Actas de las Jenui, vol. 2. 2017.
Páginas: 289-296

# *¡Acepta el reto!*: juez online para docencia en español

Pedro Pablo Gómez-Martín, Marco Antonio Gómez-Martín
Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid
{pedrop,marcoa}@fdi.ucm.es

## Resumen

En este artículo presentamos *¡Acepta el reto!*, un juez online en el que los autores ponen a disposición de los usuarios problemas en español planteados específicamente para el aprendizaje por parte de los alumnos de las diferentes asignaturas de los Grados de Informática y Ciclos Formativos. Dispone de una batería de unos 300 problemas, que pueden recorrerse por categorías, lo que simplifica encontrar aquellos que resulten más adecuados en cada momento. Muchos de los problemas están pensados para forzar soluciones con complejidades específicas no solo en tiempo, sino también en espacio, algo poco habitual en otros jueces. Con más de 100.000 envíos realizados, ha sido utilizado durante varios años en la Universidad Complutense de Madrid así como en un creciente número de institutos de Formación Profesional.

## Abstract

This paper presents *¡Acepta el reto!* (*Take on the challenge!* in Spanish), an online judge that provides programming problems written up in Spanish and created specifically for practising subjects taught in CS degrees and vocational training courses. It contains a repository with more than 300 problems, that can be browsed through a set of categories, what eases the task of finding the more suitable problems. Many of them have been designed in such a way that they must be solved using algorithms with not just a time complexity bound, but also with a concrete space complexity, something that is unusual in other online judges. It has been used during the last few years at the Universidad Complutense of Madrid, and also in different vocational training centres, reaching out more than 100.000 code submissions since then.

## Palabras clave

jueces en línea, enseñanza de la programación, programación competitiva.

## 1. Introducción

Es conocida la máxima "*a programar se aprende programando*", y prueba de ello es la cantidad de prácticas y ejercicios que los profesores requerimos a nuestros alumnos en las primeras etapas de su aprendizaje. Con el asentamiento de la evaluación continua en las universidades, esta realidad no ha hecho sino crecer.

Una forma de conseguir que los alumnos programen de manera autónoma y reciban retroalimentación sin sobrecargar de trabajo al profesorado consiste en hacer uso de herramientas que faciliten la corrección automática. Los pioneros en esta labor surgieron a raíz de los concursos de programación, pues se necesita proporcionar de manera casi inmediata una evaluación de las soluciones enviadas por los participantes.

El *software* diseñado para la gestión de concursos poco a poco fue utilizándose fuera de ellos en lo que vino a llamarse *jueces online*. Éstos nacieron con el fin de que los futuros participantes pudieran entrenar para ediciones posteriores de los concursos.

Sin embargo, la evaluación automática y los problemas que plantean pueden ser utilizados no solo por programadores expertos que quieren mejorar sus habilidades competitivas, sino también por programadores noveles que buscan un aprendizaje curricular.

En este artículo presentamos *¡Acepta el reto!*[1], un juez online destinado a ese tipo particular de usuarios. En él, los problemas están categorizados para facilitar su búsqueda, y contienen restricciones tanto en tiempo como en espacio que persiguen filtrar soluciones poco eficientes para forzar a los estudiantes a pensar en la complejidad del código que escriben.

La siguiente sección hace un recorrido por la historia de los jueces online, para pasar después, en la sección 3, a describir la historia y características de *¡Acepta el reto!*. La sección 4 describe la arquitectura del sistema y es seguida por un análisis de los problemas disponibles. La sección 6 enumera algunos usos del juez en contextos educativos. El artículo termina con unas conclusiones y trabajo futuro.

[1] https://www.aceptaelreto.com

## 2. Estado del arte

La programación es uno de los puntos clave de la informática. Para poner a prueba los conocimientos de los estudiantes universitarios, en la ya lejana década de 1970 surgió el que se considera el concurso de programación más antiguo, el ACM/ICPC (*International Collegiate Programming Contest*), que superó los 40.000 participantes en 2015 [1]. Para los alumnos de secundaria, desde 1989 se celebra la IOI (*International Olympiad in Informatics*), la segunda mayor olimpiada de las organizadas bajo la tutela de la UNESCO, en cuya final participaron, en 2015, 322 participantes de 83 países diferentes [7]. Es significativo que, a pesar de ser las Olimpiadas de la Informática en general, se centran únicamente en programación, dejando de lado el resto de áreas como *hardware* o redes.

En esos concursos, los participantes se enfrentan a problemas de programación que deben resolver en un tiempo máximo y teniendo en cuenta las restricciones específicas de cada problema. Inicialmente, las soluciones enviadas eran evaluadas manualmente por los jueces humanos, probándolas con baterías de casos preparadas con antelación. Con el tiempo, fueron surgiendo *jueces automáticos* para la gestión de los concursos, que terminaron desembocando en los llamados *jueces online* [10], en los que los usuarios pueden mandar soluciones a los problemas en cualquier momento, sin estar atados a la duración de un concurso determinado.

Específicamente, un juez online es una plataforma software que contiene la descripción de diferentes problemas de programación, junto con baterías de pruebas, secretas, con las que probar si una determinada solución es o no correcta. Los usuarios pueden registrarse y enviar sus soluciones, que son evaluadas automáticamente y reciben un veredicto.

Probablemente el juez online más antiguo y conocido es el gestionado por el profesor Miguel A. Revilla, de la Universidad de Valladolid [14], aunque con el tiempo han surgido muchos otros. Inicialmente, el objetivo principal de estos jueces era facilitar a futuros concursantes el entrenamiento a través de las baterías de problemas de concursos anteriores. Sin embargo, desde hace varios años se consideran útiles también para el *aprendizaje curricular* [9, 16], hasta el extremo de que han surgido algunos jueces online concebidos desde el principio como herramientas pedagógicas [12, 13].

A la estela del funcionamiento de los concursos de programación como el ACM/ICPC, los problemas propuestos en los jueces online suelen exigir que los programas lean de la entrada estándar un conjunto de *casos de prueba*, y para cada caso generen, por la salida estándar, un resultado. La validación de una determinada solución se realiza a través de *análisis dinámico*, de modo que el código es compilado y ejecutado en el servidor y, utilizando las capacidades de redirección del sistema operativo, se le proporciona por la entrada una batería de casos de prueba creados de antemano. La salida dada por la solución es comparada con la esperada, creada con la solución oficial implementada por los autores del problema, y dependiendo del resultado de la comparación se proporciona un veredicto u otro. El abanico de posibles veredictos varía con cada juez, aunque los más habituales son:

- *Aceptado* (AC): la salida dada por el programa encaja exactamente con la esperada.
- *Error de presentación* (PE): la salida es correcta salvo por los separadores. Ocurre, por ejemplo, si el programa tiene espacios o saltos de línea sobrantes.
- *Respuesta incorrecta* (WA): la salida no encaja con la esperada.
- *Error de ejecución* (RTE): el programa falló durante la ejecución y no llegó a terminar. Es el veredicto habitual cuando un programa en C/C++ realiza una operación inválida, o uno en Java genera una excepción no controlada.
- *Límite de tiempo excedido* (TLE): el programa estaba tardando demasiado tiempo en ejecutarse, y fue detenido antes de que pudiera terminar.
- *Límite de memoria excedido* (MLE): el programa solicitó demasiada memoria, y fue detenido antes de terminar.
- *Error de compilación* (CE): el código enviado ni siquiera ha llegado a compilar.

Es significativo que en la mayoría de los jueces los veredictos que no son de aceptación no proporcionan ayuda adicional sobre la causa que lo ha originado, ni siquiera en aquellos pensados para docencia. Algunos proporcionan la salida del compilador en el caso del veredicto *Error de compilación*, pero no es habitual que se proporcione la causa de un *Error de ejecución* (por ejemplo, la señal que ha originado la detención de la ejecución, o la excepción generada), y mucho menos aún que se proporcione el caso de prueba que ha fallado y originado la *Respuesta incorrecta*. Las razones para esta ausencia de retroalimentación son múltiples. La primera quizá sea la relación histórica de estos jueces con los concursos de programación, en los que la información a los usuarios es ínfima al realizarse en contextos competitivos. Otra causa es evitar, en la medida de lo posible, las *fugas de información*. Si un juez online proporcionara los detalles de la excepción Java que ha ocasionado la detención de la ejecución, nada impediría que un usuario generara excepciones específicas para averiguar detalles de las interioridades de los casos de prueba secretos. Por último, no puede ser olvidada la dificultad de proporcionar más información,

pues debido a la generalidad de los problemas y las diferentes opciones para gestionar la entrada y la salida que tienen las soluciones, resulta complicado técnicamente averiguar con exactitud el caso de prueba que hizo fallar al programa.

Pese a la ausencia de retroalimentación, la evaluación de los envíos usando análisis dinámico del código gracias a los conjuntos de casos de prueba es utilizado también en muchos de los tan extendidos MOOC (*Massive Online Open Courses*, Cursos Online Masivos y Abiertos). No obstante, para paliar la ausencia de retroalimentación, han surgido muchos sistemas más específicos [2, 4, 15], que la mejoran a costa de restringir el abanico de posibles problemas que el usuario puede resolver. Esto complica la autoría de los problemas y su disponibilidad, pero consiguen una enseñanza mucho más dirigida.

## 3. El juez online *¡Acepta el reto!*

*¡Acepta el reto!*[2] es un juez online con más de 300 problemas de programación en español, creados pensando en los alumnos de las diferentes asignaturas de los Grados de Informática y Ciclos Formativos. El nivel de dificultad de los problemas es amplio, existiendo problemas diseñados específicamente para las primeras semanas de los cursos de introducción a la programación donde únicamente se conocen variables y expresiones, hasta problemas complejos que requieren algoritmos sobre grafos o conocimiento de teoría de números. Permite envíos en C, C++ y Java, y para cada envío realizado informa del tiempo y de la memoria total consumida.

*¡Acepta el reto!* nació en 2014 para servir, al igual que muchos otros jueces online, como *archivo vivo* de los problemas de programación creados por los autores. En particular, en 2011 habían puesto en marcha ProgramaMe[3], el concurso español de programación para Ciclos de Formación Profesional. Además, llevaban varios años usando programación competitiva para evaluación continua en clase utilizando software de jueces de concursos [5].

El número de problemas que habían creado utilizando los esquemas de los concursos iba en aumento, y para evitar que todos esos problemas fueran "de un solo uso" decidieron embarcarse en la aventura del desarrollo de un juez online, para que pudieran seguir siendo útiles a otros estudiantes.

Desde el principio, se plantearon varios puntos diferenciadores respecto a otros jueces online, a los que el juez sigue siendo fiel:

- Los enunciados de los problemas están en español. La inmensa mayoría de los problemas de los jueces online están en inglés, lo que supone una barrera para muchos de los alumnos de los primeros cursos de los grados o de los Ciclos de Formación Profesional.
- Los problemas están *categorizados* de acuerdo a diferentes ejes, varios directamente relacionados con los contenidos impartidos en las asignaturas de programación de los Grados y los Ciclos Formativos. Esto facilita tanto a profesores como a alumnos encontrar problemas aptos para el momento del aprendizaje en el que se encuentren.
- Muchos problemas y sus casos de prueba persiguen objetivos pedagógicos específicos, por lo que los límites de tiempo y memoria están afinados en función de ellos. Si bien no es extraño que en los concursos de programación el tiempo límite se ajuste para restringir las soluciones posibles, los jueces online que actúan como archivo de problemas hechos por terceros no siempre son tan cuidadosos. En lo que se refiere al límite de memoria, ni los concursos ni los jueces online son estrictos, por lo que *¡Acepta el reto!* es, hasta lo que alcanza nuestro conocimiento, el único juez que permite forzar soluciones con complejidades en memoria específica.

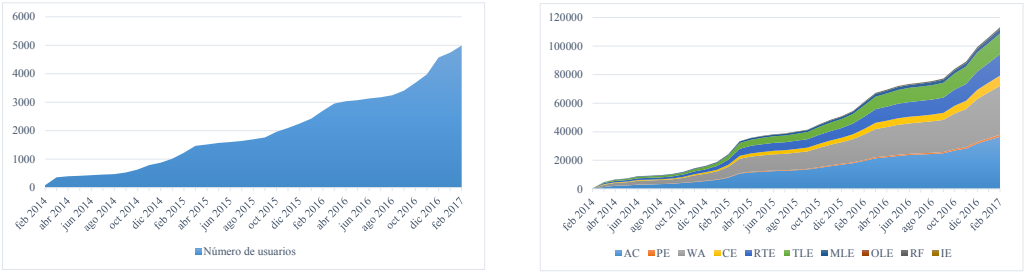
El juez recibió su primer envío el 17 de febrero de 2014, y desde entonces el número de problemas, usuarios y envíos no ha dejado de crecer. En enero de 2017 recibió su envío número 100.000 de un total de 5.000 usuarios de diferentes países hispanohablantes. La figura 1 muestra la evolución a lo largo del tiempo tanto del número de usuarios registrados como de envíos. Aparte de los veredictos habituales que se describieron en la sección 2, se añaden algunos otros más específicos disponibles en el juez, en particular *Cantidad de salida excedida* (OLE), *Función restringida* (RF) y *Error interno* (IE).

## 4. Arquitectura

Los jueces online deben realizar principalmente dos labores: proporcionar un interfaz de usuario para que los estudiantes naveguen por los problemas, lean los enunciados y hagan envíos, y ejecutar esos envíos, evaluarlos y proporcionar un veredicto. La forma en la que se implementen ambas tareas, y cómo se relacionen a nivel software, determinará la arquitectura general del juez online.

La literatura describe la arquitectura de varios de los sistemas para gestión de concursos más conocidos, así como algunos jueces online [3, 8, 11, 13]. Las aproximaciones seguidas varían enormemente, sobre todo

[2] `https://www.aceptaelreto.com`
[3] `http://www.programa-me.com`




(a) Número de usuarios registrados.

(b) Número de envíos y veredictos.

Figura 1: Estadísticas de *¡Acepta el reto!*

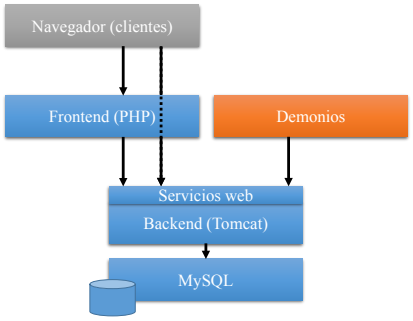



Figura 2: Arquitectura general de *¡Acepta el reto!*

dependiendo del instante en el que se iniciara su desarrollo al ser el principal factor que determina la tecnología que usa cada uno.

La arquitectura de *¡Acepta el reto!* está organizada en los siguientes elementos:

- *Backend*: se encarga del almacenamiento y acceso a los datos (usuarios, problemas, envíos, etcétera), publicando un conjunto de *servicios web* utilizados por el resto de elementos.
- *Frontend*: proporciona el *interfaz web* al juez online. Todo el acceso a los datos se realiza a través de los servicios web del *backend*.
- *Demonios*: se encargan de las tareas del juez online que están más allá de la gestión básica de los datos realizada por el *backend*. El demonio más importante es el que se encarga de evaluar los envíos que llegan, pero existen otros como el encargado de procesar los nuevos problemas que se desean publicar, o el que recorre los últimos envíos y actualiza los *rankings*. Los demonios se desarrollan utilizando un lenguaje u otro en función de su objetivo y necesidades.

La figura 2 muestra el esquema de comunicación general. El *backend* y el *frontend* se ejecutan en la misma máquina, pero los demonios están, por seguridad, en máquinas secundarias sin conexión directa a internet.

El *backend* hace uso de una base de datos en MySQL como sistema de almacenamiento principal, así como ficheros en disco para los enunciados de los problemas en sus diferentes formatos. Los servicios web están programados en una aplicación web que se despliega en una instancia de Tomcat.

El *frontend* está desarrollado en PHP, y proporciona las páginas web a los navegadores. En algunos casos, éstos se conectan directamente a los servicios web para actualizar dinámicamente el contenido, principalmente en las páginas que listan los últimos envíos.

En lo referente a seguridad, el punto más delicado está en el demonio que evalúa los envíos, dado que el juez recibe *código arbitrario* que debe ejecutar. Para mitigar los efectos de un posible código malicioso, los envíos se compilan y ejecutan en entornos de ejecución restringida. En particular:

- Cada envío se ejecuta utilizando un usuario y grupo del sistema nuevos, creados para la ocasión, sin contraseña (no puede hacer *login*), sin *shell* configurado ni directorio *home*.
- La ejecución se realiza en una *jaula chroot* que evita el acceso a ficheros sensibles.
- Se establecen límites de recursos en el sistema operativo antes de la ejecución de los envíos, restringiendo el número de procesos, consumo de CPU y memoria que pueden realizar.

La ejecución de los envíos en Java ocasiona una complicación adicional en lo referente a la restricción del uso de memoria. Como se ha comentado en la sección 3, una de las características del juez es la posibilidad de ajustar con exactitud el consumo de memoria, para poder forzar soluciones con complejidades en espacio específicas. Como se describirá más adelante, esto supone que el consumo máximo de memoria debe estar severamente restringido: aproximadamente el 75 % de los casos de prueba del juez se ejecutan imponiendo un límite de memoria de sólo 4 MiB, y el máximo límite utilizado son 64 MiB, usado en menos del 0.5 % de todos ellos.

La máquina virtual de Java (JVM), necesaria para

| Construcciones de programación | Estructuras de datos | Algoritmia |
|---|---|---|
| Expresiones | Arrays | Ad-hoc |
| Condicionales | Arrays multidimensionales | Recorridos |
| Bucles simples | Estructuras simples | Búsqueda |
| Bucles anidados | Cadenas | Búsqueda binaria |
| Procesamiento de la entrada | Pilas | Ordenación |
| Generación de la salida | Listas | Algoritmos voraces |
| Recursión | Colas | Programación dinámica |
| Límite de la representación | Colas de prioridad | Divide y vencerás |
| | Conjuntos | Búsqueda exhaustiva y vuelta atrás |
| | Árboles | Búsqueda en espacio de soluciones |
| | Tablas hash | |
| | Conjuntos disjuntos | |

Cuadro 1: Categorías principales de los problemas de *¡Acepta el reto!*

ejecutar los envíos programados en dicho lenguaje, requiere varios órdenes de magnitud por encima de ese límite para ponerse en marcha. El demonio que evalúa los envíos diferencia el lenguaje del envío de modo que si se realizó en un lenguaje compilado a código máquina (C o C++), el límite de memoria se establece para el proceso en conjunto, pero si se realizó en Java se aplica únicamente al *heap*, es decir al espacio de memoria disponible para memoria dinámica. Comparativamente esto significa que los envíos en C o C++ disponen de *menos memoria* en conjunto, porque en el mismo espacio deben incluir también el ejecutable y la pila. Sin embargo, la propia JVM hace uso del *heap* para mantener sus estructuras internas, y tras más de 300 problemas creados no se han observado diferencias que hagan pensar que con este modelo se favorece a unos lenguajes frente a otros.

El modo en el que el juez mide el tiempo y la memoria consumida por las soluciones recibidas también depende del lenguaje. La medición de la memoria máxima utilizada por la JVM se restringe también al uso del *heap*. Además, su medición tiene mucha menos precisión, dado que se ve interferida por el recolector de basura. El juez mide el consumo máximo de *heap* pero es probable que en ese pico buena parte de la memoria haya sido ya "liberada" por el programa y esté a la espera de ser reclamada por el recolector. Por tanto, existe una tendencia a que el juez informe de una memoria consumida mayor en los envíos en Java que en C o C++. Esto tiene implicaciones también durante la publicación de los problemas, pues no es fiable elegir el límite de memoria utilizando las soluciones en Java: si se pone un límite alto, la JVM tenderá a utilizarlo todo por no saltar el recolector, pero no es síntoma de que una determinada solución haga un uso excesivo de memoria. Por otro lado, decrementar la cantidad de memoria disponible tiene una repercusión negativa en el tiempo de ejecución al ocasionar más ejecuciones del recolector. Ha sido necesario paliar este problema incorporando mecanismos específicos para medir el consumo de CPU únicamente del *código de usuario* (y no de la JVM) en ese tipo de envíos.

Si bien los detalles quedan fuera del alcance de este artículo, lo que se pretende poner de manifiesto es que la inclusión de límites estrictos en el consumo de memoria, así como su medición, requieren técnicas específicas para evaluar las soluciones programadas en lenguajes que se ejecutan sobre máquinas virtuales con recolección de basura, como Java. Eso significa que incorporar nuevos lenguajes a la lista de los soportados por el juez (por ejemplo C# o Python) tiene implicaciones técnicas más allá de las sufridas por otros jueces online. Este es el principal motivo del reducido número de lenguajes soportados, pese a que su ampliación sea una característica a menudo solicitada por los usuarios.

# 5. Problemas disponibles

Los enunciados de todos los problemas tienen la misma estructura, similar a la utilizada en muchos concursos de programación. Comienzan con una ambientación para hacerlos más atrayentes. Por ejemplo, en lugar de pedir si un número es o no par se habla de la numeración de las viviendas en una calle (problema 217), para pedir ordenar tres números se habla de la devaluación del dólar zimbabuense (problema 356) y para pedir el mayor de una lista de números se recurre a los San Fermines (problema 149). Aparte de cumplir un objetivo cosmético, la ambientación también fuerza a los alumnos a *interpretar* el texto, separar lo superfluo de lo importante y deducir qué es lo que realmente se les está pidiendo. En algunas ocasiones, ésta resulta ser de hecho la parte más compleja, al esconder problemas clásicos de programación bien conocidos para que pasen desapercibidos (problema 145, "El tren del

amor", o problema 230, "Desórdenes temporales").

Tras la ambientación, se dan los detalles de la entrada que deberán procesar las soluciones, y se especifica el formato exacto de la salida. Finalmente se proporciona una batería de casos de ejemplo, cuyo objetivo no es permitir a los alumnos comprobar sus soluciones de antemano, sino sobre todo ayudarles a confirmar que han entendido lo que se está pidiendo.

La navegación por la batería de problemas se puede realizar por volúmenes o por categorías. Un volumen no es más que la agrupación de 100 problemas, sin ninguna relación lógica entre ellos más allá de que fueron incorporados al sistema en fechas cercanas. Así, por ejemplo, el primer volumen contiene los 100 primeros problemas que se publicaron (identificadores del 100 al 199), el segundo contiene los 100 siguientes, y así sucesivamente. El recorrido por volúmenes, por tanto, no está dirigido a un tipo de problema específico, y solo sirve para escoger un problema aleatorio o para hacerse una idea global del juego de problemas disponible.

A nivel pedagógico, es mucho más interesante el recorrido por categorías, pues permite una búsqueda dirigida en función de los elementos de programación o algoritmia que se quieran practicar. El cuadro 1 enumera las categorías más importantes, organizadas en tres ejes: construcciones de programación, estructuras de datos y algoritmia. Los problemas se incorporan en las categorías que mejor indican el objetivo pedagógico para el que fueron creados. Así, por ejemplo, el problema 356 de ordenar tres números pertenece a la categoría de *Condicionales* y no de *Ordenación*, dado que, al necesitar ordenar únicamente tres valores, se planteó para que los alumnos practicaran los condicionales y no la implementación de algoritmos de ordenación. Aun así, es habitual que los problemas pertezcan a más de una categoría; siguiendo con el mismo problema, también pertenece a la categoría *Límite de la representación*, al utilizar números grandes que requieren enteros de 64 bits, lo que exige a los alumnos preocuparse de los detalles de los límites.

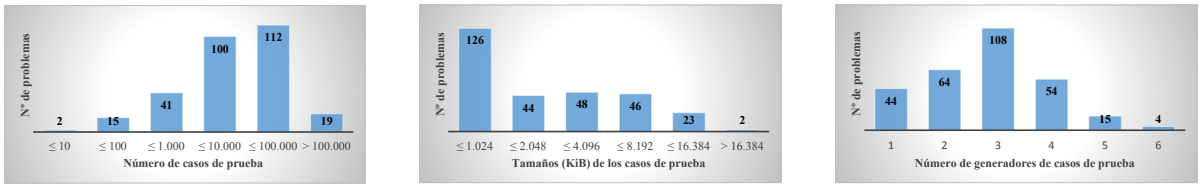
A pesar de la existencia de las categorías, cuando se muestra un problema concreto, éstas *no son visibles*; para muchos, parte de la diversión de enfrentarse por primera vez a un problema reside en averiguar su nivel de dificultad y las herramientas algorítmicas que son necesarias para su resolución. Si las categorías fueran visibles, desvelarían esa información que muchos prefieren no conocer, y que actúan como pistas. Naturalmente, los profesores tienen siempre la oportunidad de dar a sus alumnos tanta información como deseen, en particular informar sobre dichas categorías dependiendo del uso que vayan a dar a los problemas.

En lo que se refiere a la *autoría* de los problemas, aparte de crear el enunciado y los ejemplos es necesario realizar *soluciones oficiales* y *generadores de casos de prueba*. Para todos los problemas del juez los autores han creado, al menos, una solución en cada uno de los lenguajes soportados (C, C++ y Java). Esto no solo facilita la detección de errores, sino que permite medir el consumo de CPU y memoria de esas soluciones de referencia para poder especificar los límites.

Además de las soluciones oficiales, es necesario crear las baterías de casos con las que se pondrán a prueba los envíos de los alumnos. Como se describió en la sección 2, los problemas utilizan un esquema de la entrada tal que permite, en una sola ejecución, probar múltiples casos de prueba. Así, por ejemplo, en el problema de ordenar tres números cada caso de prueba serán esos tres números, y el programa deberá procesar tantos tríos como indique la primera línea de la entrada. Para proporcionar al lector una idea de la exhaustividad de las pruebas a las que se someten los envíos recibidos, la figura 3a muestra un histograma relacionando el número de problemas con la cantidad de casos de prueba (en escala logarítmica). Se puede observar que el 75 % de los problemas tiene más de mil casos de prueba, y que más del 45 % tienen más de 10.000. Estas cifras tienen repercusión en el tamaño físico de los ficheros de entrada suministrados y de salida esperados. La figura 3b muestra que más de la mitad de los problemas tienen juegos de pruebas que superan 1 MiB, y casi el 25 % supera los 8. El espacio necesario por los casos de todos los problemas roza los 800 MiB, lo que supone una media de más de 2.5 MiB por problema.

Los problemas son ejecutados múltiples veces, cada vez con una batería de casos distinta. Al menos se ejecutan con tres baterías diferentes: el ejemplo proporcionado en el enunciado, una batería vacía (sin ningún caso), y una batería adicional exhaustiva con muchos casos de prueba. Es importante que las pruebas realizadas a los envíos sean exhaustivas para no dar por buenas soluciones que no lo son, por lo que la mayoría de las veces los problemas se prueban con más baterías de casos. Es habitual disponer de una batería exhaustiva con infinidad de casos pequeños, otra con muchos casos medianos, y una última con unos pocos casos grandes, donde el significado de casos pequeños, medianos y grandes dependerá de la naturaleza de cada problema particular. La figura 3c muestra un histograma con el número de problemas que tienen diferentes cantidades de generadores de casos de prueba, sin contar el ejemplo y la batería vacía. Más del 60 % de los problemas tienen 3 o más generadores.

Las baterías de casos de prueba se especifican a través de *generadores*, es decir programas que escriben por su salida estándar la lista de casos de prueba. La creación de las soluciones y los generadores es especialmente importante en los problemas que quieren forzar soluciones con complejidades (en espacio o memoria) específicos.




(a) Número de casos de prueba. (b) Tamaño (KiB) de los casos de prueba. (c) Número de generadores.

Figura 3: Histograma del número de problemas respecto a sus casos de prueba.

Por ejemplo, el problema 132, "Las cartas del abuelo", pregunta, sobre una cadena de hasta 1.000.000 caracteres, si todas las letras de múltiples intervalos son la misma. Los límites del problema están planteados para que aquellas soluciones que sean lineales en el tamaño de cada intervalo no sean aceptadas por superar el tiempo límite. Para eso, los autores crearon no solo las soluciones oficiales correctas en cada lenguaje, sino hasta 5 soluciones más, con diferentes complejidades en tiempo, para estar seguros de que sólo las más óptimas serán admitidas en el juez. Otros problemas más complicados pueden requerir incluso más soluciones, como el problema 286, "Paradojas espacio-temporales" para el que los autores plantearon 17 soluciones diferentes.

En *¡Acepta el reto!* también hay problemas en los que la dificultad está en superar las restricciones de memoria. Esta característica se utiliza principalmente en problemas en los que se fuerza al procesamiento de la entrada sobre la marcha. Por ejemplo, el problema 248, "Los premios de las tragaperras", proporciona una secuencia de números y pide encontrar la subsecuencia contigua más larga que ocasione la mayor suma, sabiendo que se puede *reiniciar* (la secuencia de entrada funciona como un ciclo). Una de las dificultades del problema está en que el límite de memoria impide almacenar la secuencia en memoria, por lo que es necesario adaptar el algoritmo de Kadane para lidiar con esa restricción adicional.

## 6. Uso en clase

Actualmente, *¡Acepta el reto! no* proporciona ningún apoyo explícito a la docencia. En particular, el código enviado por un usuario es solamente visible por ese usuario, y por tanto no hay ninguna forma articulada que permita a un profesor ver el código enviado por sus alumnos.

Aun así, el sistema se ha utilizado con éxito en clase en diferentes asignaturas de varios Grados, y en módulos de Ciclos Formativos de Formación Profesional [6]. La disponibilidad de una batería de problemas lista para ser usada es suficiente motivación como para que muchos profesores decidan hacer uso del sistema a pesar de estas limitaciones. Cuando se utiliza para evaluación continua en el laboratorio, es habitual que el profesor exija que los alumnos que consiguen resolver el problema enseñen, sobre la plataforma, el envío que acaban de realizar. En otras ocasiones, los profesores solicitan que se les envíe el código por otros medios. Nos consta que algunos profesores prueban posteriormente las soluciones de sus estudiantes reenviándolas con sus propios usuarios.

Aunque tampoco se da soporte directamente, *¡Acepta el reto!* también se ha utilizado para organizar concursos locales en algunas instituciones educativas. En particular, algunos centros de secundaria organizan clasificatorios para elegir a sus representantes en ProgramaMe. En lugar de crear sus propios problemas y montar la infraestructura necesaria para organizar un concurso completo, utilizan directamente los problemas de *¡Acepta el reto!* y su juez para controlar los envíos de los equipos participantes.

## 7. Conclusiones y trabajo futuro

*¡Acepta el reto!* es un juez online centrado en la enseñanza de la programación, cuyo principal activo son *sus problemas*. La mayoría de ellos están planteados con un objetivo pedagógico en mente, y en función de él se encuentran categorizados. Esto permite que los profesores (y alumnos) puedan buscar fácilmente aquellos que más les interesen en función de la etapa de aprendizaje en la que se encuentren.

Mención especial merecen los problemas que fuerzan soluciones con complejidades en tiempo y espacio específicas. Si bien el control de la complejidad en tiempo es relativamente habitual en los concursos de programación, cuando esos problemas terminan en los jueces online normalmente los responsables prestan poca atención a esos detalles y las restricciones se pierden, al menos parcialmente. Mucho más novedosa es la inclusión de las restricciones en memoria, que, hasta donde llega el conocimiento de los autores, no se tiene en cuenta en los concursos de programación ni tampoco en otros jueces online.

Aunque *¡Acepta el reto!* nació para facilitar la enseñanza y el aprendizaje de la programación en español,

todavía hay puntos claros de mejora en él. La inclusión del perfil del profesor es clave para facilitar su uso en clase, y la inclusión de *pistas* y *recomendaciones* lo es para evitar la pérdida de usuarios debido a la frustración. Aunque el *backend* del sistema se planteó desde el principio para poder dar servicio a estas nuevas funcionalidades, todavía es necesario realizar trabajo adicional para tenerlas completamente operativas.

## Agradecimientos

Aunque el desarrollo y puesta en marcha de *¡Acepta el reto!* sea culpa principalmente del esfuerzo de los autores, debemos agradecer a varios antiguos alumnos, Jéssica Martín, Javier Martín, Pablo Suárez, Luis María Costero y Jesús Javier Domenech, su implementación de los primeros prototipos del *frontend* web. Éstos sirvieron como prueba de concepto mientras se desarrollaba el *backend* y todos sus servicios.

Además, estamos en deuda con varios profesores y compañeros que han colaborado con nosotros en la creación de bastantes de los problemas. Patricia Díaz comenzó en la andadura de ProgramaMe y la redacción de sus problemas, y hoy Alberto Verdejo es quién mejor ha interiorizado nuestra pasión por el cuidado en la creación de los problemas. No podemos olvidar tampoco a Isabel Pita ni a Clara Segura, o a otros que, en mayor o menor medida, han colaborado, y colaboran, en la ampliación de la batería de problemas del juez.

Este trabajo ha sido financiado por la UCM (Grupo 910494) y por el Ministerio de Economía, Industria y Competitividad (TIN2014-55006-R).

## Referencias

[1] ACM/ICPC. *Fact Sheet – The 41st Annual World Finals of the ACM Internacional Collegiate Programming Contest (ICPC)*, 2016.

[2] Xavier Baró, David Masip, Elena Planas y Julià Minguillón. PeLP: Plataforma para el aprendizaje de lenguajes de programación. En *Actas de las XIX Jornadas de la Enseñanza Universitaria de la Informática*, JENUI, 2013.

[3] California State University, Sacramento's. *PC2: Contest Administrator's Installation and Configuration Guide*, 2016.

[4] Pedro Delgado-Pérez e Inmaculada Medina-Bulo. Automatización de la corrección de prácticas de programación a través del compilador clang. En *Actas de las XXI Jornadas de la Enseñanza Universitaria de la Informática*, JENUI, 2015.

[5] Marco Antonio Gómez Martín y Pedro Pablo Gómez Martín. Uso de software de gestión de concursos de programación para evaluación continua. En *Actas de las XIX Jornadas de la Enseñanza Universitaria de la Informática*, JENUI, 2013.

[6] Marco Antonio Gómez Martín y Pedro Pablo Gómez Martín. Engánchalos antes de que escapen. Estrategias para luchar contra el absentismo. En *Actas de las XXIII Jornadas de la Enseñanza Universitaria de la Informática*, JENUI, 2017.

[7] Artem Iglikov, Mansur Kutybayev y Bakhyt Matkarimo. IOI 2015 report. *Olympiads in Informatics*, 10, 2016.

[8] Thijs Kinkhorst. *DOMjudge at Amrita*, 2013.

[9] Adrian Kosowski, Michał Małafiejski y Tomasz Noiński. Application of an online judge & contester system in academic tuition. En *Proceedings of the 6th International Conference on Advances in Web Based Learning*, ICWL'07, 2008. Springer-Verlag.

[10] Andy Kurnia, Andrew Lim y Brenda Cheang. Online judge. *Comput. Educ.*, 36(4), 2001.

[11] José Paulo Leal y Fernando Silva. Mooshak: A web-based multi-site programming contest system. volume 33, 2003. John Wiley & Sons, Inc.

[12] Yingwei Luo, Xiaolin Wang y Zhengyi Zhang. Programming grid: A computer-aided education system for programming courses based on online judge. En *Proceedings of the 1st ACM Summit on Computing Education in China on First ACM Summit on Computing Education in China*, SCE'08, 2008. ACM.

[13] Jordi Petit, Omer Giménez y Salvador Roura. Jutge.org: An educational programming judge. En *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE'12, 2012. ACM.

[14] Miguel Á. Revilla, Shahriar Manzoor y Rujia Liu. Competitive learning in informatics: the UVa online judge experience. En Elena Verdú, Rubén M. Lorenzo, Miguel Á. Revilla y Luisa M. Regueras, editors, *A New Learning Paradigm: Competition Supported By Technology*. Sello Editorial, 2010.

[15] Óscar Sapena, Mabel Galiano, Marisa Llorens y Natividad Prieto. Aprender, enseñar y evaluar con CAP, un corrector automático de tareas de programación. En *Actas de las XIX Jornadas de la Enseñanza Universitaria de la Informática*, JENUI, 2013.

[16] Gui Ping Wang, Shu Yu Chen, Xin Yang y Rui Feng. OJPOT: online judge & practice oriented teaching idea in programming courses. *European Journal of Engineering Education*, 41(3), 2016.