Feedback en Jueces Online

Trabajo de Fin de Máster
en Ingeniería Informática

Jennifer Hernández Bécares

Dirigido por el Doctor
Pedro Pablo Gómez Martín
y por el Doctor
Marco Antonio Gómez Martín

Calificación: **9**

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2017

Feedback en Jueces Online

Trabajo de Fin de Máster realizado por

Jennifer Hernández Bécares

Dirigido por el Doctor

Pedro Pablo Gómez Martín

y por el Doctor

Marco Antonio Gómez Martín

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Septiembre 2017

Copyright © Jennifer Hernández Bécares

Septiembre, 2017

La abajo firmante, matriculada en el Máster en Ingeniería en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: **Feedback en Jueces Online**, realizado durante el curso académico 2016-2017, bajo la dirección de Pedro Pablo Gómez Martín y Marco Antonio Gómez Martín en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en internet y garantizar su preservación y acceso a largo plazo.

Fdo: Jennifer Hernández Bécares

Agradecimientos

If you work really hard and you're kind,
amazing things will happen.

Conan O'Brien

A mis padres, por su apoyo continuo y por respetar mis decisiones a pesar de que a veces son repentinas y algo cuestionables.

A Viktor, fuente de inspiración y creatividad ilimitadas, ingenio infinito y paciencia insuperable, dispuesto a hacerme reír siempre, y *especialmente* en mis peores momentos.

A Álvaro, por una amistad incondicional que sólo se refuerza con el paso de los años.

A Marco Antonio y Pedro Pablo, que una vez más han demostrado que las reuniones de trabajo siempre son mejores con un café y un paquete de galletas.

Y a todos los que de un modo u otro habéis contribuido a animarme o apoyarme durante el transcurso de este máster, compartido conversaciones, sonrisas, lágrimas o complicidades.

Nada de esto sería posible sin vosotros. Gracias.

Resumen

Los *jueces online* son sistemas que surgieron inicialmente con el objetivo de actuar como *repositorios de problemas* que habían formado parte de concursos de programación. Con el tiempo, se han ido popularizando y ahora se utilizan también como *herramientas didácticas*, y permiten practicar y aprender algoritmia y métodos de programación de forma entretenida, llegando incluso a utilizarse en las aulas. Así, una situación común es que el profesor explique un tema e indique a sus alumnos qué ejercicios del juez pueden usar para practicar lo que ha contado.

Sin embargo, la mayoría de jueces existentes no están adaptados completamente a las necesidades de esos usuarios. Cuando los alumnos envían una solución a un problema, pueden recibir un veredicto positivo (solución correcta) o un veredicto negativo (incorrecta). Al recibir un veredicto negativo, el alumno tiende a sentirse perdido y desmotivado, y no sabe cómo continuar, ya que el feedback que da el juez ante los errores cometidos es inexistente.

En este trabajo se estudian las diferentes formas de incluir pistas en los jueces online, que ayudarán a los usuarios en su *camino hacia el éxito*, dándoles información sobre los errores que están cometiendo para que intenten corregirlos. Para ello, se ha realizado una implementación para el juez *¡Acepta el Reto!*, desarrollado por los directores de este trabajo. Dicha implementación utiliza las salidas generadas por las soluciones que envían los usuarios para agruparlos en función del error que han cometido y posteriormente darles una pista personalizada. Además, se proporciona una herramienta de visualización del grafo de envíos, así como una herramienta de comparación de soluciones, bajo la convicción de que ambas resultarán muy útiles a la hora de ayudar al autor o a los propios alumnos a proponer pistas para *futuros usuarios*.

Palabras clave: jueces en línea, enseñanza de programación, feedback en sistemas de enseñanza online

Abstract

Online judges are systems that initially emerged with the objective of acting as *repositories* of problems that had been part of programming contests. Over time, these judges became more popular and now they are used as *didactic tools*, allowing to practice and learn algorithms and programming methods in an entertaining way, even being used in classrooms. Thus, a common situation is for the teacher to explain a topic and tell his students what exercises from the judge they can use to practice what was explained.

However, most of the existent judges are not fully adapted to the needs of those users. When students submit a solution to a problem, they may receive a positive verdict (correct solution) or negative verdict (incorrect solution). Upon receiving a negative verdict, the student tends to feel lost and unmotivated, and does not know how to continue solving the problem, since the feedback received is non-existent.

In this work we study the different ways of including hints in online judges, which will help users on their *path to success*, giving them information about the mistakes they are making and trying to correct them. For this, an implementation for the judge *¡Acepta el Reto!*, developed by the advisors of this work. This implementation uses the output generated by the solutions sent by users to group them based on the error they have committed and then gives them custom feedback based on it. In addition, we provide a visualization tool for the graph of submissions, as well as a solution comparison tool, with the conviction that both of them will be very useful in helping the author or the students themselves to propose hints for *future users*.

Keywords: online judges, the teaching of programming, feedback in e-learning platforms

Índice

Agradecimientos vii

Resumen ix

Abstract xi

1. Introducción **1**
1.1. Antecedentes y motivación del proyecto 1
1.2. Objetivos y plan de trabajo 1
1.3. Contenido de la memoria 2

2. Sistemas de Enseñanza **5**
2.1. Introducción 5
2.2. CAI 6
2.3. ITS 8
2.4. MOOC 10
2.5. Jueces Online 12
2.6. Herramientas de detección de plagio 21

3. Feedback en Jueces Online **25**
3.1. Introducción 25
3.2. Formas Comunes de Proporcionar Feedback 27
3.3. Agrupación de Soluciones en Función de la Salida . . . 31
3.4. Clases de Equivalencia vs. Análisis Estático 36

4. Inclusión de Pistas en el Juez Online **43**
4.1. Introducción 43
4.2. Pistas a Priori 44
4.3. Pistas A Posteriori 45
4.4. Pistas Sociales 47

5. Visualización del Grafo de Envíos **49**
5.1. Introducción 49
5.2. Visualización del Grafo: Autor 50
5.3. Visualización del Grafo: Usuario 52
5.4. Comparación de Soluciones 54

6. Implementación **59**
6.1. Introducción 59
6.2. Implementación General 59
6.3. Visualización de grafos 61
6.4. Comparación de soluciones 62

7. Conclusiones y Trabajo Futuro **65**
7.1. Conclusiones 65
7.2. Trabajo Futuro 66

A. Introduction **69**
A.1. Background and motivation of this project 69
A.2. Objectives and work plan 69
A.3. Contents of this work 70

B. Conclusions and Future Work **73**
B.1. Conclusions 73
B.2. Future Work 74

Bibliografía **77**

Índice de figuras

2.1. Lista de problemas de UVa OJ agrupados por temas . . 15
2.2. Página de envíos de un usuario en UVa OJ 16
2.3. Enunciado del problema "Último dígito del factorial" de ACR 17
2.4. Ejemplo de entrada y salida del problema "Último dígito del factorial" 18
2.5. Entrada, salida y significado de un problema del juez ACR 19

3.1. Número de envíos y veredictos de ACR 32
3.2. Grafo de similitud generado por AC con una distancia máxima de 0.08 y mostrando 22 aristas 37
3.3. Comparación de los códigos con id 78285 y 88353 del problema 114 de ACR 39
3.4. Comparación de los códigos con id 77722 y 77728 del problema 114 de ACR 40

5.1. Grafo de envíos del problema número 114 de *ACR* . . . 51
5.2. Tooltip con parte del listado de ids de los envíos al problema 114 de ACR, que emerge al pasar el puntero del ratón por encima del nodo 3 52
5.3. Código del envío número 77130 en *ACR* 53
5.4. Envíos del problema 114 para el usuario escogido . . . 54
5.5. Comparación de envíos del problema 114 para el usuario escogido (parte 1) 55
5.6. Comparación de envíos del problema 114 para el usuario escogido (parte 2) 57
5.7. Comparación de envíos del problema 114 para el usuario escogido (parte 3) 58

6.1. Arquitectura general de ¡Acepta el Reto! 60
6.2. Ejemplo de comparación con la herramienta Pretty Diff 63

Capítulo 1

Introducción

It's the questions we can't answer that teach us the most. They teach us how to think. If you give a man an answer, all he gains is a little fact. But give him a question and he'll look for his own answers.

Patrick Rothfuss, The Wise Man's Fear

Antecedentes y motivación del proyecto

La idea de este proyecto surge de la necesidad latente de incorporar nuevas formas de dar feedback a los usuarios de los jueces online. Los jueces online son sistemas que surgieron inicialmente con el objetivo de actuar como repositorios de problemas que habían formado parte de concursos de programación. Sin embargo, con el tiempo, su uso se ha extendido a las aulas y se han empezado a utilizar con fines curriculares.

En contraposición con los sistemas de enseñanza online, los jueces no disponen de herramientas específicas para ayudar a los usuarios a resolver problemas de forma correcta. Este hecho hace que muchas veces esos usuarios pierdan la paciencia al intentar resolver un problema, recibiendo veredictos negativos ante las soluciones enviadas y sin modo alguno de saber qué están haciendo mal.

Objetivos y plan de trabajo

En función de lo expuesto en la sección de motivación, el objetivo principal de este trabajo es estudiar diferentes formas de incluir

pistas en los jueces online para ayudar a los usuarios proporcionándoles información sobre sus errores. Después de eso, se desarrollará una herramienta que agrupe a los usuarios en función de los errores que han cometido y permita tanto a los autores de los problemas como a los usuarios que los resuelven añadir pistas o ver las pistas añadidas para ese error concreto. De este modo se busca mejorar el proceso de aprendizaje y a la vez aumentar sus conocimientos de algoritmia y programación.

El plan de trabajo es el que sigue:

- *Estudio previo:* se realizará un estudio de los diferentes sistemas de enseñanza online que están disponibles, de las diferencias entre jueces online y de posibilidad de utilizar herramientas de detección de plagio para agrupar a los usuarios en función de los errores de programación que cometen al resolver un problema. Por otro lado, se analizará qué formas de dar feedback están presentes en los jueces online actualmente.

- *Desarrollo del proyecto:* se estudiará cuál es la mejor forma de agrupar a los usuarios en función de sus errores cometidos y qué formas novedosas de incluir pistas se pueden plantear en los jueces online. Además, nos plantearemos qué funcionalidad hay que añadir a un juez para poder incluir esos sistemas de ofrecer pistas. Posteriormente se implementará esa nueva funcionalidad para el juez online Acepta el Reto.

- *Trabajo futuro:* se concluirá con los aspectos más relevantes del proyecto y se analizará qué partes tienen aún margen para mejora y ampliación.

Contenido de la memoria

Capítulo 2: *Sistemas de Enseñanza.* Expone los principales tipos de sistemas de enseñanza, una introducción a los jueces online y habla de las herramientas de detección de plagio.

Capítulo 3: *Feedback en Jueces Online.* Introduce las ventajas de introducir feedback en los jueces en cuanto a motivación y mejora del aprendizaje e introduce las formas tradicionales de dar feedback.

Capítulo 4: *Inclusión de Pistas en el Juez Online.* Describe las tres formas de incluir pistas que consideramos útiles y lo suficientemente generales como para incorporarlas en cualquier juez online.

Capítulo 5: *Visualización del Grafo de Envíos*. Muestra la implementación llevada a cabo para el juez Acepta el Reto, que entre otras cosas permite a los autores de los problemas y a los usuarios ver una representación del grafo completo de soluciones enviadas agrupadas por su salida y que permitirá incorporar feedback para los usuarios en función de los errores que han cometido.

Capítulo 6: *Implementación*. Expone las herramientas y librerías que se han utilizado durante la implementación de este trabajo.

Capítulo 7: *Conclusiones y Trabajo Futuro*. Concluye con los aspectos más importantes mencionados y realiza un estudio de las posibles ampliaciones de este trabajo.

Capítulo 2

Sistemas de Enseñanza

The test of success is not what you do when you are on top. Success is how high you bounce when you hit the bottom.

George S. Patton Jr.

Resumen: En primer lugar, en este capítulo se exponen los principales tipos de sistemas de enseñanza: CAI (2.2), ITS (2.3) y MOOC (2.4). Posteriormente se realizará una introducción a los jueces online (2.5), que se presentan como una herramienta auxiliar novedosa en el proceso de aprendizaje de programación en ciertos contextos. Finalmente, para tener una visión completa de las diferentes posibilidades de comparación de soluciones de un problema se analizarán algunas de las herramientas de detección de plagio más conocidas (2.6) y se estudiará si son útiles para la detección de errores de programación comunes.

Introducción

Tradicionalmente, la educación y el aprendizaje se ha llevado a cabo exclusivamente en las aulas, imponiendo los roles de profesor-alumno de forma incuestionable y difícilmente modificable. Sin embargo, con el auge de los sistemas informáticos y cada vez más personas en posesión de diferentes dispositivos electrónicos, se pone de manifiesto la necesidad de impulsar los sistemas de enseñanza. En 2015, por ejemplo, un 78 % de los adultos entre 18 y 29 años en Estados Unidos disponían de ordenador personal, mientras que el 86 % tenían un smartphone [1]. Estas cifras ponen de manifiesto la importancia de llegar a los usuarios por esta vía.

Por otro lado, las desorbitadas tasas que hay que pagar para tener acceso a una educación especializada de calidad y la imposibilidad de que los profesores dediquen a cada alumno el tiempo que realmente necesitan para recibir feedback sobre su aprendizaje ha llevado a buscar mecanismos para facilitar y automatizar la docencia. Con ello se busca, entre otras cosas, reducir costes.

De esta manera surgen sistemas como los CAI (2.2), los ITS (2.3) o los MOOC (2.4). Estos sistemas permiten a todo tipo de usuarios acceder a los contenidos de forma sencilla, recibir feedback sobre sus progresos de forma inmediata y encontrar temarios hechos a medida. En contra de lo que cabría pensar, no son sólo importantes en ramas técnicas, sino que se utilizan como herramientas curriculares en múltiples disciplinas.

En este capítulo hablaremos en detalle sobre estos sistemas. Adicionalmente, se explicará también qué son los jueces online y se hablará de herramientas de detección de plagio, que son importantes en el contexto particular del aprendizaje de programación y están relacionados con el trabajo que vamos a realizar.

CAI

A mediados de los años cincuenta y principios de los sesenta, una colaboración entre la Universidad de Stanford en California y la empresa IBM introdujo la Enseñanza Asistida por Ordenador (Computer Assisted Instruction, o *CAI*, [2]) en escuelas selectas. Inicialmente, los programas de CAI eran una presentación lineal de ejercicios y sesiones de práctica. Estos primeros sistemas estaban bastante limitados por su coste y el difícil acceso a computadoras por aquel entonces. Por otro lado, el sistema PLATO (Programmed Logic for Automatic Teaching Operations), desarrollado en la Universidad de Illinois al comienzo de los años sesenta, fue otro de los sistemas CAI tempranos, utilizado en estudios superiores. El sistema PLATO consistía en un ordenador principal que soportaba hasta 1000 terminales conectados para el uso individual por parte de los estudiantes. En 1985, más de 100 sistemas PLATO operaban en Estados Unidos.

Los sistemas CAI, como su propio nombre indica, se basan en utilizar un ordenador para proporcionar y recibir formación. El formato puede ser desde un simple programa para enseñar mecanografía hasta sistemas complejos que utilizan las últimas tecnologías para enseñar nuevas técnicas de cirugía [3]. Dicho de otra manera, es una técnica de instrucción interactiva que utiliza una computadora para presentar los materiales y supervisar el aprendizaje que tiene lugar [4]. Utiliza una

combinación de texto, gráficos, sonido y vídeo para potenciar el proceso de aprendizaje, y tiene múltiples propósitos que ayudan al estudiante.

Además, estos sistemas traen consigo varios beneficios [3]; entre ellos, el usuario puede aprender a su propio ritmo de forma autodirigida, y el contenido se puede representar en una gran variedad de medios. Los estudiantes pueden avanzar tan lenta o rápidamente como gusten a través del programa de aprendizaje. Además, si quieren repetir alguna tarea o revisar material de nuevo, pueden hacerlo tantas veces como quieran. Esto también beneficia a los profesores, ya que el programa no se cansará ni se quejará de tener que repetir la información, ni se perjudicará a otros alumnos contando el mismo tema varias veces. Del mismo modo, los alumnos podrán saltarse un tema si ya conocen la información que en él se presenta, haciendo que el proceso de aprendizaje sea más eficiente.

Esas mismas ventajas que se han mencionado se pueden también poner en contra del alumno [3]. Al tener todo bajo su control, los estudiantes pueden sentirse solos y abrumados por la cantidad de información y recursos disponibles. Alternativamente, podría darse la situación contraria y que todo sea demasiado guiado si no se pone cuidado y los métodos utilizados en el aula son transferidos al programa.

Con el paso de los años, surgieron herramientas de autoría para generar contenido para sistemas educativos online. Con los sistemas de gestión de contenido aparecieron también estándares como SCORM (Sharable Content Object Reference Model, [5]), que es un conjunto de especificaciones que permite crear objetos pedagógicos estructurados. Originalmente, los sistemas de gestión de contenidos utilizaban formatos propietarios para los contenidos que distribuían, haciendo imposible el intercambio de tales contenidos. SCORM hace posible el intercambio, ya que permite crear contenidos que pueden importarse dentro de sistemas de gestión de aprendizaje diferentes, siempre y cuando soporten la norma SCORM.

Por otro lado, una herramienta de CAI típica tiene las siguientes características [4]:

- Contenido en forma de texto o multimedia
- Preguntas “multiple choice” o de múltiples respuestas
- Problemas a resolver
- Feedback inmediato
- Notas sobre las respuestas incorrectas
- Resúmenes del rendimiento del alumno

- Ejercicios para practicar
- Exámenes

Finalmente, tiene sentido hablar de CAI desarrollado específicamente para el aprendizaje de programación, ya que se ha demostrado que utilizar la tecnología para este fin puede ser una de las herramientas más eficaces para enseñar los cursos introductorios [6].

Las habilidades de programación deben ser adquiridas con una gran cantidad de práctica. Por ello, el apoyo de los CAI es muy importante para poder proporcionar a los alumnos la práctica que necesitan para consolidar el conocimiento adquirido [7]. Además, el apoyo inmediato a los estudiantes es un factor crítico para que el aprendizaje se complete con éxito. Sin embargo, introduce una gran presión en los recursos disponibles y sus capacidades, y puede que ni siquiera sea asequible para algunas universidades [8].

Como ejemplo de sistema CAI tenemos el sistema PASS [8, 9], un CAI para la enseñanza de programación basado en web que ayuda a los estudiantes a aprender programación de forma completamente automática. PASS anima a los estudiantes a practicar más sin dudar acerca de sus errores de programación. Un entorno de aprendizaje con tales características puede aumentar su motivación considerablemente y, por tanto, ayudarles a continuar practicando sus habilidades de programación.

ITS

Bajo el nombre de "Intelligent CAI", surgió en 1970 Scholar, un programa que enseñaba geografía de Sudamérica mediante diálogo. Ese fue el origen de los Sistemas de Tutoría Inteligente (Intelligent Tutoring Systems, también conocidos como *ITS*, [10]). Los ITS son herramientas que tienen como objetivo proporcionar instrucciones o feedback de forma inmediata y personalizada a los usuarios que utilizan el sistema como forma de aprendizaje, sin requerir ningún tipo de intervención de un profesor real. Estos sistemas incorporan técnicas de Inteligencia Artificial (*IA*) con el fin de simular un comportamiento que adapte los contenidos a las necesidades del usuario de forma inteligente.

Los ITS pueden considerarse los sistemas inmediatamente posteriores en el tiempo a los CAI, y se diferencian de ellos en varios aspectos [11]:

- Los ITS proporcionan una gran cantidad de conocimiento para un dominio limitado.

- A diferencia de los CAI, los ITS constan de un modelo de desempeño de los estudiantes que se mantiene de forma dinámica y se utiliza para impulsar la enseñanza.
- El diseñador del ITS define el conocimiento, pero no la secuencia de enseñanza que se va a seguir.
- Los ITS proporcionan diagnósticos detallados de los errores en lugar de ofrecer sólo ejercicios y práctica.
- Los estudiantes pueden hacer preguntas a los ITS.

Por otro lado, los ITS pueden utilizarse como una plataforma de aprendizaje adaptativo, con un ambiente idóneo para el aprendizaje individual de los estudiantes. Sin embargo, los ITS tienen un problema inherente: es muy difícil cambiar o modificar sus características. Realizar cualquier cambio, por pequeño que sea, requiere de la intervención de los desarrolladores del sistema, que son los que tienen habilidades de programación. Es muy frecuente que los profesores necesiten modificar la estructura de los cursos o incorporar nuevos materiales de aprendizaje. Para solventar esta situación, surgen los sistemas o *herramientas de autoría* (en inglés: *authoring systems*) de ITS [12]. Estos sistemas tienen que enfrentarse a varios problemas, entre ellos reducir el esfuerzo tanto en tiempo como en coste para crear nuevos contenidos, disminuir el nivel técnico requerido para ser capaces de incorporar nuevos temas o diseñar métodos de evaluación para el mismo. Por todo ello, se vuelve muy difícil crear nuevos ITS [13]. Además, por todo lo anterior, su uso está a menudo restringido a problemas pequeños y muy guiados para los que el sistema posee una solución de referencia previamente [14]. Esto, como veremos más adelante, no se puede aplicar a nuestro contexto, ya que las soluciones de referencia sólo muestran un modo de resolver el problema de entre las miles de formas que puede haber de resolverlo, por lo que no puede utilizarse con ese fin.

A pesar de todo ello, el campo de los ITS ha sido prolífico en el ámbito de la enseñanza de la programación, empezando por el famoso LISP Tutor [15], uno de los pioneros en el campo. Con el tiempo, le han seguido muchos en otros lenguajes, como C++ [16, 17] y Java [18].

La arquitectura estándar de los ITS consta de los siguientes componentes básicos [11]:

- El módulo de conocimiento experto (o experto en dominio).
- El módulo del modelo de estudiante.
- El módulo de tutorización.

Los ITS resultan diferentes a los CAI principalmente por la existencia del módulo de conocimiento experto. Este módulo sirve, en primer lugar, como fuente de conocimiento para presentarle al estudiante preguntas, respuestas y explicaciones. En segundo lugar, proporciona un estándar para evaluar el desempeño del estudiante. Para ello debe ser capaz de generar soluciones a problemas en el mismo contexto en el que está el estudiante, para que las respuestas se puedan comparar. También debe ser capaz de detectar errores sistemáticos comunes y, si es posible, identificar cualquier "agujero" en lo que ha aprendido el estudiante que pueda ser la causa de tales errores.

Por su parte, el módulo del modelo de estudiante se refiere a la representación dinámica del conocimiento y las habilidades que adquiere el estudiante, ya que no puede existir un sistema de tutoría inteligente sin entender al estudiante. Este módulo, idealmente, debería incluir los aspectos del comportamiento del estudiante que pueden tener repercusión en su aprendizaje.

El módulo de tutorización es la parte del ITS que diseña y regula las interacciones con el estudiante. En otras arquitecturas, este módulo se denomina módulo pedagógico o estrategia de enseñanza. Está muy relacionado con el modelo de estudiante, ya que el orden y la forma en la que se muestran los temas depende en gran medida de la persona a la que se expone la información.

Por todo lo anterior, los ITS son una aportación muy importante en el campo de los sistemas de enseñanza.

MOOC

MOOC [19] es el acrónimo en inglés de Massive Online Open Courses (cursos online masivos y abiertos). Los MOOC permiten a cientos de miles de estudiantes de todo el mundo acceder a contenido específico de ciertos temas sin necesidad de pertenecer a ninguna institución para acceder al contenido y sin límite de matriculaciones. El acceso es libre, abierto y gratuito, y se cursan completamente en línea. La interacción entre los alumnos o entre alumno y profesor se lleva a cabo a través de foros o herramientas de videoconferencia. Su estructura está diseñada especialmente de forma que promueva el aprendizaje autónomo de los estudiantes.

De alguna manera, los MOOC son los sucesores más directos de los CAI [20]. Los MOOC enfocados a grandes audiencias se basan esencialmente en los mismos principios pedagógicos, ofreciendo la misma funcionalidad que ofrecían los CAI pioneros pero ahora con mejores interfaces, ordenadores más rápidos y acceso a través de internet en

lugar de a través de disquetes o CD-ROMs. Adicionalmente, añaden funcionalidad como la corrección por pares para hacer frente a la gran cantidad de usuarios de los que disponen.

Probablemente la plataforma de MOOC más conocida de todas es Coursera[1], desarrollada por académicos de la Universidad de Stanford y con un catálogo de cursos de temática variada muy amplio. También son muy conocidas las plataformas Udacity[2] (especializada en programación e informática) y edX[3].

En el contexto de la programación, los MOOC proporcionan un sistema de corrección automática o, en ocasiones, de corrección por pares (entre alumnos). Sin embargo, es necesario diseñar formas de proporcionar feedback al alumno sobre qué errores está cometiendo, ya que la revisión personalizada llevada a cabo por personas es impensable debido a la cantidad de usuarios. Es por eso que estos sistemas intentan alejarse de los métodos tradicionales y hacer uso de aprendizaje máquina (*machine learning*) sobre corpus masivos de soluciones que han enviado los alumnos para intentar guiar al alumno. Haciendo uso del aprendizaje máquina, los MOOC pretenden aprovechar la gran cantidad de datos y casos que se generan por su condición de "masivos" y aprovecharlo para dar feedback. Esto se diferencia de los ITS antes mencionados, ya que en ellos el conocimiento relativo a los errores de los usuarios es explícito, y los autores tienen que incorporar IA para conseguir inferir qué está pasando.

En [21] introducen un método que utiliza una red neuronal para codificar los programas como un mapeo lineal desde un espacio de precondiciones a un espacio de postcondiciones y propone un algoritmo para dar feedback a gran escala utilizando estas características. Así, propagan los comentarios humanos en las entregas de los alumnos para que lleguen a más personas.

Por otro lado, en [22] intentan relacionar la sintaxis y la similitud funcional de los envíos para explorar las diferentes variaciones de las soluciones, bajo la suposición de que un número muy elevado de soluciones a un problema tiene un número limitado de formas (correctas) únicas de resolverse.

En [23] hablan de incorporar pistas de forma automática en un videojuego educacional llamado BOTS, que está diseñado para enseñar a estudiantes de secundaria los principios de la programación. Debido a que los rompecabezas que componen el juego crecen de forma muy rápida (ya que los propios alumnos pueden proponer problemas nue-

[1] `https://es.coursera.org/`
[2] `https://www.udacity.com/`
[3] `https://www.edx.org/`

vos), es necesario que el feedback y las pistas puedan generarse con relativamente pocos datos, ya que su revisión por parte de expertos es inviable. Así, introducen una forma de utilizar la salida que genera el código de los estudiantes para ver cómo de cerca están de la solución al problema y proporcionarle pistas que le guíen en el camino.

Finalmente, en [24] hablan de la generación de pistas a alto nivel que puedan proporcionar pistas para sub-objetivos en el problema, sugiriendo diferentes estrategias.

En los cuatro artículos mencionados, las soluciones que proponen para aportar feedback se basan en el uso de aprendizaje máquina (machine learning) para hacer "clustering" de los errores de los alumnos bajo la hipótesis de que soluciones parecidas necesitarán feedback parecido. Así, lo que hacen los tutores humanos es proporcionar pistas *a mano* para los diferentes tipos de envíos que se han reconocido, en lugar de crear sistemas expertos con reglas, como se hacía previamente. De este modo, estos sistemas se acercan al razonamiento basado en casos [25].

Sin embargo, hay desacuerdos con respecto a esta forma de proporcionar pistas [21] porque la similitud en la estructura del código entre dos soluciones distintas a un problema no siempre tiene por qué ser buen indicador de cómo de parecido puede ser el feedback que se dé a cada uno de ellos.

Jueces Online

Los sistemas descritos anteriormente tienen una intención pedagógica y son más simples de desarrollar que los ITS, pero no dejan de estar centrados en la enseñanza, y cada ejercicio tiene un objetivo específico en el contexto global.

Una aproximación diferente son los *jueces online* o online judges [26]. Como se detallará más adelante, los jueces online son repositorios de problemas de programación que incorporan correctores automáticos que permiten a los usuarios saber de forma inmediata si su resolución del problema ha sido correcta. Los jueces online surgieron a raíz de los *concursos de programación*, con el principal objetivo de almacenar los problemas que habían sido parte de ellos y permitir a los usuarios enviar y corregir sus soluciones. El concurso más antiguo, y quizá más conocido, es el ACM/ICPC, cuya participación crece anualmente y superó, en 2015, los 40.000 estudiantes [27]. Con el tiempo han surgido muchos más, algunos organizados por organizaciones gubernamentales como la International Olympiad in Informatics para estudiantes de secundaria, y otros que surgen por iniciativas de empresas privadas como

las Google Code Jam. Esta proliferación se debe a que los concursos de programación son útiles para promover el interés del estudiante en la programación y potenciar sus habilidades [28], aunque también es un buen mecanismo para identificar a los mejores estudiantes en las fases de contratación.

Aunque en los primeros concursos de programación las soluciones de los participantes eran comprobadas a mano, con el tiempo se desarrolló software de evaluación automática a partir de baterías de tests contra los que se ejecutan las soluciones enviadas (*análisis dinámico*). Esos sistemas resultaron tener interés más allá de los concursos y se terminaron convirtiendo en los ya mencionados *jueces online*.

No ha sido hasta recientemente cuando los jueces online se han empezado a ver como herramientas útiles también para el *aprendizaje curricular*. Debido a la enorme diversidad de los problemas que incluyen, pueden ser utilizados para aprender y practicar virtualmente cualquier técnica algorítmica y de estructuras de datos que se quiera enseñar. Han surgido así con éxito infinidad de iniciativas para integrar el uso de los jueces en clase [29, 30, 31, 32].

Con los jueces online generalistas, sin embargo, el profesor tiene cierta pérdida de control, ya que se trata de sistemas externos en los que la posibilidad de incluir nuevos problemas o de ver el código de los envíos de los alumnos es muchas veces inexistente. Para superar esas dificultades, algunas instituciones y profesores hacen uso de lo que se denominan *Sistemas de Evaluación de Programación* (*Programming Evaluation Systems* (PES) [33]), que no son más que instancias privadas de jueces online en los que los propietarios publican sus propios problemas y registran a sus usuarios. Un problema habitual es que estos sistemas no suelen estar preparados para su integración en los sistemas de aprendizaje virtuales que utilizan de las instituciones correspondientes, o bien no se realiza por restricciones de acceso y seguridad. Aun así, hay algunas integraciones descritas con Moodle, Sakai y otros sistemas de aprendizaje [34, 35, 36].

Todos estos usos despiertan la necesidad de incorporar *feedback* en los jueces, cuya evaluación, como se ha dicho, se basa en el análisis dinámico del código a través de casos de prueba. En contra de lo que ocurría en los sistemas mencionados anteriormente, como los ITS y MOOC, los problemas en los jueces no están creados con un objetivo pedagógico concreto, no existe una solución de referencia con la que comparar aquellas enviadas por los estudiantes y los errores de programación que pueden ocasionar no son fácilmente predecibles, tal y como es recomendable para poder proporcionar feedback en cursos de introducción a la programación [14]. Ni siquiera el espacio de proble-

mas disponibles es comparable. Por ejemplo, el Standford's Machine Learning MOOC que enseña Andrew Ng (uno de los primeros MOOC) incluye 8 conjuntos de ejercicios, y cada uno de ellos tiene varias partes, que combinadas forman un total de 42 problemas de programación [22] creados específicamente para él. Ese valor ni siquiera alcanza al 1 % del número de problemas del UVa Online Judge (juez desarrollado en la Universidad de Valladolid), recopilados a lo largo de más de 20 años.

Jueces Online Existentes

Como se ha mencionado antes, los jueces online son plataformas software para la evaluación de problemas de programación. A pesar de que surgieron inicialmente como una necesidad para el desarrollo de los concursos de programación, hoy en día tienen sentido fuera de ellos, ya sea como almacenes de los problemas de concursos pasados, como plataformas de *aprendizaje competitivo* para practicar o, más recientemente, para el *aprendizaje curricular*. Algunos de los Jueces Online más comunes son *UVa Online Judge*[4], *URI Online Judge*[5], *CodeChef*[6], *CodeFights*[7], *DMOJ*[8] o *SPOJ*[9].

Tomemos, por ejemplo, UVa Online Judge (*UVa OJ*). *UVa OJ*, al igual que el resto de sitios web que funcionan en paralelo al mismo y proporcionados por la comunidad, como uHunt o uDebug, es una plataforma que combina el llamado "servicio de juez online" con otras secciones relevantes para el usuario, como listas de problemas, una página de envíos o un ranking de usuarios. *UVa OJ* permite también navegar por la lista de problemas para encontrar uno específico relacionado con el tema de interés que queremos mejorar o practicar, ya que se pueden encontrar agrupados en temas o volúmenes. Por ejemplo, en la figura 2.1 podemos ver una de las listas de problemas agrupados por temas. La página de envíos es la que utilizan los usuarios cuando ya han programado los ejercicios y enviado una solución al juez y quieren comprobar el veredicto que ha recibido esa solución. Podemos ver un ejemplo de página de envíos en la figura 2.2. Finalmente, después de enviar una solución correcta, es posible acceder a una clasificación que muestra la posición del usuario en relación al resto de usuarios que también tienen aceptado ese problema. Así, se crea un entorno competitivo que, en general, mejora la motivación del usuario y hace que

[4] `https://uva.onlinejudge.org/`
[5] `https://www.urionlinejudge.com.br/`
[6] `https://www.codechef.com/`
[7] `https://codefights.com/`
[8] `https://dmoj.ca/`
[9] `http://www.spoj.com/`

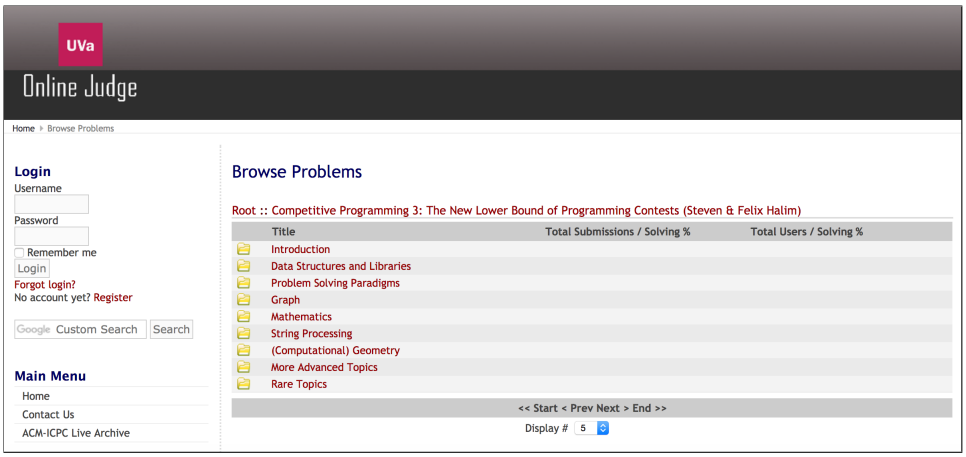


Figura 2.1: Lista de problemas de UVa OJ agrupados por temas

quiera mejorar la calidad y la eficiencia de los códigos enviados.

Por su parte, *URI OJ* [37] también tiene, además de las funciones de *UVa OJ*, un área reservada para profesores y entrenadores que se llama Academic[10]. En ella, el profesor o entrenador puede subir listas de ejercicios con ciertas fechas de entrega y añadir a sus estudiantes separándolos por equipos, clases o temas. Así, puedes seguir el rendimiento de tus alumnos y las soluciones de los problemas que les has mandado. Además, cuenta con la integración de la tecnología de MOSS 2.6.2, que verifica plagios en los envíos que han realizado los usuarios.

Tanto *UVa OJ* como *URI OJ* disponen de un foro asociado a cada problema donde se pueden discutir soluciones diferentes al mismo y errores que se están teniendo en los envíos de forma colaborativa con otros usuarios. Además, ambos están presentes en uDebug, que es una plataforma independiente que permite a los usuarios discutir los problemas con otros usuarios, ver pistas que ha compartido la comunidad, contribuir y votar las mejores y también ver si los resultados que da tu solución son iguales que los de una solución de referencia y, si no lo son, dónde está el error.

Uno de los jueces que se está popularizando en los últimos tiempos es CodeFights, una startup que pretende convertir el aprendizaje de algoritmia y matemáticas en un juego. De este modo, además de los problemas tradicionales, también da la posibilidad de competir con amigos, crear batallas, torneos o incluso enfrentarte a bots de diferentes compañías. Además, en función de las actividades que realicen los usuarios, les da diferentes insignias, que les hacen subir de categoría y escalar posiciones en la tabla de clasificaciones.

[10] `https://www.urionlinejudge.com.br/academic`

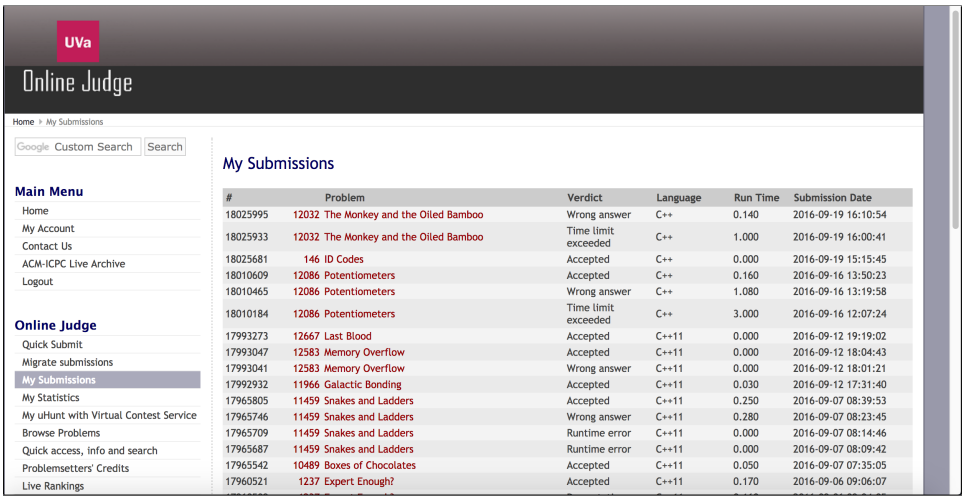


Figura 2.2: Página de envíos de un usuario en UVa OJ

Por último, DMOJ se diferencia de todos los anteriores en que también dispone de soporte específico para problemas interactivos y problemas por puntos.

Especificación de los Problemas en los Jueces Online

En los jueces online existen varios tipos de problemas: por un lado tenemos los problemas que el juez SPOJ denomina "Classic problems", que son los que tradicionalmente están disponibles en la gran mayoría de jueces. Por otro lado, también existen otro tipo de problemas que son conocidos por estar presentes en la Olimpiada Internacional de Informática (IOI[11]): los problemas por puntos (que SPOJ llama "Challenges") y los problemas interactivos.

Nosotros nos centraremos en los problemas clásicos. Veamos a continuación cómo son y qué elementos contienen:

- *Descripción del problema*: normalmente envuelta en una historia, juego o ambientación que hace más motivador el problema, a la vez que esconde lo que se pide para hacerlo más interesante.
- *Descripciones de la entrada y la salida*: especificaciones exactas del formato de los datos que el programa deberá leer de la entrada estándar y que deberá escribir por la salida estándar.
- *Entrada y salida de ejemplo*: ejemplo de una entrada específica y la salida esperada. Sirve principalmente para que el lector confir-

[11] `http://www.ioinformatics.org`

me que ha entendido el enunciado, y, en mucha menor medida, para que confirme que su solución funciona, al menos parcialmente.

Se comprueba que la solución a un problema es correcta mediante los casos de prueba. Cada caso de prueba está diseñado para que el algoritmo se pruebe *una* vez.

Los jueces online comprueban que una solución es correcta utilizando varios pares de ficheros asociados al problema: un fichero de entrada (*.in*) y un fichero de salida (*.out*). Los ficheros de entrada determinan qué va a leer el programa, mientras que los ficheros de salida especifican cuál tiene que ser la salida generada por el programa al recibir esa entrada. Estos ficheros no cambian, sino que son los mismos para cada uno de los usuarios del juez, y las salidas generadas tienen que ser iguales que aquellas que están predefinidas en lo que se considera la solución correcta. Los ficheros de entrada y salida han sido creados previamente por el autor del problema, y normalmente se mantienen secretos.

La existencia de varios pares de ficheros implica que la solución se va a ejecutar varias veces antes de proporcionarle un veredicto final al usuario. Además, para mejorar el rendimiento, la mayoría de ficheros de entrada son *multicaso*, y están compuestos por múltiples casos de prueba. Esto implica que el algoritmo será puesto a prueba varias veces en la misma ejecución.

A modo de ejemplo, mostramos el problema número 114[12] *Acepta el Reto* (*ACR*), un juez online desarrollado por los directores de este trabajo. En la figura 2.3 podemos ver el enunciado del problema.

Tu primo Luis, de 12 años, está aprendiendo a usar la calculadora. Su profesor le ha dicho que calcule el factorial de varios números. Pero, para evitar que le tengan que copiar números muy largos en el cuaderno, les ha pedido únicamente el último dígito, el de más a la derecha.

Recordando que el factorial es la multiplicación de todos los números entre el número y el uno (por ejemplo, el factorial de 8, escrito 8!, es $8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$), demuestra a tu primo Luis que tú eres capaz de hacerlo mucho más rápido que él.

Entrada

El programa recibirá en la primera línea de la entrada el número de casos de prueba. A continuación, cada caso de prueba estará compuesto de una única línea que contendrá un número (positivo).

Salida

Por cada caso de prueba n, se mostrará el último dígito (el de la derecha) de su factorial, $n!$.

Figura 2.3: Enunciado del problema "Último dígito del factorial" de ACR

Como se puede ver en la imagen, esencialmente lo único que hay

[12] `https://www.aceptaelreto.com/problem/statement.php?id=114`

Entrada	Salida	Significado
3		# de casos
2	2	Núm. positivo
3	6	
4	4	

Figura 2.4: Ejemplo de entrada y salida del problema "Último dígito del factorial"

que hacer en el problema es calcular el factorial de los números que te pidan y devolver su último dígito. Podemos ver la entrada y salida de ejemplo en la figura 2.4.

Mostramos ahora un ejemplo que tiene una entrada y una salida algo más complicadas. Se trata del problema número 132 de ACR: "Las cartas del abuelo". En él se pide que en cada caso de prueba se lea una cadena de hasta 1,000,000 letras y se diga si todas las letras que se encuentran entre varias parejas de posiciones a y b de la cadena son la misma. Debido a la potencial gran longitud máxima de la cadena de entrada, las soluciones correctas deben preprocesar la cadena como paso previo para poder responder eficientemente a las consultas de cada pareja (a, b). La figura 2.5 muestra tanto la entrada como la salida.

Las líneas discontinuas se muestran como una indicación de la separación entre los diferentes casos de prueba (*test cases*) que contiene. El último caso de prueba, con el 0, es especial, ya que marca el fin de la entrada, tal y como está descrito en la descripción de la entrada del problema completo.

Tipos de Veredictos

Cuando un usuario sube una solución al juez, las salidas que se generan, como hemos dicho, deben ser exactamente iguales. De ser así, se mostrará el problema como aceptado (*"Accepted"*) en la página de envíos del usuario. Estas comprobaciones no llevan a cabo ningún tipo de análisis de código para decidir si una solución es correcta o no.

No todos los jueces tienen los mismos veredictos, pero normalmente coinciden. La siguiente lista contiene los veredictos más comunes que podemos encontrar en los jueces:

- *Accepted (AC):* Mencionado previamente, indica que el programa ha producido la salida correcta y que ha utilizado tiempo y memoria dentro del límite aceptado.
- *Presentation Error (PE):* Las salidas son correctas, pero no se

Entrada	Salida	Significado
`Heeello`		Cadena del caso de prueba
`4`		# de casos
`0 6`	`NO`	Casos (o queries)
`1 3`	`YES`	
`3 2`	`YES`	
`1 2`	`YES`	
`Bye`		Segundo caso de prueba
`1`		
`1 2`	`NO`	
`Dummy`		Caso de prueba especial (o centinela)
`0`		

Figura 2.5: Entrada, salida y significado de un problema del juez ACR

han presentado de la forma adecuada. Esto puede ocurrir cuando hay errores de separación (espacios o intros) y éstos están repetidos o faltan. El programa no produce exactamente la misma salida que la solución de referencia y, por tanto, es incorrecta.

- *Wrong Answer (WA):* La salida que ha generado el programa al ejecutarse no es correcta.
- *Compile Error (CE):* El juez no ha conseguido compilar el programa.
- *Runtime Error (RTE):* El programa ha fallado en ejecución por motivos como un "segmentation fault", "floating point exception"...
- *Time Limit Exceeded (TLE):* El programa ha permanecido en ejecución durante un tiempo superior al permitido, por lo que no se llega a saber si la solución del usuario es correcta o no.
- *Memory Limit Exceeded (MLE):* El programa ha utilizado más memoria de la permitida.
- *Output Limit Exceeded (OLE):* El programa ha escrito demasiada información en la salida y no coincide con la esperada.

Existen dos cosas relevantes a la vista de la lista de veredictos. La primera es que de cara al usuario principiante, lo que percibe es un resultado prácticamente binario: o está bien o está mal. Un usuario más

experimentado y que suela cometer pocos errores de programación sí podrá interpretar el WA como un error de concepto sobre lo que tiene que hacer y los dos veredictos relacionados con el tiempo y memoria máxima, que pueden indicar que quizá el algoritmo seleccionado para resolver el problema no es correcto.

Podemos diferenciar dos tipos de usuarios en los jueces en línea:

- Usuarios de concursos de programación: aquellos usuarios que conocen los jueces en línea porque los utilizan para la práctica de algoritmia como preparación para concursos de programación.
- Estudiantes: utilizan el juez como preparación curricular y, en principio, la mayoría no tienen ninguna intención de aprendizaje o práctica más allá de eso.

El primer tipo de usuario no suele tener problemas a la hora de entender el funcionamiento del juez en línea. Además, en caso de tener problemas con sus soluciones, recurren al uso de foros o sistemas como el antes mencionado, uDebug. Así, son capaces de avanzar en su camino hacia la resolución de problemas en el juez. En cambio, para los usuarios que no están acostumbrados al uso de los jueces, recibir una y otra vez veredictos negativos les da motivos suficientes para abandonar el juez en línea por completo. Como se mencionó en la introducción, el objetivo de este trabajo es evitar esto.

Autoría de los Problemas

El autor del problema debe, además de redactar el enunciado y suministrar los casos de ejemplo, proporcionar los *.in* y los *.out*. En la mayoría de los jueces en línea en cada uno de los *.in* hay más de un caso de prueba, de forma que una única ejecución de la solución enviada será enfrentada a varios escenarios para ver si contesta bien a todos ellos. Algunos jueces en línea limitan a uno el número de veces que una solución se ejecuta (es el caso de la *UVa OJ*), aunque otros tienen varios ficheros *.in*. En lo sucesivo, asumiremos que ese es el caso. De hecho, daremos por hecho (porque así lo es en el juez analizado) que se realizarán al menos tres ejecuciones de la solución: una con el ejemplo que viene en el enunciado, otro con una ejecución sin casos de prueba (normalmente una entrada vacía en el que la solución debe terminar correctamente sin escribir nada) y una o más ejecuciones con *.in* secretos que ponen a prueba el algoritmo utilizado.

En problemas simples con pocos casos de prueba, el autor puede decidir crear los *.out* manualmente. Esto no es práctico en cuanto el número crece más allá de unas pocas decenas. Debido a ello, lo normal

es que la generación de los *.out* se haga utilizando la solución "oficial" proporcionada por el autor del problema. Esta solución se puede utilizar también para establecer en la plataforma los límites de tiempo y memoria que puede consumir el programa. Es por ello que lo normal es que el autor cree varias soluciones diferentes, programadas en lenguajes distintos y con, quizá, diferentes aproximaciones a la hora de resolverlo. Así, cuando el autor quiere discriminar soluciones ineficientes utilizando estos parámetros, mide los tiempos de sus soluciones y se asegura de que efectivamente con los umbrales seleccionados esas soluciones quedan fuera de las aceptadas. Un ejemplo de ello sería el problema de las cartas del abuelo, que se explicó anteriormente. Las subconsultas que tiene el problema se deben a que el autor quiere comprobar que sean resueltas en $\mathcal{O}(1)$ o en $\mathcal{O}(log(n))$.

Por otro lado, a menudo los ficheros *.in* se crean con programas generadores en lugar de crearlos a mano. Así, se consigue de forma práctica que las pruebas sean exhaustivas y que se cubran completamente todas las posibilidades que hay en los problemas, comprobando perfectamente la validez de las soluciones recibidas.

De nuevo, es importante destacar que estas soluciones no pretenden jugar el rol de soluciones canónicas y de hecho lo normal es que los jueces online ni siquiera tengan acceso a ellas, ya que lo único relevante a la hora de crear el problema son los ficheros *.in* y *.out*, independientemente del origen de los mismos. Los jueces tradicionales no están programados con el objetivo de ser un lugar al que acudir para enseñar y aprender. Es más, los autores de los problemas lo son muchas veces únicamente con el objetivo de realizar concursos de programación, y se limitan a plantear retos para poner a prueba a los participantes en diferentes materias algorítmicas o matemáticas, sin preocuparse en absoluto de que en el futuro alguien pueda querer recibir ayuda sobre sus errores. Es por esto que la incorporación de feedback en los jueces online tendrá que hacerse mediante aproximaciones diferentes a las que se utilizan en ITS, CAI o MOOC, donde el aprendizaje es primordial.

Herramientas de detección de plagio

En el contexto de la enseñanza online en el que nos estamos moviendo es muy común la presencia de *plagio de códigos fuente*. Los estudiantes o usuarios de los sistemas que hemos mencionado en las secciones anteriores se aprovechan de que las evaluaciones que se realizan son automáticas para enviar código que han escrito otras personas. Detectar manualmente el plagio en plataformas que tienen tantos usuarios y tantos envíos es inviable. Para combatir ese problema, surgen las

herramientas de detección de plagio. Es muy importante entender que el hecho de usar una cierta herramienta para detectar plagios no es decisivo para demostrar que existe plagio (o que hay ausencia de plagio). Lo que hacen las herramientas desarrolladas típicamente es devolver una cierta *medida de similitud* para cada par de envíos. Al recibir los resultados, es necesaria la intervención humana para determinar si los valores de similitud recibidos que indican plagio se deben realmente a este motivo o hay otra causa. Es común que las herramientas detecten como similares códigos que en realidad se parecen porque los alumnos parten de un ejemplo que ha contado el profesor o que aparece en el libro de referencia, porque han trabajado juntos y han utilizado la misma idea de resolución o por simple coincidencia.

Para tener una visión completa de las diferentes posibilidades de comparación de soluciones de un problema se han analizado algunas de las herramientas de detección de plagio más conocidas. El objetivo de este análisis era estudiar la dificultad y precisión en la comparación de distintos códigos fuente y ver su utilidad a la hora de ayudarnos a detectar errores de programación comunes. Las herramientas analizadas son las siguientes: JPlag (2.6.1), MOSS (2.6.2), Plaggie (2.6.3) y AC (2.6.4). A continuación se describen todas ellas con más detalle.

JPlag

JPlag [38, 39] es un sistema que busca encontrar similitudes entre varios conjuntos de ficheros con código fuente. De este modo, detecta el plagio de software. JPlag no compara bytes de texto simplemente, sino que también es consciente de la sintaxis del lenguaje de programación y de la estructura del programa, y de ese modo es robusto frente a intentos de disfrazar el plagio. Actualmente, JPlag soporta Java, C#, C, C++, Scheme y texto en lenguaje natural.

Típicamente, JPlag se utiliza para detectar y desalentar la copia indebida de los ejercicios o proyectos de estudiantes en asignaturas de programación. También puede utilizarse para detectar partes robadas de software entre cantidades grandes de código fuente o módulos que han sido duplicados y sólo modificados ligeramente.

Además, JPlag presenta sus resultados como un conjunto de páginas HTML[13]. Estas páginas se envían al cliente y se almacenan de forma local. El distintivo de JPlag es que permite llevar a cabo un clustering de pares de envíos. De este modo, resulta más sencillo ver si un envío se parece a muchos otros envíos, que ayuda a detectar casos en los que

[13]Se puede ver un ejemplo de la interfaz gráfica en la siguiente dirección: `https://jplag.ipd.kit.edu/example/index.html`

el estudiante ha copiado soluciones que otros han presentado en años anteriores.

Moss

Moss [40, 41] (*Measure Of Software Similarity*) es también un sistema para determinar la similitud entre programas. Se desarrolló en 1994 y, hasta la fecha, su aplicación principal ha sido detectar plagio en clases de programación, llevando a cabo su tarea con éxito.

Sin embargo, Moss no tiene modo de saber por qué los códigos son similares. A pesar de que, al igual que con JPlag, un humano tiene que revisar las similitudes que ha destacado el programa, ahorra mucho tiempo de comprobación mostrando qué partes son dignas de revisión exhaustiva.

Moss puede analizar código escrito en los siguientes lenguajes: C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2.

Plaggie

Plaggie [42, 43] es un motor de detección de plagios que se ha creado especialmente para ejercicios en Java. Es similar a JPlag, aunque tiene aspectos que los hacen muy distintos. Plaggie debe instalarse localmente, y su código es abierto y con licencia GNU. El algoritmo básico que se utiliza para comparar dos archivos es el mismo que en JPlag, aunque los autores mencionan que no llevaron a cabo las mismas optimizaciones.

En el caso de Plaggie, los resultados se presentan en texto plano por la salida estándar, y se guardan en formato gráfico en HTML. La salida incluye una tabla que muestra estadísticas de la distribución de los diferentes valores de similitud, el número de ficheros en los envíos, etc.

AC

AC [44] es otra herramienta más de detección de plagios en código que ayuda a detectar envíos similares en grupos de entregas en C, C++ o Java. En texto plano también funciona, aunque es menos preciso. Su autor principal, Manuel Freire, es profesor de la facultad de informática de la Universidad Complutense de Madrid. El código es libre y se encuentra publicado en Github.

Ventajas de AC con respecto a otras herramientas

AC tiene varias ventajas con respecto a las herramientas antes descritas:

- Es local y no requiere el envío de datos a servidores remotos, como es el caso de Moss, por ejemplo.
- Es robusto y utiliza la Distancia Normalizada de Compresión [45] como su medida principal de similitud, pero con la posibilidad de integrar otras. En contraposición, JPlag, Moss y Plaggie tienen análisis pre-establecidos, basados principalmente en realizar una correspondencia de sub-cadenas después de dividir en tokens (componentes léxicos).
- Es bastante potente a nivel visual, ya que proporciona varias maneras de ver los grados de similitud. No proporciona un porcentaje de copia únicamente, sino que dará explicaciones gráficas sobre lo que está pasando para que los profesores o evaluadores puedan construir sus propias explicaciones sobre los resultados. Consultar [46] y [47] para más información y artículos sobre las visualizaciones de AC.

En capítulos posteriores procederemos a analizar por qué las herramientas de detección de plagio no nos resultan útiles para nuestro propósito de incluir pistas o feedback en los jueces online, ya que se han llevado a cabo algunas pruebas de concepto con problemas reales extraídos de un juez sin resultados positivos en este contexto.

Capítulo 3

Feedback en Jueces Online

You're never too old to set another goal
or to dream a new dream.
C. S. Lewis

Resumen: En este capítulo se introduce la idea de que es interesante e imprescindible para el aprendizaje y la motivación incorporar feedback para los usuarios en los jueces online. Además, se presentan algunas de las pruebas realizadas para comparar la utilidad y precisión de los resultados cuando se lleva a cabo una agrupación mediante análisis estático del código (solución que utilizan las herramientas de detección de plagio) o cuando se utilizan clases de equivalencia de la salida (nueva aportación).

Introducción

Debido a la naturaleza de los jueces en línea, que, como hemos mencionado anteriormente, se centran en el desarrollo autónomo y no poseen una figura de profesor que guíe al alumno a medida que avanza en su aprendizaje, creemos que es importante diseñar una manera de proporcionar *feedback* a los alumnos.

Cuando un usuario empieza su camino en un juez en línea, es bastante posible que se dé el hecho de que este usuario abandone por diversos motivos. En primer lugar, a veces no les resulta fácil entender cómo funciona exactamente el juez. En sus primeros problemas enviados, el usuario no es capaz de programar correctamente lo relativo a la entrada y salida del problema, consiguiendo en numerosas ocasiones el veredicto de *Time Limit* por no leer adecuadamente los casos de prueba

y provocar que el programa no termine o el veredicto de *Presentation Error* por no imprimir la salida de forma adecuada, tal y como se pide en el problema.

Posteriormente, cuando el usuario ha superado el primer acercamiento al juez, sigue enfrentándose a veredictos erróneos por desconocimiento del error que le ha llevado a tal situación. Cuando el algoritmo escogido no es el correcto, el único *feedback* que recibe el usuario es un veredicto de *Wrong Answer* (respuesta incorrecta). Ante tal veredicto, el usuario no tiene forma alguna de descubrir qué está pasando, ya que no se le proporciona ninguna otra información al respecto. No sabe en qué caso o casos de prueba ha fallado su solución, y por tanto no puede hacer otra cosa que revisarlo para intentar descubrir su error.

Para solventar esta situación, existen páginas como uDebug [48], compuestas por una comunidad de programadores que se ayudan mutuamente proporcionando sugerencias y soluciones a problemas de varios jueces en línea, dando información de los casos de prueba y compartiendo comentarios al respecto. De este modo, un usuario de uDebug puede seleccionar un problema de uno de los jueces en línea que están soportados y para el cual ha codificado una solución previamente, proporcionar la entrada del problema y obtener la salida "aceptada" del mismo. Así, puede comprobar si la salida correcta coincide con la salida que da su solución para los mismos casos de prueba. Si las salidas coinciden, entonces es probable que la solución sea correcta. En caso contrario, el usuario tiene una indicación de que está fallando en algo y tiene que corregirlo.

En general, para poder ayudar a los usuarios, podemos abordar el problema de dos formas distintas:

- La primera opción es que el usuario haya escogido realizar un problema que está completamente fuera de su alcance, por tener un nivel de programación distinto. Para intentar evitar que los usuarios escojan realizar problemas para los cuales no tienen todavía suficiente conocimiento teórico, algunas plataformas incluyen ciertos sistemas de recomendación, que pretenden guiar al usuario proponiéndole problemas que podría realizar en función de aquellos que ha conseguido solucionar favorablemente. Así, los problemas están categorizados y los usuarios pueden explorar las diferentes categorías que aparecen en la plataforma para escoger el problema que van a realizar en siguiente lugar en función de los problemas previos que han realizado y los conocimientos que ya poseen.

- La segunda forma que hay de ayudar a los usuarios que tienen

errores al enviar soluciones para los problemas es proporcionándoles de alguna manera información adicional a alto nivel sobre los errores que están cometiendo. Esta información a alto nivel se puede proporcionar en forma de pistas.

A pesar de la existencia de páginas como uDebug, existen multitud de jueces en línea que no están soportados en ninguna de ellas. Es por ello que nosotros intentaremos llegar más allá, implementando una solución para el juez en línea *Acepta el Reto*[1], que pretende solventar el problema de desinformación de los jueces en línea y, a su vez, fomentar la participación de los usuarios y potenciar la existencia de una comunidad de usuarios que colaboren entre sí para llegar a la solución correcta de los problemas.

Conviene explicar por qué la solución que proponemos es distinta de todas las soluciones existentes que ya se pueden encontrar en el mundo de los jueces en línea.

Formas Comunes de Proporcionar Feedback

Tradicionalmente, el proceso para proporcionar información sobre los errores cometidos en los problemas de programación parte de la convicción de que existe una solución *canónica* que el estudiante debería alcanzar. Sin embargo, estos programas se basan en realizar análisis estático del código, ya que el objetivo final es que el envío del estudiante se parezca al envío canónico que se ha predefinido.

Como los jueces en línea surgen del mundo de los concursos de programación, que se centran en el aprendizaje de la algoritmia, nuestros objetivos en cuanto a proporcionar pistas son muy distintos de los que tienen los ITS y los MOOC. Como ejemplo, no podemos intentar partir en trozos o encontrar una estructura en el código que envían los estudiantes para darles pistas sobre los errores que han cometido. Las razones son las que siguen:

- Los jueces en línea admiten envíos en múltiples lenguajes de programación. La mayoría de ellos admiten al menos tres lenguajes: *C*, *C++* y *Java*. Además, también es común encontrar jueces que dan soporte a lenguajes muy variados, como *C#*, *Python*, *Haskell* e incluso *bash*. Es por ello que definir diferentes soluciones en función del lenguaje utilizado sería demasiado costoso. Además, no sólo habría que crear una herramienta que hiciese análisis estático de código para todos esos lenguajes, sino que también tendría que

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

soportar el análisis sobre las diferentes librerías de esos lenguajes. Por ejemplo, si tomamos *C++* y *Java* que son dos lenguajes relativamente similares, se pueden encontrar bastantes ejemplos que apoyen nuestra teoría. El ejemplo más claro se da con la lectura y la escritura. Otro ejemplo sería las grandes diferencias que hay en las estructuras de datos complejas o la forma de implementar los algoritmos. Por último, también en el uso de iteradores los códigos resultantes pueden ser muy distintos. Tomando el último ejemplo, por un lado tenemos *Java*, que tiene funciones predefinidas como *hasNext()* para preguntar si un iterador ha llegado al final de la estructura de datos y devolviendo un valor falso en este caso. Por su parte, *C++* no tiene nada de ese estilo, y el programador tiene que comparar los iteradores entre ellos para comprobar si la condición de finalización se cumple. Esta situación tan simple pone de manifiesto que crear una herramienta que sea capaz de *entender* cada pieza del código que se envía al juez es un gran reto, e incluso puede resultar imposible.

- A veces los usuarios intentan mejorar los tiempos que han conseguido en sus envíos previos de un problema y cambian las soluciones para intentar optimizarlas y que lleguen a posiciones altas de los rankings. Para ello, hacen cosas como utilizar manipulaciones bit a bit u optimizar las operaciones de entrada y salida, entre otras. Este tipo de comportamiento dificulta más incluso que el procesamiento de soluciones pueda ser automático.

- Como hemos dicho anteriormente, en los jueces en línea no existen soluciones canónicas a los problemas. En la mayoría de los casos, se requiere aportar una solución de referencia al crear el problema, pero sólo se utiliza para generar los archivos de salida *.out* y no como modelo. La forma en la que está incorporado el feedback en sistemas como los ITS hace que el usuario se vea dirigido hacia una solución específica que el autor ya ha planeado de antemano, y en este caso se puede confundir a usuarios que implementen soluciones diametralmente opuestas a la canónica. Esto no significa que la solución propuesta sea incorrecta, ni mucho menos. Como ejemplo, podemos tomar los diferentes algoritmos de ordenación que existen. En general, los problemas de los jueces en línea van a aceptar cualquiera de los algoritmos de ordenación, e incluso aquellos que están incluidos en alguna de las librerías disponibles en el lenguaje de programación escogido. En este caso, nuestra solución puede detectar sin problemas que las soluciones son las mismas, mientras que los métodos tradicionales de análisis está-

tico las catalogarían como distintas.

Todos los motivos anteriores hacen que dar un feedback personalizado a los usuarios en los jueces en línea sea un problema aún sin resolver. Como mucho, lo que suele ocurrir es que las plataformas existentes tengan mecanismos no estructurados para que los usuarios pueda intentar resolver sus dudas y errores. Hemos identificado varios mecanismos distintos, que listamos a continuación:

- *Foros:* en muchas de las páginas existe un hilo de discusión asociado a cada problema, donde se permite a los usuarios pedir ayuda unos a otros cuando no entienden por qué les está fallando un problema o discutir diferentes alternativas de resolución algorítmica del problema. Algunas de las páginas que incorporan foros de discusión son *UVa Online Judge*, *LightOJ*[2], o CodeForces[3].

- *Ejecución de la solución de referencia:* si el enunciado del problema no es lo suficientemente claro o si la solución del usuario devuelve los resultados correctos para la entrada de prueba pero al realizar el envío definitivo recibe repetidamente el veredicto de "Wrong Answer", es útil tener un servicio que devuelva la salida que proporcionaría una solución aceptada a partir de una cierta entrada que da el usuario. Utilizando este servicio los usuarios podrían incluso crear sus propios casos de prueba y así poder comparar sus resultados con los de la solución aceptada. Sin embargo, esta funcionalidad no está lo suficientemente extendida, y sólo hay unas pocas páginas independientes que proporcionan dicho servicio. uDebug es una de las páginas que permiten hacer esto, en la que las soluciones correctas son enviadas por los propios usuarios.

- *Ayuda general del autor:* existen otras páginas que permiten que los autores aporten información adicional a sus problemas. Así, los usuarios podrán pedir ayuda "bajo demanda" si la necesitan. Es posible que esta ayuda no sea útil para todos los usuarios, ya que no es personalizada y sólo se trata de información que el autor cree que puede resultar interesante. En cualquier caso, no se trata de "feedback" en el mismo sentido que puede tener el feedback en un ITS o un MOOC, ya que no está relacionada particularmente con el error cometido por el usuario. Esta ayuda general tiene el mismo objetivo que las etiquetas o categorías que los autores pueden poner a sus problemas: por ejemplo, se puede

[2] `http://www.lightoj.com`
[3] `http://codeforces.com/`

etiquetar un problema como "Programación dinámica" o "Algoritmo voraz". Una alternativa es ofrecer información más específica del problema, como "¡Cuidado! La entrada también puede contener números negativos.". Sin embargo, a pesar de que es innegable que esta ayuda puede resultar útil, no está enfocada a ayudar al usuario en sus errores cometidos.

- *Acceso a casos de prueba secretos u ocultos:* en la mayoría de los jueces, los casos de prueba son secretos. Sólo existen unos cuantos (como CodeForces) que publican los casos de prueba para que los usuarios puedan identificar dónde fallan sus soluciones. uDebug actúa también como un repositorio de casos de prueba proporcionados por los usuarios. Sin embargo, a pesar de la utilidad que tiene que existan estos repositorios de casos de prueba, en cuanto la entrada y la salida del problema son mínimamente complejas se vuelve muy tedioso comparar los resultados. Se han propuesto algunas soluciones para simplificar este proceso [49], pero ninguna ha sido implementada en un juez hasta ahora.

- *Pistas de la comunidad:* existen casos en los que es posible que los usuarios por sí mismos proporcionen pistas generales sobre los problemas. Dicha información no está dirigida a problemas concretos que tienen los usuarios, sino que también es un feedback general, como en el caso de la ayuda proporcionada por el autor. Sin embargo, puede resultar útil para usuarios que ni siquiera saben cómo o por dónde empezar. uDebug permite a los usuarios añadir este tipo de pistas a los problemas y, por su parte, Leetcode también incorpora la funcionalidad, aunque no con mucho éxito. Por su parte, SPOJ permite añadir "tags" públicos a los problemas[4] y también comentarios cortos, que muchos aprovechan para dar pistas.

- *Pistas genéricas en función del fichero de entrada:* otra forma de ofrecer información al usuario es hacerlo en función del fichero de entrada en el que está fallando. Esta forma de dar pistas fue estudiada en [50], trabajo de fin de máster realizado por Javier Martín durante el curso 2015-2016. La idea de este trabajo era ejecutar los casos de prueba en orden y dar pistas textuales que informen al usuario de dónde se está cometiendo el error. Por ejemplo, el primer paso era ejecutar la solución con el fichero *empty*, que es un fichero de entrada que carece de casos de prueba. Si se produce un error al ejecutar ese fichero, se informará al usuario. A continua-

[4] `http://www.spoj.com/problems/tags`

ción, se hace lo mismo con el fichero de ejemplo que se muestra en el problema. Del mismo modo, si falla al ejecutarse con esa entrada, se informa al usuario. Adicionalmente, también permite mostrar información del resto de ficheros de entrada, troceándolos en casos de prueba individuales para mostrar información relativa a ellos.

Agrupación de Soluciones en Función de la Salida

Las formas de dar feedback que se han estudiado en la sección anterior, a pesar de ser muy útiles, tienen varios problemas asociados que nos hacen intentar llegar más allá y buscar una nueva forma de proporcionar pistas al usuario. Por ejemplo, la solución propuesta por [50] (pistas genéricas en función del fichero de entrada), a pesar de ser útil, es muy costosa. Implementar esa forma de dar pistas implica llevar a cabo muchas ejecuciones distintas para finalmente sólo recibir una pista genérica que no está enfocada al error concreto que está cometiendo el usuario. Eso, a veces, puede no ser suficiente información. Por ello, en este trabajo se busca dar una solución alternativa que permita dar pistas más específicas y más relacionadas con el error que comete el usuario.

Una de las partes básicas del trabajo realizado es la agrupación de soluciones enviadas en función de la salida que generan al ejecutarse. Para poder incluir pistas en el juez, primero tenemos que buscar una forma de agrupar los envíos de los usuarios detectando cuáles tienen los mismos errores. Para ello, agruparemos automáticamente las soluciones partiendo de la siguiente hipótesis:

Si dos soluciones distintas a un problema tienen exactamente la misma salida, entonces las dos soluciones se pueden considerar equivalentes y agruparse bajo la misma clase de equivalencia, independientemente de la similitud del código fuente de ambas.

Cuando dos usuarios programan su solución a un problema, pueden escoger hacerlo en varios lenguajes de programación (los que estén soportados por el juez online en el que estén viendo el ejercicio) o adoptar enfoques completamente distintos para resolverlo. Esto provoca que intentar comparar varias soluciones realizando análisis estático sobre los códigos se torne imposible.

Sin embargo, si en lugar de utilizar el código programado utilizamos la salida que se genera al ejecutar ese código (los ficheros *.out* descritos en el capítulo anterior), podemos decidir si dos soluciones son "iguales" comparando las salidas de cada uno de ellos al ser ejecutados ante varios casos de prueba. Podemos asumir que si las salidas generadas son

distintas, la idea o concepto incorrecto que hay detrás de la solución también va a ser distinta. Al contrario, si varios envíos tienen la misma salida, es muy posible que la idea base de la que parten los usuarios sea la misma (ya sea una idea correcta o con algún error), ya que la gran cantidad de casos de prueba que se lanzan para comprobar si la solución es correcta es muy amplia y deja poco o ningún lugar para coincidencias. Es casi imposible que dos envíos produzcan la misma salida si no son lo que podríamos considerar envíos equivalentes.

Podemos ver en la figura 3.1 la evolución del número de envíos al juez Acepta el Reto a lo largo del tiempo [32]. El juez recibió su primer envío el 17 de febrero de 2014, y desde entonces el número 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.

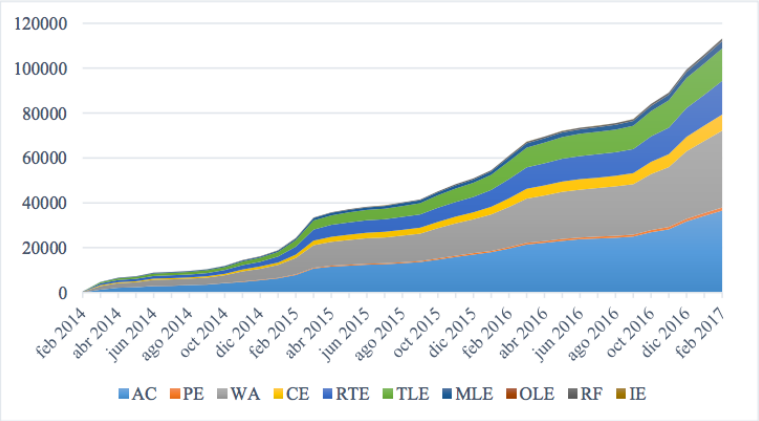


Figura 3.1: Número de envíos y veredictos de ACR

Para que la comparación de salidas sea factible, es necesario idear alguna forma que concluya si dos salidas son iguales evitando comparar todas las salidas completas entre ellas, ya que el número de envíos es elevado y la cantidad de texto en cada una de las soluciones es también arbitrariamente grande. Para agruparlos, cada envío habría que compararlo con todos los demás (o, al menos, con las clases de equivalencia ya conocidas). Para evitar la comparación con grandes cantidades de texto, el sistema genera una función *hash* (con el algoritmo MD5, que produce un valor hash de 128 bits) de forma que se compacta la salida en unos pocos bytes unívocos que harán más sencilla su comparación posterior.

Una vez que dos soluciones tienen el mismo hash a pesar de todo lo anterior, el sistema las tratará como si fueran iguales, ya que suponemos que el error o errores que tienen los usuarios que comparten hash va a

tener la misma causa base. De esta manera, podemos asegurar que si proporcionamos una pista a un usuario para un cierto problema cuya solución genera el mismo hash que las soluciones de otros usuarios, la pista va a ser útil y válida para todos los usuarios que envíen una solución con el mismo hash.

Como ejemplo de las agrupaciones de las salidas en diferentes clases se han estudiado los envíos del problema número 114 de Acepta el Reto, que ya se introdujo en el capítulo anterior. En este problema se pedía que, dado un número, se devuelva el último dígito de su factorial.

Gracias a las agrupaciones que hemos realizado, hemos detectado los errores más comunes del problema (los más repetidos por los usuarios).

A modo de guía, mostramos una de las posibles soluciones aceptadas del problema (en C++):

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

int main() {
  int ncasos;
  cin >> ncasos;
  while (ncasos--) {
    int n;
    cin >> n;
    int ret = 0;
    switch (n) {
     case 0:
     case 1: ret = 1; break;
     case 2: ret = 2; break;
     case 3: ret = 6; break;
     case 4: ret = 4; break;
     default: ;
    }
    cout << ret << '\n';
  }
  return 0;
}
```

Pasamos ahora a analizar los errores más comunes:

- El error cometido por más usuarios es devolver 0 en caso de que la entrada sea 0. Esto, como sabemos, no es correcto, ya que 0! es igual a 1. 108 usuarios distintos han cometido dicho error, al no tener en cuenta que el factorial de 0 es un caso *especial*.

- Otro de los errores de los usuarios es olvidarse de imprimir el intro final después de cada caso de prueba. 15 usuarios distintos han cometido este error.

- También ocurre que en los casos que tienen en cuenta se olvidan de tratar el 0 de forma especial y, por tanto, no imprimen nada para ese caso.

- Por otro lado, también hay usuarios que se olvidan de que *sólo* tienen que mostrar el último dígito del factorial. El código que se muestra a continuación es un envío real de un usuario programado en Java que devuelve 24 como solución cuando la entrada es 4:

```
package factorial;
import java.util.Scanner;
public class Factorial {
    public static void main(String[] args) {
        Scanner leo = new Scanner(System.in);
        int tests = 0;
        int n = 0;
        for (int i = 0; i < tests; i++) {
            n = leo.nextInt();
            switch (n){
                case 0: System.out.println("1"); break;
                case 1: System.out.println("1"); break;
                case 2: System.out.println("2"); break;
                case 3: System.out.println("6"); break;
                case 4: System.out.println("24"); break;
                default:  System.out.println("0");
            }
        }
    }
}
```

Los casos anteriores son errores bastante representativos y comunes para el problema que estamos tratando, y los usuarios que han cometido el mismo error suele ser en general porque tienen algún error de concepto en la teoría o porque se les olvida algo a la hora de llevar a cabo la implementación. Sin embargo, también puede darse que los resultados que devuelve el código del usuario sean iguales por casualidad y no porque realmente hayan fallado en lo mismo (aunque el resultado sea que el fallo es equivalente). Es el caso de los siguientes envíos:

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

int main(){
  int nCasos;
  cin >> nCasos;
  for (int i = 0; i<nCasos; i++){
    int fact;
    cin >> fact;
```

```
    if ((fact == 0) && (fact == 1)) fact = 1;
    else if (fact == 2) fact = 2;
    else if (fact == 3) fact = 6;
    else if (fact == 4) fact = 4;
    else fact = 0;

    cout << fact;
  }
  return 0;
}
```

En el caso anterior vemos que el error del usuario es que imprime como resultado un 1 sí y sólo si fact es a la vez igual a 0 y a 1, lo cual es imposible. Por tanto, de forma efectiva, se va a imprimir un 0 para todas las entradas salvo para 2, 3 y 4. Obviamente, en este caso es una errata, ya que lo que el usuario quería decir es lo siguiente:

```
if ((fact == 0) || (fact == 1)) fact = 1;
```

Por otro lado, tenemos otro usuario que envía una solución cuyo resultado está en el mismo conjunto que la solución que acabamos de ver. En cambio, en este caso es el usuario el que decide que la salida para cualquier entrada que sea distinta de 2, 3 y 4 será 0:

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

int main(){
  int numRep;
  cin >> numRep;
    for (int i = 0; i < numRep; i++){
      int fact;
      cin >> fact;
      if(fact > 4){
        cout << 0 << '\n';
      }else{
        switch(fact){
        case 4: cout << 4 << '\n';break;
        case 3: cout << 6 << '\n'; break;
        case 2: cout << 2 << '\n'; break;
        default: cout << 0 << '\n';
        }
      }
    }
  return 0;
}
```

De este modo, vemos que a pesar de que el problema y la solución proporcionada proceden de errores completamente distintos, nuestra solución consigue agruparlas de forma satisfactoria como iguales, de modo que llegado el momento se podría proporcionar una pista genérica para ambos que les haga ver que el último dígito del factorial para 0 y 1 no tiene como resultado 0.

Un dato a tener en cuenta es que las agrupaciones en función de la salida no son válidas para todos los veredictos de los jueces en línea. Esta agrupación sólo está disponible para los veredictos Accepted, Wrong Answer y Presentation Error. No se han realizado pruebas todavía para el resto de veredictos disponibles, ya que en casos como el Runtime Error o el Memory Limit Error, la ejecución termina de forma abrupta y en la práctica es posible que no se haya escrito toda la salida aún, haciendo más difícil su comparación. Sin embargo, creemos que para empezar utilizar esos tres veredictos es más que suficiente para comprobar que nuestra hipótesis de la agrupación de errores es correcta.

Clases de Equivalencia vs. Análisis Estático

Como se ha comentado anteriormente, las herramientas de plagio utilizan en muchas ocasiones métodos de comparación mediante árboles de sintaxis abstracta (AST), mientras que en los MOOC la forma de agrupar soluciones es realizar "clustering" mediante algoritmos de aprendizaje máquina. Todas ellas se basan en la hipótesis de que si los códigos son parecidos, el feedback que necesitarán será parecido, y fallarán en el mismo sitio. Hemos llevado a cabo pruebas para comprobar que en el contexto de los jueces online esta hipótesis no es válida. Las contamos a continuación.

Pruebas Realizadas

Con el objetivo de comprobar que realmente utilizar herramientas de detección de plagio no nos ayudará especialmente a agrupar los envíos de los usuarios de manera que puedan recibir pistas iguales ante un problema, hemos llevado a cabo varias pruebas con AC (véase capítulo 2, sección 2.6.4). Como ya se contó anteriormente, AC utiliza la distancia de compresión normalizada para medir la similitud entre entregas o envíos.

En la figura 3.2 mostramos el grafo de similitud que nos genera la herramienta. La distancia entre dos códigos se representa con un valor de 0 a 0.5, donde la proximidad al 0 indica que los envíos son muy

parecidos entre ellos, y la proximidad a 0.5 indica que la distancia entre ellos es casi completa. Escogemos de forma arbitraria mostrar 22 aristas distintas, que corresponde a una distancia como máximo de 0.08. Como se puede ver posteriormente en detalle en la tabla 3.1, la mayoría de envíos que se han agrupado en estas circunstancias pertenecen al mismo usuario. La excepción a esto son los envíos que figuran en el grupo 1, que cuentan con envíos del usuario 2903 y del usuario 3708. Además, como puede comprobarse en la figura 3.3, los envíos están correctamente etiquetados en el mismo grupo, ya que la forma de hacerlo es la misma y van a generar exactamente la misma salida.

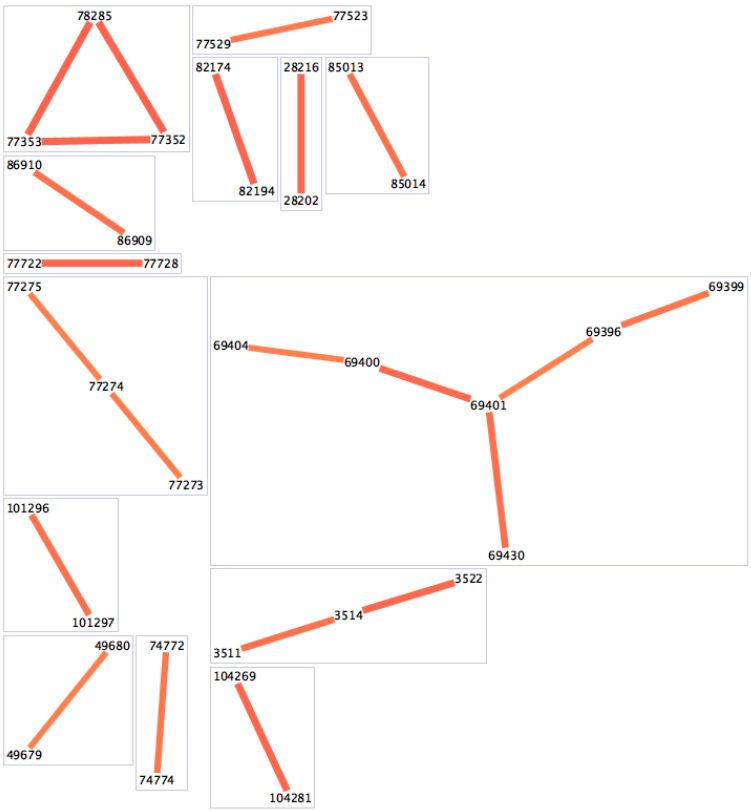


Figura 3.2: Grafo de similitud generado por AC con una distancia máxima de 0.08 y mostrando 22 aristas

En cambio, también se da el caso contrario, como puede demostrarse con los envíos 77722 y 77728. Ambos realizados por el usuario 3753, AC los coloca en el grupo 2. Sin embargo, en la figura 3.4 podemos ver que los códigos de ambos se diferencian solamente en un operador. Como este operador es clave para la resolución del problema, ya que para el envío 77722 la salida va a ser siempre vacía por ser el número de casos

Tabla 3.1: Resultados del grafo de distancias creado por AC, problema 114

Grupo	Número de envío	ID de usuario
1	78285	3708
	77353	2903
	77352	2903
2	86910	1557
	86909	1557
3	77722	3753
	77728	3753
4	77275	3724
	77274	3724
	77273	3724
5	101296	15
	101297	15
6	49679	2373
	49680	2373
7	77529	3743
	77523	3743
8	82174	3942
	82194	3942
9	28216	1261
	28202	1261
10	85013	3746
	85014	3746
11	69404	689
	69400	689
	69401	689
	69430	689
	69396	689
	69399	689
12	3511	273
	3514	273
	3522	273
13	104269	3862
	104281	2862

siempre mayor que 0, está claro que agruparlos no es acertado. Como veremos más adelante, nuestra aproximación sí que va a ser capaz de detectar cuándo dos envíos son sustancialmente distintos a pesar de tener un código prácticamente igual, como es el caso en el ejemplo que acabamos de mostrar.

Código del envío 78285

```
#include <iostream>
using namespace std ;
int main(){
    int num, resultado ,casos;
    cin >> casos ;
    for(int j = 0; j<casos;j++){

        cin >> num;
        switch (num){
            case 1: resultado = 1; break;
            case 2:resultado =2;break;
            case 3:resultado=6;break;
            case 4:resultado = 4;break;
            default: resultado =0;break;

        }
        cout << resultado << endl ;
    }
    return 0 ;
}
```

Código del envío 77353

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

int main() {
    int numCasos, num, fact;
    cin >> numCasos;
    for (int i = 0; i < numCasos; i++) {
        cin >> num;
        switch (num)
        {
        case 1: fact = 1; break;
        case 2: fact = 2; break;
        case 3: fact = 6; break;
        case 4: fact = 4; break;
        default: fact = 0; break;
        }
        cout << fact << endl;
    }
    return 0;
}

```

Figura 3.3: Comparación de los códigos con id 78285 y 88353 del problema 114 de ACR

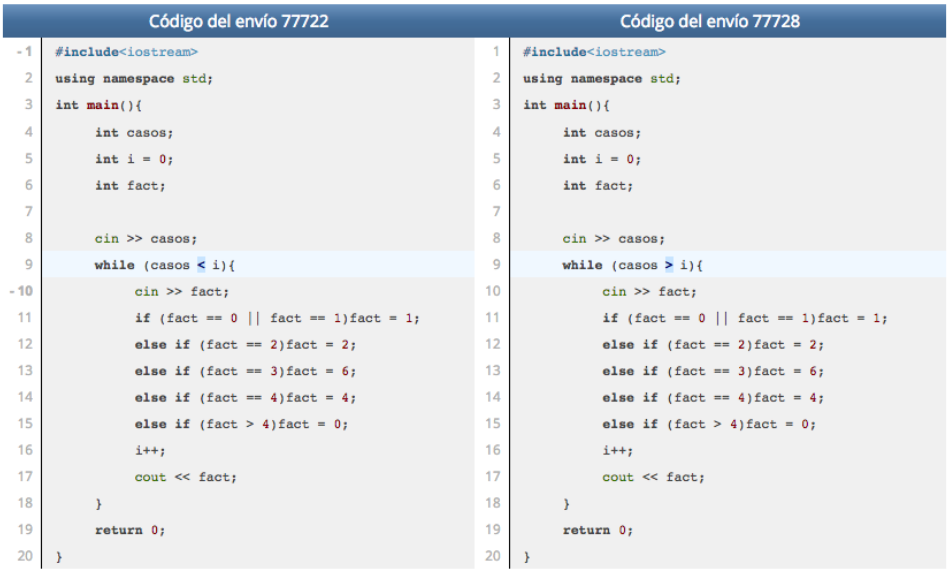
Conclusiones del Análisis

Se han realizado múltiples pruebas adicionales para comprobar que realmente los resultados de similitud que nos proporciona AC no son lo que buscamos. Las causas principales de ello son las siguientes:

- A pesar de que es capaz de tener muchos ciertos positivos (hemos comprobado que dos soluciones que detecta como similares por tener códigos parecidos son realmente equivalentes a nivel de salida) y esos usuarios podrían, por tanto, recibir el mismo feedback, la existencia de múltiples *falsos negativos* hace imposible utilizar estos sistemas en nuestro contexto.

- Cabría la posibilidad de modificar el tipo de feedback que se da para que en lugar de ofrecer pistas a alto nivel (por ejemplo: "cuidado con el factorial de 0") se ofrezcan pistas orientadas al código (por ejemplo: "el if está mal"). Sin embargo, en nuestro contexto no tenemos una solución de referencia con la que comparar el código enviado por los usuarios, por lo que resulta inviable hacerlo de este modo.

- Adicionalmente, la necesidad de revisar los grupos creados para comprobar que tienen sentido y no existen falsos negativos hace



Código del envío 77722

```
#include<iostream>
using namespace std;
int main(){
     int casos;
     int i = 0;
     int fact;

     cin >> casos;
     while (casos < i){
          cin >> fact;
          if (fact == 0 || fact == 1)fact = 1;
          else if (fact == 2)fact = 2;
          else if (fact == 3)fact = 6;
          else if (fact == 4)fact = 4;
          else if (fact > 4)fact = 0;
          i++;
          cout << fact;
     }
     return 0;
}
```

Código del envío 77728

```
#include<iostream>
using namespace std;
int main(){
     int casos;
     int i = 0;
     int fact;

     cin >> casos;
     while (casos > i){
          cin >> fact;
          if (fact == 0 || fact == 1)fact = 1;
          else if (fact == 2)fact = 2;
          else if (fact == 3)fact = 6;
          else if (fact == 4)fact = 4;
          else if (fact > 4)fact = 0;
          i++;
          cout << fact;
     }
     return 0;
}
```

Figura 3.4: Comparación de los códigos con id 77722 y 77728 del problema 114 de ACR

inviable utilizarlo como herramienta de detección de soluciones con errores similares y propensas a recibir el mismo feedback.

- Finalmente, es posible que aumentando mucho el número de aristas y permitiendo que la distancia entre envíos similares sea mayor se creen varios grupos nuevos de envíos parecidos procedentes de usuarios distintos. Sin embargo, la imposibilidad de elegir un umbral de distancia de similitud que realmente indique que dos envíos son equivalentes hace que esta aproximación sea imprecisa y difícil de automatizar.

Teniendo en cuenta que los métodos que siguen las distintas herramientas de detección de plagio para calcular las distancias entre entregas son similares y todos ellos basados en el análisis estático del código, concluimos que es necesario crear una nueva herramienta que utilice la idea de las clases de equivalencia de la salida para agrupar los envíos, lo cual nos permitirá ver los errores a alto nivel que cometen los usuarios y establecer pistas para ellos, casi con certeza de que a dos usuarios que comparten salida en sus soluciones les va a valer la misma pista.

Agrupar los envíos de esta forma no estará exento de inconvenientes. Si un usuario novato utiliza un juez online, puede cometer errores para los que no tendrá sentido añadir pistas, ya que serán únicos o irrelevantes para el resto de usuarios. Por ejemplo, uno de los errores

comunes cuando se empieza a usar un juez es no ser consciente de que no puedes imprimir mensajes auxiliares en la salida, ya que entonces la comparación de la salida será errónea y el problema será marcado con un Wrong Answer. En este caso, es muy improbable que dos usuarios distintos decidan escribir los mismos mensajes auxiliares extra, por lo que en ningún caso acabarán en la misma clase de equivalencia. Sin embargo, para el resto de usuarios, esta forma de proporcionar pistas será muy favorable, ya que no estarán enfocadas a los errores en el código, sino a la idea o concepto que está llevando al usuario a cometer el error. De este modo, serán capaces idealmente de ver qué están haciendo mal, modificarlo y conseguir que el código sea correcto.

En los capítulos siguientes veremos una implementación llevada a cabo especialmente para el juez online Acepta el Reto y que permite visualizar los envíos en forma de grafo, con las soluciones agrupadas tal y como nos será necesario para definir las pistas o "etiquetas de ayuda" para cada uno de los nodos de un cierto problema.

Capítulo 4

Inclusión de Pistas en el Juez Online

Integrity is doing the right thing even when nobody's watching.

Roy Bennett

Resumen: En este capítulo se describen los tres tipos de pistas que consideramos que pueden ser útiles en los jueces en línea, que son lo suficientemente generales como para poder aplicarlas a cualquier juez. Estos tipos de pistas están enfocados a aportar feedback en los problemas de programación tradicionales, con una entrada y una salida predefinidas, y así evitar que los usuarios de los jueces online se desmotiven por la ausencia de información ante sus errores.

Introducción

Como ya se ha mencionado previamente, cuando un usuario envía la solución de un problema a un juez en línea, es común que los veredictos proporcionados por el mismo no den suficiente información. De este modo, el usuario no tiene ningún tipo de ayuda para mejorar su solución y conseguir un veredicto positivo. Cuando envían algo que es incorrecto, tienen muy pocos datos para determinar qué está pasando y qué error tiene su solución.

En el capítulo anterior se ha introducido la idea de agrupar los envíos de los usuarios en función de la salida que generan, de forma que seamos capaces de detectar qué usuarios están cometiendo errores parecidos. Posteriormente, se habló de la posibilidad de introducir pistas

o etiquetas para cada uno de los nodos que tengan envíos erróneos y, de este modo, poder ayudar a los usuarios en su camino al éxito.

Así, consideramos imprescindible que en la propia plataforma de programación se incorpore un sistema de pistas de modo que todos los usuarios puedan beneficiarse y apoyarse entre ellos, aportando motivos extra además de la propia motivación personal para darle continuidad a su aprendizaje. Para ello, hemos definido tres tipos de pistas distintas: pistas a priori (4.2), pistas a posteriori (4.3) y pistas sociales (4.4).

Estas tres formas son lo suficientemente generales como para poder aplicarse a cualquier juez (y, en particular, a Acepta el Reto), ya que están enfocadas a aportar feedback en problemas de programación como los que se han descrito anteriormente. De este modo, siempre que la solución al problema tenga que ser equivalente a una salida predefinida para ser correcta, nosotros seremos capaces de agrupar las soluciones en función de los errores cometidos y de ofrecer información valiosa al usuario para que corrija y vuelva a enviar su solución. Contamos a continuación estas tres formas de añadir pistas.

Pistas a Priori

La forma más directa de añadir pistas a los problemas que están disponibles en un juez es pedirles a los autores que sean ellos los que lleven a cabo esta tarea. Ya que tienen que ser ellos los que especifiquen la entrada y salida del problema que quieren poner, parece razonable pedirles que también realicen una *predicción* de qué errores cometerán los estudiantes al programar una solución, y generar así ayuda textual para ellos desde el principio.

En este caso particular, la tarea no parece ser extremadamente difícil y no requiere demasiado esfuerzo. Cuando los autores quieren crear un problema, normalmente tienen que programar al menos una solución de referencia, ya que tienen que generar la salida que se espera en los problemas. Una forma relativamente fácil que tendrían los autores de generar soluciones incorrectas es llevar a cabo pequeñas modificaciones que hagan que la solución no funcione (no tener en cuenta los casos raros, no imprimir los separadores adecuados, etc.). Además, como los autores son los que ponen los casos raros de un problema, es fácil que los tengan en cuenta desde el principio y añadan una pista para cada uno de ellos. El nivel de detalle que tendrá la pista textual podrá elegirlo el autor, que decidirá cómo de explícita va a ser. Además, el detalle podría depender también de lo "cerca" que esté cada solución de la solución correcta.

Añadir este tipo de pistas requiere ampliar la cantidad de informa-

ción que describe un problema cuando se registra en un juez online. Como se ha dicho ya en la sección 2.5, un problema se especifica con un enunciado y los archivos *.in* y *.out* correctos, que se pueden crear a mano o utilizando una solución de referencia. Sin embargo, añadir pistas a priori requiere que el autor proporcione también la información del hash del nodo asociado al *.out* respecto al que va a poner la pista.

Sin embargo, anticipar los errores de los usuarios resulta una tarea extremadamente complicada, incluso si generar las pistas a priori podría considerarse una tarea sencilla para los autores que conocen bien el problema. Dependiendo de sus habilidades y de su experiencia, serán o no serán capaces de identificar los errores más comunes cometidos por los usuarios en un problema específico. Otro escenario que también es bastante posible es que el autor invierta una cantidad de tiempo elevada en introducir pistas y que finalmente no ayude a una cantidad significativa de estudiantes, lo cual haría que el esfuerzo que le lleva intentar definir cuáles son las soluciones incorrectas más "populares".

Pistas A Posteriori

Como ya hemos mencionado en la sección previa, anticipar todos los errores que pueden cometer los usuarios es un gran reto y no siempre posible. Por ello, proponemos una segunda forma de introducir feedback en el sistema, que se vuelve relevante no antes, sino *después* de haber recibido varios envíos erróneos en un problema. En este modelo, los problemas se registrarían en el sistema sin ningún tipo de pista asociada y, posteriormente, los envíos serán monitorizados y agrupados de acuerdo a la salida que generan.

De este modo, siempre que la cantidad de personas agrupadas en un nodo de error exceda un umbral, se pondrá de manifiesto la necesidad de incluir pistas para él. El autor podría revisar algunas de las soluciones de cada nodo para determinar la mejor pista posible que aplique a todos ellos, y ayudar así al mayor número de usuarios.

Con este modelo, lo que intentamos es reducir el tiempo invertido en crear pistas que, posteriormente, podrían no ser siquiera útiles para los usuarios o serlo sólo para unos cuantos, y el esfuerzo se podrá dedicar sólo a poner pistas en las clases de equivalencia que se haya demostrado empíricamente que reciben más envíos erróneos. Así, el juez online monitorizará los envíos a los problemas para identificar errores repetitivos, y podrá avisar al autor o a los administradores para que actúen en consecuencia si lo consideran apropiado, añadiendo las pistas pertinentes.

Aunque las deficiencias que habíamos encontrado en las pistas a

priori no se ponen de manifiesto en este modelo, surgen otras nuevas. La principal de ellas es el problema del arranque. Añadir pistas depende de lo concurrida que esté una clase de equivalencia. Por tanto, al principio, puede llevar demasiado tiempo que los nodos se pueblen, especialmente en problemas que reciben pocos envíos por ser muy difíciles o porque parecen menos atractivos. Los usuarios que intenten resolver problemas más difíciles, tendrán menos posibilidades de recibir ayuda.

Otro problema que surge es la imposibilidad de determinar un umbral adecuado. Podría elegirse que sea fijo para todos los problemas, que dependa del número de envíos que ha recibido el problema, de la proporción de envíos aceptados frente a envíos erróneos, etc. Sea cual sea la respuesta, una vez se escoge un cierto umbral, tras el cual el administrador o el autor del problema recibe una alerta que le sugiere que añada pistas a un problema, llegados a este punto habrá muchos usuarios que ya han cometido el error. Esto significa que, incluso aunque el autor del problema decida añadir la pista, todos esos usuarios ya "están perdidos", ya que no se les notificará de ninguna forma que un problema que habían intentado ahora dispone de pistas.

Por otro lado, a nivel de implantación real en un juez online, el modelo sufre también deficiencias. Si bien en el primer modelo las pistas eran creadas por el autor en el momento de realizar el problema, ahora esa creación *se retrasa arbitrariamente en el tiempo*. Eso significa que el autor podría verse obligado a buscar la causa de un error repetitivo en su problema mucho tiempo después de haberlo hecho, algo que dependiendo de la naturaleza del problema no necesariamente será fácil. A menudo el autor tendrá la sensación de estar enfrentándose por primera vez al problema, aunque fuera él su creador.

Esto puede llegar al extremo de que el autor, sencillamente, no tenga interés o la posibilidad de suministrar el feedback requerido. Que un problema esté en un juez online no implica que los autores del mismo estén disponibles para enriquecerlo en todo momento. Muchas veces son los propios administradores del juez los que deciden añadir a su página problemas que se han presentado en concursos de programación, piden los distintos casos de prueba a los autores y, tras publicarlo, éstos se desentienden de todo lo que pase con ese problema. En ese sentido, si los usuarios necesitaran pistas, probablemente los administradores no serían capaces de proporcionárselas, ya que no recordarán el problema y lo único que tendrán disponible serán las entradas y salidas del mismo.

Pistas Sociales

Un sistema de pistas en el que sólo se pueden añadir pistas a priori y a posteriori no es sostenible, ya que los usuarios tienen una cantidad infinita de formas de equivocarse, y es muy posible que el interés de los autores por proporcionar ayuda a sus usuarios disminuya con el tiempo. Es por ello que necesitamos idear un mecanismo distinto para añadir pistas en los jueces online.

Nuestro objetivo principal a la hora de proporcionar pistas es poder ayudar al mayor número posible de usuarios en su camino hacia la solución con veredicto AC y prevenir que abandonen un cierto problema y se desmotiven al no ser capaces de resolverlo satisfactoriamente en un periodo razonable de tiempo. Pero, aunque haya usuarios que se comportan así, en general muchos no se rinden al primer envío erróneo que tienen en un problema, sino que siguen intentando resolverlo y a medida que realizan cambios de código para solucionar los errores, la salida generada por su solución va modificándose, haciendo que el usuario pase por distintas clases de equivalencia mientras se acerca (idealmente) a la solución correcta. Intuitivamente, esto lo que genera es un grafo en el cual cada nodo representa una de las clases de equivalencia en las cuales pueden estar los usuarios, y las aristas del grafo son las diferentes transiciones que se han ido produciendo de una clase a otra al modificar sus envíos.

Este grafo proporciona información muy valiosa sobre los usuarios y sus modos de equivocarse, ya que contiene datos sobre el número de usuarios que han caído en un error en concreto, por cuántas formas de equivocarse han pasado antes de abandonar el problema o conseguir un AC e incluso deja intuir los diferentes caminos de error que existen. Conocer todos estos datos es muy útil, ya que nos permite introducir un nuevo sistema de pistas centrado en los propios usuarios.

Este tipo de pistas pretende aprovecharse del hecho de que cuando un usuario consigue una solución correcta, sabe mejor que nadie cómo funciona el problema: incluso posiblemente mejor que el autor, que puede haber escrito el problema varios años antes. En particular, utilizaremos algo parecido al *feedback por pares* [51] de una forma algo especial. Una gran parte de los usuarios que consiguen un AC lo consigue arreglando un error en el código, que podrían tener muchos más usuarios en el futuro. En particular, cuando un usuario alcance el AC después de visitar uno o varios nodos de error en el grafo, el sistema podría pedirle que escriba una pista o una pequeña explicación relacionada con el error o errores que había cometido.

Las pistas sociales pueden contribuir muy favorablemente al desa-

rrollo del sistema de pistas. Como los otros dos tipos de pistas que hemos mencionado previamente no tienen potencial de cubrir cada caso por sí mismos, consideramos que permitir a los usuarios crear y añadir sus propias pistas es una característica muy valiosa. Además, de esta forma evitaremos el problema de tener que rastrear quién era el autor original del problema para pedirle que envíe pistas para el mismo y nos aprovecharemos de que cuando el usuario consiga una solución correcta, sabrá (en general) qué errores había cometido en sus envíos anteriores. De este modo, también mantendremos el interés del usuario por resolver el problema y con la ayuda de las pistas que han puesto otros usuarios podrá llegar (idealmente) a la solución correcta.

Capítulo 5

Visualización del Grafo de Envíos

Never believe anyone who tells you that
you don't deserve what you want.

Taylor Swift

Resumen:

En este capítulo se muestra una implementación llevada a cabo para el juez Acepta el Reto, que permite a los autores o administradores de un problema visualizar un grafo de envíos que les facilitará la tarea de añadir pistas al mismo. Adicionalmente, se verá cómo sería este grafo visto desde la perspectiva del usuario. Para concluir, también se muestra el proceso de comparación de soluciones, que resultará útil a la hora de añadir pistas.

Introducción

Para apoyar las ideas que hemos planteado previamente sobre pistas, se ha llevado a cabo una implementación para el juez Acepta el Reto que permite a los usuarios y los autores del problema ver el camino que siguen al resolverlo. De este modo, lo que hacemos es agrupar las soluciones de los usuarios en función de la salida generada al ejecutar el programa y el veredicto recibido, y mostramos todos los envíos en forma de grafo.

Este modo de visualización va a resultar muy útil para los administradores o autores del problema, ya que en lugar de tener sólo la opción

de ver un envío simultáneamente, podrán ver el grafo completo de soluciones, donde cada nodo tendrá envíos de múltiples usuarios siempre y cuando la salida generada fuera la misma.

Visualización del Grafo: Autor

A continuación enseñaremos ejemplos reales sacados de Acepta el Reto, donde mostraremos el grafo que vería el autor o administrador. En la visualización se pueden ver nodos de tres colores diferentes, que corresponden a tres veredictos distintos del juez:

- *Accepted:* se muestra de color *verde*.
- *Wrong Answer:* como es natural, se muestra de color *rojo*.
- *Presentation Error:* aparece de color *azul*.

Además, las aristas del grafo indican qué orden han seguido los envíos del usuario, de modo que si existe una arista del nodo 1 al nodo 2, esto significa que en primer lugar algún usuario ha realizado un envío cuya salida corresponde a la salida del nodo 1, y posteriormente ha realizado un envío que ha ido a parar al nodo 2, por tener también la misma salida. Además, puede darse la situación de que las aristas sean bidireccionales, lo cual implica que algún usuario ha ido del nodo 1 al nodo 2 y al contrario, que el mismo usuario u otro diferente tiene envíos que han ido del nodo 2 al nodo 1.

Por otro lado, también veremos que existen nodos reflexivos. Esta situación la representaremos con un borde de color negro alrededor de cada uno de los nodos que sean reflexivos.

En la figura 5.1 se muestra el grafo que vería el autor del problema 114 ("Último dígito del factorial"). El primer problema que nos surge es la gran cantidad de envíos y de usuarios que tienen la mayoría de los problemas. Esto hace que sea difícil sacar conclusiones de lo que vemos debido a la alta cantidad de nodos del grafo. Por ello, para el ejemplo de la figura hemos decidido mostrar sólo los nodos del grafo cuyo número de envíos sea mayor o igual que 10. Esto número, algo arbitrario, se ha escogido al detectar que la existencia de nodos en los que prácticamente no hay envíos se debe a errores aislados que en realidad al autor no le interesa analizar.

Además, también podemos ver en la figura 5.2 que, como habíamos predicho, la cantidad de envíos por nodo que ve el autor del problema es muy alta.

Por otro lado, en la misma figura 5.2, en la parte derecha del grafo, el autor podrá ver una tabla con la lista de envíos. Así, podrá seleccionar

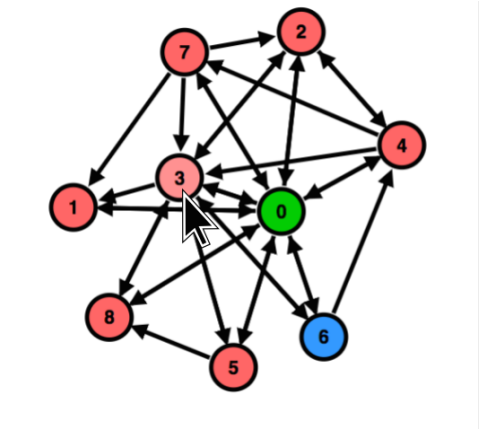


Figura 5.1: Grafo de envíos del problema número 114 de *ACR*

cualquiera de estos envíos para ver el código de los mismos. A modo de ejemplo, mostramos en la figura 5.3 lo que vería el autor al escoger el envío número 77130.

De este modo, el autor del problema podrá ver de forma rápida los códigos que envían los usuarios al problema. Bajo la hipótesis de que si varios usuarios están agrupados en el mismo nodo debe ser porque habrán cometido todos el mismo error, podrán comprobar que realmente esto es así y averiguar la causa del fallo de manera fácil y rápida, pudiendo tomar medidas (añadir pistas principalmente) de ser necesario.

Para el autor del problema, es muy interesante ser capaz de ver qué caminos siguen sus usuarios, en qué fallan y cómo van modificando sus envíos para llegar a la solución correcta. Esto les puede resultar útil a la hora de redactar problemas nuevos, para editar el problema si el enunciado tiene algún error y, en especial, para *añadir pistas a posteriori* al problema. Si lo que quieren es añadir pistas que ayuden al usuario a llegar a una solución, será completamente irrelevante analizar soluciones erróneas que han sido enviadas por un número muy reducido de usuarios, ya que lo que les interesa es llegar al mayor número posible de usuarios con sus pistas. Si añaden pistas para problemas existentes en un nodo en el que caen muchos usuarios, es bastante posible que los nuevos usuarios que lleguen al problema puedan encontrar útiles las pistas añadidas, ya que la probabilidad de que cometan los mismos errores es bastante más alta que la probabilidad de que cometan errores nuevos (aunque ambas sean perfectamente posibles). Además, incluso en caso de cometer errores nuevos, no será primordial para el autor del problema resolverlos o ayudarles a sortearlos, al ser errores aislados y que en principio otros usuarios no cometerán (al menos inicialmente, porque luego podrían convertirse en errores comunes también).

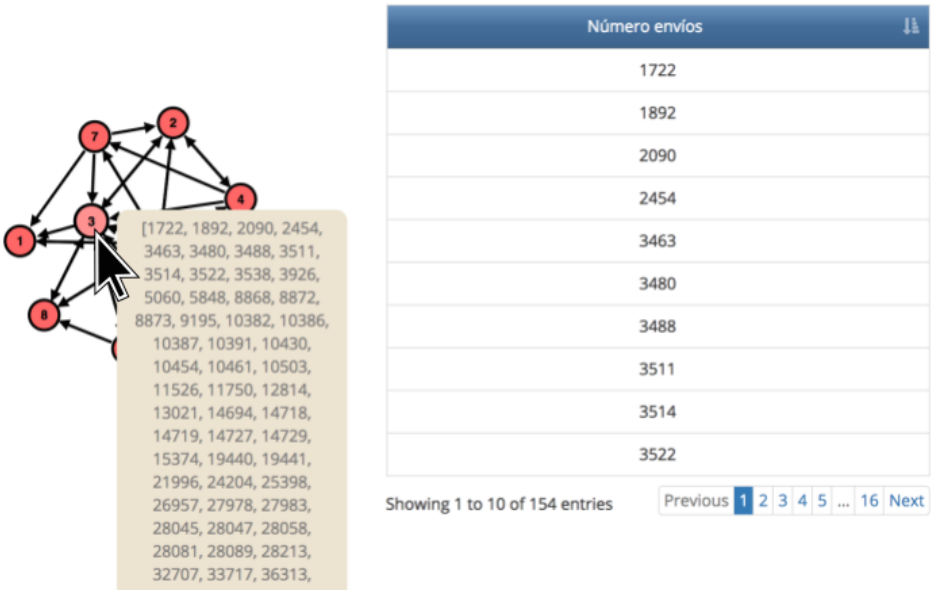


Figura 5.2: Tooltip con parte del listado de ids de los envíos al problema 114 de ACR, que emerge al pasar el puntero del ratón por encima del nodo 3

En la sección siguiente se enseñará todo esto desde la perspectiva del usuario, y se mostrará también una característica de comparación de envíos que se ha añadido con el objetivo de facilitar la tarea tanto a autores como a usuarios. Aunque se describe y se muestran ejemplos de ella en la sección del usuario, esta característica está disponible también para los autores del problema.

Visualización del Grafo: Usuario

La visualización de envíos desde la perspectiva del usuario no es sino una particularización de lo que hemos contado en el apartado anterior, que no sólo resulta útil al autor o administrador de los problemas, sino que también será útil para los usuarios.

Cuando un usuario realiza varios envíos erróneos a un mismo problema, puede ocurrir que *se pierda* en sus propios envíos y no entienda qué está haciendo mal. Comenzará a hacer cambios que a veces resultarán *un poco aleatorios* para intentar corregir sus errores, pero muchas veces ni tan siquiera estará realizando cambios sustanciales en el código, llegando incluso al punto de que la salida generada siga siendo la misma.

Visualizar el grafo de sus envíos al problema puede facilitarle esta tarea. Podrá ver si las salidas están cambiando (ya que de ser así los

Código del envío 77130

```
#include <iostream>
using namespace std;
int main()
{
  int repeticion, numero;
  cin >> repeticion;
   for(int i=0;i<repeticion;i++)
   {
      cin >> numero;
      switch(numero)
      {
      case 1: cout << '1' << '\\n'; break;
      case 2: cout << '2' << '\\n'; break;
      case 3: cout << '6' << '\\n'; break;
      case 4: cout << '4' << '\\n'; break;
      default: cout << '0' << '\\n';
      }
   }
   return 0;
  }

```

Figura 5.3: Código del envío número 77130 en *ACR*

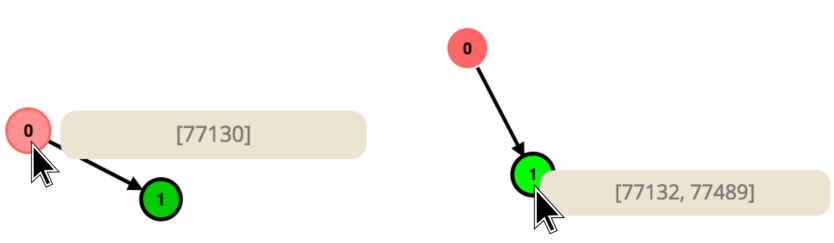
envíos estarán situados en nodos distintos del grafo) y también comparar los envíos entre ellos para recordar qué cambios había introducido en cada uno de los intentos.

En este capítulo mostraremos, en primer lugar, un ejemplo de visualización del grafo por parte del usuario. Después, veremos el proceso de comparación de envíos, que permitirá al usuario o al autor tener una visión más clara sobre lo que está ocurriendo en el problema.

Tomaremos como ejemplo el problema 114 de *ACR*. En este caso, lo que vamos a mostrar es el grafo que ve uno de los usuarios al escoger ver el grafo de envíos del problema 114. Este grafo, como puede verse en 5.4, es bastante simple y sólo muestra 3 envíos diferentes repartidos en 2 nodos. Sin embargo, lo hemos escogido porque nos muestra de forma sencilla y clara cómo es la visualización típica que vería un usuario en problemas para los que tenga envíos erróneos y envíos correctos.

En la figura 5.4 se muestra el grafo mencionado tal y como lo vería el usuario, que dispone de un envío erróneo y dos envíos correctos. Además, se puede ver que las soluciones del usuario han seguido un camino del nodo 0 al nodo 1. El nodo número 1 es, además, reflexivo. Esto se debe a que después del primer envío aceptado, el usuario, por algún motivo, ha vuelto a enviar una solución correcta al mismo.

Los tooltip que podemos ver en la misma figura 5.4 muestran los números identificadores de envío de las soluciones correspondientes a ese nodo.



(a) Envío erróneo del problema (b) Envíos correctos del problema

Figura 5.4: Envíos del problema 114 para el usuario escogido

A continuación, en la sección 5.4 veremos la comparación del código enviado por el usuario para este problema.

Comparación de Soluciones

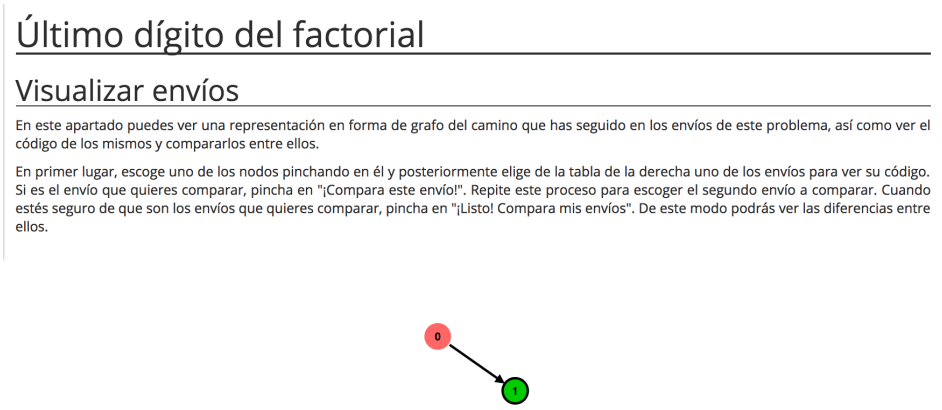
Una vez que ya hemos añadido soporte en Acepta el Reto para la visualización de envíos y la posibilidad de ver el código siguiendo el grafo de soluciones, creemos que resultará útil ir más allá y permitir tanto a los usuarios como a los autores de problemas realizar una comparación de los envíos a nivel de código. Esto puede resultar útil por varios motivos:

- Para el usuario: al poder visualizar el grafo de envíos, errores y aciertos, el usuario de la plataforma podrá hacerse una idea de cómo ha ido mejorando su solución y, llegado el caso, podrá analizar fácilmente cuál era su error viendo las soluciones (erróneas) anteriores.

- Para el autor: podrá ver de forma fácil qué envíos tienen los mismos errores y comparar los códigos de varios usuarios entre ellos para ver si el motivo por el que fallan es el mismo y, si fuera necesario, añadir pistas al problema para prevenir esos errores.

- Para el autor: si le resulta confuso ver una gran cantidad de envíos en un nodo y no sabe de dónde viene el problema, podrá comparar envíos de nodos contiguos o de un nodo con envíos erróneos y el nodo con envíos correctos para intentar ver de forma más rápida cuál ha sido el arreglo que se ha llevado a cabo para llegar a la solución correcta.

Mostraremos ahora cómo se realizaría la comparación de envíos en la sección de visualización implementada para *ACR*. En el siguiente



(a) Problema escogido y grafo asociado

(b) Activada previsualización del envío 77130

Figura 5.5: Comparación de envíos del problema 114 para el usuario escogido (parte 1)

ejemplo veremos la comparación de dos envíos del mismo usuario para uno de los problemas tal y como lo vería él.

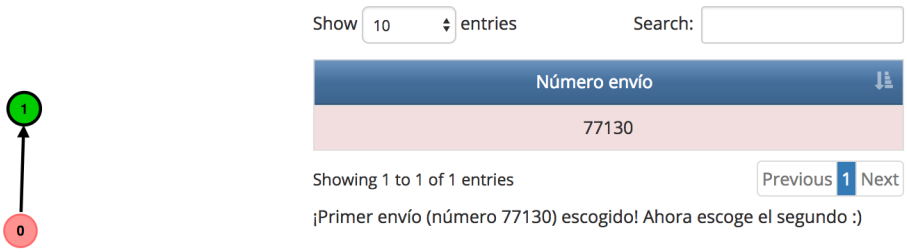
- Seleccionamos el problema número 114 de la lista de problemas.
- Una vez elegido el problema aparecerá el grafo de envíos del mismo (ver figura 5.5a), de donde se podrá elegir un nodo.
- Después de elegir un nodo aparecerá una tabla con todos los envíos correspondientes al nodo escogido. De ahí, el usuario tendrá que escoger uno de los envíos y pinchar en el botón "Compara envío X" (figura 5.5b). Adicionalmente y como hemos visto previamente, también aparecerá el código del envío en la parte inferior del grafo y de la tabla.
- Posteriormente, se le informará de que el envío ha sido escogido, el color de la fila de la tabla correspondiente a ese envío se pondrá

en rojo para indicar que ese envío ya ha sido seleccionado y se le pedirá escoger el segundo envío (ver figura 5.6a).

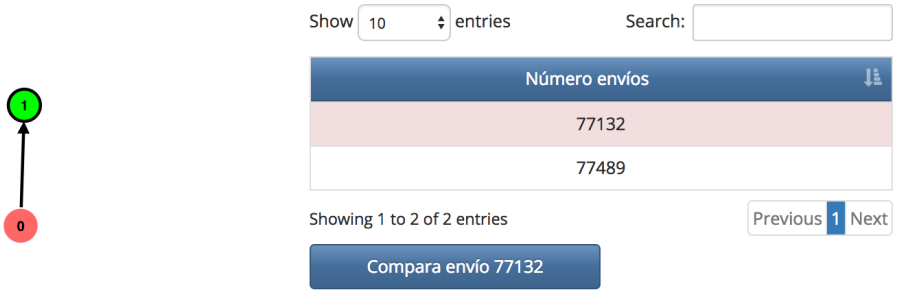
- Después de seleccionar el segundo envío a comparar (figura 5.6b), el usuario deberá pinchar en el botón que dice "¡Listo! Compara mis envíos". Esto provocará que debajo del grafo y de la tabla, en lugar de mostrar el código de uno de los envíos, se muestre la comparación de ambos envíos (ver figura 5.7).

Hasta ahora, en los jueces online no se ha pensado en el usuario hasta el punto de darle pistas sobre errores concretos. Mediante la comparación de las soluciones enviadas, se proporciona una herramienta tanto al usuario como al autor para solventar este problema y promover un mejor aprovechamiento de la plataforma, dando lugar a un intercambio de información valiosa entre usuarios o entre el autor y los usuarios que permitirá que *no se den por vencidos* si la resolución de un problema se les complica.

La idea es que, una vez que tanto usuario como autor puedan visualizar el grafo de envíos y comparar soluciones, hagan uso de esta característica para añadir pistas al problema. Como se ha mencionado en el capítulo anterior, tiene sentido añadir a la plataforma pistas a posteriori (autor) y pistas sociales (usuario). Así, no sería muy difícil implementar un apartado extra en Acepta el Reto que permita a usuarios (o autores) escoger el nodo sobre el que quieren poner una pista después de ver que han cometido un error que podría extenderse a más usuarios. De este modo, se les podría pedir que añadan una ayuda textual que explique a alto nivel cuál es el fallo que están cometiendo, lo que permitirá al resto de usuarios que compartan el fallo tener esa ayuda disponible y poder arreglar su solución de acuerdo a la información recibida.



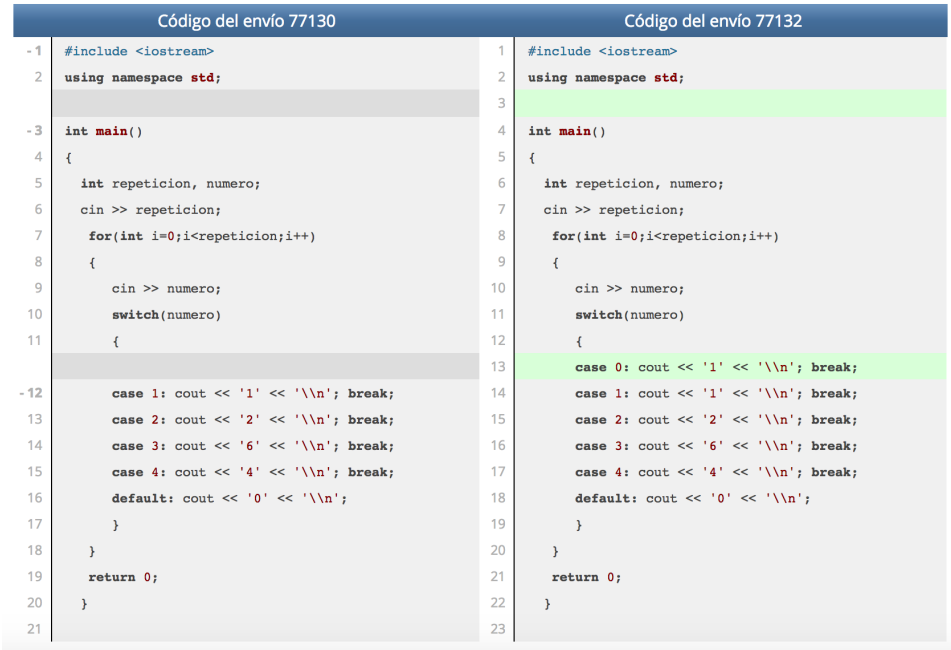
(a) Envío 77130 escogido



(b) Activada previsualización del envío 77132

(c) Envíos listos para la comparación

Figura 5.6: Comparación de envíos del problema 114 para el usuario escogido (parte 2)




```
-1  #include <iostream>
 2  using namespace std;
-3  int main()
 4  {
 5    int repeticion, numero;
 6    cin >> repeticion;
 7     for(int i=0;i<repeticion;i++)
 8     {
 9        cin >> numero;
10        switch(numero)
11        {
-12       case 1: cout << '1' << '\\n'; break;
13        case 2: cout << '2' << '\\n'; break;
14        case 3: cout << '6' << '\\n'; break;
15        case 4: cout << '4' << '\\n'; break;
16        default: cout << '0' << '\\n';
17        }
18     }
19     return 0;
20   }
21
```

```
 1  #include <iostream>
 2  using namespace std;
 3
 4  int main()
 5  {
 6    int repeticion, numero;
 7    cin >> repeticion;
 8     for(int i=0;i<repeticion;i++)
 9     {
10        cin >> numero;
11        switch(numero)
12        {
13        case 0: cout << '1' << '\\n'; break;
14        case 1: cout << '1' << '\\n'; break;
15        case 2: cout << '2' << '\\n'; break;
16        case 3: cout << '6' << '\\n'; break;
17        case 4: cout << '4' << '\\n'; break;
18        default: cout << '0' << '\\n';
19        }
20     }
21     return 0;
22   }
23
```

Figura 5.7: Comparación de envíos del problema 114 para el usuario escogido (parte 3)

Capítulo 6

Implementación

Success is the best revenge for anything.
Ed Sheeran

Resumen: Contaremos en este capítulo cómo se ha llevado a cabo la implementación de las características que se han añadido a Acepta el Reto y, adicionalmente, también se hablará de la arquitectura general del juez.

Introducción

En este capítulo se cuenta cómo se ha llevado a cabo la implementación de las diferentes características que se han añadido a Acepta el Reto. Hablaremos inicialmente de la arquitectura general de ACR (6.2), después hablaremos sobre la visualización de grafos (6.3) y finalmente hablaremos sobre la implementación de comparación de códigos (6.4).

Implementación General

La arquitectura de ¡Acepta el Reto! [32], como puede verse en la figura 6.1, 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

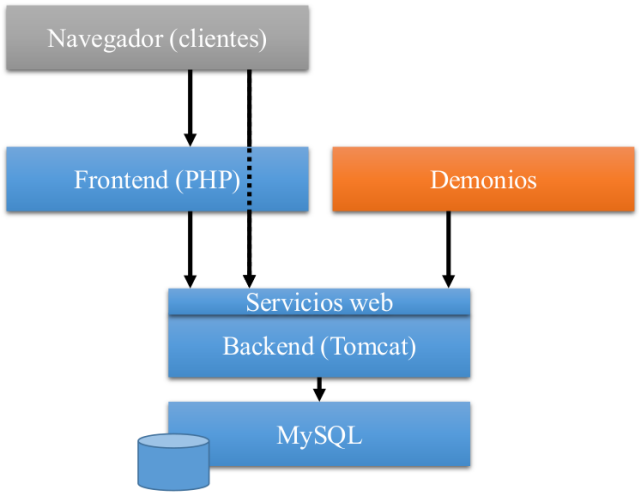


Figura 6.1: Arquitectura general de ¡Acepta el Reto!

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 este trabajo, la implementación no se ha integrado todavía con la versión en producción de ACR. En cambio, lo que se ha hecho ha sido replicar la estructura general de la página y su aspecto, para que así el diseño se asemeje lo más posible al del juez existente y en el futuro se pueda integrar la funcionalidad nueva de forma relativamente sencilla.

Por un lado, se ha utilizado Bootstrap [52] para el diseño del frontend. Por otro, se usan Bootstrap Tables [53] y jQuery [54] para las tablas nuevas. Además, los lenguajes utilizados en la implementación

han sido Javascript, HTML, CSS y Java. Finalmente, se ha hecho uso del Spring MVC Framework [55] para facilitar la comunicación entre los diferentes componentes de la web.

Visualización de grafos

La implementación de la visualización de grafos se ha hecho utilizando como base la librería D3 [56, 57]. A continuación se explica qué es D3 exactamente (6.3.1) y se cuentan las modificaciones y añadidos que se han llevado a cabo para adaptarlo a nuestro problema (6.3.2).

¿Qué es D3?

D3 es una librería en Javascript para visualizar datos con HTML, SVG y CSS. D3 permite enlazar datos arbitrarios a un DOM (Modelo de Objetos del Documento) y después aplicar transformaciones basadas en los datos al documento. Por ejemplo, D3 permite utilizar la misma matriz de números para hacer cosas como generar una tabla HTML o crear un gráfico de barras SVG interactivo con transiciones e interacciones.

Además, D3 resuelve la manipulación de documentos basados en datos de forma eficiente. De este modo, ofrece una flexibilidad extraordinaria, exponiendo las capacidades completas de estándares web como HTML, SVG y CSS. Tiene una sobrecarga mínima, y por tanto D3 es extremadamente rápido y soporta conjuntos de datos muy grandes, interacción y animación. Además, el estilo funcional de D3 permite la reutilización de código a través de una diversa colección de módulos oficiales que han sido desarrollados por la comunidad.

En nuestro caso en particular, utilizamos D3 para generar el grafo interactivo de envíos de un problema. En 6.3.2 contaremos más en detalle cosas sobre la implementación de grafos que se ha realizado.

Modificaciones y Añadidos

La implementación de grafos que se ha desarrollado parte del código que se encuentra en [58]. Sin embargo, se han llevado a cabo numerosas modificaciones, que se listan a continuación:

- Se ha eliminado la posibilidad de interaccionar con el teclado para provocar cambios de estado o modificaciones en el grafo. En la versión de Ross Kirsling, era posible cambiar la reflexibilidad del grafo simplemente presionando la tecla 'R', o borrar el nodo o las aristas seleccionadas con el 'Delete'. Por otro lado, 'L(eft)',

'R(ight)' y 'B(oth)' cambiaban la dirección de los enlaces entre nodos.

- Se ha quitado la posibilidad de crear nuevos nodos simplemente pinchando con el botón izquierdo del ratón en una zona vacía.
- Ya no se permite añadir nuevas aristas al mantener el ratón pulsado y arrastrar de un nodo a otro.
- Se ha añadido un tooltip que muestra información relativa al nodo cuando estamos realizando un 'mouseover' (pasando el ratón por encima del nodo sin pinchar en él). Además, el nodo aumenta su tamaño cuando estás sobre él, a modo de feedback al usuario.
- Se ha modificado la forma de colorear los nodos para que el color del nodo dependa del resultado del envío en lugar de ser aleatorio.
- Se ha añadido código para trabajar y mostrar la tabla con todos los envíos una vez que pinchas en uno de los nodos.

Comparación de soluciones

Para mostrar y llevar a cabo la comparación de los diferentes códigos también se ha partido de una librería llamada "Pretty Diff" [59], que se encuentra disponible en GitHub con licencia CC0 (Dominio Público) y cuyo funcionamiento puede comprobarse en [60]. En particular, se han extraído los archivos que se encuentran en la carpeta *lib* y los de la carpeta *css*, aunque estos últimos han sido modificados posteriormente para adaptar el formato de las comparaciones al estilo de Bootstrap que tiene el resto de la página.

Pretty Diff hereda su formato de salida de la aplicación jsdifflib [61]. La salida es una matriz grande que contiene matrices secundarias en cada uno de sus índices. Cada una de las matrices secundarias tiene 5 posiciones, donde se almacena la siguiente información: tipo, comienzo del código en la primera muestra, final en la primera muestra, comienzo del código en la segunda muestra y final en la segunda muestra. Los tipos son 4: "equal", "replace", "insert" y "delete". El tipo "equal" significa que ambos códigos son idénticos en las dos muestras. El tipo "replace" indica que ha habido cambios en la muestra de un código al otro. El tipo "insert" quiere decir que el código asociado es único en la segunda muestra. Finalmente, el tipo "delete" indica que el código es único en la primera muestra. Para cada uno de los tipos que acabamos de describir, el CSS asociado varía. De este modo, el usuario podrá ver de un sólo

vistazo qué ha cambiado en su código. Podemos ver un ejemplo de esto en la figura 6.2.

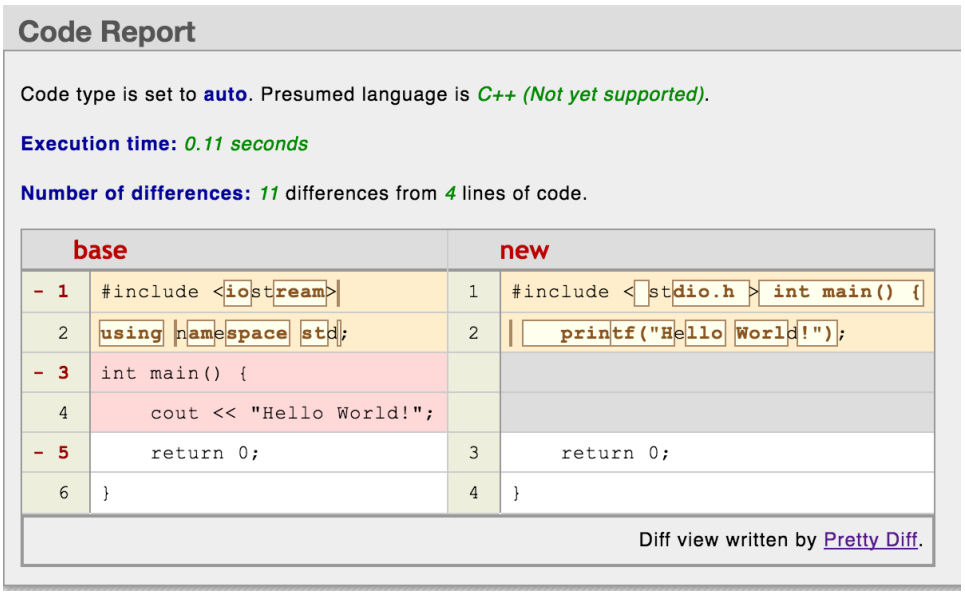


Figura 6.2: Ejemplo de comparación con la herramienta Pretty Diff

Por otro lado, el algoritmo utilizado en Pretty Diff para la comparación de envíos parte del algoritmo de Paul Heckel que está descrito en [62]. Dicho algoritmo tiene una complejidad lineal tanto en tiempo como en espacio para todos los casos. Este algoritmo tiene varios pasos, que están basados en las siguientes observaciones:

- Si una línea aparece sólo una vez en cada archivo, entonces debe ser la misma línea, aunque puede haber sido movida. Utilizamos esta observación para localizar líneas inalteradas que posteriormente excluiremos de cualquier tratamiento posterior.

- Si se ha encontrado que una línea no se ha alterado y las líneas inmediatamente adyacentes a ella en ambos archivos son idénticas, estas líneas deben ser la misma línea. Esta información se puede utilizar para encontrar bloques de líneas sin cambios.

A partir de estas observaciones, el algoritmo de Paul Hecker realiza varias pasadas diferentes para realizar la comparación entre dos envíos. El que nosotros utilizamos en este trabajo es el algoritmo que puede verse en [63], que propone una pequeña mejora al algoritmo propuesto por Paul Hecker.

Este algoritmo es como sigue:

- En primer lugar, se almacena cada una de las cadenas de entrada en un array. Para ello, lo hacemos de la forma más sencilla posible y colocamos cada línea de entrada en un índice del array correspondiente para examinarlas.
- Posteriormente, un bucle recorrerá las dos muestras y rellenará un hashmap, utilizando como claves cada una de las líneas de código y como valor un par que representa el número de veces que hemos encontrado dicha cadena en cada una de las dos muestras.
- Una vez que la tabla contiene información sobre ambas muestras, necesitamos un último bucle que tome ciertas decisiones al examinar los datos. Para ello, realiza las siguientes comprobaciones:

    1. Intenta descubrir igualdad. Todo lo demás es menos importante, por lo que sucede más adelante en el árbol de decisión.
    2. Identifica los elementos que sólo aparecen una vez en cada matriz. Si el siguiente elemento que aparece en el array no es igual pero sólo está una vez, entonces sabemos que el cambio es único.
    3. Después se intenta descubrir si el elemento actual corresponde a una eliminación. Si el elemento actual (y posiblemente los elementos posteriores a ese) están presentes en la primera muestra pero no en la segunda, entonces se trata de una supresión.
    4. Este paso es exactamente igual que el paso 3 pero a la inversa, para descubrir las inserciones.
    5. En este punto se intenta determinar si se ha llevado a cabo una supresión de un número fijo de elementos (supresión estática).
    6. Igual que el punto 5, pero ahora se pretende encontrar las inserciones de un número fijo de elementos (inserciones estáticas).
    7. Por último, se determina si los elementos actuales de cada una de las muestras corresponden a cambios no únicos.

Llegados a este punto, se tiene toda la información necesaria relativa a la comparación del código de los dos envíos.

Capítulo 7

Conclusiones y Trabajo Futuro

El destino suele estar a la vuelta de la esquina, pero lo que no hace es visitas a domicilio. Hay que ir a por él.

Carlos Ruiz Zafón

Resumen: En este último capítulo se concluye con los aspectos más importantes de los que se ha hablado previamente y se realiza un estudio de los posibles trabajos que se pueden llevar a cabo partiendo de la investigación realizada en este trabajo.

Conclusiones

En este trabajo se han tratado varios temas relacionados con la incorporación de feedback en jueces en línea. Como se ha comentado anteriormente, proveer al usuario de un sistema para que pueda ser capaz de detectar el motivo de sus errores de programación es de gran importancia para que éste no abandone, dejando su aprendizaje a medias.

En particular, se ha desarrollado una implementación para el juez Acepta el Reto que permite agrupar y visualizar las soluciones de los problemas en forma de grafo, así como una opción de comparación de las mismas. Esto permite al usuario tener una idea muy clara de cómo se ha ido moviendo por el espacio de soluciones, de ver qué cosas ha ido modificando para acercarse a la solución correcta (si es que ha llegado a ella) y de poder tener claro cuáles son los errores que ha cometido.

Por otro lado, se han propuesto tres formas distintas de añadir pistas en los jueces en línea: a priori, a posteriori (o bajo demanda) y

sociales. Cada uno de esos tres mecanismos tiene sus ventajas y sus inconvenientes. Sin embargo, los tres tipos son muy útiles, ya que cumplen su cometido de ampliar el número de pistas disponibles. Esto las hace complementarse y aumentar las oportunidades de que un usuario que en el futuro tenga algún error pueda recibir feedback personalizado.

De las tres formas propuestas, las más interesantes son las pistas a posteriori y las pistas sociales. Las pistas a posteriori surgen de forma natural, ya que al ser capaces de agrupar a todos los usuarios en función de sus errores le estamos dando al autor información privilegiada para que diseñe pistas que beneficien a esos usuarios. Finalmente, creemos que las pistas sociales son las que tienen más posibilidad de mejora, ya que incorporan un *aspecto social* al juez que hasta ahora no se había considerado y que resulta potencialmente positivo.

Trabajo Futuro

El sistema de pistas propuesto se basa en las clases de equivalencia para la salida, como ya hemos visto. Este sistema tiene una granularidad muy gruesa. Dos soluciones que se diferencien en un único caso de prueba son igual de diferentes desde el punto de vista de nuestro sistema que dos soluciones que no coinciden en ninguno. Esto implica que la reutilización de pistas [21] de una clase de equivalencia a otra es imposible. Esto podría mejorarse si separáramos las pruebas e hiciéramos un análisis más fino de los casos de prueba individuales, para ver dónde coinciden o se diferencian las soluciones. Por ejemplo, en el problema del último dígito del factorial que ha sido expuesto previamente en la memoria, existe un nodo de error donde la gente falla en el caso del 0 y en el caso del 4 (escriben 24 en lugar de escribir sólo el último dígito, que es el 4). Estos dos errores son diferentes, y pueden darse también por separado. Es decir, podría haber un nodo de error donde la gente falle en el caso del 0, un nodo donde fallen en el caso del 4 y un nodo donde fallen en ambos casos. Al no detectar esto, no podemos reutilizar pistas, por lo que nos veríamos obligados a explicar el mismo error de forma independiente varias veces.

Un punto claro de mejora sería analizar qué opciones tendríamos de proponer pistas para veredictos que no fueran *WA* y *PE*. Podríamos asumir que utilizar clases de equivalencia de salida podría aplicarse también cuando los veredictos fueran *RTE* o *MLE*, porque el programa tiene que haber escrito las soluciones de todos los casos de prueba que ya se han ejecutado de todos modos. Sin embargo, esto en la práctica no es tan fácil, ya que el hecho de haber acabado de forma abrupta la ejecución implica que posiblemente no se haya escrito toda la salida.

Esto se debe a la existencia de buffers intermedios que utilizan las librerías y el propio sistema operativo. Por ello, incorporar pistas para estos veredictos probablemente necesite una aproximación completamente distinta a la seguida hasta el momento.

Algo que hasta ahora no hemos estudiado es cómo actúa nuestra solución frente a problemas en los que se admiten múltiples soluciones. Aunque creemos que el sistema se va a comportar de la misma forma, ya que nuestra forma de dar pistas está enfocada a los envíos erróneos y no a los correctos, habría que comprobar que eso no empeora la fiabilidad de nuestro sistema.

Finalmente, añadir las pistas sociales convertiría al juez en algo más que un juez, incorporando el concepto de plataforma social y añadiendo ventajas que van mucho más allá del sistema de pistas, ya que la meritocracia está muy extendida en estos contextos de jueces en línea y programación. Prueba de ello son la existencia de plataformas como *uDebug*, los sistemas de logros en *URI OJ* o incluso, en otros contextos, las medallas y el sistema de puntuaciones existente en *Stack Overflow*. Estos mecanismos han resultado ser muy útiles para mantener a los usuarios comprometidos en todas estas plataformas, y pueden también ser útiles aplicándolo a nuestro sistema de pistas. Gracias a incentivos como insignias y clasificaciones, los usuarios serían más propensos a revisar viejos errores, documentarlos y ayudar a otros con él. Todo esto abre una línea completamente nueva de desarrollo muy interesante de estudiar y desarrollar en los jueces en línea.

Apéndice A

Introduction

It's the questions we can't answer that teach us the most. They teach us how to think. If you give a man an answer, all he gains is a little fact. But give him a question and he'll look for his own answers.

Patrick Rothfuss, The Wise Man's Fear

Background and motivation of this project

The idea of this project arises from the need to incorporate new ways of giving feedback to users in online judges. Online judges are systems that arose initially with the aim of acting as repositories of problems that had been part of programming contests. However, over time, its use has been extended to classrooms and begun to be used for curricular purposes.

In contrast to online education systems, judges do not have specific tools to help users solve problems correctly. This fact often causes these users to lose patience when trying to solve a problem, receiving negative verdicts on the solutions sent and without any way of knowing what they are doing wrong.

Objectives and work plan

According to the motivation section, the main objective of this work is to study different ways of including online hints to help users by providing them with information about their errors. After that, a tool will be developed that will group users based on the errors they made and

allow both problem authors and users who solve them to add or see the hints for that particular error. This way, we can improve the learning process and at the same time increase their knowledge of algorithms and programming.

The work plan goes as follows:

- *Previous research:* we will study the different education systems that are available, the differences between online judges and the possibility of using plagiarism detection tools to group users based on the programming errors they make when solving a problem. On the other hand, we will analyze what forms of feedback are present in online judges today.

- *Development of the project:* we will explore the best way to group users based on their mistakes and what innovative ways to include hints can be incorporated to online judges. In addition, we will consider what functionality must be added to a judge to include such systems to offer hints. Later, this new functionality will be implemented for the online judge "Acepta el Reto".

- *Future work:* it will conclude with the most relevant aspects of the project and will analyze which parts still have room for improvement.

Contents of this work

Chapter 2: *Education systems.* It shows the main eduaction systems, an introduction to online judges and talks about plagiarism detection tools.

Chapter 3: *Feedback in Online Judges.* It introduces the advantages of adding feedback in judges as to motivation and improved learning and shows traditional ways of giving feedback.

Chapter 4: *Inclusion of hints in the Online Judge.* Describes the three ways of including hints that we consider useful and general enough to be incorporated in any online judge.

Chapter 5: *Visualization of the Error Graph.* It shows the implementation carried out for the judge "Acepta el Reto", which among other things allows problem authors and users to see a representation of the complete graph of solutions sent grouped by their output and that will allow to incorporate feedback for the users depending on the mistakes they have made.

Chapter 6: *Implementation.* It exposes the tools and libraries that have been used during the implementation of this work.

Chapter 7: *Conclusions and Future Work.* We conclude with the most important aspects mentioned and studies the possible extensions of this work.

Apéndice B

Conclusions and Future Work

El destino suele estar a la vuelta de la esquina, pero lo que no hace es visitas a domicilio. Hay que ir a por él.

Carlos Ruiz Zafón

Resumen:

This last chapter concludes with the most important aspects exposed previously and a study of the possible work that could be carried out using the research carried out in this master's thesis.

Conclusions

In this work, we addressed several issues related to the incorporation of feedback in online judges. As discussed earlier, providing the user with a system so that he can be able to detect the reason behind his programming errors is of great important to help him not to leave the online judge and stopping his learning mid-way.

In particular, we developed an implementation for the judge "Acepta el Reto", which allows grouping and visualizing the solutions of the problems as a graph, as well as an option of comparing them. This allows the user to have a very clear idea of how he moved through the solution space, to see what he changed to get closer to the correct solution and to find out what mistakes he made.

On the other hand, we proposed three different ways of adding hints to online judges: a priori, on demand and social (or community) hints. Each of these three mechanisms has its advantages and disadvantages. However, all three types are very useful, since they fulfill their mission

of expanding the number of available hints. They are complementary and increase the chances of a user receiving personalized feedback in the future.

Out of the three proposed forms, the most interesting ones are on demand and social hints. On demand hints appear naturally, since being able to group all users based on their errors we are providing the author with priviliged information to design hints that benefit those users. Finally, we believe that social hints are the ones that have the best chance of improvement, since they incorporate a social aspect to the judge that until now had not been considered and that is potentially something very positive.

Future Work

The proposed hint system is based on the equivalence output classes, as we have already seen. This system has a very thick granularity. Two solutions that differ in a single test case are just as different from the point of view of our system that two solutions that do not match any. This implies that reusing hints [21] from one equivalence class to another one is impossible. This could be improved by separating tests and doing a finer analysis of the individual test cases, to see where the solutions actually match or differ. For example, in the problem of the last digit of the factorial that was exposed previously, there is an error node where people fail in the inputs 0 and 4 (they write 24 instead of writing only the last digit, which is 4). These two errors are different, and can also occur separately. That is, there could be (and there will be, most likely) an error node where people fail for the input 0, and also a different node where they fail for the input 4. If we don't detect this, we cannot reuse hints, so we would be forced to explain the same error twice.

A clear point of improvement would be analyzing what options of proposing hints we have for verdicts different from WA and PE. We could assume that using output equivalence classes could also apply when the verdicts are RTE or MLE, since the program has to write solutions for every test case that was already run anyway. However, this is not so easy in the practice, because the execution ended in an abrupt way, and that means that some intermediate buffers exist, used by libraries and the operating system on its own. For that reason, incorporating hints for those verdicts probably needs a completely different approximation from the one used until now.

Something that we have not studied so far is how our solution works for problems where multiple solutions are allowed. Although we believe

that the system is going to behave in the same way, since our way of giving hints is focused on wrong submissions and not the correct ones, it should be verified that this does not make worse the reliability of our system.

Finally, adding social hints would make the judge more than just a judge, incorporating the concept of social platform and adding advantages that go well beyond the system of hints, since meritocracy is very widespread in these contexts of online judges and programming. Proof of this is the existence of platforms like *uDebug*, achivements in *URI OJ* and even in other contexts, such as medals and scores in *Stack Overflow*. These mechanisms have proven to be very useful in keeping users engaged on all these platforms, and can also be useful by applying it to our hint system. Thanks to incentives like badges and ratings, users would be more likely to go back and check old mistakes, add documentation for them and help others with it. All this opens up a whole new line of development for online judges that would be interesting to investigate.

Bibliografía

Y así, del mucho leer y del poco dormir,
se le secó el celebro de manera que vino
a perder el juicio.

Miguel de Cervantes Saavedra

[1] Pew Research Center, "Technology device ownership: 2015." [Online]. Available: http://www.pewinternet.org/2015/10/29/technology-device-ownership-2015/

[2] D. N. Arnold, "Computer-Aided Instruction."

[3] M. Ward, "A template for call programs for endangered languages," 2007.

[4] Wikieducator, "Computer Assisted Instruction (CAI)." [Online]. Available: http://wikieducator.org/Computer_Assisted_Instruction_(CAI)

[5] Wikipedia, "Scorm." [Online]. Available: https://es.wikipedia.org/wiki/SCORM

[6] J. R. Anderson and E. Skwarecki, "The automated tutoring of introductory computer programming," *Commun. ACM*, vol. 29, no. 9, pp. 842–849, Sep. 1986. [Online]. Available: http://doi.acm.org/10.1145/6592.6593

[7] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, "On automated grading of programming assignments in an academic institution," *Computers & Education*, vol. 41, no. 2, pp. 121 – 131, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131503000307

[8] F. L. Wang and T.-L. Wong, *Designing Programming Exercises with Computer Assisted Instruction.* Springer Berlin Heidelberg, 2008.

[9] M. Choy, U. Nazir, C. K. Poon, and Y. T. Yu, *Experiences in Using an Automated System for Improving Students'Learning of Computer Programming.* Springer Berlin Heidelberg, 2005.

[10] Wikipedia, "Intelligent tutoring system." [Online]. Available: https://en.wikipedia.org/wiki/Intelligent_tutoring_system

[11] H. S. Nwana, "Intelligent tutoring systems: an overview," *Artificial Intelligence Review*, vol. 4, no. 4, pp. 251–277, Dec 1990. [Online]. Available: https://doi.org/10.1007/BF00168958

[12] S. Chakraborty, D. Roy, P. K. Bhowmick, and A. Basu, "An authoring system for developing intelligent tutoring system," in *Students'Technology Symposium (TechSym).*

[13] T. Murray, "Authoring Intelligent Tutoring Systems: An analysis of the state of the art," *International Journal of Artificial Intelligence in Education (IJAIED)*, vol. 10, pp. 98–129, 1999, part II of the Special Issue on Authoring Systems for Intelligent Tutoring Systems (editors: Tom Murray and Stephen Blessing).

[14] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13. New York, NY, USA: ACM, 2013, pp. 15–26.

[15] J. R. Anderson and B. J. Reiser, "The LISP Tutor: It approaches the effectiveness of a human tutor," *BYTE*, vol. 10, no. 4, pp. 159–175, Apr. 1985.

[16] S. S. Abu Naser, "Developing an intelligent tutoring system for students learning to program in C++," *Information Technology Journal*, vol. 7, no. 7, 2008.

[17] A. Naser and S. Samy, "Evaluating the effectiveness of the CPP-Tutor, an intelligent tutoring system for students learning to program in C++," *Journal of Applied Sciences Research*, vol. 5, no. 1, pp. 109–114, 2009.

[18] E. R. Sykes and F. Franet, "A prototype for an intelligent tutoring system for students learning to program in Java," *Advanced Technology for Learning*, vol. 1, no. 1, 2004.

[19] Wikipedia, "Massive open online course." [Online]. Available: https://en.wikipedia.org/wiki/Massive_open_online_course

[20] L. Steels, *Music Learning with Massive Open Online Courses (MOOCs)*. IOS Press, 2015, vol. 6.

[21] C. Piech, J. Huang, A. Nguyen, M. Phulsuksombati, M. Sahami, and L. J. Guibas, "Learning program embeddings to propagate feedback on student code," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 1093–1102.

[22] J. Huang, C. Piech, A. Nguyen, and L. J. Guibas, "Syntactic and functional variability of a million code submissions in a machine learning MOOC," in *Proceedings of the Workshops at the 16th International Conference on Artificial Intelligence in Education AIED 2013*, 2013.

[23] B. P. III, A. Hicks, and T. Barnes, "Generating hints for programming problems using intermediate output," in *Proceedings of the 7th International Conference on Educational Data Mining (EDM 2014)*, 2014, pp. 92–98.

[24] M. Eagle, M. Johnson, and T. Barnes, "Interaction networks: Generating high level hints based on network community clustering," in *Proceedings of the 5th International Conference on Educational Data Mining*, 2012.

[25] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI Commun.*, vol. 7, no. 1, pp. 39–59, Mar. 1994.

[26] A. Kurnia, A. Lim, and B. Cheang, "Online judge," *Comput. Educ.*, vol. 36, no. 4, pp. 299–315, May 2001.

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

[28] F. Almeida, J. Cuenca, R. F. Pascual, D. Giménez, and J. A. P. Benito, "The Spanish parallel programming contests and its use as an educational resource," in *26th IEEE International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPS 2012*. IEEE Computer Society, 2012, pp. 1303–1306.

[29] G. P. Wang, S. Y. Chen, X. Yang, and R. Feng, "OJPOT: online judge & practice oriented teaching idea in programming courses,"

European Journal of Engineering Education, vol. 41, no. 3, pp. 304–319, 2016.

[30] Y. Luo, X. Wang, and Z. Zhang, "Programming grid: A computer-aided education system for programming courses based on online judge," in *Proceedings of the 1st ACM Summit on Computing Education in China on First ACM Summit on Computing Education in China*, ser. SCE '08. New York, NY, USA: ACM, 2008, pp. 10:1–10:4.

[31] A. Kosowski, M. Małafiejski, and T. Noiński, "Application of an online judge & contester system in academic tuition," in *Proceedings of the 6th International Conference on Advances in Web Based Learning*, ser. ICWL'07. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 343–354.

[32] P. P. Gómez-Martín and M. A. Gómez-Martín, "¡Acepta el reto!: juez online para docencia en español," in *Actas de las XXIII JENUI*, 2017, pp. 77–84. [Online]. Available: http://jenui2017.unex.es/actas_jenui2017.pdf

[33] R. Queirós and J. P. Leal, "Programming exercises evaluation systems - an interoperability survey." in *CSEDU (1)*, M. Helfert, M. J. Martins, and J. Cordeiro, Eds. SciTePress, 2012, pp. 83–90.

[34] J. P. de Castro, M. A. Pérez, L. M. Regueras, and M. J. Verdú, Eds., *Edujudge System Handbook: How to Organize Programming Competitions In Moodle Courses*. Sello Editorial, 2010.

[35] H. Suleman, "Automatic marking with Sakai," in *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*, ser. SAICSIT'08. New York, NY, USA: ACM, 2008, pp. 229–236.

[36] I. Boticki, I. Budiscak, and N. Hoic-Bozic, "Module for online assessment in ahyco learning management system." *Novi Sad Journal of Mathematics*, vol. 38, no. 2, pp. 123–139, 2008.

[37] J. L. Bez, N. A. Tonin, and P. R. Rodegheri, "URI Online Judge Academic: A tool for algorithms and programming classes," *2014 9th International Conference on Computer Science & Education*.

[38] Karlsruhe Institute of Technology , " JPlag source code on Github ." [Online]. Available: https://github.com/jplag/jplag

[39] Karlsruhe Institute of Technology, "JPlag, detecting software plagiarism." [Online]. Available: https://jplag.ipd.kit.edu/

[40] Alex Aiken , "MOSS, Measure of Software Similarity." [Online]. Available: https://theory.stanford.edu/~aiken/moss/

[41] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: Local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. ACM, 2003, pp. 76–85. [Online]. Available: http://doi.acm.org/10.1145/872757.872770

[42] A. Ahtiainen, S. Surakka, and M. Rahikainen, "Plaggie: GNU-licensed Source Code Plagiarism Detection Engine for Java Exercises," in *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, ser. Baltic Sea '06. ACM, 2006, pp. 141–142. [Online]. Available: http://doi.acm.org/10.1145/1315803.1315831

[43] A. Ahtiainen, S. Surakka, and M. Rahikainen, "Plaggie." [Online]. Available: https://www.cs.hut.fi/Software/Plaggie/

[44] Manuel Freire Morán , "AC source code on github." [Online]. Available: https://github.com/manuel-freire/ac2

[45] Wikipedia , "Normalized compression distance." [Online]. Available: https://en.wikipedia.org/wiki/Normalized_compression_distance

[46] M. Freire, "Visualizing Program Similarity in the AC Plagiarism Detection System," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ser. AVI '08. ACM, 2008, pp. 404–407. [Online]. Available: http://doi.acm.org/10.1145/1385569.1385644

[47] M. Freire and A. Sopan, "Gene similarity uncovers mutation path vast 2010 mini challenge 3 award: Innovative tool adaptation," in *2010 IEEE Symposium on Visual Analytics Science and Technology*, Oct 2010, pp. 287–288.

[48] V. Shah, "uDebug 2.0: Connecting the competitive programming community," in *Competitive Learning Institute Symposium*, 2016. [Online]. Available: https://ciiwiki.ecs.baylor.edu/index.php/Main_Page

[49] A. Mani, D. Venkataramani, J. Petit, and S. Roura, "Better feedback for educational online judges," in *Proceedings of the 6th In-*

ternational Conference on Computer Supported Education, 2014, pp. 176–183.

[50] J. Martín Moreno-Manzanaro, "Pistas pedagógicas en juez online de programación." [Online]. Available: http://eprints.ucm.es/38654/

[51] P. A. Ertmer, J. C. Richardson, B. Belland, D. Camin, P. Connolly, G. Coulthard, K. Lei, and C. Mong, "Using peer feedback to enhance the quality of student online postings: An exploratory study." *J. Computer-Mediated Communication*, vol. 12, no. 2, pp. 412–433, 2007.

[52] M. Otto and J. Thornton, "Bootstrap." [Online]. Available: http://getbootstrap.com/

[53] W3 Schools , "Bootstrap tables." [Online]. Available: https://www.w3schools.com/bootstrap/bootstrap_tables.asp

[54] jQuery Foundation, " jQuery webpage ." [Online]. Available: https://jquery.com/

[55] Spring Docs, "Spring MVC Framework." [Online]. Available: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

[56] Bostock, Mike and Davies, Jason and Heer, Jeffrey and Ogievetsky, Vadim and Community , " D3, a Javascript library." [Online]. Available: https://d3js.org/

[57] Github , "D3, source code." [Online]. Available: https://github.com/d3/d3

[58] Ross Kirsling , "Directed Graph Editor." [Online]. Available: http://bl.ocks.org/rkirsling/5001347

[59] A. Cheney and Others, "Pretty diff, source code." [Online]. Available: https://github.com/prettydiff/prettydiff

[60] A. Cheney, "Pretty diff, webpage." [Online]. Available: http://prettydiff.com/

[61] C. Emerick, "JSDiffLib, Source Code." [Online]. Available: https://github.com/cemerick/jsdifflib

[62] P. Heckel, "Isolating differences between files." [Online]. Available: https://gist.github.com/ndarville/3166060

[63] Austin Cheney, "Diff algorithm on pretty diff." [Online]. Available: http://prettydiff.com/guide/unrelated_diff.xhtml