

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“DISEÑO DE UN MODELO DE CLASIFICACIÓN
AUTOMÁTICA DE LA COMPLEJIDAD DE EJERCICIOS DE
AUTOCORRECCIÓN EN UN CURSO MASIVO DE
PROGRAMACIÓN”

VICENTE DE LA ROSA SÁNCHEZ

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Andrea Vásquez Guerra
Profesor Correferente: Federico Meza Montoya

Abril - 2024

DEDICATORIA

Para mi amada familia, amigos y personas que me rodean, sin ustedes nada de esto sería posible. Con un cariño inimaginable, muchas gracias.

AGRADECIMIENTOS

Las personas que nos rodean son aquellas con las que vivimos los momentos más felices y bellos de la vida, así como también son las que nos acompañan en los pasajes grises, que no queremos recordar. Nos moldean, nos motivan a cumplir nuestros sueños, objetivos y metas, y nos recuerdan la suerte que uno tiene de disfrutar cada instancia a su lado. Mis primeras palabras van dirigidas a todos ellos y ellas.

Cuando pienso en mi familia, siempre recuerdo lo bonito de que seamos pocos. Hay veces, claro, en que a uno le gustaría tener más integrantes, pero el amor y la cercanía de una pequeña cantidad es incomparable. En primer lugar, agradecer a mi madre, por su tenaz compañía en este viaje, y no solo en este, sino que durante toda mi vida. Su caluroso apoyo e inigualable cariño corresponden a una gran causa de la persona que soy hoy. A mi hermana, Beatriz, a la cual agradezco todos los días por tener la posibilidad de ser un hermano, brindándonos consejos, aprendizajes y momentos únicos. A mi padre, Germán, por ser la persona que me enseñó a vivir cada día como si fuera el último, disfrutando cada momento, sin descuidar las enseñanzas y momentos serios. Los tres son mis pilares de vida, y sin ustedes, no sería quien soy.

Por supuesto agradezco, también, a mi familia que no está conmigo todos los días. A mi abuela, Draga, por ser el amor personificado, preocupándose de todo detalle que a uno se le escapa. A mi tía, Claudia, y mis primas, Daniela y Valentina, por su cariño inconmensurable, haciéndome sentir como en mi hogar a pesar de la distancia, y su afecto y ternura que me acompañó en toda mi estadía en la universidad. Finalmente, a mi tío, Felipe, por brindarme diferentes puntos de vista sobre la vida, y su cuidado constante por mi bienestar.

La vida no es nada sin amigos, y saber que las personas que te acompañaron en todo momento en mi estancia universitaria siguen presentes, solo me hace estar aún más agradecido. Un gran agradecimiento a Axel, Benjamín, Claudio, Elizabeth, Ignacia, Jesús, Joaquín, Pedro, Javier y Yi, por permitirme vivir junto a ustedes una de las etapas más alegres y radiantes de mi vida.

La fortuna no acaba ahí. Las amistades que me acompañaron durante mi etapa escolar todavía siguen a mi lado, lo que solamente me hace pensar que la vida me sonríe. Agradezco, en primer lugar, a mi amigo Nicolás, al cual lo conozco desde que tengo 4 años, y me sigue acompañando hasta el día de hoy. A mis amigas Constanza, Fernanda y Paula, las cuales me hacen sentir completamente afortunado de poder ser su amigo, desplegando un cariño y un sentimiento de cercanía muy profundo. A mis amigos Brandon, Felipe, Ianko, Julio y Maximiliano, por hacerme sentir en casa cada vez que estamos juntos, siendo partícipe de conversaciones y momentos extraordinarios.

Finalmente, agradecer a los profesores que me acompañaron en este proceso de memoria, Andrea Vásquez y Federico Meza, por estar siempre atentos a todo trabajo realizado, supervisando que se realizara con una calidad impecable y acorde a lo que puedo entregar como profesional.

RESUMEN

Resumen—La Universidad Técnica Federico Santa María imparte un curso de programación introductoria a gran parte de sus carreras, donde utilizan una plataforma evaluativa de ejercicios de autocorrección. Los problemas disponibles carecen de una etiqueta asociada a su nivel de complejidad, lo que ha implicado en que una porción de los estudiantes sufran desmotivación y frustración, relacionada a que los ejercicios no se ajustan a su grado de conocimiento de los contenidos.

El presente trabajo de memoria se enfoca en categorizar a los ejercicios disponibles en base a su complejidad, analizando los códigos de los estudiantes enviados a la plataforma, mediante la aplicación de técnicas de aprendizaje automático. El propósito principal busca aumentar las instancias de evaluación y ejercitación de la asignatura.

Palabras Clave—Complejidad en ejercicios; Enseñanza de la programación; Apoyo al aprendizaje; Clustering

ABSTRACT

Abstract—Universidad Técnica Federico Santa María offers an introductory programming course to the vast majority of its careers, where they use a self-correcting exercise evaluation platform. The available exercises lack a label associated with their grade of complexity, leading to a portion of students experiencing discouragement and frustration, because the exercises do not match their knowledge level of the contents.

This thesis work focus on categorizing the available exercises based on their complexity, analyzing the codes submitted by the students to the platform, using machine learning techniques. The main purpose is to increase the evaluation and exercise activities in the subject.

Keywords—Exercise complexity; Programming teaching; Learning support; Clustering

GLOSARIO

AST: *Abstract Syntax Tree*

CC: Complejidad ciclomática

CNED: Consejo Nacional de Educación

COVID-19: Enfermedad por coronavirus de 2019

CRUCH: Consejo de Rectores de las Universidades Chilenas

CSV: *Comma-separated values*

DI: Departamento de Informática

ID: Identificador único

IWI-131: Asignatura de programación de la Universidad Técnica Federico Santa María

MOOC: *Massive Online Open Courses*

LMS: *Learning Management System*

ODS: Objetivos de Desarrollo Sostenible

ONU: Organización de las Naciones Unidas

PAE: Programa Académico Especial

PYPL: *Popularity of Programming Language*

UTFSM: Universidad Técnica Federico Santa María

UVA: Unidad Virtual de Aprendizaje

SLOC: *Source lines of code*

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	IX
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Contexto	3
1.2 Organización	4
1.3 Situación actual	4
1.4 Objetivos	9
1.4.1 Objetivo general	9
1.4.2 Objetivos específicos	9
1.4.3 Alcance	9
CAPÍTULO 2: MARCO CONCEPTUAL	10
2.1 Enseñanza de la programación	10
2.2 Programación en la UTFSM	11
2.2.1 Metodología de enseñanza	11
2.3 Ejercicios de corrección automática	13
2.4 Complejidad y dificultad de ejercicios de programación	15
2.4.1 Complejidad	15
2.4.2 Dificultad	19
2.5 Aprendizaje automático	21
2.5.1 Clustering	22
2.5.2 K-means clustering	24
2.6 Stack tecnológico	26
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	28
3.1 Antecedentes	28
3.1.1 Entrevista sobre ejercicios a estudiantes	28
3.1.2 Entrevista a profesores sobre conceptos clave	31
3.2 Análisis de la complejidad de los ejercicios	32
3.3 Análisis de la dificultad de los ejercicios	38
3.3.1 Puntaje promedio	39
3.3.2 Estudiantes aceptados	40

3.3.3 Envíos aceptados	41
3.4 Clasificación de la complejidad de los ejercicios	42
3.4.1 Resultados	43
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	47
4.1 Antecedentes	47
4.2 Resultados	48
CAPÍTULO 5: CONCLUSIONES	53
5.1 Conclusiones generales	53
5.2 Cumplimiento de objetivos	54
5.3 Trabajo a futuro	55
5.4 Palabras finales del autor	55
ANEXOS	56
Anexo 1: Gráficos de completitud de ejercicios	56
Anexo 2: Gráficos de la cantidad de líneas de código por ejercicio	59
Anexo 3: Gráficos de la complejidad ciclomática por ejercicio	62
Anexo 4: Gráficos de la cantidad de conceptos por cada ejercicio	65
REFERENCIAS BIBLIOGRÁFICAS	68

ÍNDICE DE FIGURAS

1	Unidad Virtual de Aprendizaje.	5
2	Ejemplo de problemas asignados a un estudiante en la plataforma SMOJ	7
3	Árbol del problema.	8
4	Vista del estudiante de la plataforma AULA en la segunda unidad	12
5	Vista del estudiante de la página web de SMOJ	13
6	Ejemplo de un ejercicio de SMOJ, con sus componentes detallados gráficamente	14
7	Diagrama de flujo de un programa simple, que distingue si el número en la variable x es pequeño o grande.	16
8	Árbol de sintaxis abstracta (derecha) para el código correspondiente (izquierda)	18
9	Tipos de aprendizajes no supervisados	22
10	Ejemplo de la aplicación de un algoritmo de <i>clustering</i> en 3 grupos	23
11	Ejemplo de <i>clustering</i> jerárquico. De izquierda a derecha se representa el aglomerativo, mientras que de derecha a izquierda se muestra el divisivo . . .	24
12	Aplicación de algoritmo <i>K-means</i> , donde $k = 2$. Se aprecia el cálculo de los centroides y como son asignados los elementos a su grupo	25
13	Ejemplo de un <i>Dataframe</i> de la librería Pandas, donde se recopilan datos sobre la felicidad de los países	27
14	Gráfico de torta que cuantifica la cantidad de estudiantes por carrera que participaron en la entrevista.	29
15	Métricas claves en la percepción de la complejidad de los ejercicios por parte de los estudiantes.	31
16	Tabla <i>submissions</i> donde se almacenan los envíos de los estudiantes en SMOJ. .	32
17	Tabla <i>exercise</i> donde se almacenan los datos del enunciado de los ejercicios . .	33
18	Vista de SMOJ donde se puede descargar el archivo CSV de los puntajes de los estudiantes para la UVA 2	35

19	Gráfico de barras que muestra el porcentaje ejercicios que han sido completados e intentados para la UVA 2	36
20	Gráfico de barras que muestra la cantidad de conceptos utilizados, por cada UVA, para un ejercicio de la UVA 8	37
21	Gráfico de barras que representa el puntaje promedio obtenido por los estudiantes en cada ejercicio	39
22	Gráfico de barras que representa el ratio de estudiantes que completaron un ejercicio versus aquellos que lo intentaron	40
23	Gráfico de barras que representa el ratio de envíos que poseen un puntaje máximo versus el número total de envíos por ejercicio	41
24	Ejercicios de la UVA 2 ordenados según su complejidad	43
25	Ejercicios de la UVA 3 ordenados según su complejidad	43
26	Ejercicios de la UVA 4 ordenados según su complejidad	44
27	Ejercicios de la UVA 5 ordenados según su complejidad	44
28	Ejercicios de la UVA 6 ordenados según su complejidad	45
29	Ejercicios de la UVA 7 ordenados según su complejidad	45
30	Ejercicios de la UVA 8 ordenados según su complejidad	46
31	Ejercicios disponibles en la UVA 3 para el semestre 2023-2	47
32	Gráfico del ratio de completitud versus los intentos de los estudiantes en la UVA 3, para el semestre 2023-2	48
33	Ejercicios disponibles en la UVA 3 para el semestre 2024-1	49
34	Ejercicios de la UVA 3, para el semestre 2024-1, ordenados según su complejidad	49
35	Gráfico del ratio de completitud versus los intentos de los estudiantes en la UVA 3, para el semestre 2024-1, en el paralelo analizado	50
36	Gráfico del ratio de completitud versus los intentos de los estudiantes en la UVA 3, para el semestre 2024-1, contabilizando la totalidad de paralelos de la asignatura	51

ÍNDICE DE TABLAS

1 Fórmulas para el cálculo de la dificultad de un ejercicio.	19
2 Preguntas asociadas a las dificultades encontradas por los estudiantes al momento de resolver ejercicios de programación.	20
3 Estructura del <i>Dataframe</i> creado para almacenar los códigos de los estudiantes. Cada fila representa el cálculo de las métricas de un envío individual	36
4 Declaraciones que afectan el cálculo de la complejidad ciclomática en Radon. Al cómputo final se le suma 1.	37
5 Estructura del <i>Dataframe</i> creado para almacenar el conteo de palabras por cada componente del ejercicio. Cada fila representa el conteo de un enunciado individual	38
6 Ejemplo de <i>Dataframe</i> con los ejercicios y las métricas de complejidad calculadas	42
7 Tabla comparativa de los porcentajes de completitud, por ejercicio, para los semestres 2023-2 y 2024-1	50
8 Tabla comparativa de los porcentajes de completitud de los ejercicios, para el paralelo donde se aplicó la metodología versus la totalidad de paralelos de la asignatura, en el semestre 2024-1	52

INTRODUCCIÓN

El auge tecnológico evidenciado en las últimas décadas ha supuesto que la enseñanza de la programación tome una relevancia sin igual en el mundo actual. Las universidades han diversificado sus planes de estudio, acogiendo a la programación como un conocimiento base en carreras donde previamente no se impartía. Incluso, algunos colegios e instituciones de educación básica han optado por empezar a enseñar a programar a jóvenes desde edades tempranas.

Bajo este contexto, la Universidad Técnica Federico Santa María ofrece una asignatura de programación introductoria a la gran mayoría de sus carreras, sentando las bases del pensamiento lógico y algorítmico, y potenciando competencias tales como la resolución de problemas, proactividad y autogestión. Para evaluar el cumplimiento de dichas competencias, la asignatura dispone a los estudiantes de una plataforma que les permite realizar ejercicios, donde la solución es revisada de forma automática.

Sin embargo, al ser una asignatura de primer año, la diversidad de estudiantes que ingresan a la universidad supone una diferencia considerable en las bases y en los niveles de conocimiento, respaldada por un sistema académico desigual en Chile. Debido a la estandarización de la enseñanza, resulta imposible poder instruir a cada estudiante tomando en consideración sus necesidades particulares. Sin embargo, la coordinación del ramo busca tomar medidas para contrarrestar estas dificultades.

Uno de los problemas que posee la plataforma de resolución de ejercicios de la asignatura es la carencia de una etiqueta asociada a la complejidad de los ejercicios, que le permitiría a los estudiantes poder realizar aquellos que mejor se ajusten a su nivel de conocimiento de los contenidos, evitando la desmotivación y frustración inherente a no poder resolverlos.

En el marco del trabajo de memoria, se presenta una propuesta que permite clasificar a los ejercicios disponibles en la plataforma de evaluación en base a su complejidad, con el objetivo de aumentar las instancias de ejercitación y evaluación que posee la asignatura, y motivar a los estudiantes a incrementar su participación en estas actividades, buscando una mejora en su desempeño en el curso.

La estructura de este documento se compone de 5 capítulos.

El primer capítulo, denominado definición del problema, se enfoca en explicar el contexto en que se enmarca el trabajo, como también la situación actual vivida por la coordinación de programación y el problema principal a resolver. Se detallan los objetivos que se quieren alcanzar con la solución propuesta, junto con el alcance que tendría al momento de su implementación.

El segundo capítulo está compuesto por el marco conceptual que guiará al trabajo, donde se revisarán, en detalle, las bases y conceptos fundamentales para realizar la solución.

El tercer capítulo, llamado propuesta de solución, presenta los antecedentes recopilados previo al trabajo concreto a realizar, que lo sustenta y refuerza, y explica los pasos desarrollados para llegar al producto final del trabajo de memoria.

El cuarto capítulo, nombrado validación de la solución, expone la metodología seguida para la verificación de los objetivos definidos en el capítulo 1, demostrando tangiblemente que la propuesta del trabajo soluciona los problemas identificados.

Finalmente, el quinto capítulo recopila las conclusiones derivadas del trabajo de memoria realizado, junto con reflexiones y consideraciones respectivas al trabajo futuro a realizar, para una posible profundización y seguimiento del tema tratado.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contexto

El creciente interés en carreras de las áreas de la ciencia y la tecnología en los últimos años, evidenciado en un alza de las matriculas de un 81,3 % desde el año 2005 hasta el año 2020 [CNED, 2020], y la diversidad de estudiantes que ingresan a la educación superior, marcada por un sistema educativo desigual en Chile, suponen una diferencia sustancial en la formación y en los conocimientos de entrada de los alumnos de primer ingreso.

En paralelo, el primer año de universidad supone una transición desde el colegio excesivamente compleja, atendiendo un conjunto de nuevos sistemas académicos y sociales, los cuales tienen que ser enfrentados a medida que se adaptan a nuevos desafíos y se afrontan obstáculos y problemas que no han vivido antes [Millet, 2015]. Los hábitos de estudio y la gestión del tiempo para el desarrollo de sus obligaciones puede impactar positiva o negativamente dependiendo de la eficiencia con la que realizan sus tareas [Yuangga y Sunarsi, 2018].

Una asignatura transversal en las carreras de ingeniería resulta ser la de introducción a la programación, que permite a los estudiantes desarrollar sus habilidades de resolución de problemas y sentar las bases de su pensamiento lógico y algorítmico. Estos conocimientos resultan cruciales para el desenvolvimiento correcto del estudiante a lo largo de su carrera, sin embargo, las tasas de abandono y reprobación son regularmente altas [Margulieux *et al.*, 2020].

Uno de los principales motivos para el abandono de la asignatura antes de finalizarla corresponde a un uso del tiempo ineficiente por parte del estudiante, debido a que decidió realizar otra actividad, no asignó el tiempo suficiente al estudio del curso o porque alguna sección de la asignatura resultó ser mucho más difícil de lo que esperaban, designando un mayor tiempo del que habían previsto [Kinnunen y Malmi, 2006].

De esta forma, un factor crítico que influye en el desempeño del estudiante y su continuidad en la asignatura resulta ser el tiempo dedicado a las tareas y al estudio de los contenidos. Los estudiantes deben invertir el tiempo suficiente para obtener buenas notas. No obstante, este tiempo debe ser de calidad y utilizado de forma eficiente, con el objetivo de mejorar las habilidad de aprendizaje, lo que repercutirá en su desempeño académico de manera positiva [Liu, 2022].

1.2. Organización

La Universidad Técnica Federico Santa María (UTFSM) es una institución de educación superior ubicada en Chile, siendo parte de las Universidades tradicionales privadas adscritas al Consejo de Rectores de las Universidades Chilenas (CRUCH). Cuenta con cinco campus y sedes en todo el país: Campus Casa Central, ubicado en Valparaíso; Campus San Joaquín y Campus Vitacura, localizados en la capital, Santiago de Chile; Sede Viña del Mar José Miguel Carrera; y Sede Concepción Rey Balduino De Bélgica. En todas las facultades se imparten exclusivamente carreras asociadas a la ciencia y la tecnología.

El origen y fundación de la institución se remonta al año 1920, donde el empresario y filántropo chileno Federico Santa María decreta en su testamento las bases y deseos de contribuir con el progreso del país, mediante la facilitación del ingreso a estudios académicos de aquellos y aquellas que no tuvieran los recursos para su acceso, pero si el mérito, las aptitudes y un alto rendimiento académico. Es así, que en el año 1928 empieza la construcción de la casa de estudios, denominada “Escuela de Artes y Oficios”, y, a mediados de la década de 1930, se adquiere la calidad de plantel de educación superior y pasa a denominarse como Universidad Técnica Federico Santa María.

El plan formativo descrito en su modelo educativo considera, para todas las ingenierías, una asignatura de programación introductoria obligatoria para los estudiantes de primer ingreso, que es dictada a más de 2400 alumnos al año de forma coordinada en los tres Campus de la universidad [Vásquez *et al.*, 2021]. Su objetivo principal es desarrollar el pensamiento lógico y algorítmico en los alumnos, permitiendo la correcta resolución de problemas ingenieriles, mediante el uso del lenguaje de programación *Python*.

1.3. Situación actual

A raíz de la crisis sanitaria del COVID-19 en el año 2020, la UTFSM pasó de un formato educativo presencial, donde los estudiantes acuden a las diferentes sedes de la universidad a realizar sus actividades, a uno completamente remoto, debido a las restricciones de movilidad e interacción física impuestas. Por consecuencia, la asignatura de programación también tuvo que adaptarse y empezó a dictarse en formato *online*.

Pese a lo repentino de la transición, la coordinación de programación no tuvo que realizar una transformación severa de la asignatura. Desde comienzos del año 2019, ya se habían implementado diversas estrategias y cambios estructurales en su programa de estudios, adoptando una metodología constructivista para la enseñanza, donde el estudiante adquiere un rol activo en su proceso de aprendizaje, gracias al aumento de instancias de ejercitación y asimilación de los conocimientos. Así, los resultados de aprendizaje esperados al término del semestre y los métodos de evaluación son definidos con anterioridad y de manera concreta y coherente, con el objetivo de lograr una mejora en el proceso educativo de los alumnos.

Por consiguiente, el reajuste hecho a la asignatura en el periodo de la crisis sanitaria se enfocó en mantener los objetivos de aprendizaje originales del curso, considerando las diferentes realidades personales y de conectividad propias del nuevo formato educativo. Para la coordinación, resultó de vital importancia mantener constante la asistencia y la participación tanto en las clases dictadas como en las actividades formativas de los estudiantes. Para lograr esto, [Vásquez *et al.*, 2021] determinaron que un curso MOOC tradicional no sería suficiente, debido a la alta capacidad de autogestión, independencia y proactividad necesarias para su finalización.

La estrategia utilizada por parte de la coordinación de la asignatura es conocida como aula invertida (*flipped classroom*), cuya dinámica principal es la inversión del tradicional orden de enseñanza. Los estudiantes tienen contacto con los diferentes contenidos antes de su asistencia a las clases, utilizando material como vídeos o textos, en conjunto con pequeñas pruebas *online* para verificar su aprendizaje. De esta forma, las sesiones con el docente son reservadas para la discusión y resolución de dudas, y para la aplicación del conocimiento adquirido.

En base a esto, la coordinación de programación decidió dividir los resultados de aprendizaje y contenidos de la asignatura en un conjunto de 9 unidades, denominada UVA. Cada UVA se encuentra concreta y coherentemente definida en base a los resultados esperados por parte de los alumnos al término de esta, orientando el diseño de actividades y evaluaciones asociadas.

La estructura de una UVA se aprecia a continuación:

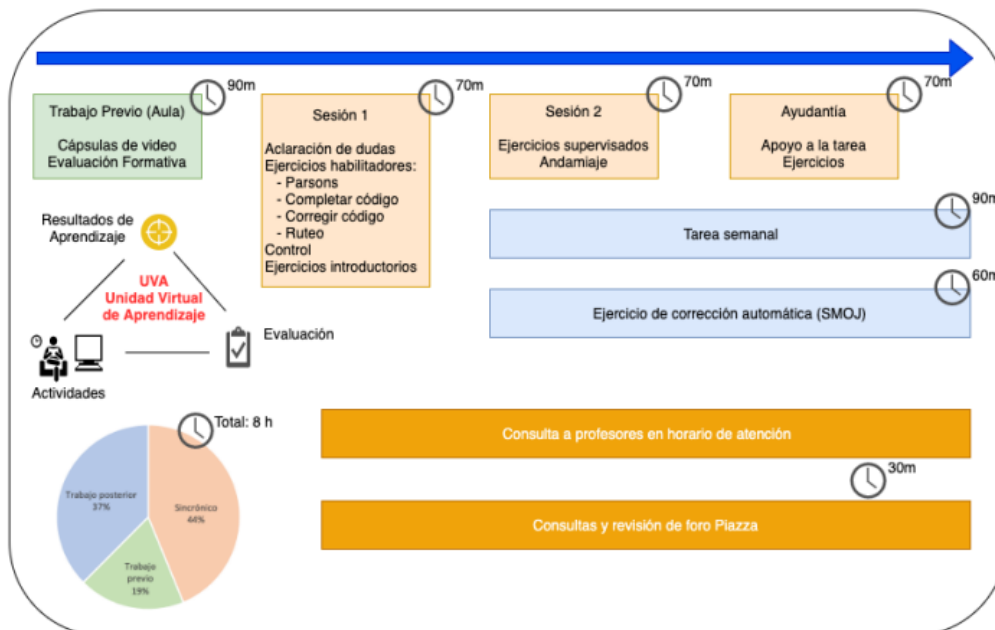


Figura 1: Unidad Virtual de Aprendizaje.

Fuente: [Vásquez *et al.*, 2021]

En la figura 1, es posible apreciar la división de la UVA en 3 tipos de actividades:

- Trabajo previo: Los estudiantes deben revisar, antes de la clase, un vídeo introductorio de los nuevos contenidos, el cual será evaluado en un cuestionario de evaluación formativa, con el objetivo de obtener un grado de comprensión inicial de la materia.
- Trabajo sincrónico: Comprende dos sesiones con el profesor de la asignatura y una sesión de ayudantía. En la primera sesión, se espera que el estudiante realice preguntas y ejercicios de complejidad baja donde aplican lo aprendido en el trabajo previo, finalizando con un control tipo *quiz*, que es parte de las evaluaciones de la asignatura. Esto permite motivar al estudiante a la realización del trabajo previo. En la segunda sesión, el profesor dispone de ejercicios a los estudiantes para que los resuelvan de manera supervisada. Finalmente, la ayudantía se efectúa después de la primera sesión, buscando que los estudiantes resuelvan sus dudas y permitir un mejor entendimiento de los contenidos tratados.
- Trabajo posterior: Los estudiantes poseen de dos tipos de actividades: resolución de ejercicios de corrección automática, disponibles en una plataforma llamada SMOJ que verifica automáticamente las entregas, generando retroalimentación inmediata; y una tarea, que busca reforzar los aprendizajes de la unidad mediante problemas de complejidad promedio para los estudiantes.

En conjunto con la estructura anterior, los estudiantes de la asignatura poseen de dos exámenes principales que engloban a una cierta cantidad de UVAs, y un tercer examen de recuperación opcional en caso de que el alumno no haya podido rendir alguno de los anteriores. Todas las evaluaciones de la asignatura (evaluaciones formativas, controles, tareas, ejercicios de corrección automática y exámenes) son desarrolladas de manera individual.

Uno de los principales problemas detectados por la coordinación de la asignatura radica en los ejercicios de corrección automática realizados en SMOJ. Dentro de la plataforma, los estudiantes disponen de enunciados de problemas elaborados previamente por los docentes, divididos por UVA. Para su resolución, los alumnos deben elaborar una solución en código, mediante el lenguaje de programación *Python*, y enviarla dentro de la página. El código es evaluado por un juez en línea con distintos casos de prueba, y el veredicto es entregado de vuelta al alumno, indicando si lo resolvió correcta o incorrectamente.

El obstáculo en cuestión se percibe en el momento que los estudiantes tienen que escoger que problemas resolverán, ya que solamente se les exige la resolución correcta de uno de los posibles que se les ofrece, pudiendo resolver, si se desea, la totalidad de estos. Al momento de la elección del ejercicio, la plataforma muestra en una tabla, mediante un orden arbitrario, la recopilación de problemas que el estudiante puede resolver. Debido a que no existe alguna etiqueta relacionada con la complejidad, los ejercicios no siempre van a ser acordes al nivel de conocimiento actual que se posee.

Resumen

Problema	Tarea	Puntaje obtenido
Números ordenados	1	0
Concurso Mechones	1	0
Menor divisor	1	0
pingXpong	1	0
Propagación de la enfermedad	1	0

Figura 2: Ejemplo de problemas asignados a un estudiante en la plataforma SMOJ
Fuente: SMOJ

Uno de los casos detectados por la coordinación de la asignatura resulta en que el estudiante selecciona el primer ejercicio de la lista, como se aprecia en 2, y resulta ser uno de complejidad mayor al nivel de conocimiento actual que posee el alumno, dando la posibilidad a que surja desmotivación y una postergación de su estudio. Otro caso es el contrario: el estudiante selecciona un ejercicio y lo resuelve de forma correcta inmediatamente, dando por cumplida la tarea del envío mínimo de un problema, cesando la práctica y adquisición de conocimiento con los ejercicios restantes que ofrece la plataforma.

Debido a que la plataforma SMOJ resulta el principal medio de ejercitación que poseen los estudiantes en la asignatura, los problemas encontrados y las consecuencias que tienen en los alumnos desembocan en una falta de motivación, la cual se va acumulando a medida que los contenidos avanzan y el estudiante se va quedando atrás. Todo esto termina en el estudiante abandonando la asignatura o reprobándola al término del semestre académico.

De manera paralela a la detección de los problemas mencionados por la coordinación, y en el contexto de las asignaturas “Gestión de proyectos informáticos” y “Taller de Desarrollo de proyectos informáticos” de la carrera ‘Ingeniería Civil Informática’ de la UTFSM, donde se le pide a los estudiantes la creación de un proyecto que siga los lineamientos de los ODS de la ONU, nace *PySano*, una herramienta de selección inteligente de ejercicios para la asignatura de programación, que considera las necesidades, fortalezas y debilidades de los estudiantes. Considerando el rendimiento académico en las evaluaciones del ramo y el rendimiento en la resolución de problemas dentro de la plataforma, proporciona material de estudio y ejercicios acorde al ritmo de aprendizaje y conocimientos que posee el estudiante.

Los ejercicios que dispone *PySano* a los estudiantes son extraídos del banco de preguntas que posee la coordinación, y que son ocupados en, actualmente, una de las cuatro diferentes instancias de ejercitación y evaluación durante el semestre académico, siendo esta los ejercicios disponibles en la plataforma SMOJ. Lo anterior resulta en que las cuatro instancias restantes (evaluaciones formativas, controles, tareas y certámenes) no extraen ejercicios directamente de este banco, sino que los profesores y ayudantes deben diseñar y crear ejercicios ad hoc manualmente, lo que deriva en grandes ineficiencias al momento de tomar en cuenta las necesidades de los estudiantes en la creación de ejercicios adaptables y de calidad.

Complementando lo anterior, la creación de los ejercicios por parte del equipo de docentes y ayudantes debe considerar el proceso cognitivo que realiza el estudiante para su resolución, es decir, estimar y valorar la complejidad inherente del problema, para que se ajuste correctamente al nivel de conocimiento del alumno, y no repercuta en el curso de su aprendizaje ni desemboque en los problemas mencionados previamente. La inadvertencia de la complejidad del ejercicio en su proceso creativo corresponde a otra dificultad encontrada por la coordinación de la asignatura, resultando en ineficiencias al momento de crearlos, y los profesores optando por el uso de ejercicios almacenados en el banco de preguntas, pertenecientes a períodos académicos previos, reciclándolos y reutilizándolos en las instancias de trabajo de los alumnos.

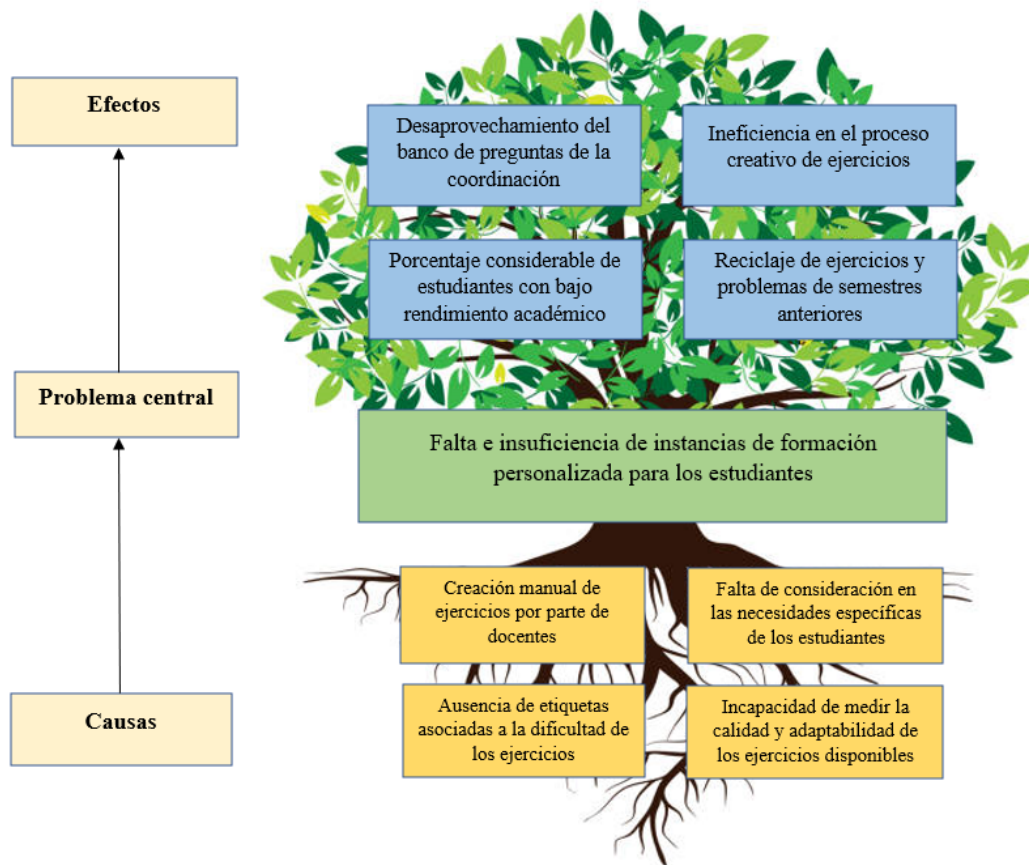


Figura 3: Árbol del problema.
Fuente: Elaboración propia.

Por lo anterior, resulta de importancia y urgencia buscar una solución que permita aprovechar el banco de preguntas de la asignatura de programación en su totalidad, generando un etiquetado a priori de la complejidad de un ejercicio para su disposición como material de estudio, permitiendo un aumento de las instancias de ejercitación y evaluación acordes y adaptadas a las necesidades de los estudiantes.

1.4. Objetivos

1.4.1. Objetivo general

Aumentar las instancias de ejercitación y evaluación en un curso masivo de introducción a la programación a través del desarrollo de un modelo de clasificación automática de la complejidad de ejercicios de autocorrección en un curso masivo de programación.

1.4.2. Objetivos específicos

- Analizar los conceptos clave referidos a la medición de la complejidad de ejercicios de programación en un contexto de una asignatura universitaria introductoria.
- Diseñar un modelo de clasificación automática de la complejidad de ejercicios de programación basado en técnicas de inteligencia artificial y aprendizaje automático.
- Evaluar el rendimiento del modelo en base a diferentes métricas y encuestas aplicadas a los estudiantes de programación de la universidad.

1.4.3. Alcance

El trabajo de memoria busca repercutir en una de las cinco instancias de ejercitación y evaluación de la asignatura: los ejercicios de corrección automática dispuestos en la plataforma SMOJ. Considerada la principal herramienta que poseen los estudiantes para poner en práctica los conocimientos adquiridos en las instancias restantes, se busca abstraer a los estudiantes de la tarea de seleccionar manualmente los problemas, permitiendo encarrilarlos y adecuarlos a su ritmo de aprendizaje y nivel de conocimiento personal que poseen, aumentando su motivación y desempeño académico.

Si bien solamente se opta por el análisis proporcionado por los ejercicios de la plataforma SMOJ, debido a su naturaleza de problema a resolver mediante una solución en código *Python*, las demás instancias de aprendizaje, ejercitación y evaluación claramente influyen en los resultados que el estudiante obtendrá en la asignatura. En el futuro, el trabajo y análisis a realizar sobre la complejidad de estos ejercicios puede ser ampliada a todas las instancias de trabajo que posee el alumno, influyendo de una mejor y amplia manera en los estudiantes y su rendimiento en la asignatura.

CAPÍTULO 2

MARCO CONCEPTUAL

2.1. Enseñanza de la programación

La programación corresponde al acto en que una persona comunica instrucciones a realizar por un computador [Sobral, 2021]. A través de un lenguaje de programación, el usuario puede concretarlas y resolver aquello que desea. Si bien la definición propuesta solo hace referencia al uso de un elemento tecnológico para el desarrollo de una acción, los cursos de introducción a la programación no se enfocan solamente en esto. Antes del diseño de un programa que resuelva un problema en específico, las personas deben comprender cuál es el problema en cuestión y cómo usar las herramientas proporcionadas por la tecnología para llevarlo a cabo.

El curso de introducción a la programación de la UTFSM, denominado “Programación”, con siglas IWI-131, enfoca la asignatura en que los estudiantes apliquen el pensamiento computacional para la resolución de problemas de ingeniería; y la manipulación de datos y establecimiento de relaciones para la obtención de nueva información, mediante el uso de algoritmos. Cuando el estudiante finaliza el curso, se espera que haya desarrollado correctamente la habilidad de resolución de problemas, por medio del análisis y la evaluación de soluciones efectivas y eficientes.

Así, la instrucción de las bases del pensamiento lógico y algorítmico en los estudiantes que empiezan sus carreras universitarias resulta de gran importancia para la adquisición de habilidades de resolución de problemas, las cuales serán indispensables para su desarrollo profesional. El contenido enseñado, los objetivos de aprendizaje y la forma en que se practica la docencia necesita una preparación acorde a su complejidad. Debido a esto, enseñar programación a estudiantes que no poseen un conocimiento de entrada sobre el área es una tarea compleja. Mientras algunos pueden aprender de manera rápida y sencilla, otros sienten que es casi imposible entender los contenidos y aprobar la asignatura, lo que a menudo lleva al abandono de esta [Sobral, 2021].

En consecuencia, la metodología de aprendizaje y enseñanza a utilizar en un curso introductorio de programación debe ser cautelosamente elegida para potenciar el aprendizaje de los estudiantes y permitir que se desenvuelvan de manera correcta a medida que avanzan en la asignatura. El enfoque y estrategias acordadas por la coordinación de programación para el desarrollo del curso son discutidas en el siguiente capítulo.

2.2. Programación en la UTFSM

Como fue introducido en el capítulo 1, la coordinación de programación efectuó diversos cambios estructurales a la asignatura tras la crisis sanitaria del COVID-19, transicionando a una modalidad completamente en línea desde una presencial. Sin embargo, con el cese de gravedad de la pandemia a inicios del año 2022 y la atenuación de las restricciones impuestas al distanciamiento físico en el país, la UTFSM decidió que todas las asignaturas dictadas en la casa de estudios volverían a efectuarse en formato presencial.

La estructura de la asignatura recibiría ligeros cambios, enfocados en que las clases impartidas por los docentes ya no serían realizadas a través de una plataforma *online*, si no que en las salas de clases dispuestas por la universidad. De esta forma, se procede a detallar y describir la organización del curso, cuyas bases y conceptos se revisarán en la literatura.

2.2.1. Metodología de enseñanza

La elección del modelo de aula invertida por parte de la coordinación de programación se debió, principalmente, al contexto vivido de enseñanza remota, y al limitado tiempo e interacción existente entre estudiante y profesor [Vásquez *et al.*, 2021].

La literatura describe el modelo de aula invertida como una metodología que, como su nombre indica, invierte el orden tradicional de enseñanza, en el cual los estudiantes adquieren nuevos contenidos mediante clases expositivas por parte de los docentes, por uno donde los tópicos de estudio son presentados con anterioridad a la interacción con el docente, mediante el uso de videos, textos, u otro tipo de materiales, apoyado por pequeños *tests* para medir el conocimiento logrado (cuestionarios de evaluación formativa). Así, las clases son reservadas para aclarar dudas, permitir la discusión de los contenidos tratadas y la aplicación de las competencias adquiridas [Sobral, 2021].

La coordinación de programación ejecuta esta estrategia mediante el uso de un Sistema de Gestión del Aprendizaje (LMS, por su sigla en inglés, Learning Management System) denominado AULA, implementado por la UTFSM y basado en Moodle. El sistema permite a los estudiantes disponer de material de estudio para cada asignatura que cursa, informarse de noticias por parte del cuerpo docente, participar en foros con sus pares para generar discusión y consultas sobre el curso y rendir evaluaciones de manera *online*, entre otras funciones.

La plataforma AULA permite a la coordinación de programación alinearse con el planteamiento constructivista que sirvió de base para la reestructuración de la asignatura, permitiendo la definición coherente de las reglas del curso y su sistema de evaluación, los aprendizajes esperados por cada UVA y los métodos y actividades de enseñanza que dispondrán los estudiantes.

Como fue introducido en el capítulo 1, el curso de programación posee 9 unidades de aprendizaje, las cuales se encuentran a disposición de los alumnos a medida que se van enseñando los contenidos en el trabajo previo y sincrónico. Los temas que tratan las diferentes UVAs de la asignatura se enumeran a continuación:

1. Algoritmos / Programación con bloques y gráficos de tortuga
2. Programas secuenciales con Python
3. Condiciones lógicas y estructuras condicionales: Decisiones
4. Estructuras repetitivas: Ciclos y ciclos anidados
5. Manejo de datos tipo *string*
6. Creación y uso de funciones
7. Listas como colecciones de elementos
8. Diccionarios
9. Procesamiento de texto y archivos

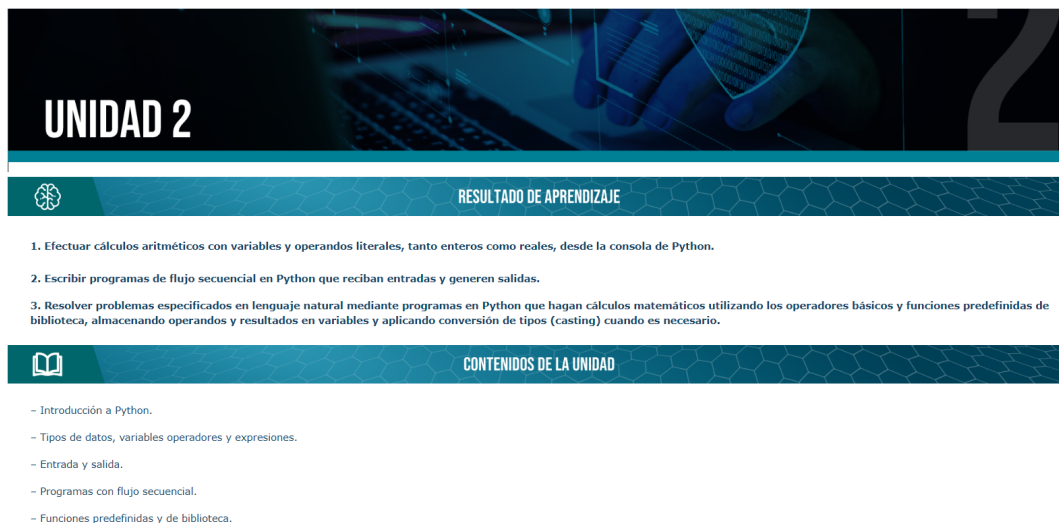


Figura 4: Vista del estudiante de la plataforma AULA en la segunda unidad
Fuente: AULA USM

A través de AULA, los estudiantes pueden ingresar a SMOJ (Santa María Online Judge), plataforma que será el centro de la investigación a realizar y que fue introducida en el capítulo 1. Mediante este sistema *web*, se desarrolla el trabajo posterior que los alumnos tienen que realizar por cada UVA de la asignatura, resolviendo ejercicios de corrección automática de diferentes complejidades, utilizando los contenidos vistos en el trabajo previo y sincrónico.



Figura 5: Vista del estudiante de la página web de SMOJ
Fuente: SMOJ

Los ejercicios de corrección automática, cuya solución involucra la creación de un código en *Python* por parte del estudiante, corresponden al foco del trabajo de memoria, en base a los problemas detectados por la coordinación de programación. La estructura y funcionamiento de estos son explicados en el siguiente capítulo.

2.3. Ejercicios de corrección automática

Los ejercicios de corrección automática son aquellos que no requieren intervención manual para su evaluación (la cual podría ser por parte de un profesor o ayudante de la asignatura), utilizando un sistema encargado que permite la automatización de esta tarea, generalmente llamado juez en línea. La resolución de estos ejercicios, en el contexto de la programación, se basa en el diseño de un código escrito en un lenguaje de programación acordado por la asignatura (en el caso de IWI-131, *Python*), que efectivamente soluciona el problema, en base a diferentes casos de prueba que la plataforma provee.

Los ejercicios de corrección automática dispuestos en SMOJ para su resolución poseen una estructura estándar que permite a los estudiantes entender correctamente lo que el problema requiere para ser solucionado. Los componentes del ejercicio, los cuales serán claves para la propuesta de solución, son detallados a continuación:

- **Nombre del ejercicio**

- **Enunciado:** Presenta una descripción detallada del contexto que abarca el ejercicio, ofreciendo al estudiante todo lo que debe saber para su correcta resolución.
- **Instrucciones:** Indica al estudiante lo que su código debe realizar para solucionar de manera correcta el ejercicio.
- **Entrada:** Presenta los valores de entrada al programa, junto a su tipo de dato, cuyo uso permitirá al código resolver el problema. En los ejercicios de la UVA 6 (funciones), este componente se denomina **Parámetros**, debido a su naturaleza.
- **Salida:** Corresponde lo que el código debe imprimir por pantalla, posterior a los procesamiento y ejecuciones hechas. Los casos de prueba se contrastan con la salida, y permiten evaluar, de manera cuantitativa, el desempeño del código en una escala del 0 al 100. En los ejercicios de la UVA 6, este componente se denomina **Retorno**.
- **Observaciones:** Otorga detalles sobre el problema para que el estudiante los considere dentro de su creación del código. Es un componente opcional, debido a que no está presente en todos los ejercicios.
- **Ejemplo:** Muestra al estudiante la salida que debería entregar el programa en base a la entrada.

Un ejemplo de un ejercicio de corrección automática en SMOJ es el siguiente:

Área de un triángulo (Fórmula de Herón)
Título

La fórmula de Herón permite calcular el área de un triángulo a partir de sus lados (a , b , c). Sea s el semiperímetro del triángulo: $s = \frac{a+b+c}{2}$, el área del triángulo corresponde a: $\sqrt{s \times (s - a) \times (s - b) \times (s - c)}$.

Enunciado

Tarea 1

Tarea 1

Escriba un programa que lea como entrada las longitudes de los lados de un triángulo, y determine su área utilizando la fórmula de Herón. El programa debe entregar como salida el área redondeada a dos posiciones decimales.

Instrucciones

Entrada

Tres números positivos **con o sin decimales**, correspondientes a las longitudes de los tres lados de un triángulo. No escriba ningún mensaje antes de leer los datos.

Salida

Un único número redondeado a dos decimales, correspondiente al área del triángulo. En la salida, no escriba mensajes ni nada adicional a lo indicado.

Observaciones

Puede suponer que las entradas serán siempre correctas, es decir, números positivos en rangos razonables, y con los que se puede formar un triángulo. Es decir, cumplen con la desigualdad triangular.

Ejemplo

Entrada
4
3
5
Salida
6.0

Figura 6: Ejemplo de un ejercicio de SMOJ, con sus componentes detallados gráficamente

Fuente: SMOJ

La estructura propuesta permite la división de un ejercicio de programación en dos partes: el **enunciado** y el **código**. En el siguiente capítulo, se detallará como se han aplicado los conceptos de complejidad y dificultad estudiados en la literatura para este tipo de ejercicios.

2.4. Complejidad y dificultad de ejercicios de programación

La estructura que la coordinación de programación ha diseñado para su modelo educativo implica un escalado creciente de complejidad en las actividades y evaluaciones que los estudiantes tienen que realizar. El trabajo previo trata contenidos introductorios, que deben ser evaluados por cuestionarios sencillos. Por otro lado, las tareas de trabajo sincrónico empiezan con ejercicios y problemas de complejidad similar a las del trabajo previo, aumentando hasta llegar a la segunda sesión. Finalmente, en las actividades de trabajo posterior, se debe disponer de ejercicios de una complejidad más elevada y acorde que permita medir efectivamente el correcto nivel de comprensión de los contenidos enseñados en la unidad.

La medición de la complejidad y dificultad de ejercicios de programación ha sido estudiada previamente en contextos de asignaturas introductorias a la programación, donde los estudiantes utilizan plataformas de apoyo para el aprendizaje para la resolución de problemas, con el objetivo de poder reforzar y evaluar sus propios conocimientos. Si bien estos términos son utilizados como sinónimos, [Effenberger *et al.*, 2019] afirma que existe una distinción clave entre ambos conceptos.

2.4.1. Complejidad

La **complejidad** de un ejercicio de programación es entendida como las características intrínsecas de los problemas que influyen en el rendimiento de los alumnos, pero que son independientes al contexto (por ejemplo, las personas que lo resuelven). La complejidad puede ser medida en base al enunciado y solución del ejercicio, como también utilizando el número de líneas que posee o la cantidad de contenidos diferentes que involucra para su resolución.

A medida que los estudiantes aprenden a programar e interiorizan el pensamiento lógico y algorítmico, se vuelven incrementalmente más capaces de resolver problemas más desafiantes. En cierto punto, la complejidad de un ejercicio puede correlacionarse con las habilidades del estudiante [Ihantola y Petersen, 2019]. Los estudiantes demuestran su conocimiento en la resolución de problemas no triviales y que mezclan diferentes contenidos. Sin embargo, un ejercicio simple también puede resolverse con una respuesta o código complejo. En este caso, la complejidad indica falta de habilidades por parte del estudiante.

La medición y evaluación de la complejidad de ejercicios de programación ha sido ampliamente estudiada en el contexto de problemas de código, la cual se ha analizado mediante evaluación de expertos o un uso conveniente de heurísticas, como el número de elementos

sintácticos en el código.

Algunas métricas importantes a definir para la estimación de la complejidad de un ejercicio de programación, en base al código de respuesta que lo resuelve, son las siguientes:

- **Cantidad de líneas de código:** Corresponde a la métrica de medición más directa del tamaño del código. Sin embargo, la funcionalidad y complejidad de este no tienden a relacionarse bien, puesto que un estudiante habilidoso puede proveer la misma funcionalidad con un código más corto.
- **Complejidad ciclomática:** Corresponde a una métrica cuantitativa del número de caminos linealmente independientes a lo largo del código de un programa. Matemáticamente, está definida en base al diagrama de flujo del problema.

El diagrama de flujo de un código es una representación, utilizando grafos, de todos los caminos que pueden ser recorridos a través de la ejecución de un programa. Cada nodo del grafo modela un bloque básico, es decir, una parte del código sin ramas o saltos (punto del programa donde el flujo se ve alterado). Los arcos dirigidos son utilizados para representar los saltos entre bloques.

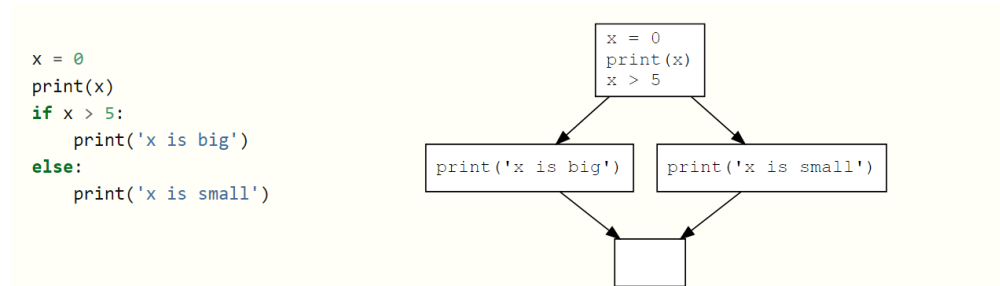


Figura 7: Diagrama de flujo de un programa simple, que distingue si el número en la variable `x` es pequeño o grande.

Fuente: [Liu y Badr, 2020]

El cálculo de la complejidad ciclomática M es el siguiente:

$$M = E - N + 2P$$

Con:

- E : Número de arcos del grafo.
- N : Número de nodos del grafo.
- P : Número de componentes conexos del grafo.

Debido a que en la asignatura de programación introductoria de la UTFSM se utilizan programas simples, el valor de P siempre será igual a 1. De esta forma, el valor de M se calcula de la siguiente forma:

$$M = E - N + 2$$

- **Métricas de Halstead:** Se deben definir ciertos conceptos para su correcto entendimiento:

- **Operador:** Símbolo que indica la resolución de una tarea matemática, relacional o lógica, produciendo un resultado final.
- **Operando:** Elemento que puede ser modificado o manipulado por el programa, tales como las variables.

De esta forma, para un cierto problema, definimos:

- η_1 : El número de operadores distintos presentes.
- η_2 : El número de operandos distintos presentes.
- N_1 : El número total de operadores.
- N_2 : El número total de operandos.

A partir de estos valores, podemos calcular diversas métricas para el código de un programa:

- Vocabulario del programa: $\eta = \eta_1 + \eta_2$
- Longitud del programa: $N = N_1 + N_2$
- Volumen: $V = N \cdot \log_2 \eta$
- Dificultad: $D = \frac{\eta_1}{2} \cdot \frac{N_2}{\eta_2}$
- Esfuerzo: $E = D \cdot V$

- **Cantidad de conceptos:** Corresponde a cuantos conceptos de programación diferentes fueron utilizados en el código de resolución del ejercicio. [Luxton-Reilly y Petersen, 2017] realizan una distinción clave para la medición de los conceptos, resultando las siguientes denotaciones:

- **Componente conceptual:** Hace referencia a un elemento sintáctico que puede ser extraído del código, los cuales se corresponden directamente con los *tokens* de sintaxis del lenguaje (*reserved keywords*, o palabras reservadas en español), en conjunto con las funciones incorporadas (por ejemplo, la función *print*).
- **Etiqueta de concepto:** Permite identificar grupos de elementos que poseen un significado conceptual, pero que no se corresponden con un *token específico*.

Así, esta definición permite diferenciar al *token while* (componente conceptual) del concepto de ciclos, el cual engloba al anterior y posibilita un análisis más preciso que se relaciona directamente con los contenidos enseñados en cada UVA de la asignatura.

El trabajo de evaluar la complejidad de un código ha requerido, a lo largo del tiempo, evaluación por parte de expertos o el uso de métricas como las expuestas previamente. Las tres primeras métricas descritas forman parte del área de ingeniería de *software* y, aunque han perdido cierta aceptación, siguen siendo de uso común debido a ser un método cuantitativo conveniente para medir el código [Voas y Kuhn, 2017]. Así, aunque no sean aplicables a todos los lenguajes de programación o sean el mejor predictor de la calidad del código, su mayor virtud radica en su facilidad de cálculo al utilizar los envíos de los estudiantes (en este caso, de SMOJ) [Jones, 1994].

Para el caso de la cantidad de conceptos, y tomando en cuenta la naturaleza tanto de la asignatura, enseñando los contenidos de forma secuencial a lo largo del semestre; y de los ejercicios utilizados en SMOJ, de carácter retroactivo con lo instruido en cada UVA, resulta una métrica más natural con el progreso de los estudiantes. [Ihantola y Petersen, 2019] demostraron una correlación fuerte entre el número de conceptos que posee un ejercicio con el progreso de los estudiantes en un curso de programación introductoria en universidades de Estados Unidos. Esto va de la mano con la retroactividad propia de los problemas, donde los contenidos vistos previamente siguen siendo relevantes, aún cuando no estén siendo evaluados directamente (por ejemplo, un ejercicio de la unidad de ciclos, muy probablemente también utilice condicionales, contenido enseñado anteriormente).

La extracción de los conceptos involucrados en un ejercicio de programación resulta mucho más sencilla de realizar automáticamente desde el código de solución. [Pelánek *et al.*, 2021] proponen la transformación del código a su correspondiente árbol de sintaxis abstracta (AST, por sus siglas en inglés) para elaborar técnicas de extracción que sean independientes al ejercicio. Este árbol corresponde a una simplificación de la sintaxis presente en el código, cuyos nodos denotan elementos de este. Por ejemplo, para el siguiente código, su árbol de sintaxis abstracta sería el que se muestra a continuación:

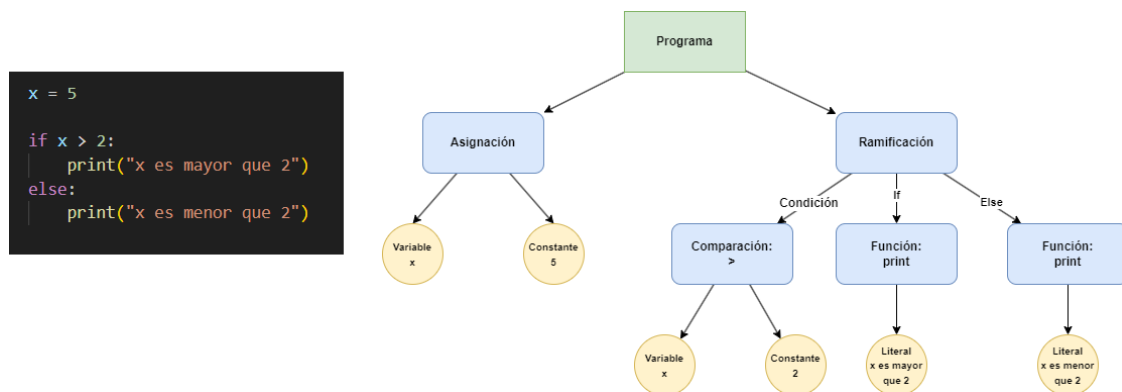


Figura 8: Árbol de sintaxis abstracta (derecha) para el código correspondiente (izquierda)
Fuente: Elaboración propia

Utilizando el AST, es posible la obtención de conceptos tales como la asignación de variables, el uso de operadores de comparación y de la estructura condicional *if - else*.

2.4.2. Dificultad

La **dificultad** de un ejercicio de programación, por otro lado, está completamente relacionada al rendimiento observado por parte de los estudiantes, tales como la tasa de fallos, el número de intentos requeridos para su resolución o el puntaje promedio obtenido.

[Tran *et al.*, 2022] genera un análisis en base a diferentes métricas del proceso de desarrollo de ejercicios en la Universidad Tecnológica de la Ciudad de Ho Chi Minh, tales como el puntaje promedio, número de estudiantes que resolvieron correctamente el problema y el puntaje máximo, entre otros, para la creación de un sistema de clasificación de la dificultad. El sistema permite un manejo del banco de preguntas de la asignatura, asignando ejercicios a grupos de estudiantes que se ajusten correctamente a su nivel de conocimiento. Junto con esto, permite la evaluación de la resolución de los problemas asignados, calculando automáticamente el puntaje de las preguntas al momento del envío de su solución, mediante un sistema desarrollado por la universidad.

Una vez calculado el puntaje, es posible la obtención de diferentes métricas específicas que describen el desempeño de los estudiantes, las cuales se describen a continuación y se detallan en la siguiente tabla:

Notación	Nombre	Rango de valores	Propiedades
F_0	Puntaje promedio	[0,1]	Más grande, más fácil
F_1	Estudiantes aceptados	[0,1]	Más grande, más fácil
F_2	Envíos aceptados	[0,1]	Más grande, más fácil
F_3	Mejor envío	[0,1]	Más grande, más fácil
F_4	Número de envíos	[0,1]	Más grande, más difícil

Tabla 1: Fórmulas para el cálculo de la dificultad de un ejercicio.

Fuente: Elaboración propia, a partir de [Tran *et al.*, 2022]

1. Puntaje promedio: Corresponde a la media aritmética de los puntajes de los estudiantes, normalizado al rango [0,1].
2. Estudiantes aceptados: Se refiere al cociente entre los estudiantes que lograron resolver el ejercicio versus la cantidad total de estudiantes que intentaron el problema.
3. Envíos aceptados: Corresponde al cociente entre la cantidad de envíos aceptados en un ejercicio en particular, versus la cantidad total de envíos realizados en el mismo ejercicio.
4. Mejor envío: Considera el mejor puntaje registrado por un estudiante en un ejercicio en particular, en relación al puntaje máximo posible de la pregunta.
5. Número de envíos: Se refiere al cociente entre la cantidad de envíos que el estudiante ha efectuado hasta resolver el ejercicio, versus la cantidad máxima de envíos permitidos por el problema.

Como se puede apreciar, para el caso anterior consideramos métricas exclusivas en base al desempeño de los estudiantes en los ejercicios.

[Effenberger *et al.*, 2019] afirma que la medición de la dificultad de las tareas encomendadas a los estudiantes permite una estimación de la habilidad para una correcta orientación en el comportamiento adaptativo de los sistemas. Un enfoque común en las asignaturas es seguir un orden de tareas, donde se presentan primero las más fáciles y después las más difíciles, o las más simples a las más complejas. Para realizar esto, es necesario saber qué ejercicios son los fáciles y difíciles para los estudiantes. En conjunto con lo anterior, la presencia de un etiquetado correcto es de suma importancia para el manejo del contenido en la asignatura, permitiendo la creación, actualización y eliminación de tareas en base a la pertinencia encontrada, resultando en la disposición de mayor y mejor material de estudio para los alumnos.

En relación con lo previamente mencionado, también hay que tomar en consideración qué dificultades poseen los estudiantes al momento de realizar un ejercicio de programación.[Ahmad *et al.*, 2019] afirma que la posesión de habilidades de resolución de problemas es fundamental al momento de empezar a aprender a programar. Estas competencias permiten un correcto desarrollo de la lectura y comprensión de los problemas. Los autores desarrollaron un cuestionario de percepción de las principales dificultades encontradas en la resolución de ejercicios de programación, tomando facetas como el análisis y comprensión de los problemas. Los estudiantes deben responder en una escala del 1 al 5, donde 1 implica estar fuertemente en desacuerdo con la afirmación, y 5 implica estar fuertemente de acuerdo con la afirmación. Algunas preguntas del cuestionario y sus resultados se detallan a continuación:

Ítem	Media del puntaje	Desviación estándar
¿Es difícil entender un programa sin antes derivar su algoritmo?	3.74	1.03
¿Es difícil entender un cierto problema de programación?	3.11	1.07
¿Es difícil interpretar un problema de programación?	2.91	0.95
¿Es difícil entender que es requerido para resolver un ejercicio?	2.77	0.95
¿Es difícil entender la lógica de un programa largo?	3.48	1.03
¿Es difícil entender los pasos a seguir al momento de resolver un problema?	2.99	1.02

Tabla 2: Preguntas asociadas a las dificultades encontradas por los estudiantes al momento de resolver ejercicios de programación.

Fuente: Elaboración propia, a partir de [Ahmad *et al.*, 2019]

2.5. Aprendizaje automático

La propuesta de solución contempla, en su parte final, la clasificación de los ejercicios de corrección automática en base a su nivel de complejidad, con el objetivo de que los estudiantes dispongan de una guía sobre que problemas se adecúan al conocimiento que poseen en el trabajo asíncrono de la asignatura. La tarea de clasificar y etiquetar a los ejercicios es realizada a través de un área de la inteligencia artificial, denominada aprendizaje automático (*machine learning*, en inglés).

El aprendizaje automático es un área de las ciencias de la computación que se deriva del estudio de reconocimiento de patrones y la teoría del aprendizaje computacional en la inteligencia artificial [Simon *et al.*, 2016]. Su uso principal radica en la creación de algoritmos que permitan aprender desde los datos, y realizar predicciones sobre estos. El procedimiento principal opera en la construcción de un modelo que, por medio de determinadas entradas de un conjunto de datos, obtenga las salidas requeridas.

El aprendizaje automático se divide en 3 áreas principales:

1. **Aprendizaje supervisado:** El programa es entrenado en un conjunto predefinido de muestras de entrenamiento, que contiene las entradas y las salidas, con el objetivo de adquirir información sobre la relación entre ambas [Liu y Wu, 2012]. Se denomina supervisado debido a la presencia de las salidas en los datos, comúnmente llamado como etiqueta.
2. **Aprendizaje no supervisado:** El programa es entrenado en un conjunto de datos que no posee etiquetas o clasificaciones. Así, el objetivo principal de esta técnica corresponde a identificar patrones y relaciones ocultas. Al contrario del área previa, no posee la salida en los datos [El Boucheffry y de Souza, 2020].
3. **Aprendizaje por reforzamiento:** El programa es entrenado tomando acciones en un cierto contexto, generando una recompensa para el sistema y moviéndose a otro estado. Los modelos aprenden por prueba y error, basado en las observaciones y la retroalimentación de las acciones pasadas. El modelo no utiliza datos predefinidos; la interacción con el ambiente por sí mismo es suficiente [Buffer *et al.*, 2020].

De esta forma, la diferencias clave radican en como los modelos utilizan los datos para aprender. Como fue explicado en el capítulo 1, los ejercicios de corrección automática presentes en SMOJ carecen de una etiqueta referente a su complejidad, por lo que el trabajo a realizar requiere el uso de un modelo presente en el área del aprendizaje no supervisado. Existen diversas técnicas al momento de trabajar los datos, las cuales son revisadas a continuación:

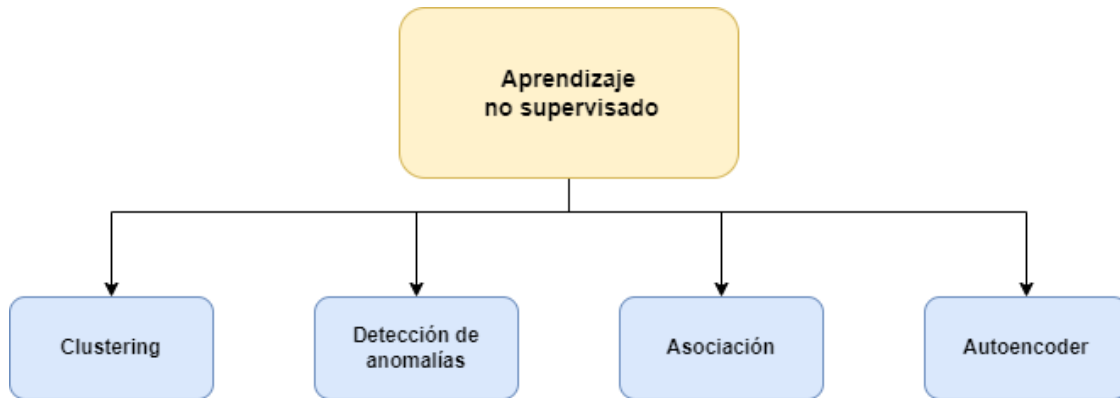


Figura 9: Tipos de aprendizajes no supervisados
Fuente: Elaboración propia, a partir de [Naeem *et al.*, 2023]

- **Clustering:** También llamado análisis por grupos o agrupamiento, corresponde a la práctica de clasificar elementos en grupos a los que pertenecen. Los datos son particionados con el objetivo de que cada uno sea contenido en un solo grupo.
- **Detección de anomalías:** Es el procedimiento de detectar datos anómalos en el conjunto de datos. Son identificados o predichos mediante la búsqueda de elementos que difieran significativamente del modelo estándar.
- **Asociación:** Permite detectar asociaciones no descubiertas en el conjunto de datos. Este método está interesado en revelar los posibles vínculos entre las variables presentes.
- **Autoencoder:** Corresponde a un enfoque que utiliza redes neuronales, cuya arquitectura presenta un cuello de botella que obliga a la red a utilizar el conocimiento de forma comprimida. Si los datos presentan cierta estructura, esta puede ser aprendida y utilizada para manejar la entrada en la red.

El método que permite clasificar a los ejercicios según su complejidad, o en otras palabras, encontrar grupos de complejidad (sencillo, intermedio, complejo), resulta ser el *clustering*, por lo que se profundizará en cómo funciona y que técnicas existen para elaborar los grupos.

2.5.1. Clustering

Como fue introducido previamente, el *clustering* es el proceso de identificar grupos de forma natural, en base a una métrica de similitud (por ejemplo, la distancia euclidiana). Los algoritmos de agrupamiento son utilizados en diversas áreas, tales como la segmentación de imágenes, minado de datos, compresión de datos, entre otros. Un grupo es generalmente identificado mediante el centro del grupo (o centroide) [Omran *et al.*, 2007].

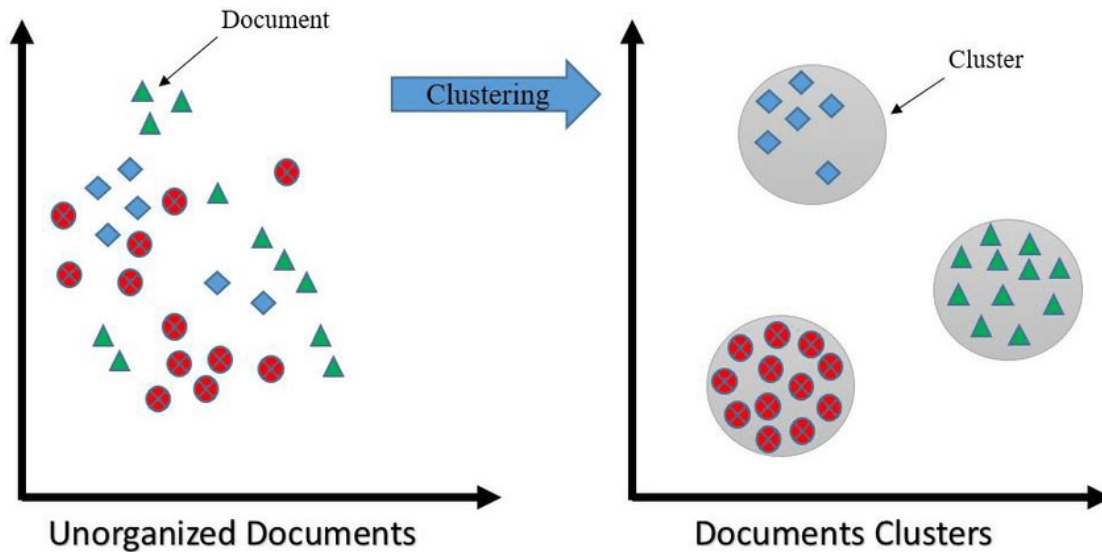


Figura 10: Ejemplo de la aplicación de un algoritmo de *clustering* en 3 grupos
Fuente: [Abualigah et al., 2021]

Algunas definiciones útiles se presentan a continuación:

- Patrón: También llamado vector de características, corresponde a un elemento individual utilizado por el algoritmo de *clustering*.
- Característica: Representa un componente individual de un patrón.
- Grupo: Corresponde a un conjunto de patrones similares, donde los patrones de diferentes grupos no son iguales.
- Algoritmos duros: Asignan un patrón a un, y sólo un grupo.
- Algoritmos difusos: Asignan cada patrón a un grupo con algún grado de afiliación.
- Métrica de distancia: Corresponde a una métrica que permite evaluar la similitud entre patrones.

La mayoría de algoritmos de *clustering* están basados en dos técnicas conocidas como **clustering jerárquico** y **clustering particional** [Omran et al., 2007].

El *clustering* jerárquico buscan establecer una jerarquía entre los diferentes grupos, mediante la combinación o división de estos. El algoritmo que combina los grupos es llamado aglomerativo, donde cada elemento representa inicialmente un grupo, e iterativamente van siendo fusionados hasta que la estructura deseada es completada. Por otra parte, el algoritmo que divide los grupos es denominado divisivo, donde se inicia con un único grupo que contiene todos los datos, y, recursivamente, los va dividiendo hasta lograr el número de grupos deseado.

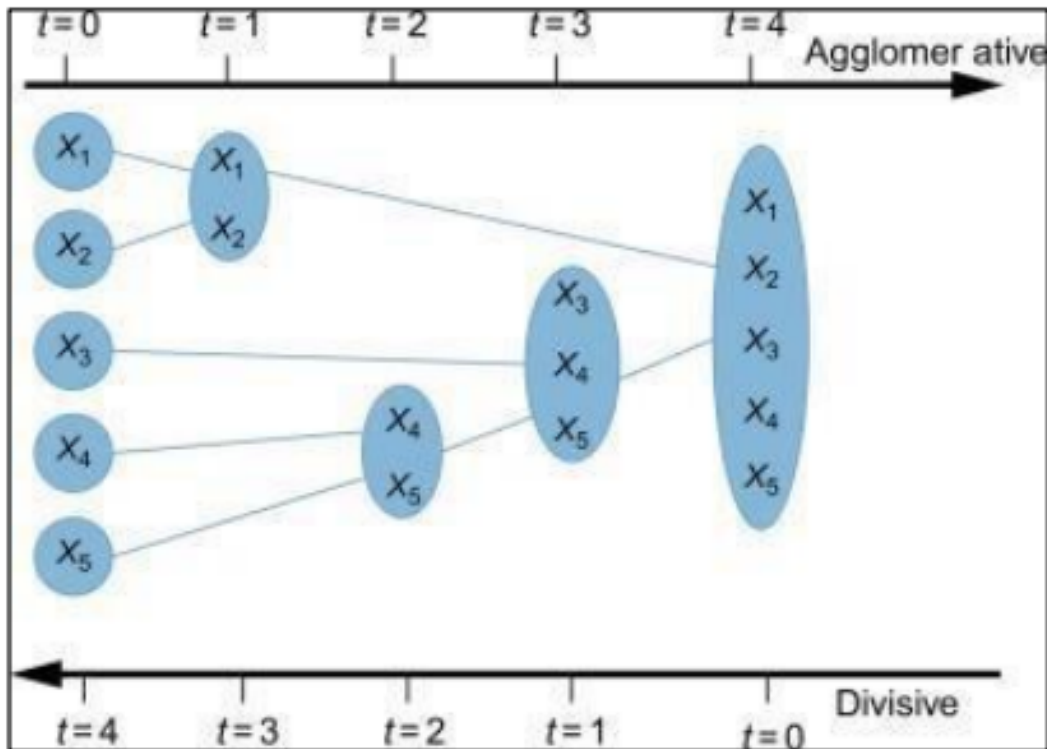


Figura 11: Ejemplo de *clustering* jerárquico. De izquierda a derecha se representa el aglomerativo, mientras que de derecha a izquierda se muestra el divisivo
Fuente: [Shetty y Singh, 2021]

El *clustering* particional, por otro lado, divide los datos en un número específico de grupos, con el objetivo de minimizar cierto criterio. Los elementos agrupados poseen características similares, mientras que los elementos de diferentes grupos poseen características diferentes [Gandhi y Srivastava, 2014].

Un algoritmo en particular del *clustering* particional es utilizado en la propuesta de solución, debido a la capacidad de agrupar los ejercicios en los grupos de complejidad que se requieren. Este algoritmo se denomina **K-means**, y es descrito a continuación.

2.5.2. K-means clustering

El algoritmo denominado *K-means* divide los datos en k grupos, donde k es elegido previamente a su aplicación. La formación de los grupos es realizada a través del cálculo de los centroides, donde cada grupo posee uno. Por cada elemento del conjunto de datos, el algoritmo utiliza métricas como la distancia euclidiana para identificar el grupo al que pertenece. Los centroides son recalculados en base a los patrones asociados. El proceso se repite hasta que la convergencia es lograda.

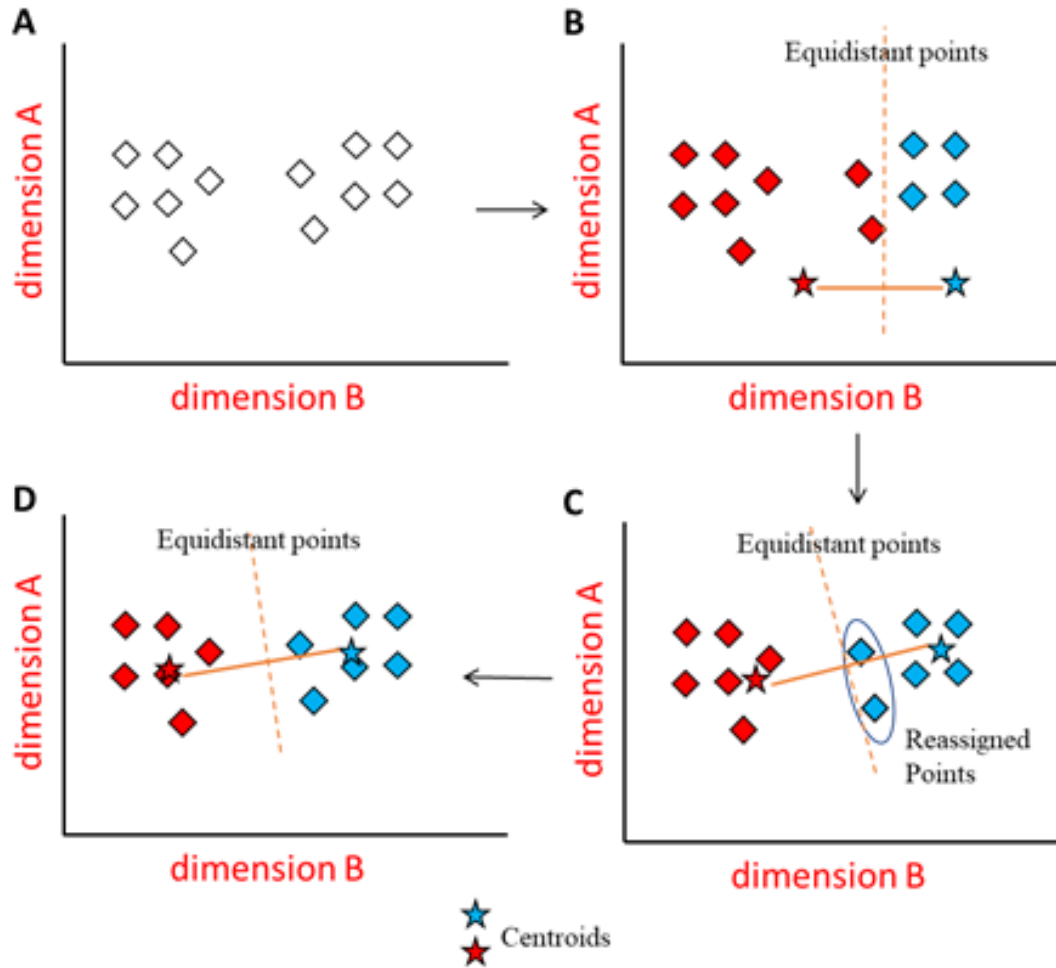


Figura 12: Aplicación de algoritmo *K-means*, donde $k = 2$. Se aprecia el cálculo de los centroides y como son asignados los elementos a su grupo
Fuente: [Muzhingi, 2020]

Las ventajas de este algoritmo radican en su facilidad de uso, escalabilidad, velocidad de convergencia y adaptabilidad a datos dispersos [Oyelade *et al.*, 2010]. Sin embargo, un gran problema radica en que se debe especificar el número de grupos a formar por adelantado. Si no se tiene noción de cuantos grupos se quieren obtener antes de aplicar el algoritmo, su uso puede resultar en soluciones no óptimas o que no se ajusten al objetivo deseado.

2.6. Stack tecnológico

La creación de una solución tecnológica involucra un conjunto de herramientas y componentes que, utilizados de manera compuesta, permiten la obtención de un resultado funcional y robusto. En el contexto de las ciencias de la computación, involucran la elección de un sistema operativo, capas de arquitectura, protocolos, entornos de ejecución, bases de datos y llamadas de funciones, entre otros [Semilof y Montgomery, 2020].

La solución propuesta, que es detallada en el capítulo 3, debe ser capaz de obtener y analizar códigos de estudiantes, que se encuentran almacenados en una base de datos de la plataforma SMOJ. Los datos deben ser explorados a cabalidad para encontrar patrones y tendencias que permitan aportar al análisis a realizar sobre la complejidad de los ejercicios. Además, debe utilizar medidas estadísticas para verificar que los resultados representan a la población total de estudiantes que resuelven los ejercicios.

Así, el lenguaje de programación escogido para ejecutar la solución es Python, un lenguaje de alto nivel y de propósito general. Las principales características que posee Python radican en su simpleza y facilidad de aprendizaje, multitud de librerías de código abierto, independencia de plataformas y portabilidad [Chaudkar, 2022]. Según los índices PYPL y TIOBE, que miden la popularidad de los lenguajes de programación en base a que tan frecuente son buscados en motores de búsqueda como Google, Python se encuentra en el primer lugar en ambos, correspondiente a un 28,59 % y 15,63 % del total de búsquedas, respectivamente.

Gracias a la gran diversidad de librerías que Python posee a su disposición, es que la labor del análisis y la ciencia de datos resulta de una ejecución sencilla en este lenguaje de programación. Tareas como el preprocesamiento y visualización de los datos, análisis estadístico y el despliegue de un modelo de aprendizaje automático pueden realizarse utilizando librerías de código abierto y uso libre.

De esta forma, las librerías escogidas para el desarrollo de la solución son las siguientes:

- **NumPy:** Considerada una de las librerías más populares de Python, ofrece una colección extensa de estructuras y objetos para el análisis de los datos, además de permitir el cálculo de complejas operaciones con fórmulas matemáticas.
- **Pandas:** Usada de forma paralela con la librería anterior, Pandas posee herramientas que permiten al programador trabajar con conjuntos de datos enormes y procesarlos para cumplir el objetivo deseado. La lectura de los datos es realizada utilizando esta librería, debido a que soporta archivos tales como CSV, tablas de bases de datos, archivos de texto plano, entre otros. Una de sus características principales corresponde a la estructura de datos denominada *Dataframe*, un objeto especial que consiste en un arreglo bidimensional indexado, que permite tratar a los objetos como una tabla [Team, 2023].

	RANK	Country	Happiness score	Whisker-high	Whisker-low	Dystopia (1.83) + residual	GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices	Generosity	Perceptions of corruption
0	1	Finland	7.821	7.886	7.756	2.518	1.892	1.258	0.775	0.736	0.109	0.534
1	2	Denmark	7.636	7.710	7.563	2.226	1.953	1.243	0.777	0.719	0.188	0.532
2	3	Iceland	7.557	7.651	7.464	2.320	1.936	1.320	0.803	0.718	0.270	0.191
3	4	Switzerland	7.512	7.586	7.437	2.153	2.026	1.226	0.822	0.677	0.147	0.461
4	5	Netherlands	7.415	7.471	7.359	2.137	1.945	1.206	0.787	0.651	0.271	0.419
...	...	...	...	...	...	...	...	...	...	...	...	...
141	142	Botswana*	3.471	3.667	3.275	0.187	1.503	0.815	0.280	0.571	0.012	0.102
142	143	Rwanda*	3.268	3.462	3.074	0.536	0.785	0.133	0.462	0.621	0.187	0.544
143	144	Zimbabwe	2.995	3.110	2.880	0.548	0.947	0.690	0.270	0.329	0.106	0.105
144	145	Lebanon	2.955	3.049	2.862	0.216	1.392	0.498	0.631	0.103	0.082	0.034
145	146	Afghanistan	2.404	2.469	2.339	1.263	0.758	0.000	0.289	0.000	0.089	0.005

146 rows x 12 columns

Figura 13: Ejemplo de un *Dataframe* de la librería Pandas, donde se recopilan datos sobre la felicidad de los países
Fuente: [Chaudekar, 2022]

- **Matplotlib:** Después de procesar los datos, estos deben ser presentados de una forma en que sean fáciles de entender. Esta librería, considerada la más popular en relación a la visualización de datos, permite la creación de gráficos de alta calidad, con funcionalidades diversas en la personalización de estos.
- **scikit-learn:** La creación de un modelo de aprendizaje automático es realizada a través de esta librería. Construida sobre las mencionadas anteriormente, permite la creación de modelos de regresión, clasificación, *clustering*, entre otros. Se escogió en función de su facilidad de uso y velocidad de aplicación, además de destacar en funciones generales del aprendizaje automático. Otras librerías cumplen roles más específicos en esta área, como lo son TensorFlow (*deep learning*), PyTorch (tareas computacionalmente complejas) y Keras (redes neuronales).

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

3.1. Antecedentes

Previo a desarrollar el análisis de la complejidad de los ejercicios de programación en la plataforma SMOJ, se realizaron dos entrevistas: la primera hecha a estudiantes de la universidad y la segunda hecha a profesores de la asignatura IWI-131.

3.1.1. Entrevista sobre ejercicios a estudiantes

La multitud de elementos extraíbles de los problemas, tanto en su enunciado como en su código, ofrecen un abanico amplio de posibilidades al momento de realizar el análisis de complejidad. La utilidad concreta de realizar una entrevista sobre aquellos componentes que los estudiantes perciben como más complejos permite dilucidar y centrar el estudio y análisis hecho, y ligarlo fuertemente a las necesidades que los alumnos poseen, permitiendo, efectivamente, aumentar su interés por la asignatura y la realización de una mayor cantidad de ejercicios en la plataforma SMOJ.

La creación de la entrevista se realizó considerando que los estudiantes debían tener alguna experiencia previa total con SMOJ por al menos algún semestre, es decir, estudiantes que habían aprobado la asignatura o aquellos que la habían reprobado y la estaban cursando por segunda o más veces. Por facilidad de contacto y de realización, se optó por la segunda opción, aprovechando que la universidad, al finalizar el segundo semestre del año 2023, inauguró el Programa Académico Especial (por sus siglas, PAE), que permite impartir asignaturas en el periodo de vacaciones (entre el fin del segundo semestre del 2023 y el inicio del primer semestre del 2024) para alumnos que hayan reprobado la asignatura previamente.

De acuerdo con lo anterior, la entrevista se formuló en base a preguntas cortas, además de consultar la carrera del estudiante y las veces que ha tomado el ramo durante su estadía en la universidad. Las preguntas en cuestión son las siguientes.

- Cuando tienes que realizar los problemas de SMOJ para el trabajo asincrónico de la asignatura, ¿Cuántos realizas y finalizas de manera correcta?
- Al momento de seleccionar un ejercicio para su realización, ¿Cuál escoges y bajo qué criterios?
- Si un ejercicio escogido resulta demasiado difícil, ¿Qué factores añaden dificultad al problema?

La primera pregunta permite cuantificar el uso de SMOJ por parte de los estudiantes, tomando en consideración que deben realizar, de manera mínima, uno de los ejercicios disponibles correctamente por cada UVA para obtener la calificación máxima en el trabajo asíncrono de la asignatura.

La segunda pregunta permite entender qué criterios poseen los estudiantes para realizar un ejercicio, partiendo de la base que el orden en el cual son visualizados por los estudiantes es aleatorio. De esta forma, se puede extraer aquellos elementos que resultan más atractivos de un ejercicio para realizarlo correctamente.

La tercera y última pregunta proporciona aquellos elementos que los estudiantes consideran más difíciles cuando realizan un ejercicio e, incluso, aquellos que producen que el estudiante abandone el problema y lo cambie por otro. Así, el análisis realizado los utilizará para la medición de la complejidad teórica, en base a la correlación existente entre dificultad y complejidad.

La entrevista fue realizada a 17 de 19 estudiantes matriculados en la asignatura de programación durante el PAE, correspondiente al Campus Casa Central Valparaíso. De los 17 estudiantes, 16 habían reprobado el ramo una vez, por lo que el programa académico correspondía a su segunda experiencia con la asignatura, y solo 1 lo había reprobado dos veces previamente. Las carreras de estudio de los entrevistados se pueden apreciar en el siguiente gráfico:

Carreras de los estudiantes entrevistados

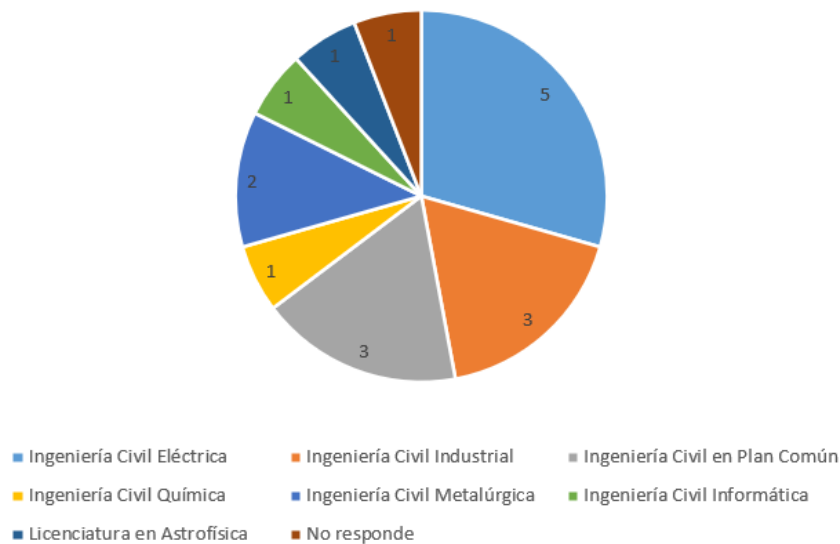


Figura 14: Gráfico de torta que cuantifica la cantidad de estudiantes por carrera que participaron en la entrevista.

Fuente: Elaboración propia.

Los resultados de la primera pregunta se manifiestan en una cantidad reducida de ejercicios resueltos por parte de los estudiantes, donde la mayoría (16), realiza menos de la mitad de ejercicios disponibles en cada laboratorio de la asignatura, siendo las respuestas más populares el realizar 1, 2 o 3 ejercicios. Considerando que, para el semestre 2023-2, la cantidad promedio de ejercicios disponibles por cada UVA es de 5,875, resulta evidente la pequeña cuantía de problemas que los estudiantes se dedican a resolver de forma correcta, desperdiciando material de estudio y de ejercitación considerable, que los podría preparar de mejor forma para su desempeño en la asignatura.

Los resultados de la segunda pregunta, cuya naturaleza es de respuesta cualitativa, permiten establecer los criterios que poseen los estudiantes al momento de realizar un ejercicio de SMOJ. Como fue apreciado en la pregunta anterior, los estudiantes no resuelven una gran parte de los problemas que son ofrecidos para su trabajo posterior, por lo que determinar cuáles son resulta en un elemento clave para el diseño del modelo de la complejidad de los ejercicios. Así, los estudiantes presentaron dos criterios básicos que les permite discernir el ejercicio que van a resolver: el enunciado y el código. Respecto al primero, mencionan que la facilidad de lectura y su extensión resultan en la norma principal a seguir para elegir un ejercicio; para el segundo, la cantidad de procesamiento de datos y una poca cantidad de líneas de código aseguran la resolución del problema.

Los resultados de la tercera y última pregunta, que también presenta un carácter cualitativo, resume las dificultades encontradas en el proceso de resolución de los ejercicios de SMOJ. De igual forma que en la pregunta anterior, se identifican los mismos factores clave: el enunciado y el código del ejercicio, referido a la cantidad de procesamiento de los datos a realizar. Además, una minoría de estudiantes afirmaron que visualizar el código, es decir, el procedimiento de transformar lo que el enunciado está pidiendo a un código funcional que resuelva el problema, resulta en la tarea más difícil a afrontar en el trabajo posterior de la asignatura.

De esta forma, podemos englobar las afirmaciones de los estudiantes en dos aspectos clave de los ejercicios: **el enunciado y el procesamiento realizado a los datos**. Así, podemos confirmar que el análisis efectuado en el capítulo 3 apunta efectivamente a las complejidades percibidas por parte de los estudiantes al momento de realizar un ejercicio de corrección automática. Las métricas asociadas al enunciado corresponde a **la extensión** de este, mientras que las métricas relacionadas al procesamiento de los datos corresponden a la **complejidad ciclomática** y la **cantidad de líneas de código**.

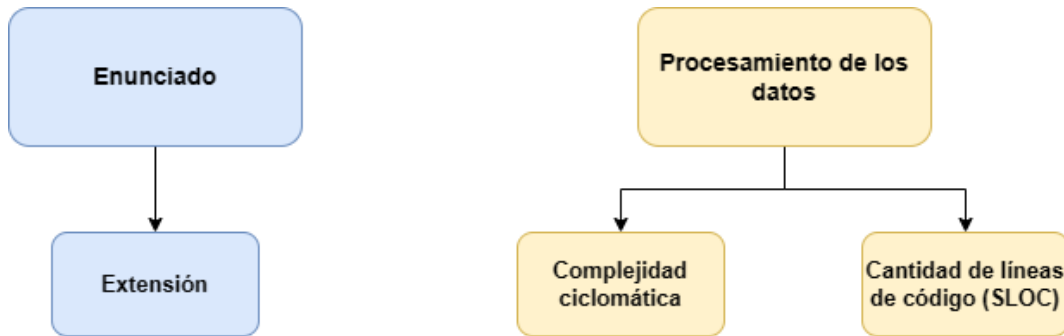


Figura 15: Métricas claves en la percepción de la complejidad de los ejercicios por parte de los estudiantes.

Fuente: Elaboración propia.

3.1.2. Entrevista a profesores sobre conceptos clave

Las métricas obtenidas en la entrevista realizada a los estudiantes de la asignatura de programación fueron estudiadas en el capítulo 2 del presente texto. Sin embargo, una métrica que no fue extraída directamente corresponde a la cantidad de conceptos presentes en los ejercicios de SMOJ. De esta forma, se procedió a desarrollar una entrevista con un profesor en la materia, con el objetivo de conocer los conceptos clave que se enseñan en programación y que son medidos en las diferentes instancias de evaluación, para extraerlos del código de los estudiantes y realizar el análisis correspondiente.

La entrevista se inició con la explicación del trabajo de memoria, abarcando los tópicos centrales en que gira, con el objetivo de contextualizar al profesor y que esté en línea con lo que se espera de él. Así, se definió el trabajo a realizar con los códigos de los estudiantes, las métricas utilizadas y el proceso de análisis de los ejercicios. Luego, se procedió a revisar por cada UVA de la asignatura los conceptos clave que se enseñan, resultando ser los siguientes:

- **Programas secuenciales con Python:** Asignación de variables, operaciones unarias, operaciones binarias, importación de librerías.
- **Condicionales:** Operaciones *booleanas*, operadores de comparación, estructuras condicionales simples y estructuras condicionales anidadas.
- **Ciclos:** Ciclo *while*, ciclo *for*, variables de aumento.
- **Strings:** Acceso a índices, operaciones de forma algorítmica.
- **Funciones:** Definición de funciones, argumentos, retorno.
- **Listas:** Uso de listas, acceso a elementos, indexación, listas multidimensionales.
- **Diccionarios:** Creación de diccionarios, uso de llave-valor.

3.2. Análisis de la complejidad de los ejercicios

La plataforma SMOJ posee una base de datos consistente de los envíos realizados de todos los ejercicios que han formado parte de un laboratorio, que corresponde a la instancia de evaluación del trabajo posterior de la asignatura y que se ha realizado desde el primer semestre del año 2020. La coordinación de programación es quien administra esta base de datos, por lo que se contactó con esta para hacer uso de ella y poder obtener el código de respuesta de todos los ejercicios hechos por los estudiantes. La decisión de utilizar el código de los alumnos para analizar su complejidad radica en la ausencia de un código oficial de respuesta al ejercicio, y en la gran cantidad disponible de envíos almacenados en la base de datos de SMOJ.

La base de datos consiste en un tabla llamada **submissions**, cuya estructura se revisa a continuación:

submissions	
PK	<u>id_submission</u>
	id_problem
	subproblem
	id_lab
	id_user
	id_language
	source_code
	score
	qualifiable
	verdict

Figura 16: Tabla *submissions* donde se almacenan los envíos de los estudiantes en SMOJ.

Fuente: Elaboración propia

Los atributos de relevancia para el trabajo son los siguientes:

- **id_problem**: Es el identificador único del problema, siendo de tipo entero.
- **subproblem**: Los ejercicios de SMOJ tienen la capacidad de soportar 1 o más subproblemas para que el estudiante los realice, es decir, para el mismo enunciado, tendrá tareas diferentes que realizar. Este campo indica si corresponde al problema principal, señalado con un 1, o a los siguientes subproblemas, indicado con un número en orden creciente.

- **id_language**: SMOJ admite problemas en tres lenguajes de programación diferentes: Python, C++ y Java. Nuestro interés es obtener aquellos ejercicios realizados en Python, cuyo identificador único en la tabla es el 71.
- **source_code**: Corresponde al código de respuesta enviado por el estudiante. Será el centro de atención de la investigación.
- **score**: Indica el puntaje, en una escala del 0 al 100, que obtuvo el código del estudiante en el ejercicio. Resulta de interés al analizar la complejidad obtener solamente los ejercicios que hayan sido solucionados correctamente.
- **qualifiable**: Los laboratorios se habilitan en plazos establecidos dentro del calendario de la asignatura. Este atributo indica si el ejercicio ha sido enviado en un plazo habilitado (señalado con un 1) o si no lo ha sido (señalado con un 0). Para efectos de la investigación, se centrará en obtener aquellos ejercicios que posean este campo en 1.
- **verdict**: Indica el veredicto que el juez en línea dictaminó para el envío del ejercicio. Se recopilarán solamente aquellos códigos cuyos envíos hayan sido aceptados, lo cual es representado por un valor de 3 en este campo.

Para el caso de los enunciados de los ejercicios, no se pudo obtener con la coordinación de programación alguna base de datos en su posesión que contuviera la información relevante a la estructura del problema, por lo que se tuvo que crear de forma manual la base de datos referente a los enunciados.

Se creó una tabla, denominada **exercise**, cuya estructura se aprecia a continuación:

exercise	
PK	<u>id</u>
	title
	statement
	instructions
	input
	output
	observations
	smoj_code
	uva

Figura 17: Tabla *exercise* donde se almacenan los datos del enunciado de los ejercicios

Fuente: Elaboración propia

Los campos que posee hacen referencia a los componentes descritos en el capítulo 2.3, además de otros dos de relevancia:

- `smoj_code`: Corresponde al identificador único que posee el problema en SMOJ, relacionado directamente con el campo `id_problem` en la tabla *submissions*.
- `uva`: Indica a que UVA pertenece el ejercicio, cuyos valores van desde el 2 al 9.

Con las dos bases de datos disponibles para las consultas necesarias, el siguiente trabajo a realizar corresponde al cálculo de las métricas de complejidad a analizar, tanto en el código como en el enunciado.

Para el caso del código de los estudiantes, las métricas a calcular serán las siguientes, en base a los resultados de la entrevista a los estudiantes y a lo utilizado en la literatura:

- **Cantidad de líneas de código (SLOC)**
- **Complejidad ciclomática**
- **Cantidad de conceptos**

Para el cálculo de las primeras dos métricas, se hizo uso de una librería de Python denominada **Radon**, la cual permite computar una variedad de métricas del código. Para la cantidad de conceptos, se hizo uso de una librería de Python llamada **ast**, la cual posibilita el procesamiento del árbol de sintaxis abstracta para un código en específico, permitiendo la extracción de los conceptos relevantes obtenidos en la entrevista con los profesores de la asignatura.

En el caso del enunciado de los problemas, se procedió a consultar a la tabla *exercise* por los componentes de la estructura de los ejercicios, y sumar la cantidad total de palabras para obtener la extensión del ejercicio.

El procedimiento realizado, para cada UVA, inicia con un análisis correspondiente a los estudiantes que rindieron los laboratorios pertenecientes al semestre académico 2023-2, con el objetivo de visualizar el porcentaje de estudiantes que intenta y completa cada ejercicio del laboratorio en cuestión. La plataforma SMOJ posee una herramienta que permite obtener los máximos puntajes que los estudiantes alcanzaron en cada ejercicio, y exportar dichos resultados en un archivo de valores separados por comas (o CSV, por su sigla en inglés, *comma-separated values*) que permitirá extraer los datos.

Scoreboard 2023-2 UVA 2: Programas secuenciales

Campus / Sede	Todos	Paralelo	Todos	Filtrar	Descargar			
Estudiante	Paralelo	#21	#71	#93	#95	#124	#125	#127
		Tarea 1	Tarea 1	Tarea 1	Tarea 1	Tarea 1	Tarea 1	Tarea 1
NOVOA FUENTEALBA VICTOR MARCELO ALFONSO	301	100 1	100 1	100 1	100 1	100 1	100 1	100 1
VERGARA CONTRERAS BENJAMIN PATRICIO	1	100 2	100 1	100 1	100 1	100 1	100 1	100 1
MONARDES GONZALEZ BENJAMIN ESTEBAN	11	100 1	100 2	100 1	100 1	100 3	100 1	100 1
LUNA MOYANO ALEXANDER SEBASTIAN	7	100 3	100 2	100 1	100 2	100 1	100 1	100 2
SOTO SAAVEDRA TOMAS HERNANDO	212	100 3	100 1	100 1	100 1	100 1	100 1	100 1
GALINDO ACUÑA ISMAEL ABDON	301	100 2	100 1	100 1	100 1	100 1	100 1	100 1
MORAGA GONZALEZ MARTIN DANIEL	11	100 4	100 1	100 1	100 1	100 1	100 1	100 1

Figura 18: Vista de SMOJ donde se puede descargar el archivo CSV de los puntajes de los estudiantes para la UVA 2

Fuente: SMOJ

Para el manejo de archivos CSV, se utilizó la librería **Pandas** de Python, especializada en el análisis y manipulación de datos. Esta librería pone a disposición una estructura denominada **Dataframe**, cuya principal característica es poder manejar datos heterogéneos en una tabla bidimensional, consistente de filas y columnas, la cual resulta sencilla de recorrer y utilizar, y posibilita operar sobre los datos con métodos propios de la librería, como el cálculo de la media aritmética, la mediana o la distribución seguida, por ejemplo.

De esta forma, podemos obtener el porcentaje de estudiantes que ha intentado un ejercicio y no lo ha resuelto correctamente, y el porcentaje de estudiantes que lo ha completado. Cabe destacar que se realiza una suposición fuerte en el primer caso, puesto que se asume que aquellos estudiantes que poseen un 0 como puntaje no han intentado el ejercicio, aún cuando podrían haberlo hecho y no tener ningún punto en el problema. Así, el cálculo se realiza considerando a la totalidad de estudiantes que se han inscrito en el laboratorio, y restándole los que posean un 0 en el ejercicio en particular. Para la obtención del porcentaje de estudiantes que han completado el problema, se divide la cantidad de alumnos que consiguieron un puntaje de 100 con la cantidad total de estudiantes inscritos. Los resultados se plasman en un gráfico de barras, donde es posible comparar, visualmente, la diferencia entre estudiantes que han intentado el ejercicio versus los que lo han completado. Un ejemplo del gráfico, obtenido de la UVA 2, se muestra a continuación:

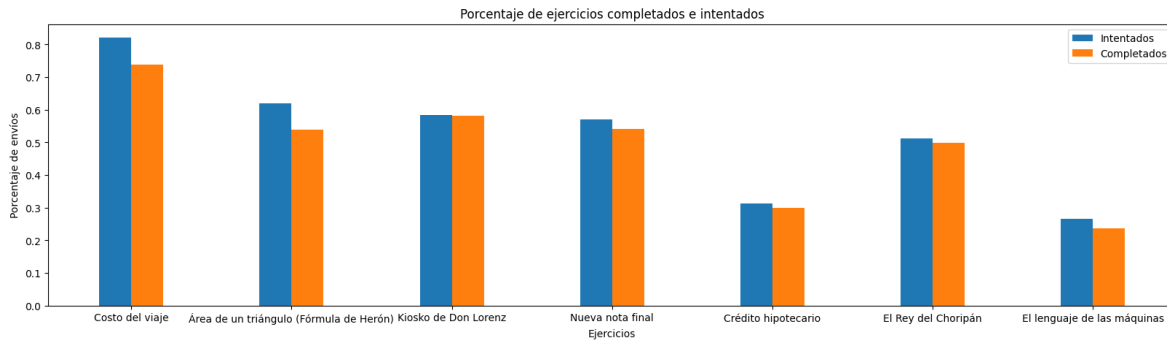


Figura 19: Gráfico de barras que muestra el porcentaje ejercicios que han sido completados e intentados para la UVA 2
Fuente: Elaboración propia

Posterior a la visualización, comienza el análisis de las métricas de complejidad del código de los estudiantes. Para esto, se realiza una consulta a la tabla *submissions* mediante un diccionario, donde la llave corresponde al nombre del ejercicio y su valor al identificador único del problema (*id_problem*), recuperando el código fuente del envío, considerando siempre que debe poseer un puntaje igual a 100, estar escrito en Python y ser calificable. De esta forma, obtenemos todos los códigos enviados para cada ejercicio perteneciente a la UVA. Después, se crea un *Dataframe* consistente en el nombre del ejercicio y las métricas propuestas para el código, con el objetivo de acceder y manipularlas de forma sencilla.

	Nombre del ejercicio	SLOC	Complejidad ciclomática
0	Alfombra	30	3
1	Alfombra	17	3
2	Alfombra	20	4
3	Alfombra	24	5

Tabla 3: Estructura del *Dataframe* creado para almacenar los códigos de los estudiantes. Cada fila representa el cálculo de las métricas de un envío individual
Fuente: Elaboración propia

Para la cantidad de líneas de código, la librería Radon posee una función denominada *analyze*, que permite, como su nombre lo indica, analizar el código y realizar el cálculo de diferentes métricas crudas de este, tales como la mencionada al inicio, los comentarios o líneas en blanco. Utilizada la función, se accede al componente SLOC que retorna y se añade al *Dataframe*.

La complejidad ciclomática requiere un paso previo antes de su cálculo, debido a que la librería Radon la calcula solamente si el código se encuentra inserto dentro de una función o una clase. Naturalmente, no todos los ejercicios utilizan funciones, y el uso de clases es un contenido que no se enseña en la asignatura. Para solucionar esto, a todos los códigos obtenidos mediante la consulta hecha a la tabla *submissions* se le agregó una línea de código en su encabezado consistente en la declaración **def: func()**, que permite la inclusión en una

función, además de añadir las indentaciones correspondientes a esta adición. Esto se realiza solamente para el cálculo de esta métrica y no afecta a la métrica de cantidad de líneas de código. Hecho esto, se utiliza la función de Radon `cc_visit` que efectúa el cómputo de la complejidad ciclomática correctamente y se agrega al *Dataframe*. Las siguientes declaraciones en Python afectan el cálculo de la métrica:

Palabra clave	Efecto en el cálculo	Justificación
<i>if</i>	+1	Una instrucción <i>if</i> genera una decisión
<i>elif</i>	+1	La instrucción <i>elif</i> genera otra decisión
<i>else</i>	0	La instrucción <i>else</i> no genera una nueva decisión, debido a que esta se encuentra en el <i>if</i>
<i>while</i>	+1	Existe una decisión al inicio del ciclo <i>while</i>
<i>for</i>	+1	Existe una decisión al inicio del ciclo <i>for</i>
Operador booleano	+1	Los operadores booleanos <i>and</i> y <i>or</i> añaden un punto de decisión

Tabla 4: Declaraciones que afectan el cálculo de la complejidad ciclomática en Radon. Al cómputo final se le suma 1.

Fuente: <https://radon.readthedocs.io/en/latest/intro.html#cyclomatic-complexity>

Finalmente, para calcular la cantidad de conceptos presentes en el código, se hace uso de la librería *ast* previamente mencionada, que permite la transformación de un código en Python a su correspondiente árbol de sintaxis abstracta. La librería maneja los nodos del árbol con diferentes etiquetas, las cuales permiten la recolección de los conceptos claves obtenidos de la entrevista con los profesores de la asignatura. Para esto, se hace uso de la función *dump* del módulo, que retorna el contenido formateado con los etiquetas, el cual se recorre en busca de las que se relacionan con los conceptos. Finalmente, un diccionario es creado con el conteo de los conceptos, separados por UVA para una mejor visualización. Un ejemplo de un gráfico de barras de la cantidad de conceptos es el siguiente:

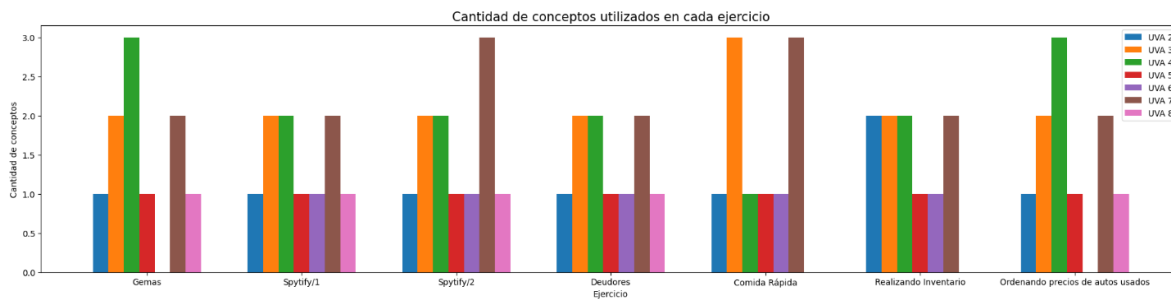


Figura 20: Gráfico de barras que muestra la cantidad de conceptos utilizados, por cada UVA, para un ejercicio de la UVA 8

Fuente: Elaboración propia

Cabe destacar que, para las tres métricas de complejidad relativas al código de los estudiantes, se almacena la **mediana** de los datos, debido a la presencia de *outliers* (estudiantes con

excesivas líneas de código), que sesgan el análisis estadístico, resultando más fiable que la media aritmética típicamente usada.

Con respecto al enunciado del problema, y como fue mencionado previamente, se utiliza la tabla *exercise* para consultar los componentes que conforman la estructura del ejercicio, almacenándolos en un *Dataframe*, cuyas columnas poseen el conteo de palabras por cada elemento del enunciado. El ejemplo proporcionado por SMOJ no es considerado, debido a la posibilidad que existe de que contenga estructuras o tipos de datos utilizados en programación, tales como listas o números, que no se consideran palabras como tal, además de ser una ayuda para los estudiantes al momento de probar su código y verificar que esté funcionando correctamente.

	Nombre del ejercicio	Enunciado	Instrucciones	Entrada	Salida	Observaciones
1	Medianas	103	26	39	19	21
2	Filtrar lista	114	54	7	7	20
3	Notas alumnos	63	46	4	4	24

Tabla 5: Estructura del *Dataframe* creado para almacenar el conteo de palabras por cada componente del ejercicio. Cada fila representa el conteo de un enunciado individual

Fuente: Elaboración propia

3.3. Análisis de la dificultad de los ejercicios

La base de datos que posee SMOJ relativa a los envíos realizados por parte de los estudiantes en los diferentes ejercicios disponibles otorga la posibilidad de efectuar un estudio de la dificultad, debido a que almacena, efectivamente, todos los envíos que un alumno ha hecho en la plataforma, aún si ya ha obtenido la calificación máxima. Debido a que las métricas de dificultad de los problemas requieren la existencia de la totalidad de intentos y éxitos para un ejercicio en particular, es posible la realización de un análisis con el objetivo de contrastar los resultados obtenidos en el apartado de la complejidad.

La obtención de los datos se realiza consultando a la tabla *submissions* los envíos de los ejercicios por cada UVA, creando un diccionario que almacene los puntajes obtenidos por cada estudiante. Como fue mencionado en el párrafo anterior, la plataforma SMOJ faculta a los estudiantes a que intenten el ejercicio cuantas veces lo deseen, sin ninguna restricción, considerando para el cálculo de su nota en el trabajo posterior la calificación máxima obtenida. No obstante, pueden existir envíos múltiples que corresponden al máximo, es decir, de 100. Para evitar que se distorsionen los datos por la presencia de múltiples 100 en los envíos de los estudiantes, el diccionario admite un solo puntaje máximo, si es que el estudiante lo posee.

Las métricas de la dificultad de los ejercicios a calcular elegidas son las tres primeras mencionadas al inicio del capítulo 2.4.2: **puntaje promedio, estudiantes aceptados y envíos acep-**

tados, obtenidas de [Tran *et al.*, 2022].

3.3.1. Puntaje promedio

El cálculo del puntaje promedio (F_0) se realiza a través de la siguiente fórmula:

$$F_0 = \frac{1}{N} \sum_{i=1}^N \left(\frac{1}{C_i} \sum_{j=1}^{C_i} \frac{c_{ij}}{C_{max}} \right)$$

Donde:

- N = Número de estudiantes que intentaron el ejercicio.
- C_i = Número de envíos realizados por el i -ésimo estudiante.
- c_{ij} = Puntaje correspondiente al j -ésimo envío del i -ésimo estudiante.
- C_{max} = Puntaje máximo del ejercicio, correspondiente a un 100 para la totalidad de problemas en SMOJ.

Los puntajes promedio por cada estudiante se almacenan en un diccionario, cuya llave corresponde al ejercicio que representa, y su valor a otro diccionario con el usuario y una lista de puntajes asociados. Esto permite la creación de gráficos de barra que permitan comparar los promedios, como se muestra a continuación:

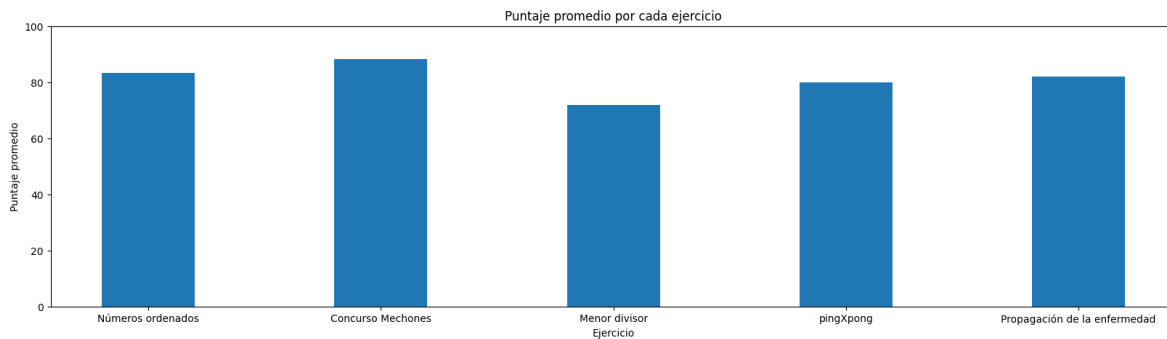


Figura 21: Gráfico de barras que representa el puntaje promedio obtenido por los estudiantes en cada ejercicio

Fuente: Elaboración propia

3.3.2. Estudiantes aceptados

El cálculo de los estudiantes aceptados por ejercicio (F_1) se realiza a través de la siguiente fórmula:

$$F_1 = \frac{S}{S_{tot}}$$

Donde:

- S = Número de estudiantes que resolvieron el problema.
- S_{tot} = Número de estudiantes que intentaron el problema.

De igual manera que la métrica anterior, los estudiantes que completaron el ejercicio son almacenados en un diccionario, cuya llave corresponde al ejercicio y su valor a lo calculado. Luego, cada valor es dividido por el número total de estudiantes que intentaron el problema. El gráfico de barras creado permite comparar este ratio, como se muestra en el siguiente ejemplo:

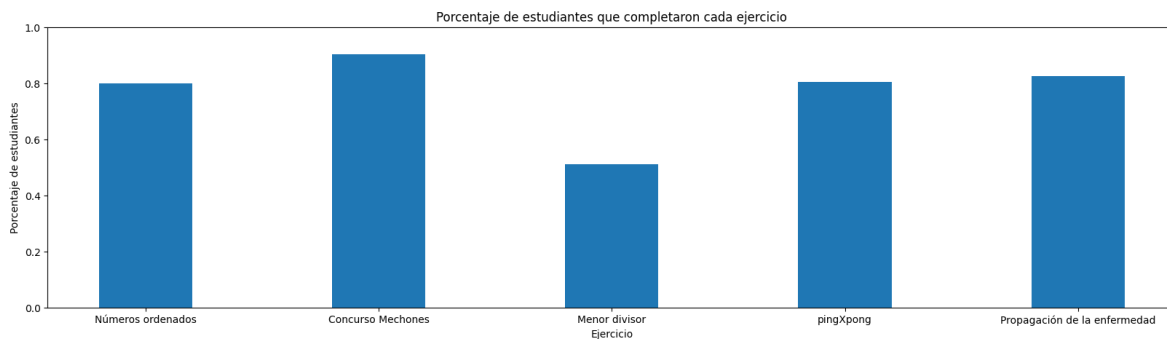


Figura 22: Gráfico de barras que representa el ratio de estudiantes que completaron un ejercicio versus aquellos que lo intentaron

Fuente: Elaboración propia

3.3.3. Envíos aceptados

El cálculo de los envíos aceptados por ejercicio (F_2) se verifica con la siguiente fórmula:

$$F_2 = \frac{U}{U_{tot}}$$

Donde:

- U = Número de envíos con puntaje máximo.
- U_{tot} = Número total de envíos realizados.

Finalmente, se crea un diccionario cuya llave representa el ejercicio y su valor es el número de envíos con puntaje máximo. Luego, para obtener la razón de envíos aceptados, se divide cada valor por el número total de envíos. El gráfico de barras que permite la comparación de este ratio en los diferentes ejercicios se visualiza a continuación:

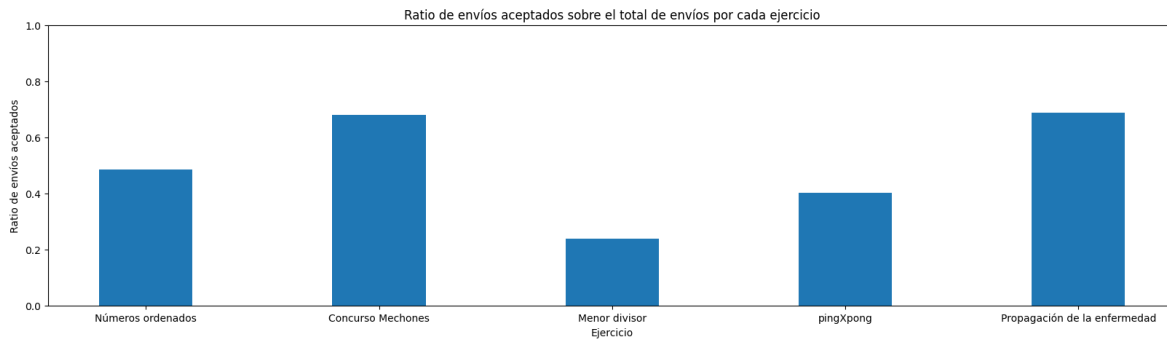


Figura 23: Gráfico de barras que representa el ratio de envíos que poseen un puntaje máximo versus el número total de envíos por ejercicio

Fuente: Elaboración propia

3.4. Clasificación de la complejidad de los ejercicios

El análisis de la complejidad de los ejercicios realizado previamente, tanto para el enunciado como para el código de resolución de los estudiantes, permite la agrupación de la totalidad de las métricas calculadas para cada problema disponible. De esta forma, es posible la aplicación del algoritmo K-means, discutido en el capítulo 2.5.2, tomando como características las medidas de la complejidad, resultando en que los patrones corresponden a las métricas particulares de cada ejercicio.

El algoritmo K-means implica la elección de un valor para k , donde se discutió en reuniones con la profesora de las dos posibilidades existentes: un valor de $k = 2$, resultando en la clasificación de ejercicios sencillos y complejos, o un valor de $k = 3$, obteniendo ejercicios etiquetados como sencillos, intermedios y complejos. Se optó por el valor de $k = 3$, debido a la amplitud que otorga a la clasificación y a la obtención de mejores resultados después de aplicar el algoritmo.

De esta forma, se procedió a la creación de un *Dataframe* que contuviera los ejercicios de cada UVA de la asignatura, con las métricas calculadas de la complejidad del código y del enunciado. Un ejemplo de *Dataframe* se aprecia a continuación:

	Nombre del ejercicio	SLOC	CC	Cantidad de conceptos	Cantidad de palabras
1	Alfombra	21	4	6	290
2	Gimnasio	19	10	5	171
3	Empresa de telefonía ClaroStar	19	5	6	182
4	Número palíndromo	18	7	6	158

Tabla 6: Ejemplo de *Dataframe* con los ejercicios y las métricas de complejidad calculadas
Fuente: Elaboración propia

Utilizando la librería scikit-learn, se aplica el algoritmo K-means al *Dataframe* presentado previamente, especificando el valor de $k = 3$, inicializando una semilla aleatoria para asegurar reproducibilidad de los resultados, y otorgando las características a utilizar para encontrar a qué grupo pertenece cada ejercicio, correspondiente a las métricas de complejidad calculadas. Finalmente, los resultados son obtenidos, siendo estos los valores 0, 1 y 2.

La interpretación de los valores es un aspecto crucial al utilizar este algoritmo de *clustering*, puesto que se le entrega solamente la cantidad de grupos a conformar, mas no el significado que posee el grupo. Por consiguiente, se realizó la asignación de los valores 0, 1 y 2 a las etiquetas definidas previamente (sencillo, intermedio, complejo). La definición de esta relación es independiente en cada UVA, puesto que, por ejemplo, el valor 0 en la segunda unidad puede representar a los ejercicios sencillos, mientras que el mismo valor en la tercera puede indicar que son ejercicios complejos. Así, se filtraron en un orden ascendente las métricas de complejidad, permitiendo que aquellos ejercicios que poseían los menores valores asignarse a los ejercicios sencillos, y así hasta los complejos.

3.4.1. Resultados

Los resultados obtenidos de la clasificación de la complejidad de los ejercicios se detallarán, por cada UVA, mediante una lista en orden ascendente de complejidad, es decir, los ejercicios del principio poseen una menor complejidad que aquellos que están en el final. Además, se precisará a que grupo (sencillo, intermedio o complejo) pertenece cada problema.

Los resultados para la UVA 2 son los siguientes:

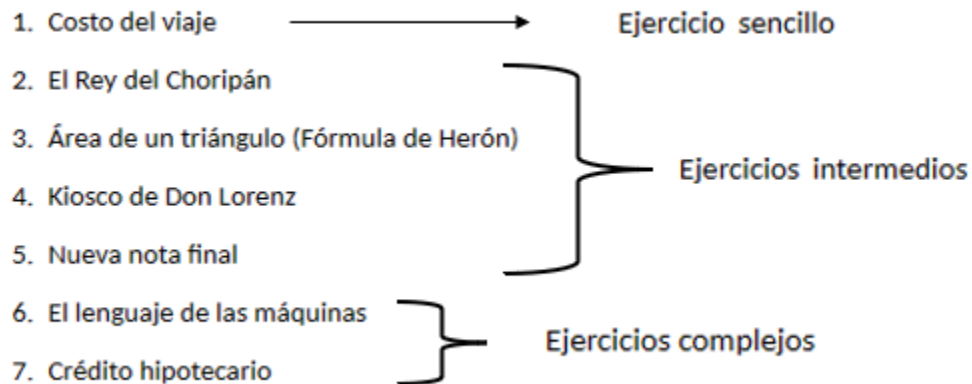


Figura 24: Ejercicios de la UVA 2 ordenados según su complejidad
Fuente: Elaboración propia

Los resultados para la UVA 3 son los siguientes:

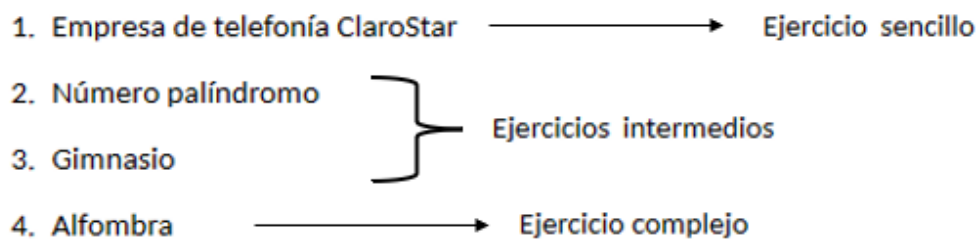


Figura 25: Ejercicios de la UVA 3 ordenados según su complejidad
Fuente: Elaboración propia

Los resultados para la UVA 4 son los siguientes:

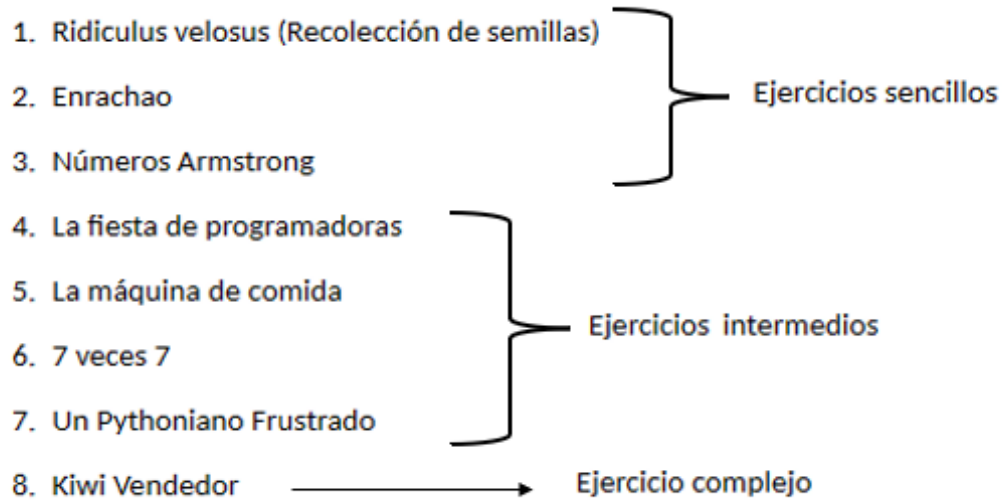


Figura 26: Ejercicios de la UVA 4 ordenados según su complejidad
Fuente: Elaboración propia

Los resultados para la UVA 5 son los siguientes:

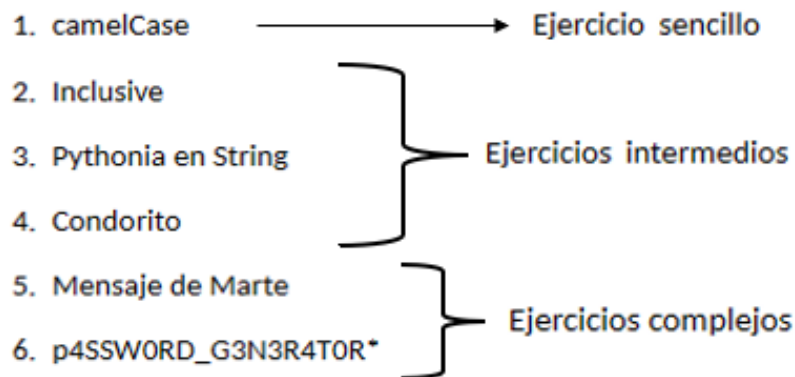


Figura 27: Ejercicios de la UVA 5 ordenados según su complejidad
Fuente: Elaboración propia

Los resultados para la UVA 6 son los siguientes:

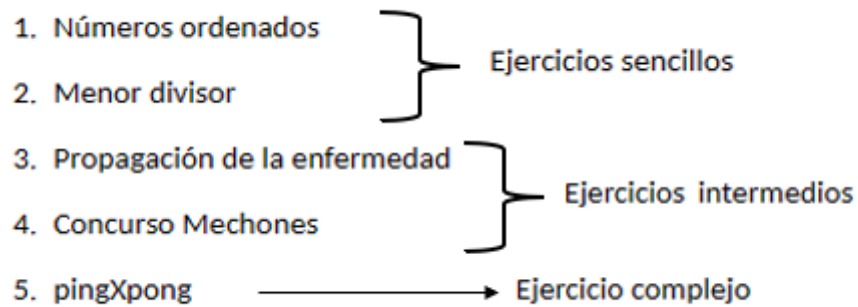


Figura 28: Ejercicios de la UVA 6 ordenados según su complejidad
Fuente: Elaboración propia

Los resultados para la UVA 7 son los siguientes:

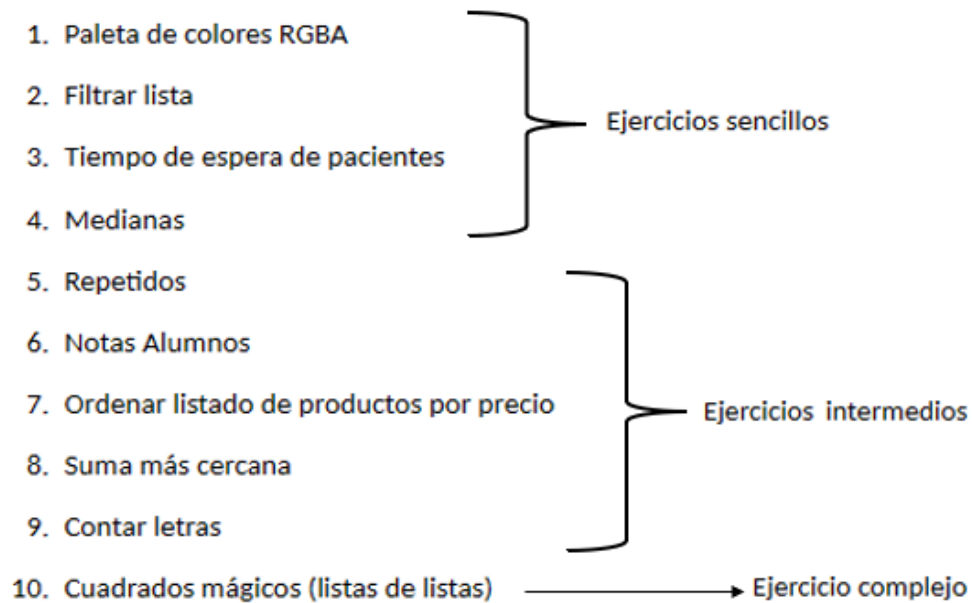


Figura 29: Ejercicios de la UVA 7 ordenados según su complejidad
Fuente: Elaboración propia

Los resultados para la UVA 8 son los siguientes:

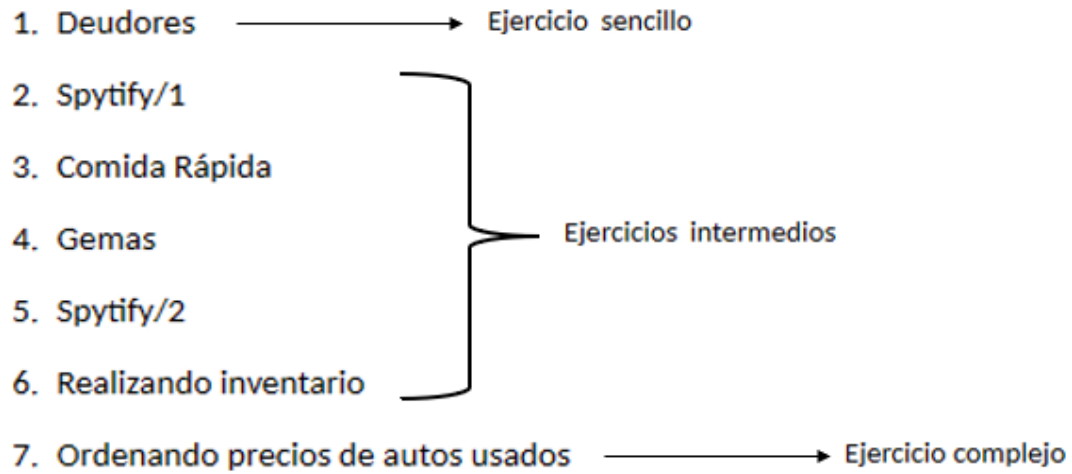


Figura 30: Ejercicios de la UVA 8 ordenados según su complejidad

Fuente: Elaboración propia

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

El proceso de validación del trabajo de memoria realizado, siendo este del carácter de una propuesta de modelo y metodología a sugerir a la asignatura de programación, involucra la demostración de que ambas ideas permitan el cumplimiento del objetivo general definido en el capítulo 1, a saber: **aumentar las instancias de ejercitación y evaluación en un curso masivo de introducción a la programación a través del desarrollo de un modelo de clasificación automática de la complejidad de ejercicios de autocorrección en un curso masivo de programación.**

De acuerdo a esto, la principal actividad a evaluar se enmarca en el mismo trabajo posterior de la asignatura. El objetivo presente busca que los estudiantes realicen más ejercicios en la plataforma SMOJ, disminuyendo la desmotivación y frustración inherente cuando encuentran problemas que no se ajustan a su nivel de conocimiento. Así, se procedió a comparar los resultados del laboratorio de la tercera unidad de la asignatura del período 2023-2, con los resultados del mismo laboratorio pero en el semestre 2024-1, aplicado a un paralelo en particular. La diferencia radica en que, para la actividad del año 2024, los estudiantes la realizarán siguiendo el orden de complejidad ascendente obtenido en la propuesta de solución, es decir, se les presentará primero los ejercicios sencillos de la unidad, luego los intermedios, y finalmente los complejos.

4.1. Antecedentes

El laboratorio de la tercera unidad, correspondiente al contenido de condicionales, para el semestre 2024-1, ofrece a los estudiantes 4 ejercicios para su resolución (aquellos que fueron analizados en el capítulo 3), que se muestran a continuación:

Laboratorios	2023-2 UVA 3: Condicionales	Mis envíos	LABORATORIO FINALIZADO
Te recomendamos ver el vídeo Laboratorios en SMOJ antes de intentar un laboratorio.			
Resumen			
Problema	Tarea	Puntaje obtenido	
Alfombra	1	0	
Gimnasio	1	0	
Empresa de Telefonía ClaroStar	1	0	
Número palindromo	1	0	

Figura 31: Ejercicios disponibles en la UVA 3 para el semestre 2023-2

Fuente: SMOJ

Los ejercicios disponibles se muestran en un orden arbitrario, es decir, no siguen algún criterio en el orden presentado a los estudiantes. De hecho, el orden obtenido en la propuesta de solución dista considerablemente con el presente este semestre. El ejercicio más complejo de la lista, “Alfombra”, es exhibido al inicio de la tabla, siguiéndolo un ejercicio intermedio, uno sencillo y otro intermedio. Considerando esto, se presenta a continuación el gráfico de completitud de los problemas por parte de los estudiantes:

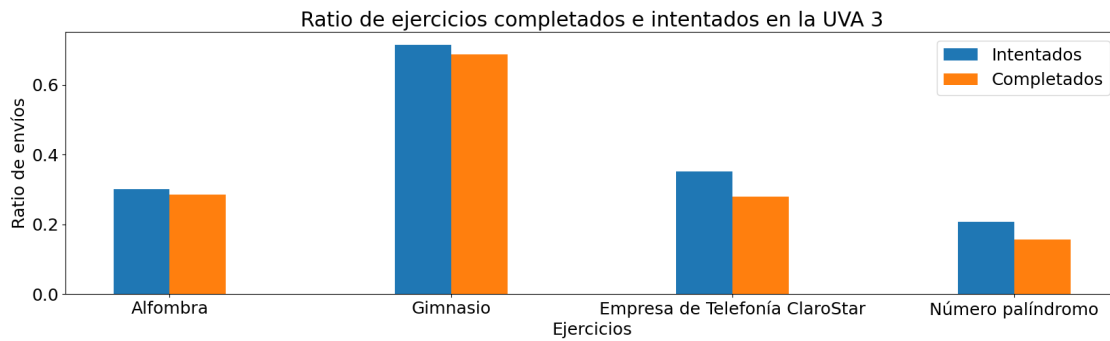


Figura 32: Gráfico del ratio de completitud versus los intentos de los estudiantes en la UVA 3, para el semestre 2023-2
Fuente: Elaboración propia

Como se puede apreciar, el primer ejercicio de la lista fue resuelto por una cantidad pequeña de estudiantes, a comparación del segundo ejercicio, poseedor de una menor complejidad. Además, como se logra observar en la mayoría de las UVAs, la mayoría de estudiantes resuelve un solo ejercicio (el más sencillo para ellos, a priori) con el objetivo de obtener la calificación máxima en el trabajo posterior, sin solucionar los que restan en la unidad. Esto explica la menor cantidad de estudiantes que resolvieron el ejercicio “Empresa de Telefonía ClaroStar”, siendo el de menor complejidad de la lista, pero realizando su aparición después del ejercicio “Gimnasio”, con una complejidad notablemente menor que el primero.

Los porcentajes de completitud, por ejercicio, permitirán efectuar una comparación con la metodología de ordenar los problemas en base a su complejidad, para verificar efectivamente que se esté aumentando la realización de ejercicios por parte de los estudiantes.

4.2. Resultados

La realización del laboratorio de la tercera unidad, en el semestre 2024-1, fue llevado a cabo a partir del día 18 de marzo, hasta el día 25 del mismo mes. Para esta actividad, la coordinación de programación decidió agregar 3 nuevos ejercicios, y eliminar 1 de los presentes en el laboratorio del semestre 2023-1. Así, la lista de problemas disponibles a los estudiantes se muestra a continuación:

Laboratorios	2024-1 UVA 3: Condi...	Mis envíos	Termina en 22h 5m 55s
Te recomendamos ver el vídeo Laboratorios en SMOJ antes de intentar un laboratorio.			
Resumen			
Problema	Tarea	Puntaje obtenido	
Triángulos	1	0	
Alfombra	1	0	
Línea de la pobreza	1	0	
Gimnasio	1	0	
Empresa de Telefonía ClaroStar	1	0	
Maniana llueve	1	0	

Figura 33: Ejercicios disponibles en la UVA 3 para el semestre 2024-1

Fuente: SMOJ

Los ejercicios introducidos tienen por título “Triángulos”, “Línea de la pobreza” y “Maniana llueve”, mientras que el ejercicio removido es “Número palíndromo”. Cabe destacar que los ejercicios que no estaban incluidos en la lista del semestre 2023-2 si han estado en otros laboratorios, esto con el motivo de poder analizar la complejidad correctamente, tomando como base la metodología utilizado en el capítulo 3. De esta forma, se recopilieron los códigos de resolución de laboratorios pasados en que los ejercicios se encontraban disponibles, con el objetivo de obtener las métricas de complejidad tanto para el enunciado como para el código. Luego, se aplicó el algoritmo K-means, tal como se explicó en el capítulo 3.4, a toda la lista de ejercicios del semestre 2024-2, obteniendo la siguiente clasificación de la complejidad:

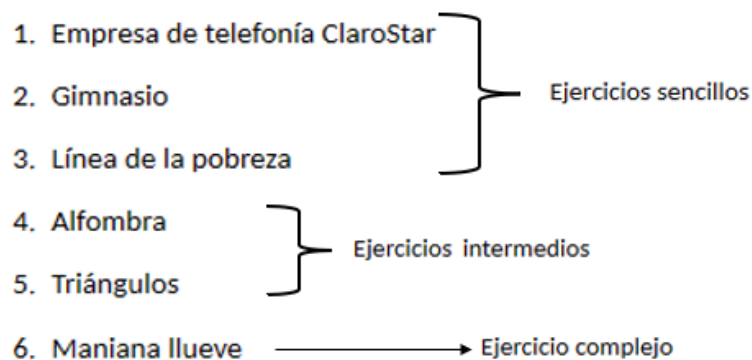


Figura 34: Ejercicios de la UVA 3, para el semestre 2024-1, ordenados según su complejidad

Fuente: Elaboración propia

Obtenida la clasificación de la complejidad para los ejercicios, se informó al paralelo a analizar sobre el orden a seguir cuando resuelvan los problemas, sin indicar la etiqueta de complejidad del ejercicio, con el objetivo de que no tengan preconceptos sobre los problemas, pudiendo alterar los resultados.

De esta forma, al momento de finalizar el laboratorio, se procedió a analizar los resultados de completitud e intentos de los ejercicios, tal como se realizó en el semestre 2023-2. El gráfico se presenta a continuación:

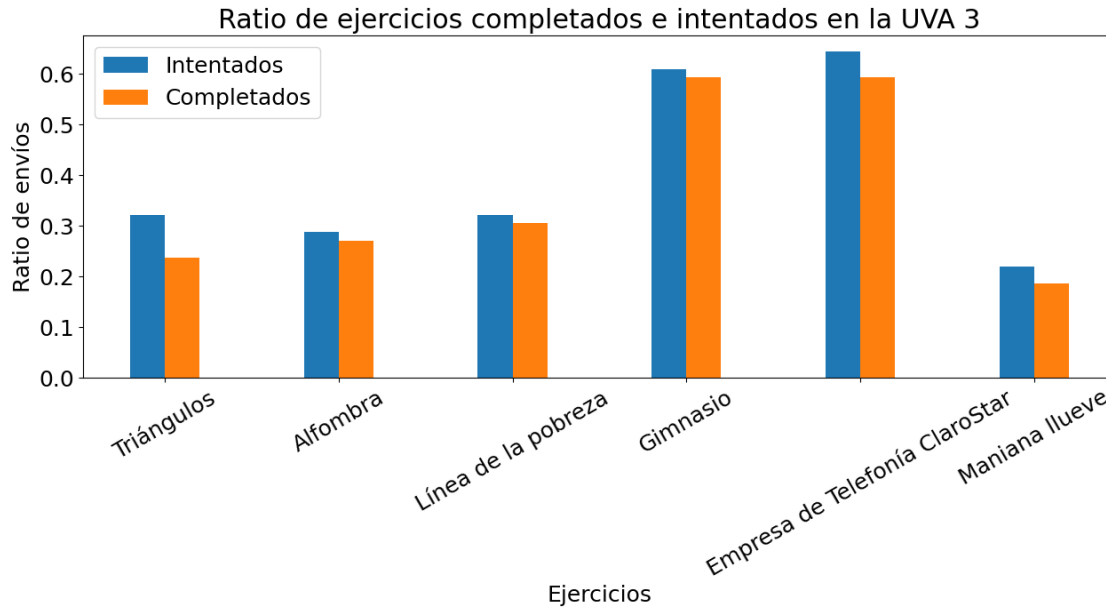


Figura 35: Gráfico del ratio de completitud versus los intentos de los estudiantes en la UVA 3, para el semestre 2024-1, en el paralelo analizado
Fuente: Elaboración propia

Analizando el gráfico, se aprecia de inmediato que el ejercicio más intentado y completado por los estudiantes, es el primero de la lista de complejidad. Más aún, es visible que el orden de los ejercicios propuesto se empareja perfectamente con los porcentajes de completitud de los problemas en este laboratorio, siguiendo un orden descendente.

Realizando la comparación con el laboratorio del semestre 2023-2, se obtienen los siguientes resultados:

Nombre del ejercicio	% completitud 2023-2	% completitud 2024-1
Triángulos	N.A	23,7 %
Alfombra	28,6 %	27,1 %
Línea de la pobreza	N.A	30,5 %
Gimnasio	68,8 %	59,3 %
Empresa de telefonía ClaroStar	28 %	59,3 %
Maniana llueve	N.A	18,6 %

Tabla 7: Tabla comparativa de los porcentajes de completitud, por ejercicio, para los semestres 2023-2 y 2024-1
Fuente: Elaboración propia

Si bien existe una diferencia porcentual de un 9,5 % en los ejercicios más resueltos, entre ambos semestres, se aprecia claramente que existe una distribución más pareja en los porcentajes de completitud del semestre 2024-1, donde fue aplicada la metodología de complejidad de los ejercicios. La media del porcentaje de completitud para el semestre 2023-2 es de $35,2\% \pm 20\%$, mientras que para el semestre 2024-2 es de $36,4\% \pm 16,4\%$, comprobándose tanto una menor dispersión de la completitud, como también un leve aumento en la completitud de los ejercicios para el semestre analizado, considerando que el total de problemas aumentó en 2.

Ahora bien, como el análisis del orden de los ejercicios según su complejidad fue aplicado a solo un paralelo de la asignatura de programación, es posible también efectuar la comparación entre este paralelo en particular con los demás, los cuales no siguieron el orden obtenido, sino que el tradicional y arbitrario. De esta forma, los resultados de la completitud e intentos de los ejercicios para aquellos paralelos que realizaron el laboratorio de forma normal se aprecian en el siguiente gráfico:

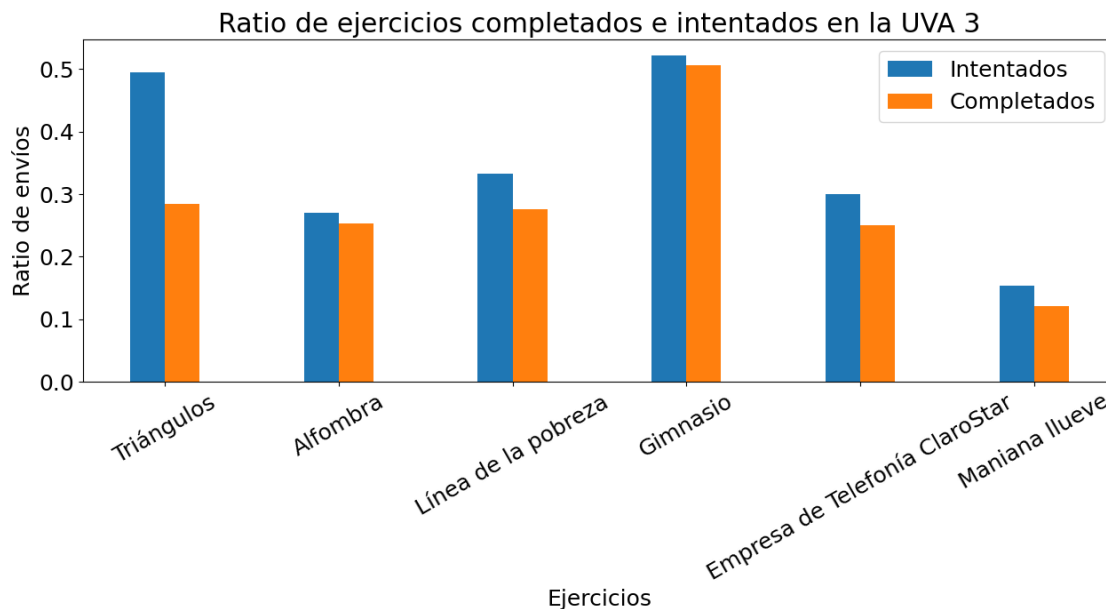


Figura 36: Gráfico del ratio de completitud versus los intentos de los estudiantes en la UVA 3, para el semestre 2024-1, contabilizando la totalidad de paralelos de la asignatura
Fuente: Elaboración propia

Posiblemente uno de los resultados más impactantes al analizar el gráfico se aprecia en el ejercicio “Triángulos”, donde la mitad de estudiantes que lo intentó lo completó satisfactoriamente. Este problema, según el análisis realizado, era el más complejo de la categoría intermedio, lo que explica tal disimilitud entre los intentos y la completitud. Más aún, ejemplifica uno de los mayores problemas encontrados por la coordinación de la asignatura, discutido en el capítulo 1, sobre los ejercicios que resultaban complicados para los estudiantes y que no se ajustaban a su nivel de conocimiento, siendo, en este caso, una barrera de entrada al laboratorio, y posible causante de frustración y desmotivación en su realización.

La comparación de resultados entre el paralelo al que fue aplicado la metodología de complejidad, versus la totalidad de paralelos de la asignatura, se muestra en la siguiente tabla, donde se añadió una columna denominada $\Delta \%$, que permite evaluar la diferencia de los porcentajes de completitud entre los diferentes casos:

Nombre del ejercicio	% completitud paralelo	% completitud totalidad	$\Delta \%$
Triángulos	23,7 %	28,4 %	-4,7 %
Alfombra	27,1 %	25,3 %	1,8 %
Línea de la pobreza	30,5 %	27,6 %	2,9 %
Gimnasio	59,3 %	50,5 %	8,8 %
Empresa de telefonía ClaroStar	59,3 %	25 %	34,3 %
Maniana llueve	18,6 %	10,1 %	8,5 %

Tabla 8: Tabla comparativa de los porcentajes de completitud de los ejercicios, para el paralelo donde se aplicó la metodología versus la totalidad de paralelos de la asignatura, en el semestre 2024-1

Fuente: Elaboración propia

Si bien el primer ejercicio fue realizado por una menor cantidad de estudiantes en el paralelo donde se aplicó la metodología de la complejidad, en los demás ejercicios existe una disparidad notoria a favor de este paralelo, donde todas las diferencias en los porcentajes son positivas. El resultado más significativo se aprecia en el ejercicio “Empresa de telefonía ClaroStar”, siendo este el ejercicio más sencillo de la lista en base al orden propuesto. Los estudiantes del paralelo analizado empezaron realizando este problema, mientras que los alumnos de los demás paralelos lo tenían disponible como quinto ejercicio del laboratorio, e, incluso, comenzando con uno de los más complejos, lo que explica los resultados analizados.

CAPÍTULO 5

CONCLUSIONES

5.1. Conclusiones generales

En este trabajo de memoria se realizó un análisis de la complejidad de ejercicios de autocorrección, enmarcados en un curso masivo de programación, con el objetivo de aumentar las instancias de ejercitación y evaluación disponibles para los estudiantes. Esto le da la posibilidad a los alumnos de realizar más ejercicios en la plataforma SMOJ, preparándolos mejor para las actividades evaluativas de la asignatura.

Para lograr tal objetivo, se estudiaron las bases de la enseñanza de la programación, permitiendo entender correctamente la elección del modelo de aula invertida por parte de la universidad al momento de impartir la asignatura, y poder conocer cada etapa que vive el estudiante cuando está siendo partícipe del curso. Junto con esto, se analizó en profundidad a los ejercicios de programación, estudiando su estructura y contexto en que se ubican, con el propósito de comprender qué es lo que los vuelve complejos y difíciles, resultando en la obtención de métricas y conceptos claves a medir en la propuesta de solución.

El análisis realizado a los códigos de los ejercicios enviados por parte de los estudiantes a la plataforma SMOJ contempló la totalidad de estos, para todos los ejercicios. Se obtuvo una observación global de los envíos realizados, desde que la plataforma está activa como actividad de aprendizaje en la asignatura. Con esto, se procedió a computar y obtener las métricas resultantes del trabajo conceptual de los ejercicios, respaldados y validados por entrevistas tanto a estudiantes como a profesores, resultando en un análisis potente tanto en lo teórico como en lo práctico.

La clasificación de la complejidad de los ejercicios se obtuvo exitosamente, para los ejercicios presentes en cada unidad de la asignatura. Cabe destacar que los problemas analizados corresponden a aquellos que han sido parte de algún laboratorio de la asignatura, debido a la existencia de ejercicios almacenados en la plataforma que no han estado disponibles a los estudiantes. Sin embargo, esto no es una limitante, debido a que es posible categorizar a los ejercicios apenas estén disponibles en un nuevo laboratorio, extrayendo los códigos de los estudiantes y aplicando la metodología descrita en el capítulo 3.

Si bien la validación de la solución fue aplicado a sola una unidad de la asignatura, los resultados son prometedores, y se espera que la coordinación de la asignatura aplique la metodología detallada en la propuesta de solución a todos los laboratorios que se realicen en el futuro, debido al gran aumento de ejercicios que los estudiantes realizaron. También se espera que este incremento resulte beneficioso para los alumnos, tanto en el desempeño en la asignatura como en el aprendizaje de la programación.

5.2. Cumplimiento de objetivos

El trabajo de memoria presente se puede medir en función de los resultados obtenidos tras su aplicación. A continuación, se expone brevemente el cumplimiento de los objetivos específicos descritos en el capítulo 1, y el cumplimiento del objetivo general:

- **Analizar los conceptos clave referidos a la medición de la complejidad de ejercicios de programación en un contexto de una asignatura universitaria introductoria**

El marco conceptual del trabajo de memoria fue desarrollado, desde un inicio, con el propósito de entender a qué se refiere que un ejercicio sea complejo. Para esto, como se mencionó previamente, se estudiaron todas las bases relacionadas a los ejercicios de programación, para llegar a aquellos conceptos y métricas que definen la complejidad de un problema. Lo obtenido en la literatura fue, además, respaldado con una entrevista a alumnos sobre las complicaciones que tienen al momento de realizar los ejercicios de SMOJ, y con una entrevista a profesores sobre qué conceptos resultan claves para el aprendizaje de los alumnos, cuando se están enseñando los contenidos de programación.

- **Diseñar un modelo de clasificación automática de la complejidad de ejercicios de programación basado en técnicas de inteligencia artificial y aprendizaje automático**

El trabajo realizado en la propuesta de solución contempla el uso de un modelo de clasificación automática, enmarcado en el aprendizaje no supervisado, denominado K-means, que permite agrupar elementos en grupos con características similares.

- **Evaluar el rendimiento del modelo en base a diferentes métricas y encuestas aplicadas a los estudiantes de programación de la universidad**

La validación de la solución fue planteada en base a la comparación del porcentaje de completitud de ejercicios en laboratorios donde no se ha aplicado la metodología versus uno donde si haya sido así. Las encuestas, sin embargo, fueron utilizadas para respaldar el marco conceptual estudiado que, indirectamente, apoyaron la creación y rendimiento del modelo propuesto.

En base a lo anterior, y concluyendo que cada objetivo específico ha sido logrado correctamente, es que podemos afirmar que el objetivo general, que consiste en **aumentar las instancias de ejercitación y evaluación en un curso masivo de introducción a la programación a través del desarrollo de un modelo de clasificación automática de la complejidad de ejercicios de autocorrección en un curso masivo de programación** ha sido cumplido apropiadamente.

5.3. Trabajo a futuro

El modelo propuesto utiliza una de las técnicas de *clustering* más básicas, pero más efectivas, disponibles para su uso. Para todo aquel o aquella que busque seguir con esta investigación, se propone que compare los resultados obtenidos de aplicar diferentes modelos de *clustering*, con el objetivo de mejorar la clasificación y que resulte en mayores beneficios a los estudiantes.

La obtención de las etiquetas, hasta ahora, es solamente visible para los profesores de la asignatura, debido a la posibilidad de que los estudiantes, al tener conocimiento de las etiquetas, predispongan sus actividades de ejercitación en base a esto. Se propone, también, que se pueda realizar un estudio de si es realmente beneficioso que los alumnos sepan la complejidad del problema que están resolviendo, y si es así, poder relacionarlo con una visualización oportuna en la plataforma SMOJ.

Finalmente, se sugiere la investigación de diferentes métricas a medir, tanto en el enunciado del ejercicio como en el código, con el objetivo de mejorar y robustecer el análisis realizado.

5.4. Palabras finales del autor

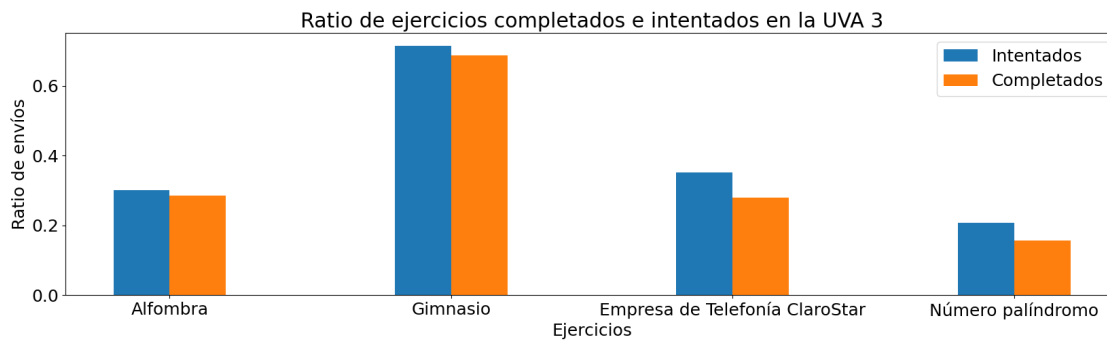
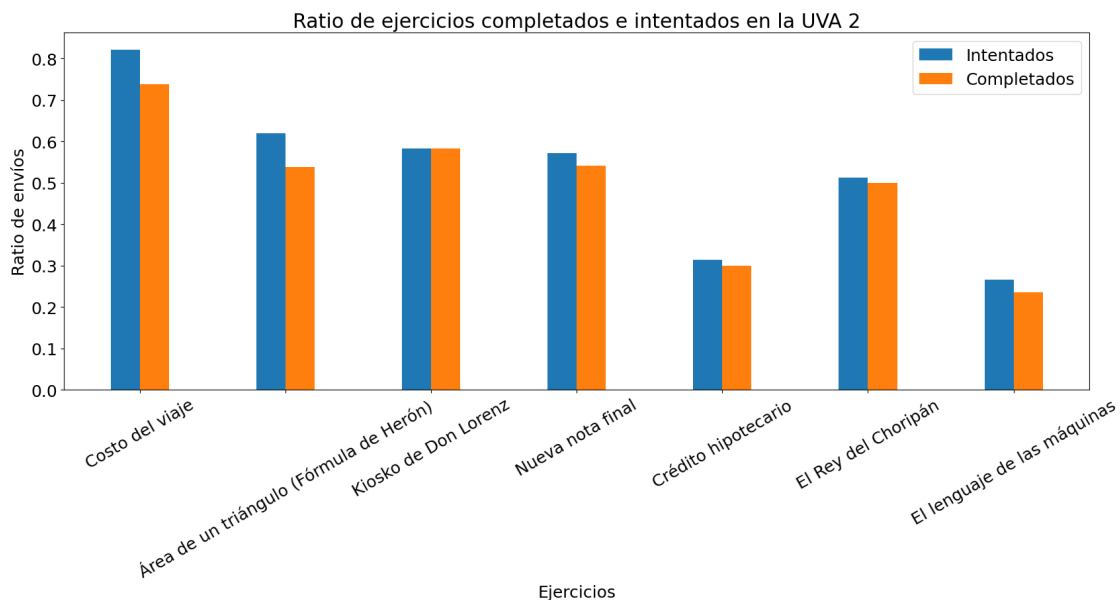
El desarrollo del trabajo de memoria fue un desafío en múltiples aristas. La realización de un análisis de este nivel me permitió poner a prueba los conocimientos adquiridos en mi estadía en la universidad, asociados al análisis exploratorio de los datos, visualización, estadística y habilidades de programación generales. Para esto, se dio lugar a múltiples reuniones con los profesores, velando por la calidad y el estándar del trabajo realizado, acorde a un profesional.

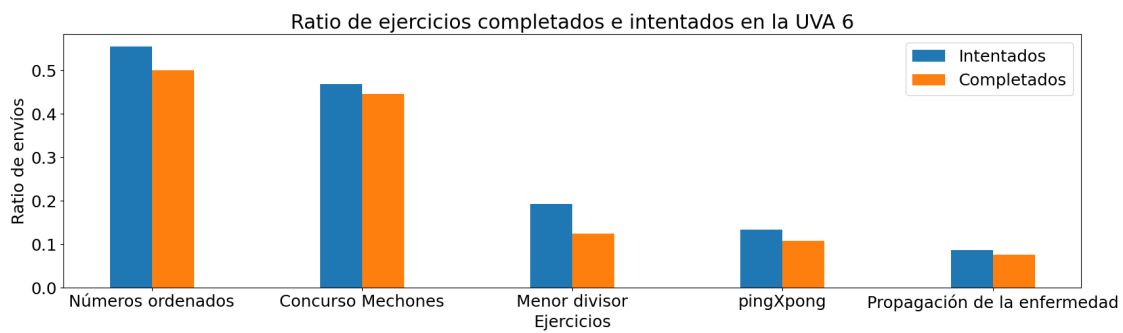
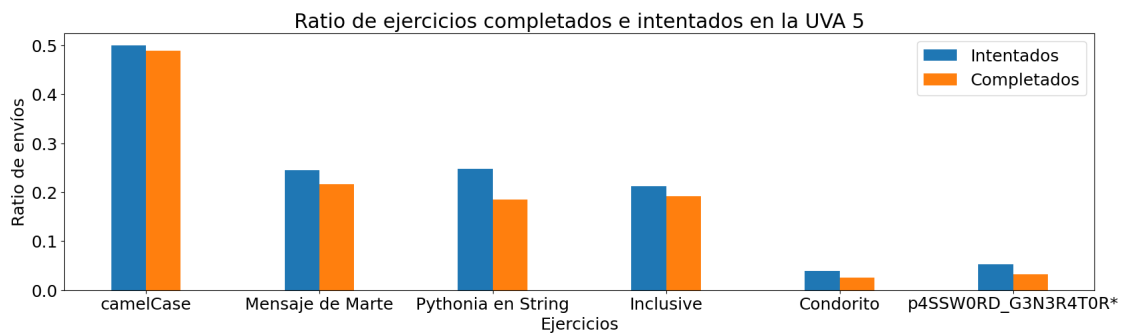
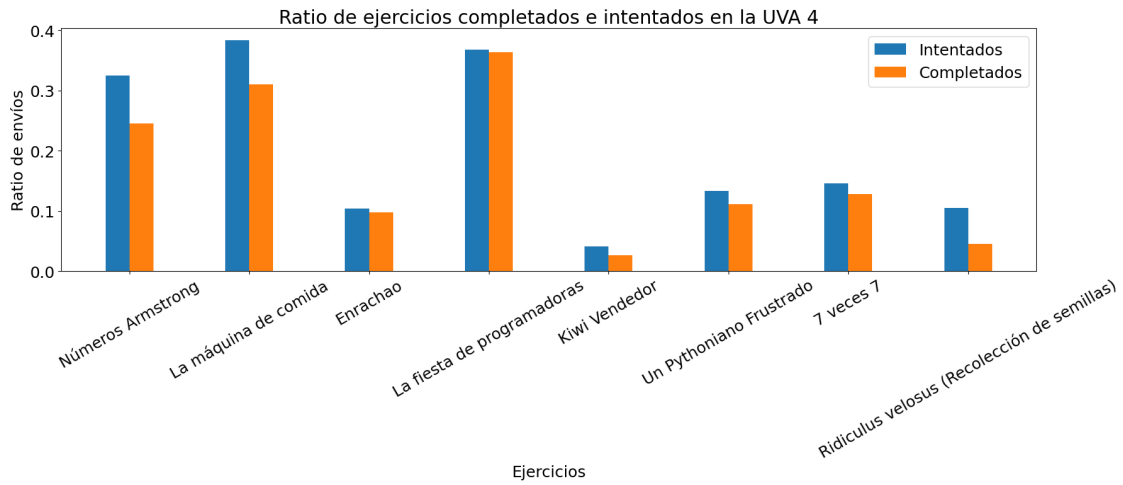
La finalización del trabajo me deja tremendamente orgulloso y feliz, no solo por el estudio mismo, si no que también por la posibilidad de mejorar la enseñanza de la programación a los estudiantes, logrando tener un impacto positivo tanto en la coordinación de la asignatura como en el desempeño, motivación y conocimiento de los alumnos.

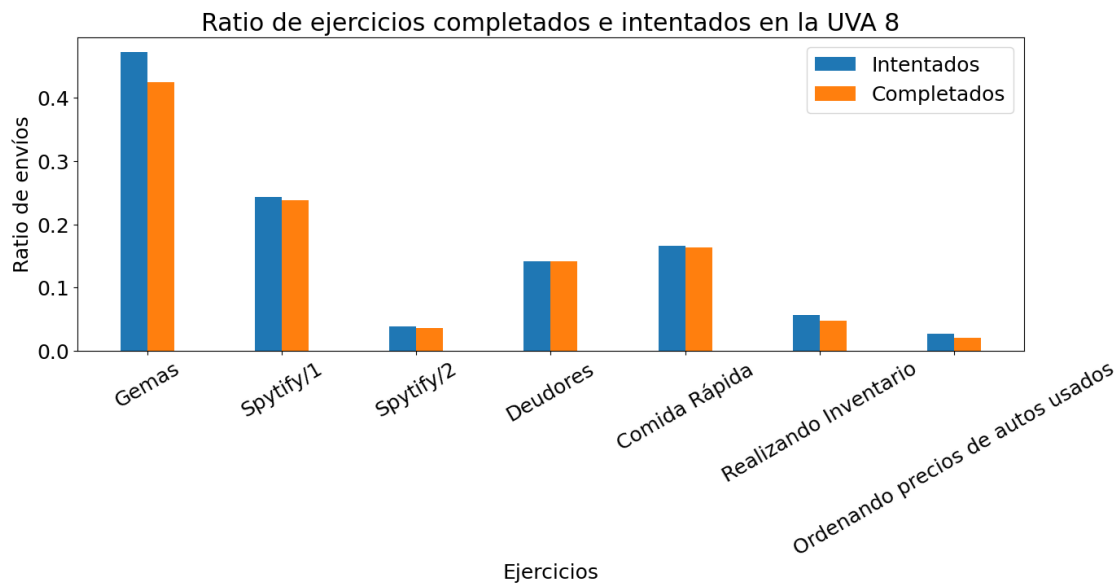
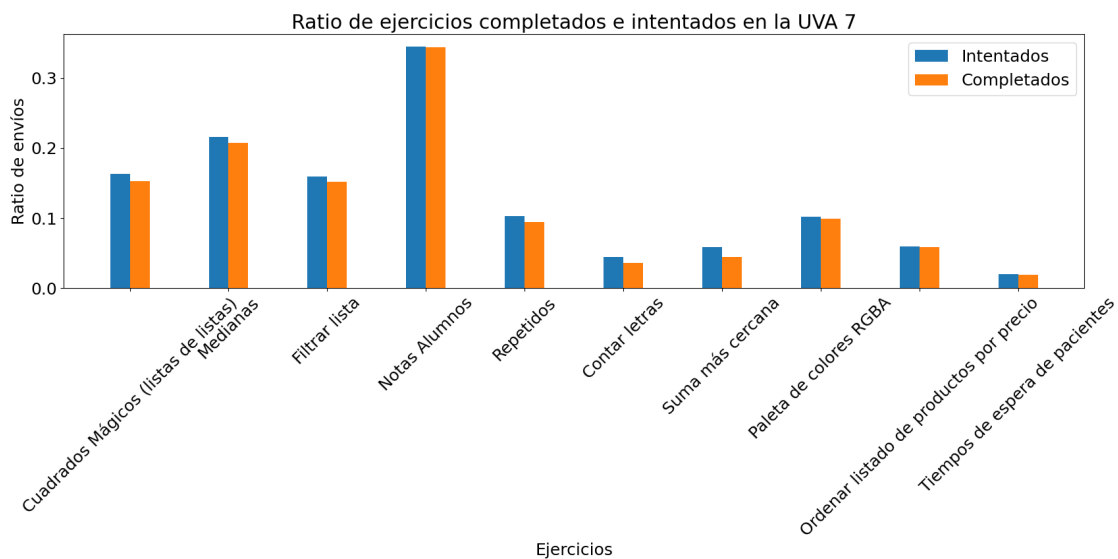
Mis últimas palabras van dirigidas a todas las personas que dedican su tiempo a la mejora continua de los procesos de aprendizaje, siendo conscientes de la relevancia que posee en la construcción de bases concretas y firmes para la adquisición de nuevos conocimientos y formación de profesionales de calidad.

ANEXOS

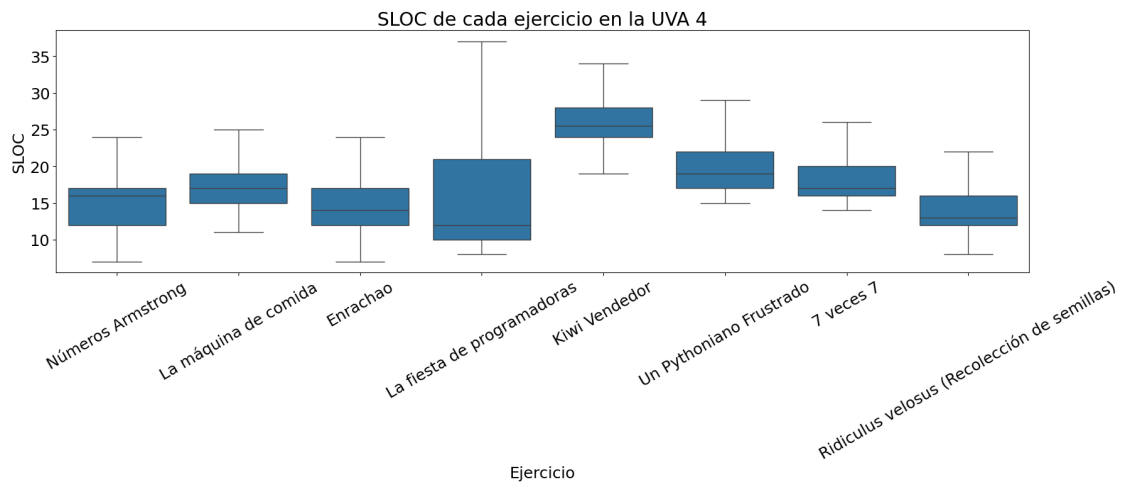
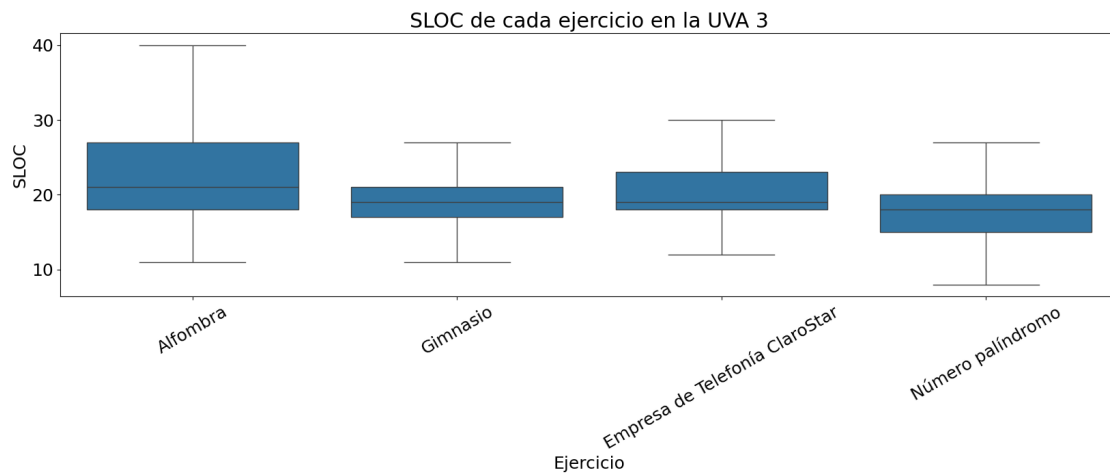
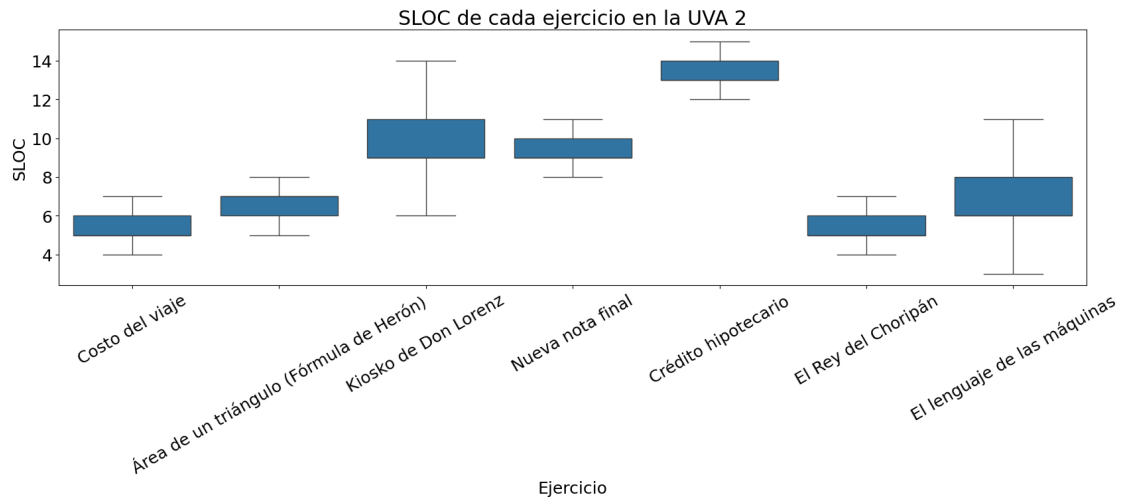
Anexo 1: Gráficos de completitud de ejercicios



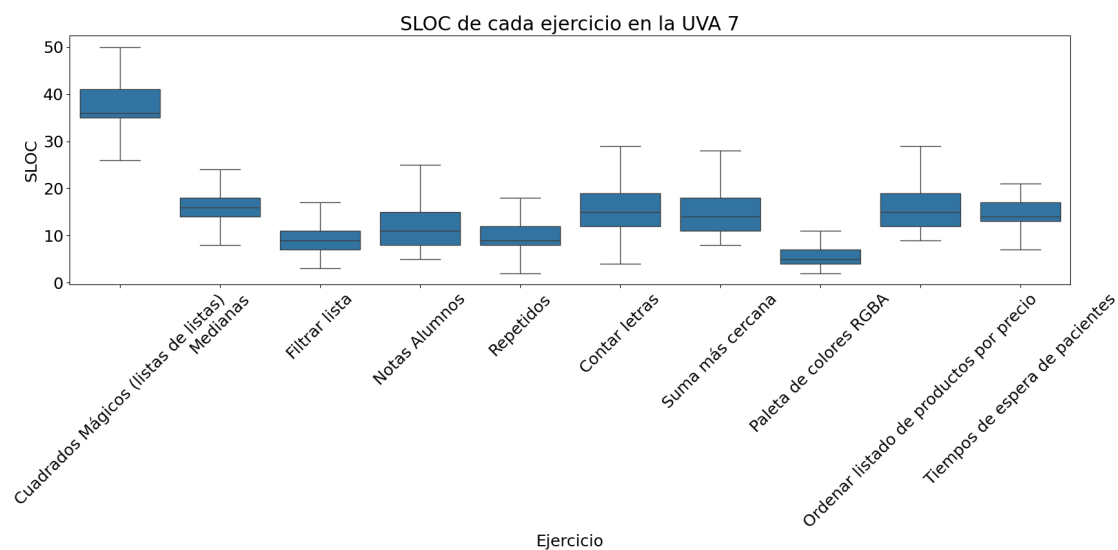
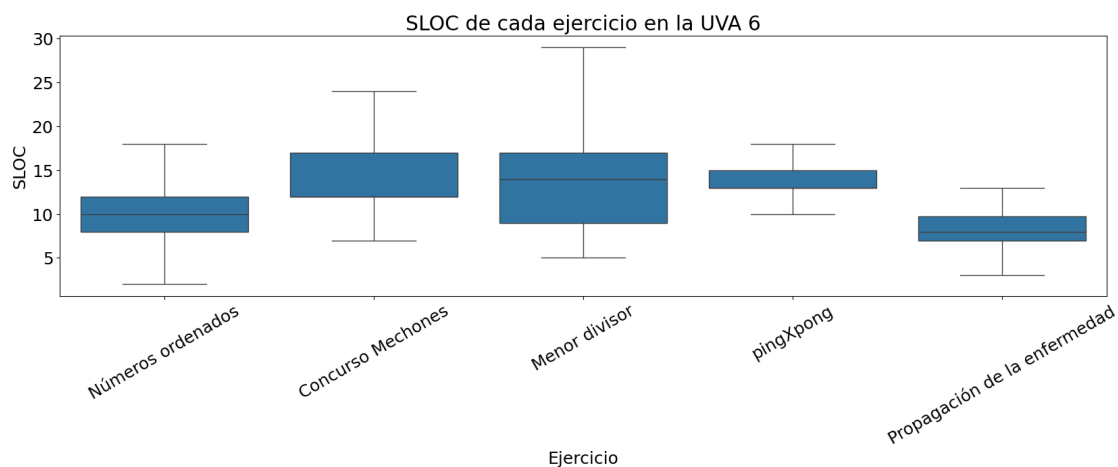
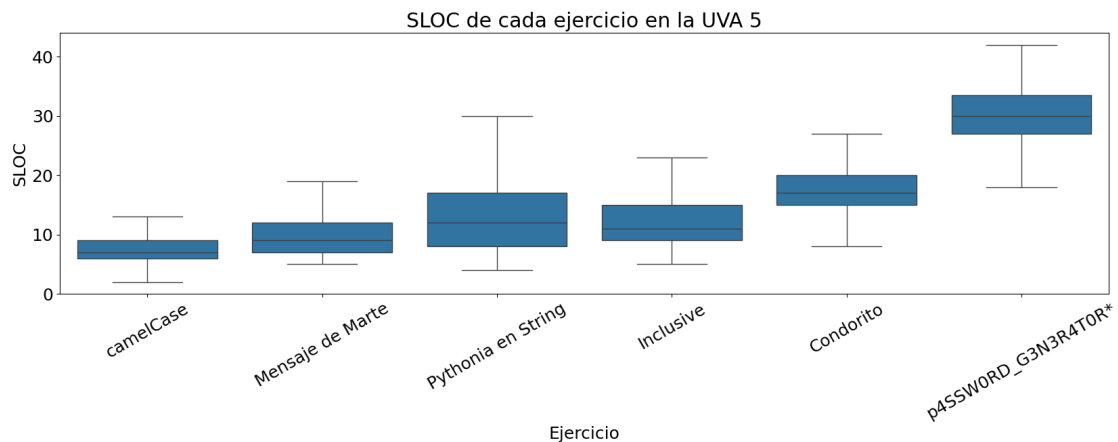


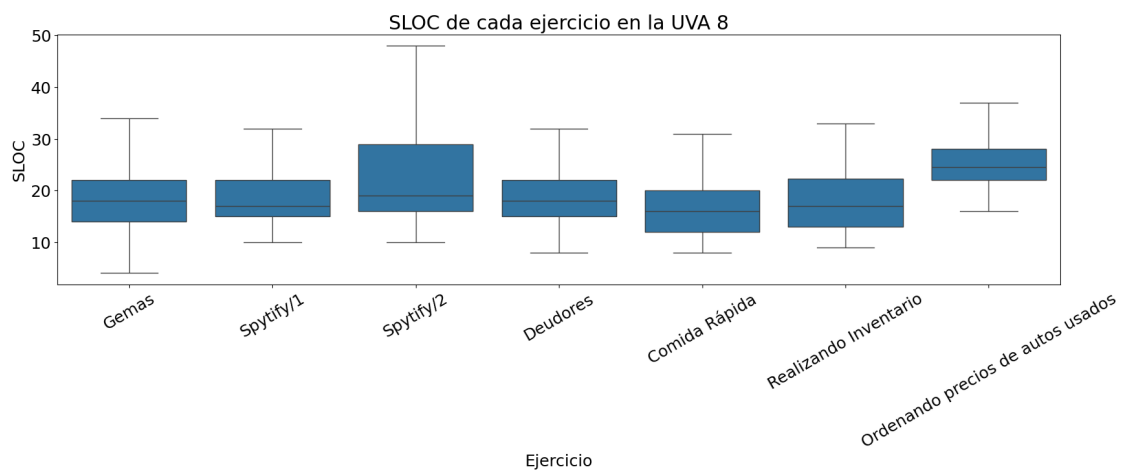


Anexo 2: Gráficos de la cantidad de líneas de código por ejercicio

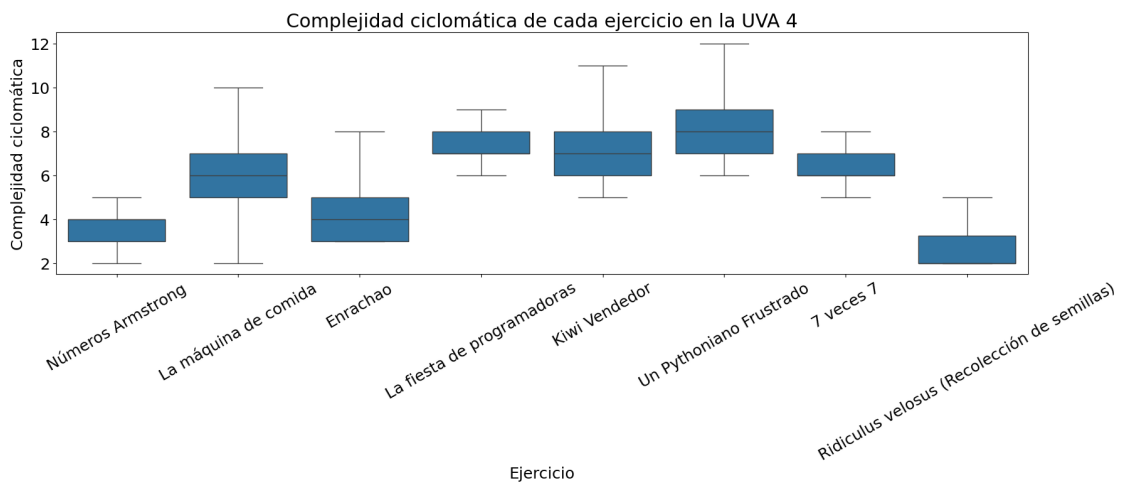
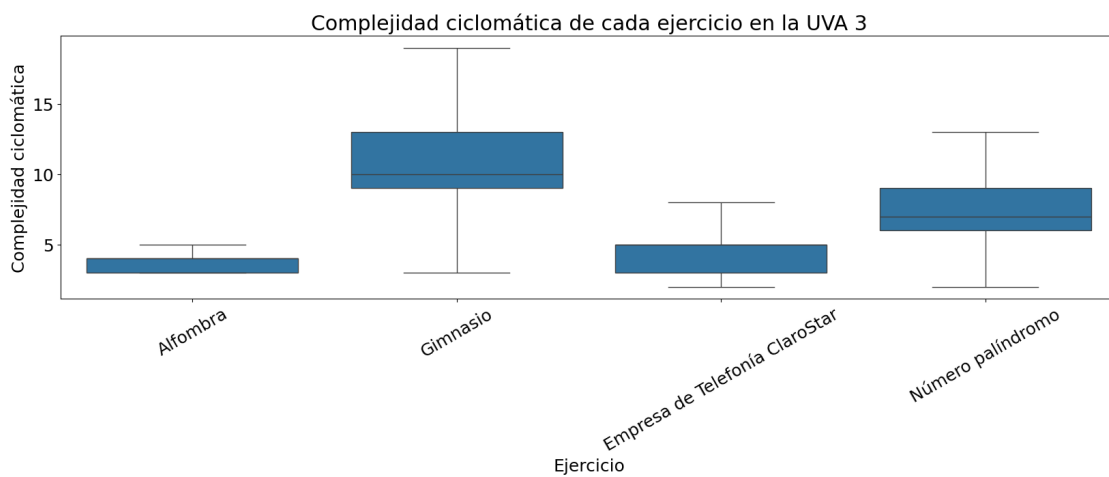
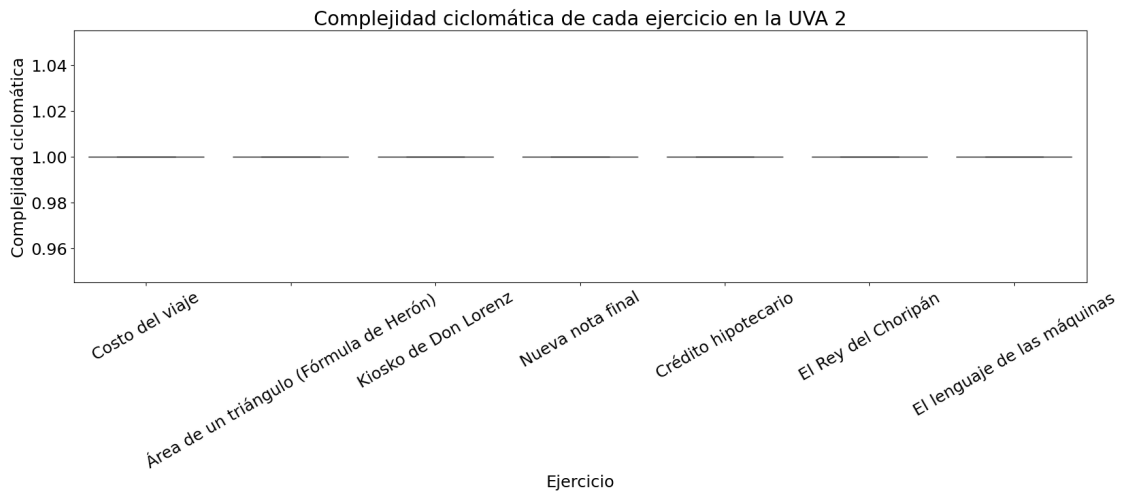


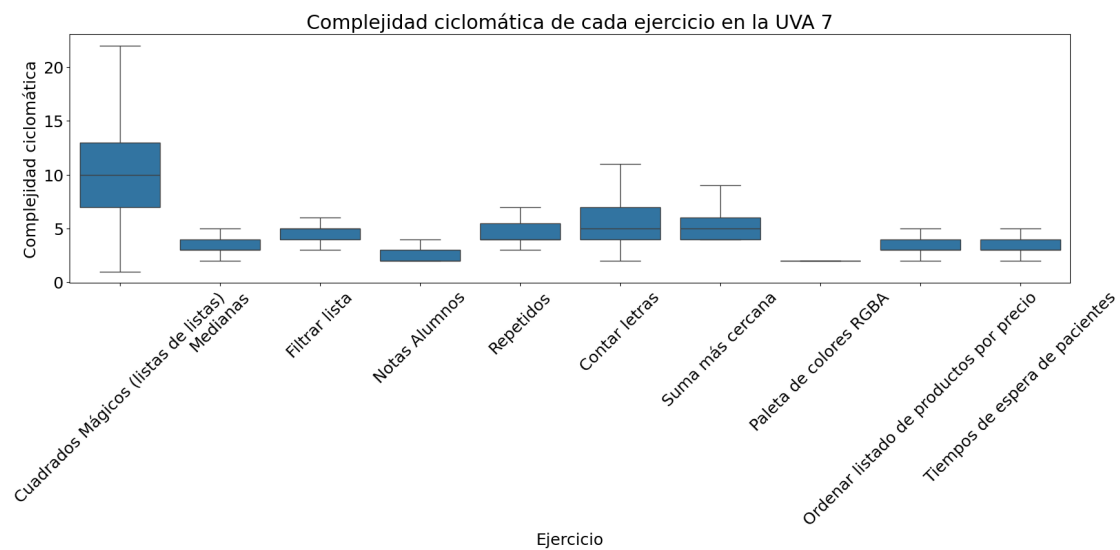
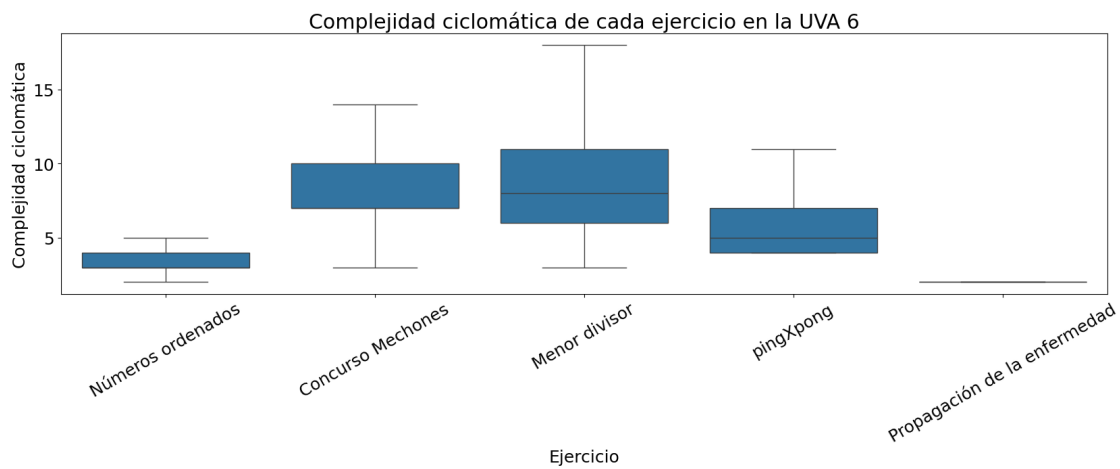
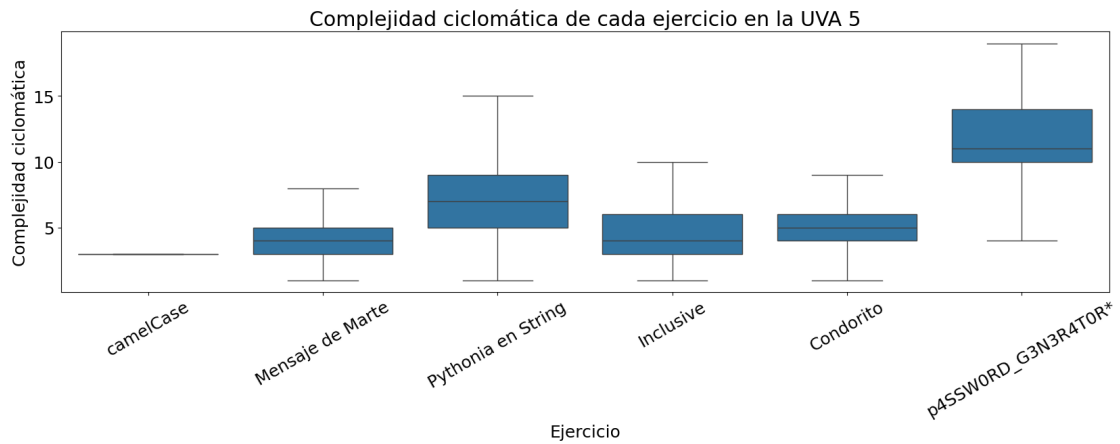
DISEÑO DE UN MODELO DE CLASIFICACIÓN AUTOMÁTICA DE LA COMPLEJIDAD DE EJERCICIOS DE AUTOCORRECCIÓN EN UN CURSO MASIVO DE PROGRAMACIÓN

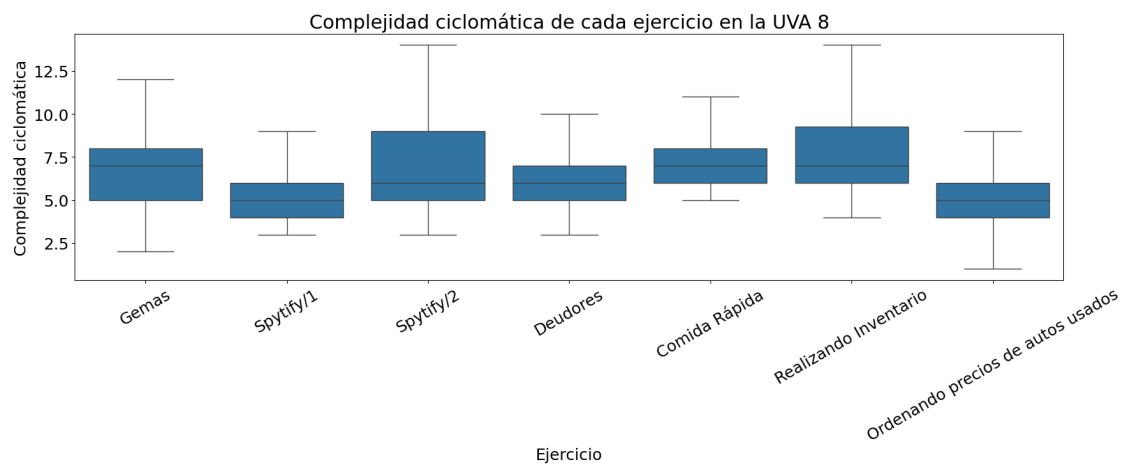




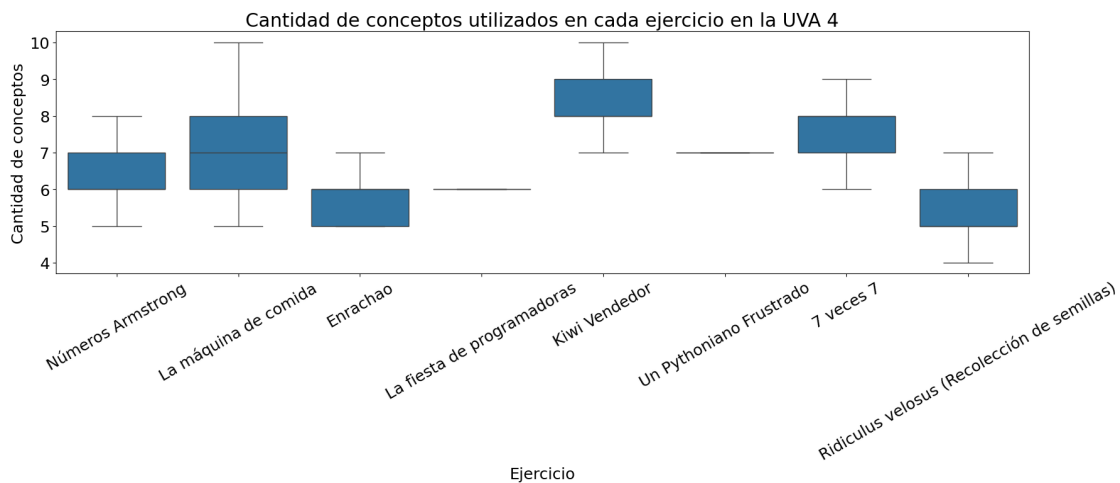
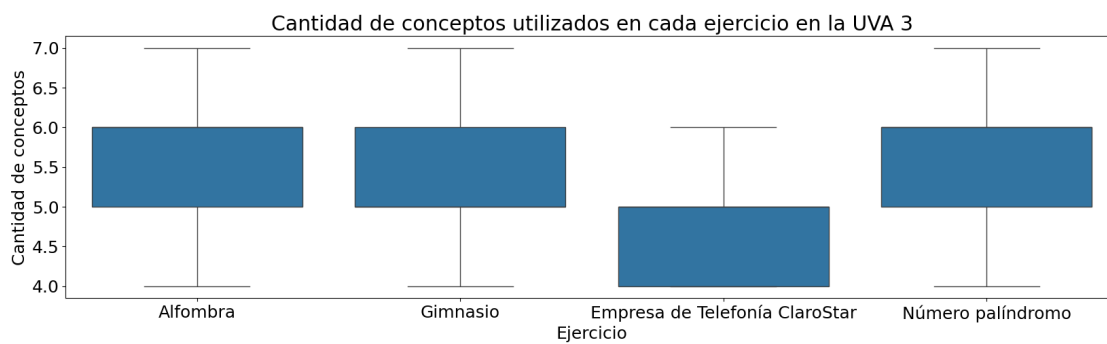
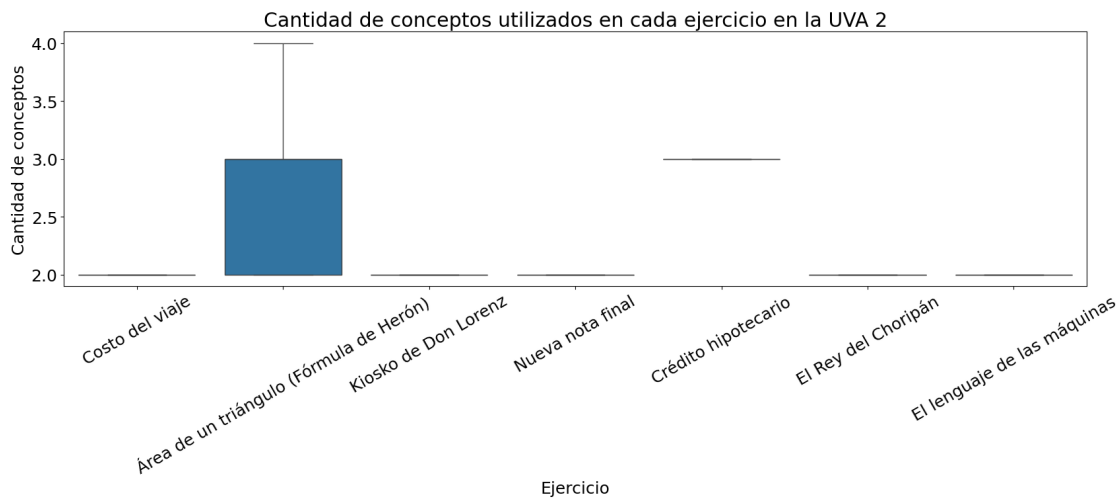
Anexo 3: Gráficos de la complejidad ciclomática por ejercicio

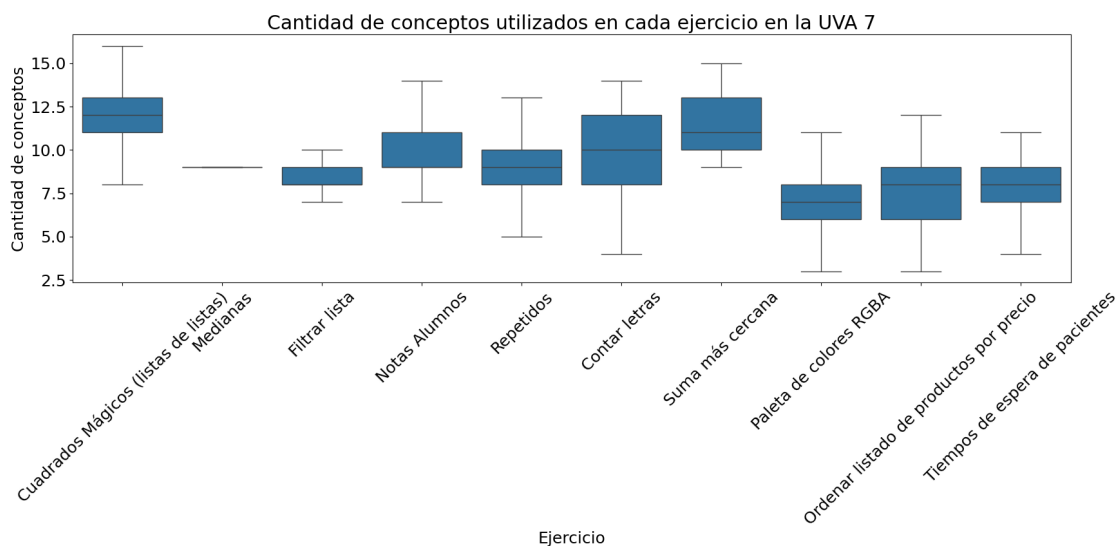
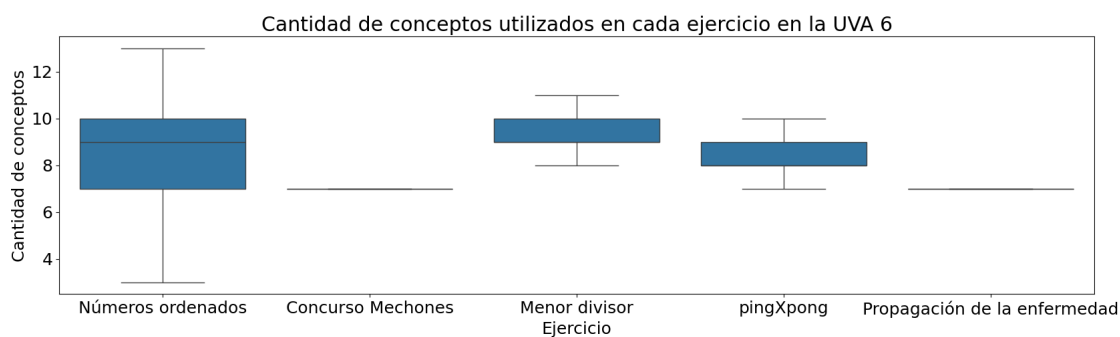
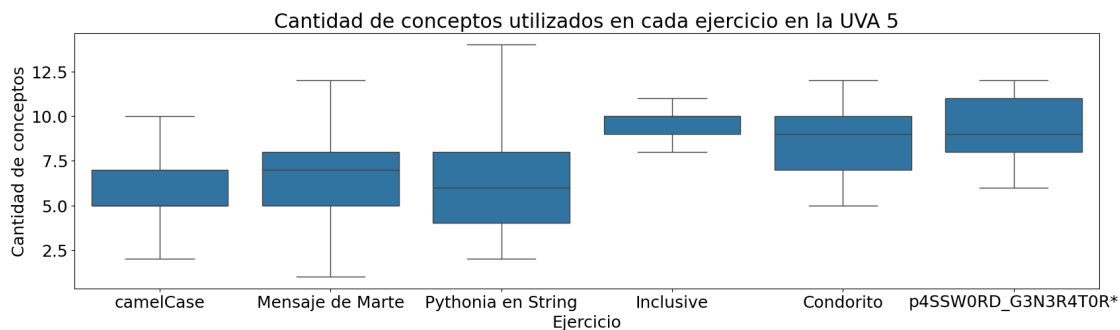


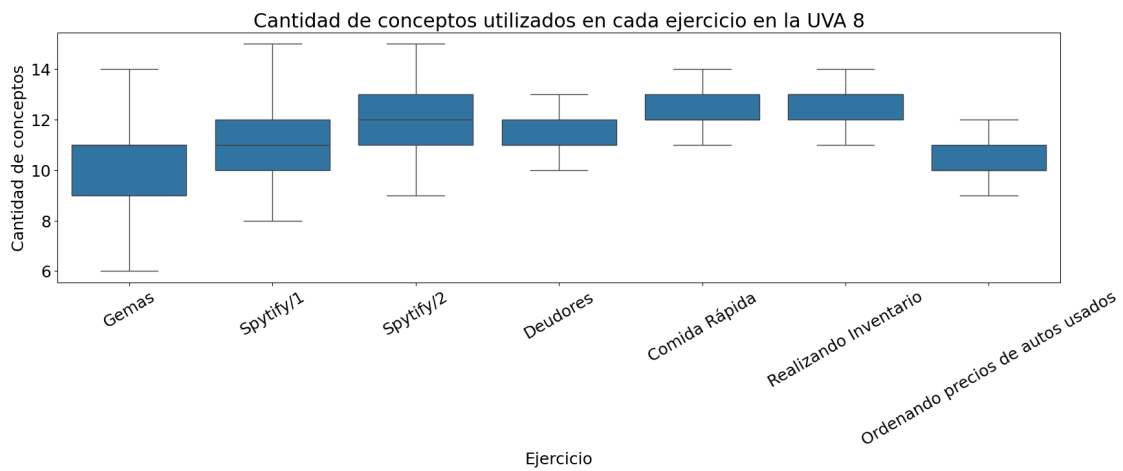




Anexo 4: Gráficos de la cantidad de conceptos por cada ejercicio







REFERENCIAS BIBLIOGRÁFICAS

- [Abualigah *et al.*, 2021] Abualigah, L., Gandomi, A., Elsayed Abd Elaziz, M., AlHamad, H., Omari, M., Al Shinwan, M., y Khasawneh, A. (2021). Advances in meta-heuristic optimization algorithms in big data text clustering. *Electronics*, 10:101.
- [Ahmad *et al.*, 2019] Ahmad, A., Abdi, A., Abuhassan, A., y Khalid, M. (2019). What is difficult in learning programming language based on problem-solving skills? pp. 18–22.
- [Buffer *et al.*, 2020] Buffer, O., Pietquin, O., y Weng, P. (2020). Reinforcement learning. CoRR, abs/2005.14419.
- [Chaudhkar, 2022] Chaudhkar, S. (2022). An overview of python for data analytics. *International Research Journal of Engineering and Technology (IRJET)*, 9.
- [CNED, 2020] CNED (2020). Informe de tendencias de educación por sexo.
- [Effenberger *et al.*, 2019] Effenberger, T., Cechák, J., y Pelánek, R. (2019). Difficulty and complexity of introductory programming problems.
- [El Bouchefry y de Souza, 2020] El Bouchefry, K. y de Souza, R. S. (2020). Chapter 12 - learning in big data: Introduction to machine learning. En *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, pp. 225–249. Elsevier.
- [Gandhi y Srivastava, 2014] Gandhi, G. y Srivastava, R. (2014). Review paper: A comparative study on partitioning techniques of clustering algorithms. *International Journal of Computer Applications*, 87.
- [Ihantola y Petersen, 2019] Ihantola, P. y Petersen, A. (2019). Code complexity in introductory programming courses.
- [Jones, 1994] Jones, C. (1994). Software metrics: good, bad and missing. pp. 98–100.
- [Kinnunen y Malmi, 2006] Kinnunen, P. y Malmi, L. (2006). Why students drop out cs1 course?
- [Liu y Badr, 2020] Liu, D. y Badr, M. (2020). Application: Control flow graphs.
- [Liu, 2022] Liu, M. (2022). The relationship between students' study time and academic performance and its practical significance. *BCP Education & Psychology*, 7:412–415.
- [Liu y Wu, 2012] Liu, Q. y Wu, Y. (2012). Supervised learning.
- [Luxton-Reilly y Petersen, 2017] Luxton-Reilly, A. y Petersen, A. (2017). The compound nature of novice programming assessments. pp. 26–35.

- [Margulieux *et al.*, 2020] Margulieux, L., Morrison, B., y Decker, A. (2020). Reducing withdrawal and failure rates in introductory programming with subgoal labeled worked examples. *International Journal of STEM Education*, 7.
- [Millet, 2015] Millet, H. (2015). The year of change: Challenges faced by first year students based on individual identities.
- [Muzhingi, 2020] Muzhingi, I. (2020). K-means clustering made simple.
- [Naeem *et al.*, 2023] Naeem, S., Ali, A., Anam, S., y Ahmed, M. (2023). An unsupervised machine learning algorithms: Comprehensive review. *IJCDS Journal*, 13:911–921.
- [Omran *et al.*, 2007] Omran, M., Engelbrecht, A., y Salman, A. (2007). An overview of clustering methods. *Intell. Data Anal.*, 11:583–605.
- [Oyelade *et al.*, 2010] Oyelade, J., Oladipupo, O., y Obagbuwa, I. (2010). Application of k means clustering algorithm for prediction of students academic performance. *International Journal of Computer Science and Information Security*, 7.
- [Pelánek *et al.*, 2021] Pelánek, R., Effenberger, T., y Čechák, J. (2021). Complexity and difficulty of items in learning systems. *International Journal of Artificial Intelligence in Education*, 32:1–37.
- [Semilof y Montgomery, 2020] Semilof, M. y Montgomery, J. (2020). Definition of software stack.
- [Shetty y Singh, 2021] Shetty, P. y Singh, S. (2021). Hierarchical clustering: A survey. *International Journal of Applied Research*, 7:178–181.
- [Simon *et al.*, 2016] Simon, A., Deo, M., Selvam, V., y Babu, R. (2016). An overview of machine learning and its applications. *International Journal of Electrical Sciences & Engineering*, Volume:22–24.
- [Sobral, 2021] Sobral, S. (2021). *Strategies on Teaching Introducing to Programming in Higher Education*, pp. 133–150.
- [Team, 2023] Team, C. (2023). Statistical modeling with python: How-to & top libraries.
- [Tran *et al.*, 2022] Tran, H., Vu, T., Nguyen, H., Tran, D., Nguyen, T., y Le, V. (2022). An efficient approach to measure the difficulty degree of practical programming exercises based on student performances. *REV Journal on Electronics and Communications*.
- [Vásquez *et al.*, 2021] Vásquez, A., Meza, F., y Barrera, P. G. (2021). Emergency remote teaching model for massive programming classes. pp. 1–9.
- [Voas y Kuhn, 2017] Voas, J. y Kuhn, D. (2017). What happened to software metrics? *Computer*, 50:88–98.

[Yuangga y Sunarsi, 2018] Yuangga, K. y Sunarsi, D. (2018). The influence of procrastination and low time management on student self efficacy (at ma soebono mantofani). *PINISI Discretion Review*, 2:85.