esta respuesta la solución del usuario se va aproximando a la respuesta final.

Al principio este concepto puede resultar un poco confuso, pero retomemos el ejemplo anterior en el que buscábamos un número mediante búsqueda binaria.

Ahora la solución del usuario no va a leer todo el rango de números del 1 al 23, sino que, por ejemplo, va a leer un 10 y eso le indica al usuario que la longitud del array en el que tiene que buscar el número es 10, además de leer cuántos elementos hay es necesario saber cuál es el número a buscar, para seguir con el ejemplo el 3.

Con estos datos la solución del usuario no es capaz de saber si el número está o no, por ello es necesario la colaboración del juez. Como el problema es de búsqueda binaria, preguntaríamos al juez cuál es el elemento situado en la quinta posición del array, en este caso es un 8. Éste respondería 8, y al ser 8 mayor que 3, nos quedaríamos con la mitad izquierda y repetiríamos la operación hasta encontrar la respuesta.

De esta manera se solventa el problema de la lectura de entrada, la cual ya no es lineal y ahora sí se puede ver si la solución del usuario es una solución eficiente o no.

Problemas técnicos de los ejercicios interactivos

Es cierto que los ejercicios interactivos permiten plantear otro tipo de problemas, pero éstos también tienen unos inconvenientes. Como he comentado, para solucionar el problema es necesario que la solución del usuario esté conectada con el juez y viceversa. Lo que significa que van a tener que compartir un canal de comunicación.

Por ejemplo, el juez escribe por la salida estándar lo que quiera que la solución del usuario lea por la entrada estándar. Para realizar esta comunicación se utiliza un buffer y cada vez que el juez escribe en él es necesario limpiarlo para evitar mandar información incorrecta. Por eso cada vez que se hace una escritura por la salida estándar es necesario utilizar la función flush para liberar el buffer y que se pueda utilizar sin problemas en la siguiente comunicación. La función flush está disponible en multitud de lenguajes, entre ellos, C++, C y Java, que son los más usados en este tipo de ejercicios. Este problema se repite en el sentido contrario, de la solución del usuario, al juez.

Enunciado

En el apartado anterior vimos cómo es un enunciado para un ejercicio no interactivo, pero ahora ese modelo de enunciado no es válido, ya que antes el usuario sólo tenía que leer unos datos de entrada para tratarlos y dar un resultado final. Sin embargo, en este tipo de problemas para obtener la solución es necesario interactuar con el juez mediante la realización de preguntas. Por lo que, además de describir lo que el ejercicio debe hacer, es necesario explicar cuál va a ser el protocolo de comunicación para que el código del usuario hable con el juez. Hay que tener en cuenta, que la implementación que realiza el usuario no sólo tiene que realizar preguntas al juez, sino que también debe poder recibir las respuestas que éste le vaya a proporcionar.

Más adelante profundizaremos sobre cómo funciona un juez para problemas interactivos. Pero lo más importante es saber que la solución que envía el usuario debe comunicarse con el juez y viceversa, y que la respuesta que la solución del usuario devuelve sólo es posible obtenerla a través de preguntas que la implementación del usuario haga al juez y las respuestas que éste dé.

Jueces interactivos

Como comentaba en el apartado de la introducción, existen otros jueces de programación aparte de Acepta el Reto, pero no todos soportan ejercicios interactivos, de hecho, hay muy pocos jueces que los soporten.

A continuación, vamos a ver cuatro jueces que ofrecen este tipo de ejercicios. Cabe destacar que pese a que incorporan estos ejercicios, representan un porcentaje minoritario frente a los ejercicios tradicionales. Para hacernos una

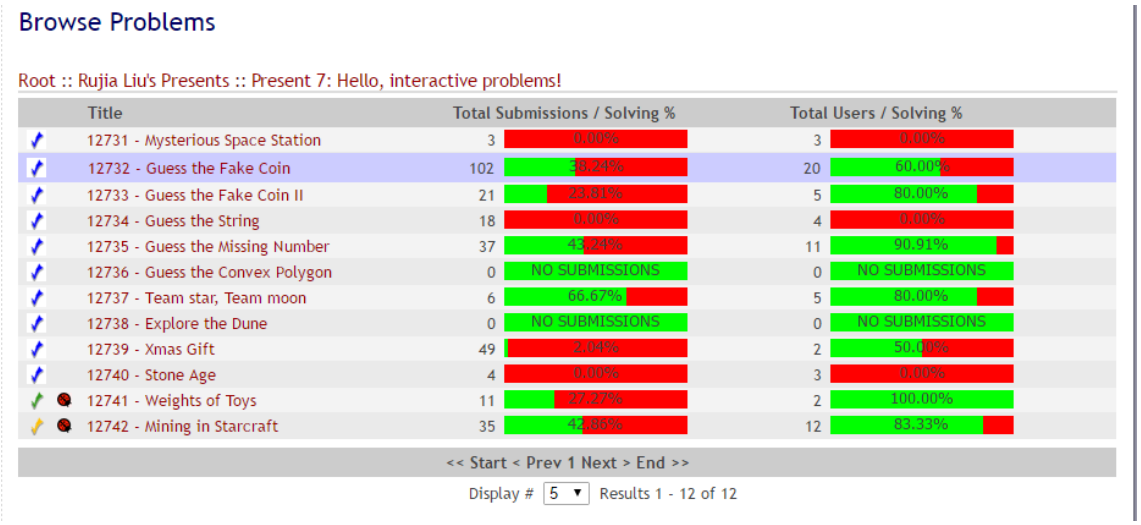
idea de lo poco integrados que están todavía, la mayoría de los jueces que vamos a ver en los siguientes apartados organizan concursos pero en ninguno introducen este tipo de ejercicios.

UVa Online Judge

Juez online [2] desarrollado en la Universidad de Valladolid por Miguel Revilla en 1995 y que empezó a estar operativo para todo el público en 1997. Tienen más de 4.300 problemas y más de 800.000 usuarios, esto en parte es debido a que los enunciados están en inglés, lo que permite que puedan llegar a todo el mundo.

En este juez no sólo está la posibilidad de resolver los problemas, sino que también organizan concursos online en los que se puede participar desde cualquier parte del mundo.

Aunque Miguel fue el que desarrolló este juez, ha tenido diferentes colaboradores que han ayudado a mejorarlo, uno de ellos es Rujia Liu's, con el que se ha conseguido introducir los ejercicios interactivos. Actualmente sólo disponen de un conjunto de 12 problemas, que si los comparamos con los más de 4.300 que tienen de los problemas tradicionales, supone un porcentaje muy pequeño. Además, hay que destacar que están separados del resto de problemas y no es sencillo encontrarlos a través de la navegación de su web. De hecho, pese a que el número de usuarios es muy alto, hay problemas que todavía no tienen ningún envío, como se puede apreciar en la siguiente imagen.



Para cada problema se dispone de un enunciado, éstos son similares a los de los problemas tradicionales, pero existe una diferencia, es necesario que el código del usuario siga unas pautas concretas para comunicarse con el juez. Para explicar el funcionamiento de la comunicación, añaden una tabla con los comandos que debes utilizar, así como una breve descripción de lo que significan.

There are n coins. One of the coins is fake: it is slightly heavier than the other coins, and all other coins have the same weight.

Your task is to find out the fake coin.

Interaction Protocol

Your program should read from standard input, and write to standard output. After printing each line to the standard output, you should flush the output, by calling `fflush(stdout)` *or* `cout << flush` *in C/C++,* `flush(output)` *in Pascal and* `System.out.flush()` *in Java. Please read general instructions for interactive problems for more information.*

First, read the number of test cases T ($1 \leq T \leq 100$). For each test case, read an integer n ($3 \leq n \leq 120$) in the first line, then issue one or more '`Test`' command, followed by an '`Answer`' command.

Command	Description
`Test` L_1 $L_2 \ldots R_1$ $R_2 \ldots$	Places coins $L_1, L_2 \ldots$ on the left side and $R_1, R_2 \ldots$ on the right side ($1 \leq L_i, R_i \leq n$), and returns the result a. $a = 1$ means $left > right$, $a = -1$ means $left < right$, $a = 0$ means $left = right$. There should be an even number of coins, and the first half are placed on the left side. No coin should appear in a command twice.
`Answer` b	Tell us your answer. This command does not return anything.

If your program violated any of these rules (bad format, invalid arguments etc), the server will exit immediately, and you will receive Protocol Violation (PV).

Protocol Limit

For each test case, you can issue at most 5 '`Test`' commands, **otherwise you'll get Protocol Limit Exceeded (PLE).**

Sample Interaction

```
1
9
                        Test 1 2 3 4 5 6
-1
                        Test 4 5
0
                        Answer 6
```

En la imagen de arriba vemos un enunciado[3] real del juez online de la UVa. En este caso la descripción del problema es muy corto y sencillo, se trata de adivinar cuál es la moneda falsa. Sólo hay una y se diferencia de las otras porque su peso es distinto, pero lo que en realidad nos importa, es ver los cambios en comparación con los ejercicios tradicionales.

En la sección Interaction Protocol se describe cómo va a ser la comunicación para este problema, en el que la solución del usuario empezará leyendo por la entrada estándar el número de casos de prueba que se van a ejecutar. Además recuerda que después de realizar una escritura es necesario hacer

[3]https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=826&page=show_problem&problem=4585

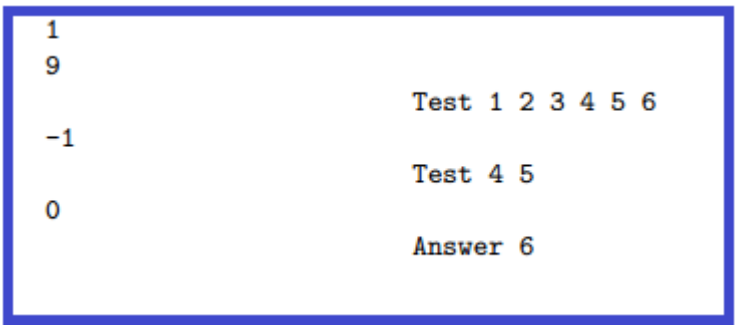
flush, como ya hemos visto en el apartado de Problemas técnicos de los ejercicios interactivos.

Como hemos estado viendo la solución del usuario realiza preguntas al juez para averiguar la respuesta, y cuando la encuentra la devuelve. Por lo que es inevitable tener un protocolo de comunicación para que el juez sepa cuándo la solución está preguntando y cuándo le ha enviado una respuesta.

Esa es justo la información que vemos en la tabla, para este caso concreto se van a utilizar dos comandos, Test y Answer, como sus nombres indican Test se utilizará para realizar la pregunta, para saber cuál es el formato de la pregunta en la columna Description de la tabla se explica la manera correcta de formular la consulta.

Cuando la solución ya ha encontrado la respuesta se la envía al juez precediendo el comando Answer.

Para ayudar a entender cómo funciona la comunicación, en cada enunciado se escribe un ejemplo de cuál es la comunicación, en la sección Sample Interaction. La parte de la izquierda representa la información que el juez manda y la parte derecha es un ejemplo de lo que el código del usuario debe mandar al juez, como podemos ver la comunicación la empieza el juez indicando que sólo hay un caso de prueba y que contiene 9 monedas. Con esta información la implementación del usuario ya puede empezar a preguntar al juez y eso es justo lo que vemos en la parte derecha:

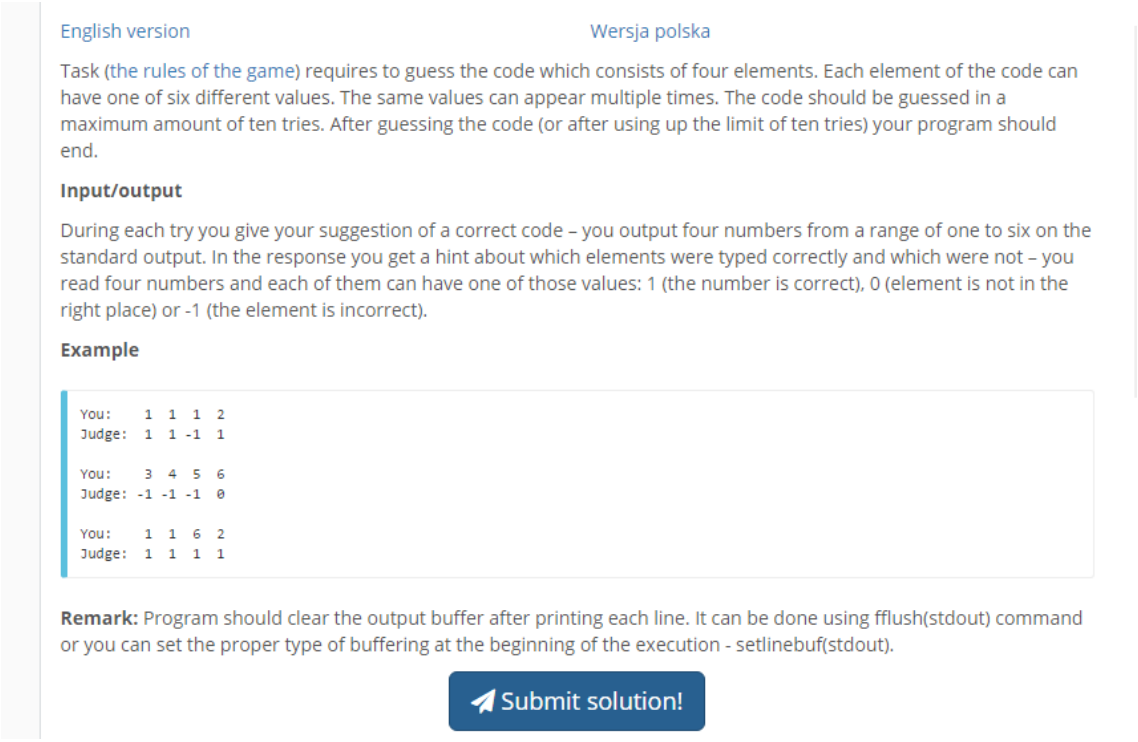


Hasta este punto podemos pensar que la solución siempre se va a encontrar, ya que el usuario realiza preguntas hasta que da con la solución, para evitar una comunicación infinita, los ejercicios interactivos tienen un número máximo de preguntas que se pueden realizar, si pasado ese número de intentos no se proporciona una solución, este juez da el veredicto Protocol Limit, aunque hay jueces que dan otro tipo de veredicto para estos casos.

SPOJ (Sphere Online Judge)

Es un juez evaluador de problemas [4] que soporta 45 lenguajes de programación y varios compiladores. Además de ser una herramienta que permite resolver problemas, también organizan concursos y retos y disponen de un foro en el que los usuarios pueden comentar las soluciones e incluso pueden pedir ayuda. Según consta en su página web, desde 2012 han organizado más de 2.400 concursos. Tiene más de 200.000 usuarios y más de 20.000 problemas para resolver.

Este juez ha empezado a introducir los ejercicios interactivos, aunque dispone de menos ejercicios que el juez anterior, en concreto sólo tiene cuatro problemas. Todos los enunciados siguen el mismo patrón, un texto que describe lo que el programa debe hacer, una descripción de cómo debe ser la entrada y la salida, una imagen que ilustra un ejemplo de comunicación y por último una nota que te recuerda que debes usar la función flush para asegurarte de que la comunicación se realiza. A continuación vemos un enunciado[4] a modo de ejemplo:



English version Wersja polska

Task (the rules of the game) requires to guess the code which consists of four elements. Each element of the code can have one of six different values. The same values can appear multiple times. The code should be guessed in a maximum amount of ten tries. After guessing the code (or after using up the limit of ten tries) your program should end.

Input/output

During each try you give your suggestion of a correct code – you output four numbers from a range of one to six on the standard output. In the response you get a hint about which elements were typed correctly and which were not – you read four numbers and each of them can have one of those values: 1 (the number is correct), 0 (element is not in the right place) or -1 (the element is incorrect).

Example

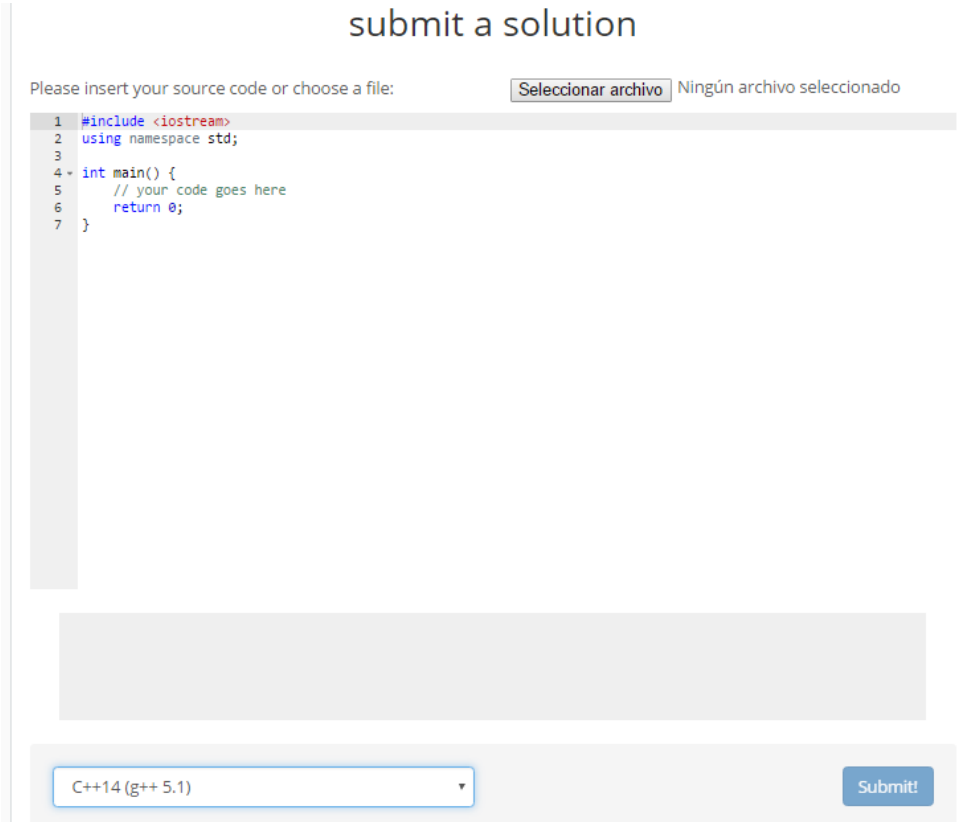
```
You:    1  1  1  2
Judge:  1  1 -1  1

You:    3  4  5  6
Judge: -1 -1 -1  0

You:    1  1  6  2
Judge:  1  1  1  1
```

Remark: Program should clear the output buffer after printing each line. It can be done using fflush(stdout) command or you can set the proper type of buffering at the beginning of the execution - setlinebuf(stdout).

Submit solution!

El envío de la solución del problema es como la del resto de problemas. Existen dos maneras de enviar el código. La primera es escribirlo directamente en el editor de texto. La segunda forma es subir un fichero. También debes seleccionar el lenguaje y por último darle a submit (Enviar).

[4] http://www.spoj.com/problems/MSTRMND/



Este juez no sólo permite a los usuarios resolver los problemas interactivos, sino que también es posible que los usuarios desarrollen sus propios problemas e incluso los pueden subir, para ello proporcionan una dirección de correo electrónico para solicitar permiso para subir ejercicios, aunque bien es cierto, que yo no lo he solicitado, por lo que desconozco los requisitos que pueden pedir.

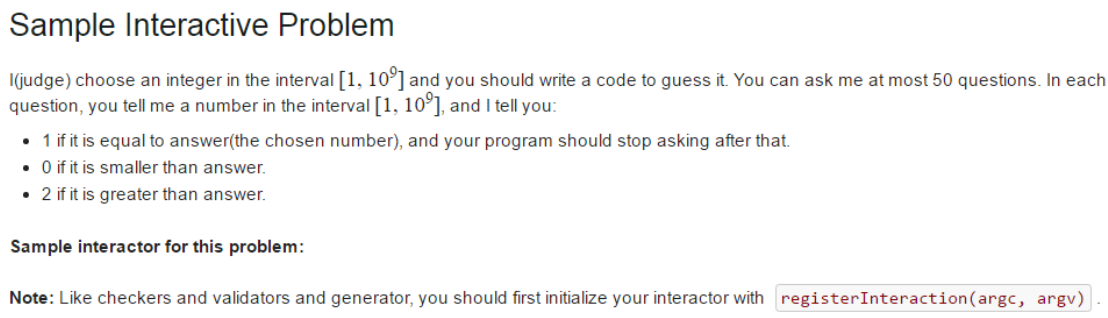
Para conseguir que los usuarios puedan subir ejercicios y que todos sigan una estructura similar, en la web hay un apartado con pautas a seguir para que puedas desarrollar tu juez evaluador. En esa guía aparece una interfaz para homogeneizar el desarrollo de los problemas.

CodeForces

Juez online [3] de origen Ruso creado y mantenido por un grupo de programadores liderado por Mikhail Mirzayanov de la Universidad Estatal de Saratov. Está orientado principalmente a los concursos, pero aunque los usuarios no participen, sí que pueden resolver los problemas como en un juez normal.

Lo más interesante de esta web es que tiene un blog, y en él se pueden encontrar entradas sobre los problemas interactivos. De hecho, una de las entradas, no sólo explica lo que son sino que proporciona información para desarrollar un juez que resuelva estos problemas. Además, muestra un ejemplo de un problema y su posible solución.

En la imagen de abajo podemos ver un enunciado[5] de un problema interactivo. Es el clásico Adivina mi número, el juez "piensa" un número entre un rango que el usuario también conoce y éste debe averiguar cuál es. Como se puede leer en el enunciado, hay un límite de 50 intentos para resolver el problema. También proporciona información sobre los datos que te va a devolver el juez, así como una pequeña descripción de lo que significan.



Sample Interactive Problem

I(judge) choose an integer in the interval $[1, 10^9]$ and you should write a code to guess it. You can ask me at most 50 questions. In each question, you tell me a number in the interval $[1, 10^9]$, and I tell you:

- 1 if it is equal to answer(the chosen number), and your program should stop asking after that.
- 0 if it is smaller than answer.
- 2 if it is greater than answer.

Sample interactor for this problem:

Note: Like checkers and validators and generator, you should first initialize your interactor with `registerInteraction(argc, argv)`.

A continuación, podemos ver un ejemplo de código del juez que podría interactuar con las soluciones que envíe el usuario. Para implementar el código de un juez es importante que el usuario siga unas pautas y utilice palabras reservadas concretas, de esta manera cuando el juez se ejecute lo hará como ejercicio interactivo. En el ejemplo de código de la imagen de abajo se ve perfectamente. En la segunda línea de la implementación, justo debajo del main vemos la siguiente línea de código:

registerInteraction(argc,argv);

Esta línea de código es necesaria ponerla en toda implementación de un juez porque es la que se encarga de reconocer que es un ejercicio interactivo y crear la comunicación.

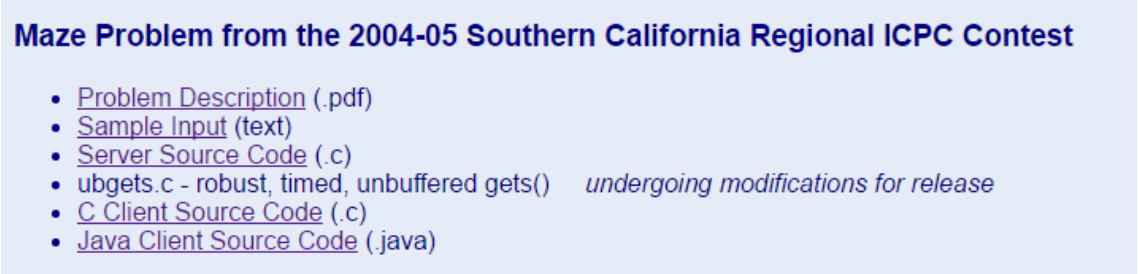
[5] http://codeforces.com/blog/entry/18455

```
int main(int argc, char ** argv){

        registerInteraction(argc, argv);

        int n = inf.readInt();
        cout.flush();
        int left = 50;
        bool found = false;
        while(left > 0 && !found){
                left --;
                int a = ouf.readInt(1, 1000000000);
                if(a < n)
                        cout << 0 << endl;
                else if(a > n)
                        cout << 2 << endl;
                else
                        cout << 1 << endl, found = true;
                cout.flush();
        }
        if(!found)
                quitf(_wa, "couldn't guess the number with 50
questions");
        ouf.readEof();
        quitf(_ok, "guessed the number with %d questions!", 50 -
left);

}
```

Otra de las normas que se utilizan para homogenizar las implementaciones es para dar el veredicto, que se utiliza quitf.

ACM International Collegiate Programming Contest

Hasta ahora sólo se han mencionado jueces online que habían incorporado los ejercicios interactivos, pero también es posible encontrar ejercicios interactivos en concursos de programación. Es el caso del concurso ICPC [5], un concurso anual e internacional con sede en la Universidad de Baylor y que permite a universidades de todo el mundo competir entre ellas. Además de concursos internacionales también organizan concursos regionales, y fue en uno de estos concursos, en concreto en uno del Sur de California en el año 2004, uno de los ejercicios que tenían que resolver era interactivo.



Como podemos ver en la imagen de arriba es posible ver el enunciado del problema en pdf, así como un ejemplo de un caso de entrada, que representa la información que debe leer el código del usuario para poder resolver el problema con éxito.

2004/2005 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
Escape from Swamp County Park

(requires an interactive server)

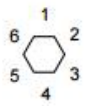
A chill wind descended upon Swamp County. It swept out of the north hills like a flood, then settled in the heart of the swamp. There it festered among the warm moist air of the cursed bog, and it stewed a thick, toxic fog. Your team wandered into the park (before the wind and fog) for a relaxing picnic. Now, however, you are lost, choking for breath, and must escape before you suffocate.

Swamp County Park is carved out of the swamp according to a hexagonal grid, with pathways cut through barbed vines. You must grope through the blinding fog to find the exit. You have a limited number of movement attemps before the toxic fumes overtake you; you must move with efficiency to escape the park. Fortunately, the sheer size of the park dilutes the toxicity of the fog, so the bigger the park, the more moves you have to escape. Sadly, you don't know the exact size of the park, other than it cannot exceed the size of the swamp, which is 20-by-20 hexagonal grid locations.

Input and Output

Your program must converse with a server, issuing movement attempts to standard output and receiving responses through standard input. Your program starts the conversation. You will never start on the exit location. You may attempt to move in any of six directions:

1 attempt to move one hex location up
2 attempt to move one hex location up and to the right
3 attempt to move one hex location down and to the right
4 attempt to move one hex location down
5 attempt to move one hex location down and to the left
6 attempt to move one hex location up and to the left



Each movement attempt sent to standard output must be a single decimal digit '1'..'6' followed by end-of-line. *It is very important that your program flush standard output after issuing a movement attempt.* e.g.

C	`fputs("1\n",stdout); fflush(stdout);`
C++	`cout << "1\n" << flush;`
Java	`System.out.println("1"); System.out.flush();`

If your program encounters the exit, it should terminate normally. Successful escape from the park generates a Correct condition. Any characters sent to the server that do not follow the protocol are deemed a Runtime Error. If the server has to wait more than one second (real time) between your movement requests, your program will trigger a Time Limit Exceeded condition. If your program continues to move around after reaching the exit, it will receive a Presentation Error. If the number of move attempts exceeds the limit ($2 \times rows \times columns$), your program will elicit a Wrong Answer condition (you suffocated).

In response to your attempted moves, the server answers with either

`n` (you ran into a hex location with barbed vines, and did not move)
`y` (you successfully moved in your intended direction)
`E` (you successfully moved in your intended direction and found the exit)

Each response from the server consists of one of the characters shown above, followed by end-of-line. You must read these responses from standard input.

Este problema[6] nos pide que encontremos la salida de un laberinto en el cual nos hemos perdido.

En la imagen de abajo podemos ver el caso de entrada, que representa, en primer lugar las dimensiones del laberinto, 4 de altura y 8 de ancho. Los

[6] http://www.socalcontest.org/current/interactive/maze.pdf

asteriscos representan los límites, el guion representa la posición inicial en la que aparecemos y la E donde se encuentra la salida.

```
4 8
**E*****
* *** -*
*       *
********
```

Si leemos el enunciado de este problema, vemos que el patrón es similar a los que habíamos visto anteriormente. Además de describir lo que el código del usuario va a tener que resolver, se proporciona información sobre cómo debe ser el protocolo de comunicación entre la implementación y el juez.

C	`fputs("1\n",stdout); fflush(stdout);`
C++	`cout << "1\n" << flush;`
Java	`System.out.println("1"); System.out.flush();`

Como vemos en la imagen, el usuario debe tener en cuenta que después de enviar una pregunta o la respuesta al juez, es necesario que se limpie el buffer y para ello te explica cómo debes hacerlo dependiendo del lenguaje en el que programes la solución.

- **n** (you ran into a hex location with barbed vines, and did not move)
- **y** (you successfully moved in your intended direction)
- **E** (you successfully moved in your intended direction and found the exit)

Otro de los puntos importantes que hay que tener en cuenta cuando se intenta resolver un problema interactivo, es saber cómo interpretar las respuestas que el juez envía. En la imagen de arriba, se describe para este problema cuales son los tres valores que el juez puede devolver y lo que significa cada uno.

Pero sin duda, la mayor diferencia de esta web respecto a lo que hemos visto anteriormente, es que pese a proporcionar el enunciado y un ejemplo del caso de entrada al no ser un juez online no es posible enviar soluciones.

Sin embargo, sí que proporcionan la solución a este problema, de hecho la proporciona en C y en java. Desde mi punto de vista, me parece una buena idea, no sólo para que los participantes de ese concurso pudieran ver cómo se resolvía, sobre todo si el concursante no fue capaz de resolverlo. Pero también me parece una buena idea para el resto de personas interesadas en estos problemas, ya que tienen a su disposición un ejemplo resuelto, lo que les puede ayudar a ver cómo se resuelven este tipo de problemas.

Aunque en esta web no sólo podemos ver cómo se resuelve este ejercicio, sino que también es posible ver cómo está implementado el juez. De esta manera, cualquier persona puede ver y entender cómo funciona la comunicación entre solución y juez.

Esto tal vez pueda parecer poco importante para el usuario final, pero realmente, creo que ayuda a entender mejor en qué consisten los ejercicios interactivos y cómo funcionan al completo. Si nos fijamos, vemos que el juez solamente está desarrollado en C, mientras que se proporcionan las soluciones en C y en java. La razón por la que no está puesta la implementación del juez también en java es porque aunque el juez y las soluciones estén en distintos lenguajes, ambos deben ser capaces de comunicarse sin problemas.

4 Estado inicial

Como hemos visto en apartados anteriores, el juez que se ha desarrollado a lo largo de este proyecto está integrado en un proyecto mayor que es Acepta el reto. Debido a que el juez interactivo se iba a integrar en un proyecto existente, el desarrollo del mismo debía ser compatible con la arquitectura que ya estaba hecha, con el fin de que se pudiera aprovechar y reutilizar. Además de que la integración sería más sencilla.

Por este motivo, es importante saber de qué estructura se partía para poder desarrollar el juez con una estructura similar.

ACEPTA EL RETO

Acepta el reto [1] es un juez online desarrollado en la Facultad de Informática de la Universidad Complutense de Madrid. Fue un proyecto que iniciaron dos profesores y que además, son los directores de este trabajo. La idea surgió como repositorio a los problemas que ellos proponían en el concurso de programación para módulos de informática de Formación Profesional, el concurso se llama Programame! Y se realiza todos los años en la Facultad de Informática para la final nacional. La idea de este concurso es resolver unos problemas, generalmente 10, en 4 horas y la manera en la que se envían las soluciones es a través de un juez online.

De este concurso nació la idea de Acepta el Reto y en el año 2013 empezaron con la implementación, para ya en febrero de 2014 ponerlo en funcionamiento.

Además de ser un juez online que almacena los problemas de los concursos que organizan, también contiene numerosos problemas que se utilizan como apoyo en asignaturas de las titulaciones de informática. Como ya he contado anteriormente, mi trabajo de fin de grado fue sobre este juez, cuando estaba en sus inicios, y por entonces, les mostramos a un número de profesores este juez y les hicimos una encuesta. El resultado fue que la mayoría veían muy positivo introducirlo como herramienta de apoyo en sus clases y a día de hoy, ya se usa de manera habitual en diferentes asignaturas y en diferentes cursos.

La captura de arriba muestra la página principal del juez. En la cabecera aparece un menú y un apartado para autenticarse. Generalmente para usar un juez online es necesario tener una cuenta, para que así los rankings se actualicen en base a usuarios y además, éste podrá tener un área propia en la que poder ver sus envíos y progresos.

Problemas

La web es sencilla e intuitiva como podemos ver desde el primer momento.

Como podemos ver los problemas están agrupados por volúmenes o por categorías según le sea más cómodo al usuario.

Enunciado

Problema número 216

Goteras

Tiempo máximo: 4,000 s Memoria máxima: 4096 KiB

http://www.aceptaelreto.com/problem/statement.php?id=216

Con la llegada de las lluvias, has descubierto una molesta gotera en el salón. Con precisión suiza, las gotas caen una vez por segundo desde el techo hasta un improvisado cubo que te ves obligado a vaciar periódicamente hasta que encuentres una solución.

Convivir con una gotera es complicado porque tienes que sincronizar tu vida alrededor de los vaciados del cubo...

Entrada

La entrada estará compuesta de un primer número indicando cuántos casos de prueba vendrán a continuación.

Cada caso de prueba será un número mayor que cero con el número de gotas que entran en el cubo.

Salida

Para cada caso de prueba, el programa escribirá en una línea el tiempo máximo que puedes estar sin cambiar el cubo en el formato `HH:MM:SS`, donde `HH` indica el número de horas, `MM` el número de minutos y `SS` el número de segundos.

Ningún cubo es tan grande como para poder estar más de un día completo sin cambiarse.

Entrada de ejemplo

```
3
70
3600
3661
```

Salida de ejemplo

```
00:01:10
01:00:00
01:01:01
```

En el enunciado[7] que acabamos de ver, la primera parte y la más evidente es el texto que describe al usuario de manera literaria qué es lo que va a tener que implementar. A partir de ahí, es el momento de identificar partes que son necesarias para que el usuario tenga en cuenta para que el juez sea capaz de ejecutar la solución enviada y el veredicto que muestre sea AC, lo que significará que el envío es correcto.

En la parte superior, lo primero que vemos es el título del problema. Justo en la línea de abajo aparecen dos valores que son muy importantes para el juez. Si un usuario, cuando programa la solución no tiene en cuentas dichos valores puede darse el caso de que su solución esté bien pero que el juez no la dé por buena.

Los valores de los que hablo son, tiempo máximo y memoria máxima. El significado del valor tiempo máximo, es que para que la solución del usuario sea correcta además de que las respuestas que se obtengan sean iguales a las obtenidas con la solución oficial es necesario que se ejecuten en menos tiempo que el máximo permitido, que en el caso concreto de este ejercicio es 1 segundo.

El segundo valor, indica la memoria máxima que la implementación enviada por el usuario puede utilizar. Aunque la solución funcionase, si mientras se está ejecutando la solución se sobrepasan estos valores, para el juez la solución no es correcta.

Cuando describía la tabla en la cual el usuario puede ver si su solución ha sido correcta o no, vimos que tenían varias columnas, el nombre del problema, el veredicto, el tiempo en ejecutarse, la memoria usada… En aquel momento era posible pensar que estos valores no eran más que informativos, pero como acabamos de ver, la realidad es que marcan restricciones que el usuario debe tener en cuenta a la hora de implementar la solución.

Después de la descripción del problema vemos una nueva sección llamada Entrada, ésta describe cual es el formato y lo que representa la información. En este caso concreto la entrada consta de varios casos de prueba y cada caso de prueba ocupa una línea. Además, cada línea está compuesta por números separados por un espacio. Toda esta información es muy importante, porque el usuario cuando quiera proponer su solución debe poder saber cómo procesar la entrada para poder utilizarla correctamente.

Debajo de la Entrada hay otro apartado que es la Salida, en la cual se explica cómo es el formato de salida que debe devolver la solución propuesta. Esta sección también es muy importante para el usuario, ya que si no es capaz de dar una solución con el formato que el juez espera éste dará la respuesta como errónea aunque estuviera bien implementada.

[7] https://www.aceptaelreto.com/problem/statement.php?id=216

Para ayudar al usuario a entender el formato de entrada y de salida en los dos últimos apartados se muestra un ejemplo de una posible entrada y una posible salida.

Aunque el ejemplo de enunciado pertenece a Acepta el reto, la estructura del enunciado es similar en el resto de los jueces.

Ranking

La última sección que vamos a ver del juez es la parte de las estadísticas.

Últimos envíos recibidos

Envío	Usuario	Problema	Resultado	Lenguaje	Tiempo	Memoria	Pos	Fecha
72713	Alfonsoelmas	Sudokus correctos (345)	WA	C++	0.072	1684	-	Hace 4 horas
72712	Sleidom	Pintando fractales (167)	AC	C++	0.74	1680	148	Hace 4 horas
72711	Cherry	Evaluando expresiones (198)	TLE	C++	-	-	-	Hace 5 horas
72710	Cherry	Evaluando expresiones (198)	RTE	C++	-	-	-	Hace 5 horas
72709	Alfonsoelmas	Sudokus correctos (345)	WA	C++	0.128	1684	-	Hace 5 horas
72708	jlmaroto	Pintando fractales (167)	AC	C	0.116	1128	37	Hace 7 horas
72707	Alfonsoelmas	Sudokus correctos (345)	WA	C++	0.064	1684	-	Hace 16 horas
72706	Alfonsoelmas	Sudokus correctos (345)	WA	C++	0.068	1684	-	Hace 16 horas
72705	Alfonsoelmas	Sudokus correctos (345)	WA	C++	0.064	1684	-	Hace 16 horas
72704	Alfonsoelmas	Sudokus correctos (345)	WA	C++	0.072	1684	-	Hace 16 horas
72703	Alfonsoelmas	Sudokus correctos (345)	WA	C++	0.068	1684	-	Hace 17 horas
72702	Alfonsoelmas	Sudokus correctos (345)	WA	C++	0.064	1684	-	Hace 17 horas

En la imagen de arriba podemos ver una tabla con los últimos envíos. En la tabla podemos ver diferentes columnas, en este juez, además de las columnas básicas, el usuario, el problema, el veredicto, el lenguaje en el que se ha implementado, el tiempo que ha tardado en ejecutarse la solución enviada y la memoria consumida. También aparecen las columnas, número de envío, fecha del envío y posición en el ranking general.

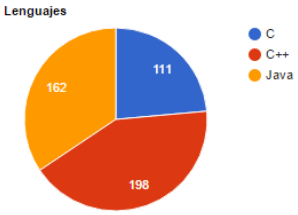
Estadísticas de un problema

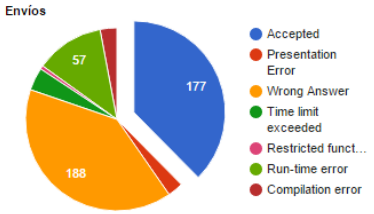
Para cada problema se dispone de estadísticas que muestran el porcentaje de los lenguajes en los que se ha resuelto el problema, el porcentaje de los veredictos, una tabla con el ranking de ese problema.

Todas estas estadísticas resultan muy útiles, porque permiten ver si un problema es más sencillo resolverlo con cierto lenguaje o si tiene un porcentaje bajo de AC es que el problema no es sencillo.

Goteras

Estadísticas





Número de envíos	Usuarios que lo han intentado	Usuarios que lo han conseguido
471	146	115

Clasificación

Pos	Envío	Usuario	Lenguaje	Tiempo	Memoria	Fecha
1	42495	MDuran	C	0.02	1112	2015-10-18 04:55:54
2	70067	Apereza	C	0.02	1112	2016-05-04 20:49:54
3	10630	polacoprO	C	0.024	988	2014-10-05 19:10:18
4	10992	marcant94_	C	0.024	988	2014-10-13 22:34:04
5	11922	IvanRobles92	C	0.024	988	2014-11-02 13:27:04
6	12707	Team_Rocket	C	0.024	988	2014-11-13 17:02:56
7	14230	Juanjovalencia	C	0.024	988	2014-11-26 21:07:26
8	35649	ElGraju	C	0.024	988	2015-04-27 10:44:16
9	35991	Davinchillo	C++	0.024	988	2015-05-07 17:15:24
10	38805	jlmaroto	C	0.024	1112	2015-07-31 12:12:10

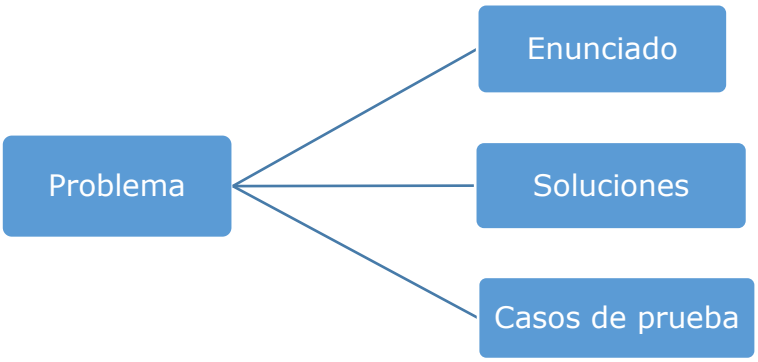
Últimos envíos

Envío	Usuario	Resultado	Lenguaje	Tiempo	Memoria	Pos	Fecha
70067	Apereza	AC	C	0.02	1112	2	2016-05-04 20:49:54
70066	Apereza	AC	C	0.024	1112	23	2016-05-04 20:30:07
70065	Apereza	AC	C	0.068	1112	75	2016-05-04 20:19:51
70064	Apereza	AC	C	0.028	1112	32	2016-05-04 20:17:38
70063	Apereza	AC	C	0.024	1112	22	2016-05-04 20:12:11
69022	FerDev	RTE	Java	-	-	-	2016-04-25 03:58:27
68055	ivanspasov	TLE	Java	-	-	-	2016-04-10 17:58:36
67610	margual	RTE	C++	-	-	-	2016-04-05 09:30:14
67259	Apereza	AC	Java	3.398	3585	171	2016-04-01 13:53:03
65508	Adonis	AC	Java	1.749	3585	163	2016-03-15 20:32:28
65505	Adonis	WA	Java	1.587	3584	-	2016-03-15 20:30:14

Estructura de un problema

Empezaremos viendo la estructura que tiene un problema y los ficheros que lo componen. Un problema para este juez tiene una estructura XML, con unos campos determinados, un enunciado, soluciones y casos de prueba.

A continuación podemos ver un esquema que representa la arquitectura de un problema:



Un enunciado, es el documento que el usuario verá para poder resolver el problema y se compone de varios campos que al ser un XML serán etiquetas.

Título: este campo representa al título que el autor quiere poner a su problema.

Descripción: es la explicación del problema y en él se dice lo que el usuario debe hacer. Es la parte más extensa y más literaria.

Descripción de la entrada: en este campo se explica qué es lo que la solución del usuario va a recibir.

Descripción de la salida: en este campo se explica qué es lo que la solución del usuario debe devolver.

Pistas: en Acepta el reto es posible dar pistas estáticas, aunque éstas se darán de manera ocasional en un futuro.

Información para el profesor: este campo está pensado para que el autor del problema ponga información para el profesor como, formas alternativas de resolverlo, cómo usarlo en clase… Aunque actualmente no se utiliza.

Notas: Es posible añadir notas adicionales al final del enunciado.

Otro de los apartados que vemos son las soluciones, en él se almacenan varias soluciones que resuelven el problema. Hay que destacar que existen dos tipos de soluciones, la solución oficial y el resto.

La solución oficial es la que el autor del problema implementa como la solución de referencia y es a ésta a la que se le pasan los generadores para generar los .out. Todas las demás soluciones compararán sus resultados con los obtenidos por la solución oficial.

El resto de soluciones son opcionales, pero pueden servir para ver la complejidad del problema resuelto en diferentes lenguajes. Es posible que un problema resuelto en java sea sencillo, pero que implementado en C fuera muy complicado.

Los casos de prueba se utilizan para probar la solución y así poder corroborar que es correcta. Existen tres tipos de casos de prueba:

Empty: representa a un fichero sin casos de prueba.

Sample: representa un fichero con los casos de prueba que aparecen en el ejemplo de entrada del enunciado y otro fichero con las respuestas esperadas

Testcase: son los ficheros que se generan mediante generadores, esto significa que no se realizan de manera manual, en el siguiente apartado se explican con más detalle. Los generadores generan los .in y para obtener los .out es necesario ejecutar los ficheros de entrada con la solución oficial.

Generadores

Un generador de casos de prueba no es más que un código, que puede estar escrito en C++. A continuación vamos a ver el código de un posible generador de casos de prueba.

```
#include <stdlib.h>
#include <iostream>
#include <time.h>
#include <stdio.h>

using namespace std;


#define LIMITE 10
#define num 20

int main() {

  srand(time(NULL));
  int s;

  cout << LIMITE << "\n";

    for (int c = 0; c < LIMITE; ++c) {
      s = rand() % (num*(c+1)) + 1;
      cout << num*(c+1) << " " << s << "\n";
    }

  return 0;

}
```

El generador cuyo código tenemos escrito, genera números aleatorios.

Como podemos ver en la imagen, lo primero que se escribe es el número total de casos de prueba que va a haber, en este caso 10, que corresponde al valor de la variable LIMITE y a continuación genera tantos números aleatorios como determine la variable LIMITE, pero para que el rango de números sea diferente se utiliza el contador del bucle y se le multiplica por una variable anteriormente declarada.

Tipos de entrada

En el apartado 2 vimos los dos tipos de ficheros que había, .in y .out, pues en este juez también se disponen de estos ficheros, sólo que para crear el fichero .in es necesario conocer los tres tipos de entrada que se permiten.

La razón por la que se han creado tipos de entrada en lugar de escribir los casos de prueba sin más es por practicidad. Cuando una solución es enviada por un usuario ésta no se ejecuta por cada caso de prueba, sino que en una ejecución se ejecutan todos los casos de prueba de un testcase.

Si se pretende crear un fichero .in con entrada multicaso debemos tener en cuenta que es necesario avisar al juez de alguna manera cuantos casos de prueba va a tener que leer.

El primer tipo es en el que la primera línea será un entero que representará el número de casos que ese testcase va a tener. En el ejercicio de las "Goteras" se utiliza este tipo de entrada.

Entrada de ejemplo

```
3
70
3600
3661
```

En este caso se observa claramente como el primer número que es un 3 indica que las siguientes tres líneas serán tres casos de prueba.

Otra manera de indicar que ya no hay más casos de prueba para leer es colocar un caso especial en la última línea, de esta manera cuando el juez llegue a este caso sabrá que debe parar. En el enunciado de "¿Qué lado de la calle? Que vimos en el apartado del Estado inicial, se utiliza este tipo de entrada.

Entrada de ejemplo

```
3
10
41
0
```

La última opción para identificar que ya se han leído todos los casos es encontrar el fin de fichero (eof), de esta manera cuando el juez lea dicha marca sabrá que el testcase ha terminado.

En qué momento se debe utilizar un tipo de entrada u otro dependerá de la elección de la persona que está desarrollando el problema y del propio problema. Con esto quiero recalcar que un problema no tiene por qué tener sólo un tipo de entrada, sino que para uno es posible que los tres tipos de entrada sean aplicables y el autor se ha decantado por una de ellas por decisión propia.

En el ejemplo que hemos visto en el que el caso de prueba terminaba con un 0, también podía haberse aplicado el primer tipo, en el que el primer número indica el número de casos, pero al leer el enunciado vemos que el problema

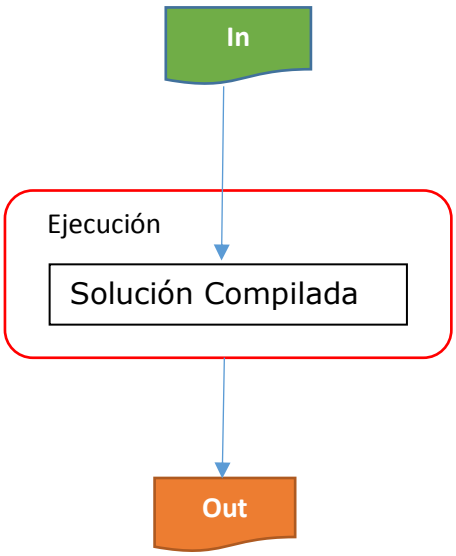
trata de números de portales, por lo que nunca va a haber un portal con el número 0 y por eso el tipo de entrada que se eligió parece más apropiado.

Estructura del proyecto

El proyecto está compuesto por varias herramientas y una de ellas es Execution, que se compone de un conjunto de clases que permiten crear un Executor, que es un objeto que contiene el código de la solución ya compilado y convertido en una aplicación, además de otros parámetros de configuración, como el directorio de trabajo, la ruta del compilador o el tiempo máximo en el que puede ejecutarse.

Solución

Uno de los ficheros que iba a ser necesario manejar era la solución del usuario, la cual se ejecuta un número de veces, tantas, como casos de prueba haya, y el resultado se va almacenando en un fichero. Además de la solución del usuario se dispone de una solución oficial la cual al ejecutarse también genera una solución que se guarda en otro fichero.

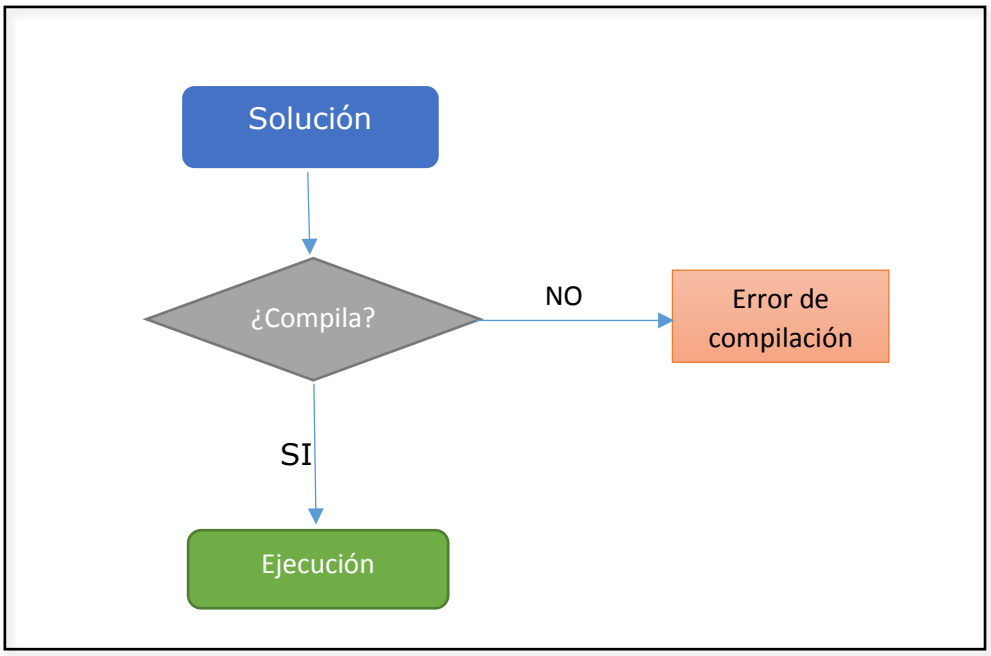


Como vemos en la imagen de arriba la solución compilada se ejecuta y para ello recibe unos datos de entrada, que debido a cómo está desarrollado este proyecto es un fichero, en concreto es el fichero generado con el generador que previamente habíamos visto.

Una vez que la ejecución ha finalizado, ya tendremos en el de salida (out) las repuestas obtenidas. Este proceso se realiza sólo para la solución del usuario, la solución oficial se ejecuta sólo una vez para generar los .out.

Antes de seguir ahondando en la explicación de todos los componentes que tiene la estructura, es importante entender cómo funciona en términos generales.

Cuando un usuario envía una solución, ésta no sólo es compilada para detectar cualquier error de compilación, sino que la solución también es ejecutada. En el supuesto caso de que la solución termine y proporcione un resultado es necesario comprobar si es correcto. Es en este punto donde el juez cobra sentido.



Después de realizar el proceso anterior, tanto para la solución oficial como para la solución enviada por el usuario y suponiendo que ambas han obtenido respuestas que están almacenadas en ficheros, es el momento de realizar las comparaciones para que el juez pueda determinar el veredicto correspondiente.

VEREDICTOS

Otro de los factores importantes a tener en cuenta son los veredictos que el juez devuelve, ya que el juez de los ejercicios interactivos también debe devolver veredictos.

Accepted (AC): Aceptado

Este es el resultado más esperado por el usuario, quiere decir que la solución es correcta porque ha superado todos los casos de prueba.

Presentation error (PE): Error de presentación

Este resultado informa al usuario de que al escribir la solución hay diferencias con los espacios, saltos de línea o tabuladores y por ello los casos de prueba han fallado.

Wrong Answer (WA): Respuesta Incorrecta

Este veredicto informa al usuario que al ejecutar los casos de prueba los resultados obtenidos son diferentes a los esperados y por ello no ha superado las pruebas.

Compilation error (CE): error de compilación.

Este veredicto informa al usuario de que su solución tiene algún error de compilación, lo que ha provocado que no haya sido posible obtener ningún resultado porque ese código no ha podido ser compilado.

Run-time error (RTE): Error durante la ejecución

Este error se produce cuando el juez, después de ejecutar la solución del usuario, lo ha ejecutado y ha terminado antes de tiempo, es decir, no se han podido probar todos los casos.

Time limit exceeded (TLE): Tiempo límite superado

Cuando el juez devuelve este veredicto significa que el código de la solución del usuario se ha podido compilar y ejecutar, pero durante la ejecución ha tardado demasiado tiempo en dar una respuesta y esto ha provocado que la ejecución se haya cancelado.

Memory limit exceded (MLE): Límite de memoria superado

Este veredicto aparece durante la ejecución, después de haber compilado el código correctamente. El motivo por el que se produce es porque mientras la solución estaba probando los casos de prueba, ésta ha consumido más memoria de la que estaba permitida, por ello ha sido necesario cancelar la ejecución.

Output limit exceeded (OLE): Límite de salida superado

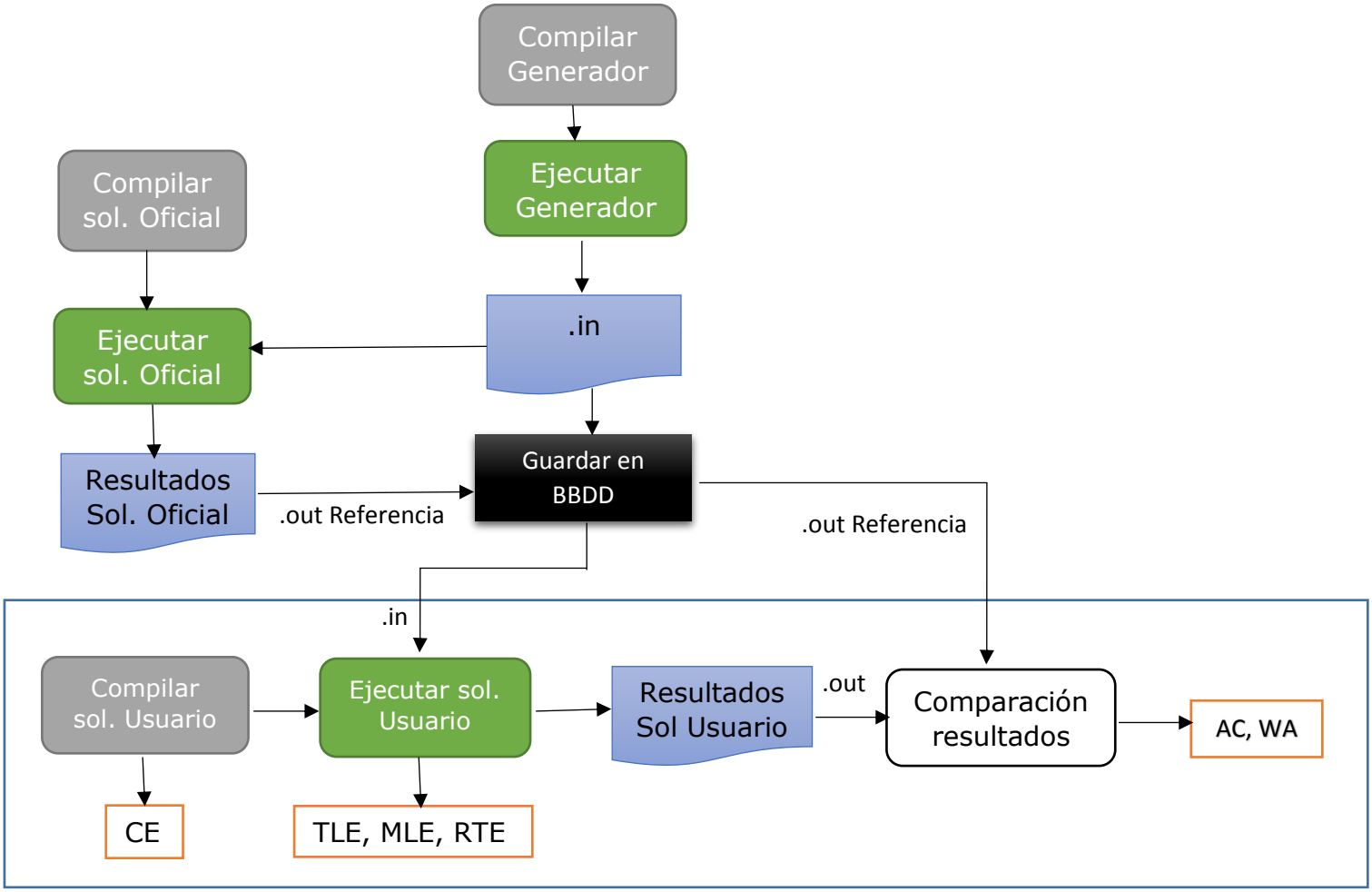
Este resultado se produce durante la ejecución de la solución enviada por el usuario, lo que quiere decir que se pudo compilar sin problemas, pero al pasar

los casos de prueba la respuesta que da el código es más larga que la permitida, por lo que se ha cancelado la ejecución.

Restricted function (RF): Función restringida

Si el juez devuelve este veredicto, es debido a que el usuario ha utilizado en su código una operación que puede dañar al juez y por este motivo ha sido cancelada su ejecución.

Visión general de un juez



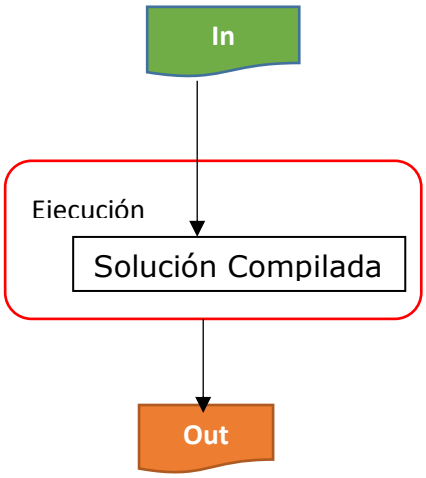
En este esquema se ve claramente el funcionamiento del juez, los componentes que necesita y los pasos que debe seguir para dar el veredicto. Como vemos hay dos partes diferenciadas. Por un lado, la parte de la generación de los ficheros de entrada y la parte de la ejecución de la solución oficial que se ejecutan sólo una vez y sus ficheros resultantes se almacenan en base de datos, y por otro lado, el tratamiento que recibe la solución del usuario, en la que sus resultados no se almacenan.

5 Desarrollo del proyecto

Después de estudiar cómo funcionaba el juez para problemas no interactivos, era el momento de empezar a pensar cómo desarrollar el juez para problemas interactivos.

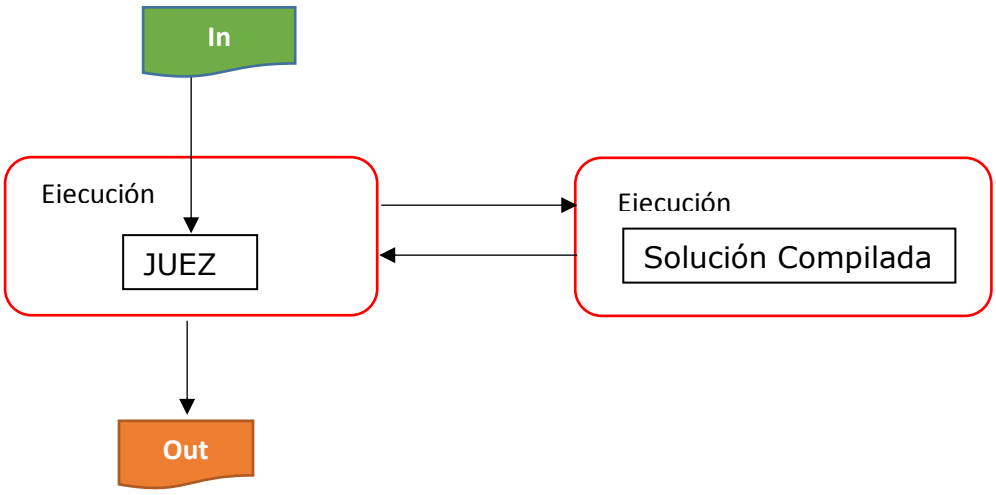
Arquitectura

Lo primero que se debe hacer es adaptar la arquitectura existente. Si recordamos la ejecución de una solución era la siguiente:



Ahora no sólo va a haber una solución ejecutándose sino que habrá dos ejecuciones simultáneas. Una será el juez y la otra la solución que resuelve el problema, ya sea la solución del usuario o la solución oficial.

De esta manera la arquitectura se modificaría de la siguiente manera

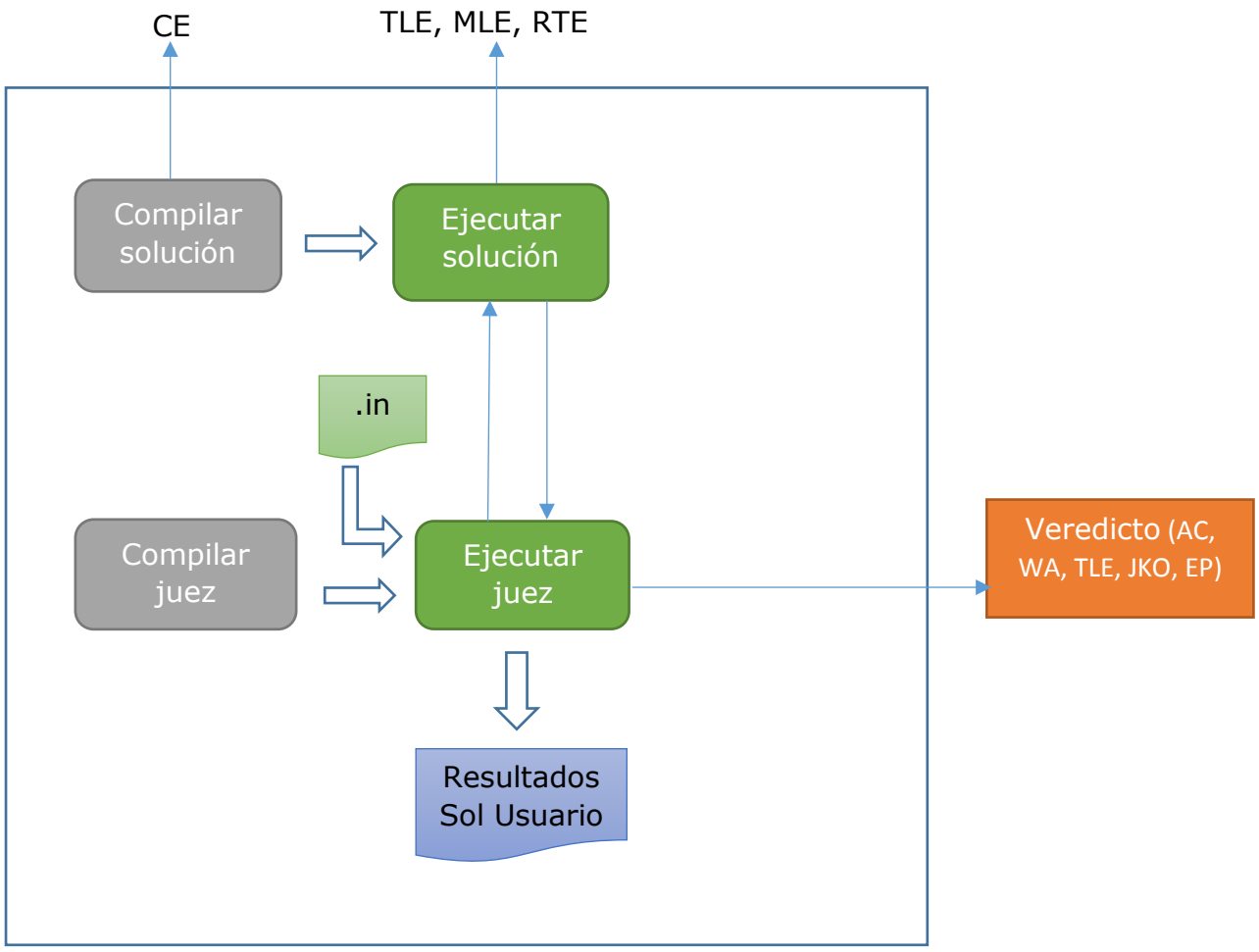


Como vemos el Juez sigue teniendo la entrada del fichero .in, que es el que almacena los casos de prueba, y el fichero .out que almacena la respuesta generada por la solución. Pero además, tiene otras dos conexiones, que son para comunicarse con la otra ejecución.

Podríamos pensar que con una flecha bidireccional sería suficiente, pero como veremos en el apartado de la implementación los canales de comunicación son diferentes.

Otro detalle a tener en cuenta es la función del fichero .out, si bien, en los ejercicios tradicionales su función estaba bien definida, en los ejercicios interactivos no está tan claro, ya que es el juez el que da el veredicto, es decir, que el juez ya sabe si la solución es correcta o no, por lo que la función de este fichero es más ambigua. La razón por la que se mantiene es para mantener una homogeneidad entre los ejercicios tradicionales y los ejercicios interactivos.

En la imagen de abajo vemos actualizada la arquitectura global del juez.



Es muy importante tener en cuenta que en los ejercicios tradicionales sólo había un juez, pero en los ejercicios interactivos habrá tantos jueces como problemas, esto quiere decir, que cada vez que se desarrolla un ejercicio interactivo, no sólo es necesario realizar una solución oficial, sino que es necesario desarrollar un juez capaz de comunicarse con cualquier solución que le envíen.

Aun así, el funcionamiento básico no cambia, la solución debe compilarse antes de ejecutarse y se ejecutará tantas veces como casos de prueba haya. Como ya se ha comentado en un apartado anterior, el juez puede estar implementado en un lenguaje concreto, pero debe poder comunicarse con soluciones realizadas en diferentes lenguajes. Es decir, un juez implementado en C, debe comunicarse con las soluciones de los usuarios que pueden estar implementadas en C, C++ o java.

Implementación

Una vez pensado cómo implementar el proyecto era el momento de empezar a programar. La parte más importante de este proyecto es la creación de una comunicación entre el juez y la solución enviada por el usuario.

Para ello, ha sido necesario ampliar la herramienta Execution y en concreto he tenido que ampliar la clase Executor que es la encargada de lanzar la ejecución de la aplicación de la solución. En ella, he necesitado implementar un método llamado execComunica que recibe dos parámetros uno representa al juez y el otro a la solución a ejecutar.

Es posible que pensemos que los dos parámetros son las propias implementaciones, pero esto dista mucho de la realidad, no hay que olvidar que antes de proceder a ejecutar la solución con el juez, han tenido que ser compiladas previamente con el compilador correspondiente al lenguaje de programación utilizado en las implementaciones.

Una vez compiladas las soluciones se procede a crear un objeto que contiene un ejecutable o un .class y un conjunto de parámetros cómo la ruta donde se encuentra el ejecutable, la ruta del fichero de entrada, la ruta del fichero de salida, el directorio de trabajo o si se quiere guardar la salida de error. Ahora que ya hemos creado estos objetos, éstos sí que los podemos pasar como parámetros al método execComunica().

Dentro de este método, es donde se realiza la comunicación entre ambos ejecutables pero antes es necesario crear una estructura que lo permita.

Lo primero que es necesario hacer es crear dos Process Builder [6], uno por cada ejecutable.

```
ProcessBuilder pb = new ProcessBuilder(e1._command);
ProcessBuilder pb2 = new ProcessBuilder(e2._command);
```

La clase ProcessBuilder es una clase java que permite crear procesos del sistema operativo, cada proceso creado por el constructor tiene los siguientes atributos:

- command, que almacena el ejecutable y una lista de argumentos que pueden ser necesarios a la hora de lanzar la ejecución.
- workingdir que representa a un directorio de trabajo que se puede especificar mediante una ruta.
- standard input que por defecto lee de una tubería pero que es posible redirigirla, standard output es similar al anterior pero con la salida, también es posible redirigir la salida de error.
- redirectErrorStream que permite enviar la salida estándar y la salida de error por la misma vía, aunque por defecto esta opción viene deshabilitada.

Si nos fijamos en los atributos que necesita el ProcessBuilder, vemos que son muy parecidos a los que hemos creado en el objeto anterior. Al principio parecía que sólo con compilar los códigos y generar ejecutables era suficiente para poder comunicarlos pero no es cierto, porque se necesita un manejo de hilos y de procesos ejecutándose en segundo plano como veremos más adelante.

Después de haber creado los dos processBuilder con los command de los objetos Executor configuramos el directorio de trabajo.

```
pb.directory(new File(e1._workingDir));
pb2.directory(new File(e2._workingDir));
```

A continuación, se declaran dos objetos de la clase Process de Java.

```
Process p = null;
Process p2 = null;
```

La clase Process va ligada a la anterior, ya que los processBuilder tienen un método llamado start() que crea un proceso que se ejecuta en segundo plano. Por esto al declarar las variables sólo se inicializan a null.

Ahora es el momento de lanzar los dos procesos, uno para cada aplicación. Por eso era tan importante que los procesos se ejecutaran en segundo plano. Como podemos ver en el código primero se lanza pb y después se lanza pb2. Si no se lanzasen en segundo plano lo que sucedería es que p se ejecutaría y hasta que no terminase no se ejecutaría p2 lo que provocaría que nunca se pudieran comunicar. Además, en este caso concreto los códigos de las aplicaciones están desarrollados para que exista la comunicación, esto quiere decir que los códigos se quedan esperando a que el otro les hable y si no reciben la contestación se quedan bloqueando el sistema.

```
try {
      p = pb.start();
      p2 = pb2.start();
} catch (IOException eI) {
      eI.printStackTrace();

}
```

Debido a la complejidad, es necesario ejecutar estos métodos dentro de una sentencia try – catch, por si no es posible lanzar alguno de los dos procesos o por si lanzan alguna excepción.

De manera similar a la de las variables processBuilder se crean las variables de la clase Thread [7].

```
Thread inputThread = null;
Thread outputThread = null;
Thread errorThread = null;
```

La clase Thread de Java permite a la máquina virtual de java que una aplicación tenga varios hilos ejecutándose al mismo tiempo, que es lo que se necesita para comunicar los dos procesos.

Por último, nos faltaría realizar la comunicación, pero este paso hay que hacerlo con cuidado, ya que puede resultar bastante lioso. La idea principal es que la salida del juez sea la entrada del código del usuario y la salida del código del usuario sea la entrada del juez, pero no es una conexión trivial, ya que la manera en la que están implementadas las hebras en java puede llevar a confusión.

El problema al que te enfrentas con este tipo de comunicación es que no es posible realizarla directamente, se necesita un buffer u otro tipo de almacenamiento temporal para que la hebra que debe escribir escriba en ese buffer y la hebra que tiene que leer lo haga del mismo sitio.

Para que esta tarea sea más sencilla se ha hecho uso de una clase privada llamada TeeRunnable a la que le pasas un inputStream, un outputStream y el tamaño del buffer que se va a usar de intermediario y automáticamente conecta la hebra de la salida con la hebra de la entrada.

```
if (pb.redirectOutput() == ProcessBuilder.Redirect.PIPE) {
      // Necesitamos la hebra que lee y escribe.
      outputPipeRunnable =
                        e1.new TeeRunnable(p.getInputStream(),
                        p2.getOutputStream(),
                        e1._storeOutput?MAX_STRING_SIZE_IN_BYTES:0);
      outputThread = new Thread(outputPipeRunnable,
                              "OutputPipeRunnable");
      outputThread.start();

}
```

Este paso se debe realizar dos veces mínimo, uno que redirija la salida del juez a la entrada de la solución, otra en sentido contrario y también se puede realizar otra redirección, aunque esta ya sería de manera opcional. Esta última redirección que se puede hacer es la de la salida de error, realizarla es muy útil ya que si no, resulta muy complicado seguir la conversación que están manteniendo las dos aplicaciones, pero si redirigimos la salida de error a un fichero, por ejemplo, podemos ver lo que está sucediendo.

Una vez que tenemos configuradas todas las hebras con sus correspondientes redirecciones, ya podemos arrancarlas para que se ejecuten.

Ahora que ya tenemos los dos procesos comunicados entre sí y ejecutándose a la vez, es necesario tener en cuenta que mientras que alguno de los procesos no haya terminado el otro debe esperar. Esto es muy importante tenerlo en cuenta, ya que un proceso depende del otro, es decir, hay instantes en que un proceso debe esperar a que el otro le hable para que éste pueda realizar su tarea.

```
while (!procesoTerminado && !procesoTerminado2) {
	try {
		retValue = p.waitFor();
		retValue2 = p2.waitFor();
		// Terminó
		procesoTerminado = true;
		procesoTerminado2 = true;
	}
	catch (InterruptedException e) {
		Thread.currentThread().interrupt();
	}

}
```

Si todo ha ido bien las dos aplicaciones se habrán ejecutado y el juez ya podrá dar el veredicto correspondiente.

Enunciado

En estos enunciados también existe un tiempo límite de ejecución (TLE), aunque en esta ocasión estos veredictos, además de poder aparecer porque el código ha tardado demasiado en ejecutarse, es posible que se produzcan porque la implementación que ha realizado el usuario esté preguntando más veces de las permitidas. No debemos olvidar, que para encontrar la solución es necesario hablar con el juez y si no se controlan el número de preguntas que pueden hacerse, el usuario podría estar preguntando de manera infinita y encontrar la solución por casualidad.

Problema interactivo número 1

Adivina el número

Tiempo máximo: 1,000 s Memoria máxima: 4096 KiB

Gabriel y Marta estaban esperando el autobús, ambos se estaban aburriendo pero sabían qué en cualquier momento éste llegaría y que no les iba a dar tiempo a jugar una partida con su Smartphone.

Entonces Gabriel tuvo una idea, Miró a Marta y le dijo: "He pensado un número entre el 1 y el 50, a ver si adivinas cuál es."

A Marta también le pareció entretenido y no sólo jugaron durante la espera del autobús, sino que jugaron durante todo el trayecto.

Cuando Gabriel llegó a casa, quería seguir jugando pero estaba solo, así que decidió buscar en internet a ver si podía seguir jugando, y encontró el juego, sólo que en lugar de jugar con una persona, jugaba con un ordenador. El problema era que había veces que antes de que Gabriel encontrase el número, el ordenador le decía que había perdido porque había preguntado demasiadas veces. Esto le hizo pensar que tal vez había un algoritmo para adivinar el número con un número menor de intentos.

Ayuda a Gabriel a encontrar un algoritmo que encuentre el número que el ordenador ha pensado en el menor número de intentos.

Comunicación

Tu programa debe conversar con el ordenador y debe leer por la salida estándar y debe escribir por la salida estándar. Es importante que después de cada escritura se utilice la función flush() para limpiar el buffer.

La primera vez que leas será el número de partidas que vas a jugar, la segunda vez será hasta qué número el ordenador ha decidido pensar. El rango de números será desde el 1 hasta el máximo que delimite el ordenador.

Para comunicarte con él sólo debes escribir cuál crees que es el número que ha pensado por la salida estándar y él te devolverá un entero para indicarte si el número propuesto es menor, mayor o igual al que ha pensado.

Ejemplo de comunicación

Ordenador	Tú
5	
100	
	50
1	75
0	

Respuestas del ordenador:

Respuesta	Descripción
-1	El número pensado es menor del propuesto
1	El número pensado es mayor del propuesto
0	Solución encontrada

Solución

En el apartado anterior hemos visto un posible ejemplo de un ejercicio interactivo, pero ahora debemos saber cómo enfrentarnos a este tipo de problemas desde el punto de vista del usuario.

Con el formato tradicional de ejercicios el usuario tenía que leer los datos de entrada con los que resolvería el problema, pero con este tipo de problemas el planteamiento cambia. Es cierto, que el usuario debe implementar una solución que lea, pero con ese o esos datos no va a poder obtener la solución a la primera, sino que para llegar a la solución correcta debe saber que pregunta hacerle al juez y en qué momento de la implementación para que la respuesta le sea útil y le ayude a resolver el problema.

A continuación vamos a ver una posible solución del problema Adivina el número, que hemos visto en el apartado anterior.

En este código se observa como el usuario espera a que el juez inicie la conversación. Como el ejercicio está pensado para que se adivine un número entre el 0 y n y están ordenados de mayor a menor la opción más óptima es resolverlo mediante una búsqueda binaria.

Una de las partes de este código que refleja que es un ejercicio interactivo es cuando se procesa la respuesta que el juez envía, según la respuesta el código hace diferentes cosas.

```
import java.util.Scanner;


public class AdivinaNumeroSol {

	public static void main(String args[]) {

		Scanner in = new Scanner(System.in);

		int numPartidas = in.nextInt();
		int sol;
		int respuesta=-1;
		int ini =1;
		int fin =-1;
		for (int i = 0; i<numPartidas;i++){
			boolean encontrada = false;
			boolean valido = true;
			fin = in.nextInt();
			int mitad = fin/2;
			System.out.println(mitad);
			System.out.flush();

			while (!encontrada && in.hasNextInt() ) {
				respuesta = in.nextInt();
				switch (respuesta){
					case -1 :
						fin= mitad;
						mitad = (fin+ini)/2;
						System.out.println(mitad);
						System.out.flush();
						break;
					case 1:
						ini= mitad;
						mitad = (fin+ini)/2;
						System.out.println(mitad);
						System.out.flush();
						break;
					case 0:
						sol= mitad;
						encontrada = true;
						break;
				}
			}
		}
	}

}
```

Juez

Acabamos de ver una posible solución del problema implementada en java, pero ahora vamos a ver una posible implementación de un juez que sea capaz de interactuar con las diferentes soluciones.

Como se comentó en el apartado de la implementación el juez puede estar implementado en un lenguaje distinto al de la solución y en este caso está desarrollado en C++.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <fstream>
#include <iostream>
using namespace std;
int main() {
   int respuesta=-1;
   bool encontrada = false;
   cout << numNumeros << endl;
   tle = log2(numNumeros);
   ret = 0;
   while(!encontrada && ret == 0){
      cin >> respuesta;
      tle--;
      if ((respuesta < 0) || (respuesta > numNumeros)){
         ret = 2; // WA
         cerr << "WA el volor recibido no es correcto" << endl;
      }
      else{
         if (tle > 0){
            if (respuesta < solucion){
               cout << 1 << endl;
               cerr << "Recibido: "<<respuesta<<", Es mayor" << endl;
            }else if( respuesta > solucion){
               cout << -1 << endl;
               cerr << "Recibido: " <<respuesta<<", Es menor" << endl;
                  }
                  else{
                     encontrada = true;
                     cout << 0 << endl;
                     cerr << "Solución encontrada: " <<respuesta<< endl;
                  }
         }
         else{
            ret = 1; //TLE
            cerr << "TLE, has preguntado demasiado" << endl;
         }
      } //end if
   } //end while

      return ret;
}
```

Como podemos observar en el código, es el juez el que inicia la conversación y según la pregunta que realiza en usuario el juez responde una cosa u otra.

Es importante recalcar, que el juez no sólo tiene que responder al usuario sino que debe dar un veredicto, para ello la variable ret será la encargada de almacenar el veredicto que se va a devolver.

- Si ret vale 0 significa que todo ha sido correcto y que el veredicto es AC

- Si ret vale 1 significa TLE y este veredicto se produce cuando se preguntan más veces de las necesarias para resolver el problema, un claro ejemplo de Time Limit es que el usuario haya optado por realizar una búsqueda lineal.

- Si ret vale 2 significa que la solución no es correcta, en este ejemplo esto puede parecer confuso ya que el problema se llama adivina el número, por lo que se entiende que la implementación del usuario irá dando respuestas hasta que dé con la solución. Pero puede ser que la solución esté mal implementada y, por ejemplo, de como respuesta un número fuera del rango del que se está jugando. Esto sería Wrong Answer.

Otro detalle importante que se puede ver es que el juez utiliza la salida de error, de esta manera se puede trazar la conversación.

VEREDICTOS

Siguiendo las similitudes con los ejercicios tradicionales es necesario hablar de los veredictos. En el apartado anterior ya hablé del tiempo límite de ejecución, pero con estos tipos de ejercicios han aparecido nuevos veredictos además de los que ya existían.

Time Limit Exceeded (TLE)

Como vimos en los ejercicios normales este veredicto ya existía, pero en los ejercicios interactivos además tiene un nuevo significado, cuando la solución de un usuario realice más preguntas del máximo permitido, el juez dará este veredicto.

Protocol Error (EP)

Este es uno de los nuevos veredictos que pueden aparecer cuando enviamos un problema interactivo.

Este veredicto indica al usuario que la comunicación entre su solución y el juez ha sido interrumpida, esto puede haberse producido por varios factores, como por ejemplo, que en el código del usuario se haya introducido una función para interrumpir la comunicación como close(); otra posible causa de

este veredicto es que la comunicación se haya visto interrumpida por motivos ajenos, como la caída del servidor.

Judge Error (JKO)

Como hemos visto anteriormente es posible que el usuario mande un código pero éste sea incorrecto y desencadena errores. Pues bien, este error aparece cuando es el juez el que tiene los problemas durante la ejecución. Es cierto, que no es un error que vaya a darse con mucha facilidad, ya que la persona que desarrolla el juez lo hace con cuidado, contemplando todos los casos y probándolo para evitar estos problemas, pero debido a que existe esta posibilidad es necesario tener controlado este veredicto.

6 Conclusiones

Una vez finalizado el proyecto es necesario echar la vista atrás para poder valorar el trabajo realizado y los conocimientos aprendidos.

Valoración personal

Cuando empecé con el proyecto no tenía muy claro cómo iba a terminar. Si bien es cierto que conocía el proyecto Acepta el reto, los jueces online y el formato de los ejercicios, nunca había oído hablar de los ejercicios interactivos ni conocía acrex y su conjunto de herramientas.

El proyecto en el que me disponía a trabajar ya era un proyecto terminado, operativo y estable. Mi trabajo consistía en hacer una ampliación, por lo que los cambios que fuera a realizar no sólo tenían que ser funcionales sino que además no debían romper lo que ya estaba funcionando.

Con esta situación me encontraba ante dos grandes retos, el primero, entender lo que eran los ejercicios interactivos, cómo funcionan y para qué son útiles.

El segundo reto era comprender y entender el proyecto que ya estaba implementado para poder realizar las modificaciones pertinentes.

Ahora que el proyecto ya está terminado, me doy cuenta de los obstáculos a los que me tuve que enfrentar. Sobre los ejercicios interactivos no hay mucha información y la primera vez que intenté solucionar uno de ellos no entendía lo que tenía que programar y eso me llevó a una pregunta: ¿para qué sirven? Realmente no le encontraba ninguna mejora respecto a los ejercicios tradicionales, además cada problema tenía un juez lo que implicaba más carga de trabajo a la hora de desarrollar un problema, lo cual era claramente una desventaja a frente a los ejercicios normales. Cuando comprendí que eran capaces de resolver un grupo de problemas que con los otros ejercicios no era posible, todo empezó a tener sentido y poco a poco me fui familiarizando con ellos hasta el punto de crear uno desde cero.

En cuanto al código, la mayor dificultad, dejando de lado que el código era bastante extenso, fue entender cómo debía funcionar la comunicación, aunque el concepto era bastante sencillo, la salida de uno es la entrada de otro, yo nunca había trabajado con hebras y supuso para mí un obstáculo.

Respecto al proyecto en general, debo decir que es un proyecto con un gran recorrido y que pese a que está dentro de Acepta el Reto, sería fácil trasladar esa funcionalidad a otro juez o aplicación, lo que permitiría a un usuario utilizar sólo la parte de los ejercicios interactivos obviando el resto del proyecto. De hecho, si observamos cómo están pensados los problemas para los jueces que vimos en el estado del arte, vemos como encajan con el proyecto que he realizado.

Tal vez en un primer momento podemos pensar que esto no parece de mucha utilidad, pero realmente sí que lo es.

Trabajo futuro

Pese a que la idea principal del proyecto se ha realizado y se ha completado la evolución más lógica es introducirlo en Acepta el reto.

Esta nueva mejora en el juez online, no sólo le hará crecer en relación al número de usuarios que lo utilicen, ya que como hemos visto en este trabajo, no hay demasiados jueces que ofrezcan soporte a los ejercicios interactivos ni mucha variedad y además el juez está en español, lo que para muchos puede parecer una desventaja, desde mi punto de vista no lo es. El español es una de las lenguas más habladas del mundo y los jueces que hemos visto están en inglés, lo cual abriría un mercado interesante en el que la competencia sería menor.

7 Conclusions

Once the project is completed it is necessary to look back to assess the work done and the knowledge learned.

Personal Ratings

When I started the project I was not sure how it would end. Even I knew the project Acepta el Reto, the online judges and format of the exercises, I had never heard of interactive exercises or its toolkit ACREX.

The project in which I was about to work was already a finished, operational and stable project. My job was to make an extension, so that the changes I was going to perform not only had to be functional but also should not break what was already working.

With this situation I faced two major challenges, the first, to understand what the interactive exercises were, how they work and why they are useful.

The second challenge was to understand the project Acepta el Reto to make the necessary changes.

Now that the project is finished, I realize the obstacles I had to face. On the interactive exercises there is little information and the first time I tried to solve one of them I did not understand how to solve it and that led me to a question: What is an interactive problem? Really I could not find any improvement over traditional exercises, moreover, every problem had a judge what implies an extra workload and requires additional time to develop a problem, which was clearly a disadvantage compared to normal exercises. When I realized that they were able to solve a set of problems that the other exercises was not possible, everything started to make sense and gradually I became familiar with them to the point of creating one from scratch.

Relating to the code, the greatest difficulty, leaving aside the code was quite extensive, was to understand how communication should work, although the concept was quite simple, the output of one is the input of another, I had never worked with threads and it supposed an obstacle for me.

Regarding the project in general, I must say it is a project with a great future and although it is within Acepta el Reto, It would be quite simple to move this functionality to another judge or another standalone application, which would allow a user to use only the part of the interactive exercises obviating the rest the Acepta el Reto project. In fact, if we look at how they are designed the judge’s problems we saw in the state of the art, we see how they fit with the project I've done. Perhaps we may think that this does not seem very useful, but really it is.

Future work

Although the main idea of the project was implemented and completed, the most logical evolution is to introduce it in Acepta el Reto.

This will improve the online judge, not only making it grow in relation to the number of users who use it, since as we have seen in this document, there are not many judges who offer support to interactive exercises. Besides, it offers so much variety and also the judge is in Spanish, what can be a disadvantage to many people, but from my point of view it is not. Spanish is one of the most widely spoken languages in the world and the judges I have seen are in English, for this reason the Spanish market sounds interesting in order to have less competitors.

Bibliografía

[1] «Acepta el Reto,» [En línea]. Available: https://www.aceptaelreto.com/. [Último acceso: 10 Junio 2016].

[2] «UVa Online Judge,» [En línea]. Available: https://uva.onlinejudge.org/. [Último acceso: 14 Junio 2016].

[3] «CodeForces,» [En línea]. Available: http://codeforces.com/. [Último acceso: 14 Junio 2016].

[4] «Spoj,» [En línea]. Available: http://www.spoj.com/. [Último acceso: 12 Junio 2016].

[5] «ICPC,» [En línea]. Available: http://www.socalcontest.org/current/interactive.shtml. [Último acceso: 7 Junio 2016].

[6] H. Schildt, Java: The complete Reference, McGraw-Hill Education, 2014.

[7] E. R. Harold, Java I/O, O'Reilly, 2006.