Pensamiento Computacional en México

El Pensamiento Computacional incluye un conjunto de competencias que permiten desarrollar representaciones de información, formular modelos mentales y soluciones algorítmicas necesarios para plantear soluciones a problemas, usando o no computadoras; el cual es fundamental para obtener un mejor provecho de las tecnologías digitales que inundan ya el entorno de vida, de trabajo y de estudio. Para un país como México es de gran importancia que se haga conciencia del valioso aporte que brinda el pensamiento computacional en todos los niveles educativos, así como de los esfuerzos realizados a la fecha en esa dirección. Su incorporación en una estrategia educativa nacional coordinada abriría las puertas a elevar el nivel académico de nuestros estudiantes, impulsando el progreso y beneficiando al desarrollo del país. Es este contexto surgió la iniciativa de organizar el Seminario de Pensamiento Computacional en México (PENCOMX) con el objetivo de identificar y documentar el estado actual de la investigación y la práctica en torno al pensamiento computacional en nuestro país, a fin de coadyuvar a su comprensión y desarrollo en beneficio de la educación y la sociedad mexicana. Las versiones revisadas y extendidas de los artículos en extenso presentados originalmente en el seminario, enriquecidos con los comentarios y discusiones a lo largo de los dos días de trabajo, conforman este libro que ponemos a su consideración.

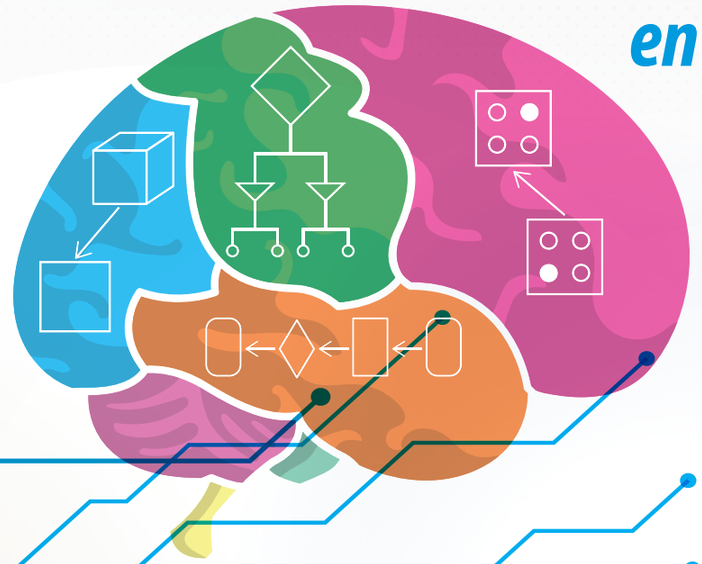
ISBN 978-607-98941-4-6

PENCOMX

amexcomp

Pensamiento Computacional en México | Academia Mexicana de Computación

Pensamiento Computacional en México



Editado por

Rafael Morales Gamboa
Eduardo Morales Manzanares
Alberto Pacheco González
Marcela Quiroz Castellanos
Luis Enrique Sucar Succar

amexcomp

Academia Mexicana de Computación A.C.

Pensamiento Computacional en México

Seminario de Pensamiento Computacional en México (PENCOMX).
28 y 29 de enero de 2021.

Comité organizador

Rafael Morales Gamboa (UDG)
Eduardo Morales Manzanares * (INAOE)
Alberto Pacheco González (TecNM campus Chihuahua)
Marcela Quiroz Castellanos * (UV)
Luis Enrique Sucar Succar * (INAOE)

Comité del programa

Gabriel López Morteo (UABC)
Rafael Morales Gamboa (UDG)
Eduardo Morales Manzanares (INAOE)
Juana Julieta Noguez Monroy * (Tec)
Alberto Pacheco González (TecNM campus Chihuahua)
Marcela Quiroz Castellanos UV)
Luis Enrique Sucar Succar (INAOE)
Jorge Luis Zapotécatl * (INAOE)

* Miembro de la Academica Mexicana de Computación.

Pensamiento Computacional en México

editado por

Rafael Morales Gamboa
Eduardo Morales Manzanares
Alberto Pacheco González
Marcela Quiroz Castellanos
Luis Enrique Sucar Succar

Academia Mexicana de Computación

Primera edición: 2021

ISBN: 978–607–98941–4–6

Diseño de portada: Mario Alberto Vélez Sánchez

Esta edición y sus características son propiedad de la
Academia Mexicana de Computación, A. C.

Impreso y hecho en México.
Printed and made in Mexico.

Prefacio

Existe actualmente un interés global por el desarrollo del Pensamiento Computacional en los estudiantes, la fuerza laboral y la sociedad en general. Entendido como un conjunto de competencias que permiten desarrollar representaciones de información y modelos mentales necesarios para plantear soluciones algorítmicas a problemas, el pensamiento computacional es considerado fundamental para obtener un mejor provecho de las tecnologías digitales que inundan ya el entorno de estudio, de trabajo y de vida.

El desarrollo de estas competencias entre la mayoría de la población de nuestro país nos habilitaría para resolver una gran variedad de problemas en diversos contextos, disciplinas y profesiones, utilizando estrategias diversas, las cuales podrían ser implementadas en sistemas computacionales para obtener rápidamente soluciones que sean más precisas y más confiables. Para algunos expertos, el pensamiento computacional constituye un nuevo paradigma para la ciencia, que complementa tanto a la teoría, como la experimentación.

Para un país como México es de gran importancia que se haga conciencia del valioso aporte que brinda el pensamiento computacional en todos los niveles educativos, así como de los esfuerzos realizados a la fecha en esa dirección. Su incorporación en una estrategia educativa nacional coordinada abriría las puertas a elevar el nivel académico de nuestros estudiantes, impulsando el progreso y beneficiando al desarrollo del país, como ha sido ya planteado e implementado en otros países desde sus niveles educativos básicos, con iniciativas tales como CSTA CS para K-12 (2011), European Digital Competence Framework (DigComp, 2013-2016), the British Computer Society Computational Thinking iniciative (2015-2020), ISTE computational thinking competences standars (2016-2018), NSF STEM + Computing K-12 Education y CS for All (STEM+C, 2017-2020).

Es este contexto que surgió la iniciativa de organizar el Seminario de Pensamiento Computacional en México (PENCOMX) con el objetivo de identificar y documentar el estado actual de la investigación y la práctica en torno al pensamiento computacional en nuestro país, a fin de coadyuvar a su comprensión y desarrollo en beneficio de la educación y la sociedad mexicana. Se recibieron un total de quin-

ce propuestas de ponencias, de las cuales diez fueron aceptadas para ser presentadas en las sesiones del seminario los días 28 y 29 de enero de 2021. Son las versiones revisadas y extendidas de los artículos en extenso presentados originalmente en el seminario, enriquecidos con los comentarios y discusiones a lo largo de los dos días de trabajo, las que conforman este libro que ponemos a su consideración.

Nuestro mayor agradecimiento a todos los autores por su disposición a compartir sus experiencias y conocimientos, a participar activamente en las sesiones del seminario y a revisar sus respectivos documentos hasta que todos estuvimos satisfechos.

Rafael Morales Gamboa
Eduardo Morales Manzanares
Alberto Pacheco González
Marcela Quiroz Castellanos
Luis Enrique Sucar Succar

Agradecimientos

Los autores y editores de este libro agradecemos el apoyo de los demás miembros del Comité del Programa del Seminario de Pensamiento Computacional por sus valiosas aportaciones a mejorar la calidad de lo que aquí se presenta. Al Sistema de Universidad Virtual de la Universidad de Guadalajara por hospedar el sitio web del seminario (pencomx.org). A los revisores externos por la identificación de los numerosos detalles que tuvimos que corregir para hacer esta obra presentable al público y a la Academia Mexicana de Computación por publicarla.

Contenido

Lista de figuras

Lista de tablas

Lista de algoritmos

Lista de códigos

Parte I

Enseñanza del Pensamiento Computacional

Capítulo 1

Hacia la Enseñanza del Pensamiento Computacional como materia básica de la Preparatoria en México

Alejandro A. Torres-García [1], *Pedro Tecuanhuehue-Vera* [1]
y *Gisela Yanett López Juárez* [1]

[1] Centro de Aprendizaje y Desarrollo de Habilidades "Mind Up".
alejandro.torres@ccc.inaoep.mx, pedrish@gmail.com,
cadh.mindupdespertandomentes@gmail.com

Resumen. En la actualidad la enseñanza de la informática en México es generalizada, enfocándose principalmente en la enseñanza de conceptos teóricos y en el aprendizaje del uso de software de ofimática. Sin embargo, para la sociedad y empleos del futuro, se requiere que los alumnos desarrollen habilidades tanto para identificar y comprender problemas de la realidad como para plantear y evaluar soluciones. En ese sentido la enseñanza del pensamiento computacional como materia en el tronco común de bachillerato (preparatoria) ofrecería a los estudiantes el fomento de estas habilidades. Por ello, dicha idea se está implementando en la preparatoria del Centro de Aprendizaje y Desarrollo de Habilidades "Mind Up Despertando Mentes" de Tuxtla Gutiérrez, Chiapas.

Se trata, según los datos recabados, de la primera institución que imparte dicha materia en el bachillerato en México. Por ello, en este trabajo se presenta una estrategia inicial sobre el proceso utilizado para introducir a los estudiantes al Pensamiento Computacional en el nivel bachillerato así como también los avances alcanzados hasta ahora. La estrategia tiene 3 frentes y son llevados a cabo de forma paralela: 1) enseñanza de los elementos del pensamiento computacional, 2) charlas de expertos que ilustran el uso del pensamiento computacional en su campo, y 3) el desarrollo de proyecto integrador en el cual los alum-

nos eligen un problema de la sociedad para el cual les interesa proponer una solución. Esto último permite fomentar las habilidades del pensamiento crítico en los alumnos, así como también integrar el conocimiento de lo aprendido en otras materias del semestre.

Palabras clave. Pensamiento computacional, bachillerato, método de enseñanza, aprendizaje basado en proyectos.

1.1 Introducción

En la actualidad, las cantidades de datos e información generadas diariamente hacen necesarias toda una serie de habilidades y sistemas automáticos, robustos a estas cantidades de datos, que permitan procesarla y analizarla para extraer información útil y valiosa lo más rápido posible. Es por esta razón que la economía global está transitando de una economía basada en la información/conocimiento a otra impulsada por la computación [1]. Además, los avances tecnológicos actuales en el campo de la robótica, sensores, drones y electrónica junto con la disminución de sus costos hacen posible que estas áreas puedan ser empleados/considerados para solucionar/automatizar problemas existentes en nuestras comunidades.

Lo anterior ha hecho que en países como Reino Unido, Lituania, Finlandia, Corea del sur, Japón y Singapur se estén introduciendo políticas educativas para preparar a sus ciudadanos para estar listos para la sociedad del futuro [1]. Particularmente, en estos países las iniciativas y políticas son hechas para introducir las habilidades de Pensamiento Computacional y Programación en sus escuelas [1].

El pensamiento computacional, en palabras de la investigadora Jeannette Wing, se define como: 'los procesos de pensamiento involucrados en la formulación de problemas y representación de sus soluciones, de manera que dichas soluciones puedan ser ejecutadas efectivamente por un agente de procesamiento de información (humano, computadora o combinaciones de humanos y computadoras)' [2].

El pensamiento computacional se basa en las siguientes piedras angulares o elementos clave [3]:

a) la descomposición de problemas (dividir un problema en partes más pequeñas),
b) la abstracción (obtener los elementos relevantes para modelar el problema),
c) el reconocimiento de patrones (identificar similitudes entre los subproblemas con problemas previos en los elementos a estudiar),
d) el diseño de algoritmos (plantear claramente los pasos de una solución al problema), y
e) evaluación de las soluciones planteadas (o proyectos).

Estas habilidades son fundamentales no sólo para la programación de computadoras sino también para poder sistematizar el proceso encaminado a la solución de problemas en dominios diversos. Dentro de las ventajas que ofrece el pensamien-

to computacional están la oportunidad de desarrollar el pensamiento crítico en los alumnos, así como el pensamiento abstracto, la toma de decisiones, la adquisición de conocimientos y el desarrollo de habilidades requeridas para la sociedad del futuro, además, envuelve a los alumnos en un proceso descriptivo que fomenta la resolución de problemas complejos y los ayuda a tener una visión global. Además, el pensamiento computacional permite que el educando demuestre actitudes como la confianza, resiliencia, tolerancia, observación, análisis y comprensión del comportamiento humano.

También, Jeannette Wing visualiza al pensamiento computacional como una habilidad que llegará a ser tan universal como las matemáticas y el lenguaje [2, 3]. Esto sería muy relevante para tener un lenguaje común y un conjunto de habilidades compartidas por las personas que sean útiles durante todo el proceso que deriva en el diseño de una metodología, método o algoritmo o el desarrollo de un prototipo para abordar una problemática dada. Asimismo, el pensamiento computacional permitiría compartir el conocimiento de forma universal con otras personas (independientemente de sus idiomas maternos).

1.1.1 El pensamiento computacional en el nivel medio superior en México

A pesar de las posibilidades que la enseñanza del pensamiento computacional ofrecería a los estudiantes del bachillerato como una materia del tronco común, ésta aún no está incluida en los planes de estudio actuales. Lo anterior, pese a ser una habilidad para las actividades y profesiones de este siglo. Por ejemplo, haciendo un análisis de los planes de estudio a nivel bachillerato actuales [4, 5], encontramos que las materias más relacionadas con el pensamiento computacional son Informática [6] y las materias adicionales que son enseñadas en los módulos de especialidad de algunos sistemas de bachillerato.

Particularmente, uno de los módulos más interesantes lo encontramos en el módulo 2 de la materia informática II [7] del Colegio de Bachilleres, en la cual se da una introducción a los diagramas de flujo y pseudocódigos. Sin embargo, no hay un seguimiento en los siguientes módulos de la materia. Además, conviene recordar que en el pensamiento computacional la habilidad de diseño de algoritmos es complementada con las habilidades de descomposición de problemas, abstracción y reconocimiento de patrones.

En lo relacionado con informática en el tronco común, esta materia se centra más en la enseñanza de conceptos de teoría informática y tecnologías de la información y, en la práctica, ésta se enfoca en la enseñanza de software de ofimática (Word, Excel y Power Point). Lo anterior en detrimento de fomentar que los alumnos desarrollen habilidades que les permitan identificar y comprender problemas, reconocer patrones en estos problemas, y proponer y desarrollar proyectos.

La motivación para comenzar la enseñanza de pensamiento computacional en el tronco común en vez de, o complementado con, las materias de Informática (orientadas al aprendizaje de ofimática) es que los alumnos desarrollen habilidades encaminadas hacia el entendimiento y solución de problemas. Además de que permite mejorar la experiencia de los alumnos al ser el pensamiento computacional capaz de integrar conocimientos y habilidades de otras materias concurrentes en el plan de estudios de una preparatoria como: Taller de investigación, Taller de lectura y redacción, Inglés, Matemáticas, entre otras.

Otro factor que justifica la prevalencia del pensamiento computacional sobre la enseñanza de software de ofimática es el hecho de que, en las ciudades y en muchas poblaciones del país, se cuenta con acceso a computadoras e internet (al menos ciber café). Por ende, es más probable que los estudiantes de preparatoria estén familiarizados con el uso de software de edición de textos, diapositivas y hojas de cálculo. Adicionalmente, con el aprendizaje de las habilidades del pensamiento computacional, los alumnos posteriormente podrían ser capaces de ir de una forma autodidacta encontrando de entre los tutoriales disponibles en la red, el que pueda guiarlos en el aprendizaje de algún software de ofimática.

En las siguientes secciones presentaremos los esfuerzos encaminados a responder la pregunta cómo abordar la enseñanza del pensamiento computacional como materia del tronco común en el nivel de preparatoria, siguiendo un enfoque de aprendizaje basado en proyectos. Este tipo de aprendizaje tiene varias etapas que van desde definir el propósito del proyecto, identificar su mercado, investigar sobre el tema en detalle; pasando por crear un plan de gestión del proyecto y el diseño de éste hasta la elaboración/implementación del proyecto (o producto) [8]. Por lo que la materia de pensamiento computacional no sólo es de gran apoyo para los alumnos en las etapas iniciales del aprendizaje basado en proyectos sino que será de gran ayuda en todo el proceso del aprendizaje basado en proyectos. Con esto se logra que los alumnos puedan adquirir y tener una experiencia más enriquecedora con el pensamiento computacional y que los siguientes módulos de especialidad sean enfocados en cómo guiar a los alumnos en el desarrollo de sus proyectos.

1.2 El Pensamiento Computacional en la Preparatoria

La implementación de esta idea se lleva a cabo en el primer semestre de la preparatoria "Mind Up Despertando Mentes" de Tuxtla Gutiérrez, Chiapas. Actualmente, la estrategia se está aplicando con la participación de seis alumnos del primer semestre. Cabe señalar que ninguno de los estudiantes tenía conocimientos previos de programación o computación. Adicionalmente, esta materia será complementada con materias de especialidad en programación como parte de la preparatoria.

El eje central de la propuesta radica en el aprendizaje basado en proyectos por

las siguientes dos razones. La primera radica en el hecho de que el aprendizaje basado en proyectos está orientado a la acción y consolida las bases del constructivismo, en el cual los alumnos se apoyan de sus conocimientos e ideas previas. Mientras que la segunda razón se debe a que, como se menciona en [9], este enfoque permite que los alumnos puedan abordar problemas no triviales (pero de su interés) planteando una solución a estos. La idea en mente es lograr que los alumnos tengan una experiencia de aprendizaje más cercana a la realidad y al mundo profesional, en la cual no siempre hay una ruta única o clara para resolver un problema particular.

El aprendizaje basado en proyectos también permite que los alumnos generen y refinen sus preguntas, debatan sus ideas, diseñen planes y experimentos, recolecten y analicen datos, y establezcan conclusiones. Asimismo, éste permite que los alumnos comuniquen sus ideas y necesidades a otros, realicen nuevas preguntas y creando o mejorando productos y procesos [8, 9]. En suma, el aprendizaje basado en proyectos favorece que los estudiantes puedan apropiarse de su propio conocimiento, y logren desarrollar el pensamiento reflexivo y crítico, y un interés por la investigación.

Por otra parte, según los datos recabados, Mind Up es la primera institución que ofrece la materia de pensamiento computacional en el nivel bachillerato en México. Por esta razón, a continuación presentamos una estrategia de implementación que podrían seguir otras instituciones del país. Básicamente, la estrategia de la enseñanza del pensamiento computacional en "Mind Up Despertando Mentes" aborda los siguientes tres frentes:

1. Introducción al pensamiento computacional.
2. Charlas de profesionistas mostrando el uso del pensamiento computacional.
3. Proyecto integrador de las materias del semestre.

En el frente 1 se busca desarrollar paulatinamente las habilidades del pensamiento computacional. El frente 2 busca que los alumnos tengan una experiencia más auténtica, como es sugerido en [8], en la que aprenden como el pensamiento computacional es usado por expertos en proyectos/ambientes multidisciplinarios. Finalmente, el frente 3 es el desarrollo de un proyecto que permite incorporar las habilidades de otras materias del semestre como Pensamiento Crítico, Taller de investigación, Inglés, Caricatura, Taller de lectura y redacción, Economía doméstica, Química, y Filosofía y Ética. A continuación, se presenta a detalle cada una de estas etapas.

1.2.1 Introducción al pensamiento computacional

En este frente se presentan los conceptos de pensamiento computacional, abstracción, descomposición de problemas, reconocimiento de patrones y diseño de algoritmos. La idea fundamental es que el estudiante aprenda estas habilidades para,

a partir de un problema dado, abstraer los elementos relevantes del problema, descomponer el problema en partes más pequeñas y manejables, reconocer si algunos de estos subproblemas (o inclusive elementos del mismo) ya los ha resuelto antes, o bien, si identifica soluciones previamente abordadas que puedan ayudarle a solucionar el problema actual y diseñar una estrategia de solución, generalmente, desde el punto de vista computacional.

En la parte del diseño y análisis de algoritmos se ha trabajado de la mano de dos herramientas disponibles en la red como hourofcode y Scratch. Esto por la duración de la materia y la experiencia de los alumnos en computación, y con la idea de privilegiar la creatividad y la propuesta de soluciones un poco antes que la pulcritud de los códigos o habilidad en el manejo de la computadora. También se optó por la enseñanza del pensamiento computacional con herramientas de computadora por las situaciones actuales relacionadas con la pandemia de la COVID-19, en la cual las clases son impartidas en línea.

Aunque en el caso de los ejercicios de hourofcode son menos flexibles, éstos permiten homogeneizar y evaluar más rápidamente las soluciones ya que son para resolver problemas ya planteados y con los bloques necesarios para hacerlo. Asimismo, hourofcode tiene un mapeo bastante nítido con respecto a pseudocódigos y códigos de algún lenguaje de programación. Mientras que en el caso de Scratch, éste facilita más la implementación de ideas libremente y tiene un mapeo bastante transparente con el aprendizaje de diagramas de flujo.

En particular nuestra estrategia comenzó con los ejercicios en hourofcode, y conforme los alumnos adquirieron habilidades comenzamos a trabajar con Scratch, en donde hemos desarrollado algunos juegos e implementado diagramas de flujo ya definidos. La siguiente etapa es aprender a trasladar las ideas de solución de un problema dado tanto a diagramas de flujo como en pseudocódigos, pero pasando por todas las etapas del pensamiento computacional. Finalmente, la idea es concluir el curso comenzando a implementar algunos diagramas de flujo en Python debido a que será el lenguaje de programación a usar en el siguiente bloque de la especialidad en programación de "Mind Up". Todo el esquema de tareas, prácticas y contenidos pueden consultarse a detalle en el anexo 1 del material suplementario.

En la Figura 1.1 se observa un ejemplo de juego elaborado durante el curso en el cual se aplicó la descomposición de problemas al realizar la implementación de cada personaje por separado, la abstracción para pensar en cómo representar un salto, entre otras cosas.

El refuerzo de lo aprendido en las prácticas se lleva a cabo mediante reportes donde se hace hincapié en que el alumno identifique en qué momento utilizó las habilidades del pensamiento computacional. Asimismo, que los alumnos identifiquen elementos comunes en el desarrollo de programas como las entradas, las salidas, las variables, los operadores aritméticos y lógicos entre otras cosas.

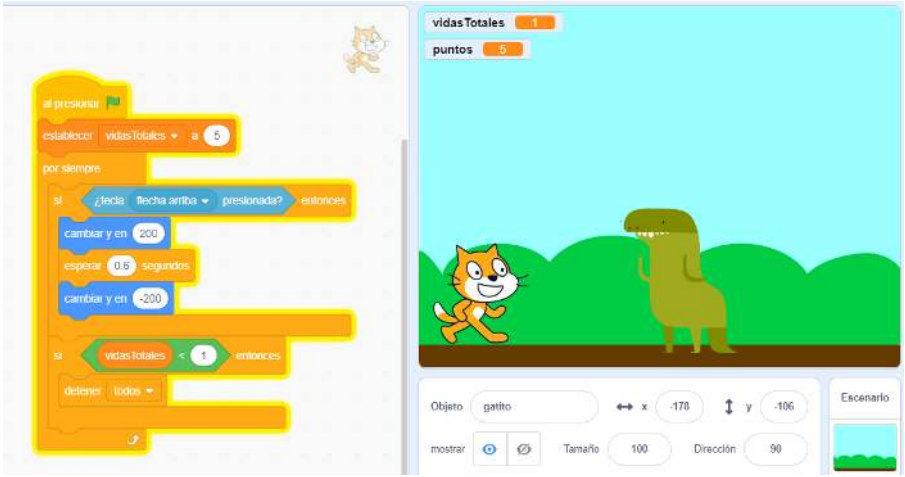


Figura 1.1: Ejemplo de un juego realizado en clase con Scratch.

1.2.2 Charlas sobre el uso del pensamiento computacional

En este frente se realizaron varias charlas con el objetivo de mostrar a los estudiantes el proceso de resolución de problemas tanto en proyectos de investigación como en empresas. Además, este frente logra una sinergia entre dos mundos tradicionalmente separados, por un lado, el de los profesionistas y científicos, y por otro el de los estudiantes de preparatoria. Asimismo, permite ejemplificar a los educandos el concepto de aldea global, donde las oportunidades y las profesiones de su interés pueden estar en otras regiones diferentes a su localidad con lo cual se apoya a ampliar el horizonte de opciones para los estudiantes.

Particularmente, las charlas tuvieron una duración aproximada de 30 a 40 minutos con oportunidad para preguntas y respuestas. Se recomendó a los ponentes orientar la charla hacia cómo abordaron su proyecto de investigación o cómo abordan problemas en las empresas en las que laboran, cómo diseñaron la estrategia de solución, algunos resultados y conclusiones. Para ello, se contó con la participación de cinco expertos que ayudaron a los estudiantes para reforzar los conceptos vistos en clase y ejemplificar la forma en la cual el pensamiento computacional es aplicado en la práctica.

Las charlas estuvieron relacionadas con el Uso del Kinect para motivar el aprendizaje de matemáticas [10], diagnóstico del cáncer mamario [11], sonificación de señales de electroencefalogramas (EEG) [12], comparativa de similitud entre plantas y medicamentos de patente [13] y validación de software. Asimismo, algunos ejemplos de qué actividades realizan y qué retos tienen que resolver en empresas como Oracle, Banco Azteca y Axxis Solutions. En próximas sesiones también se invitará

a un médico para que platique acerca del proceso que sigue para poder diagnosticar un padecimiento.

Estas charlas fueron de mucha utilidad para ejemplificar varios conceptos del pensamiento computacional desde la abstracción, la descomposición de problemas, el diseño de algoritmos y el reconocimiento de patrones. Algo positivo para los estudiantes es que se dieron cuenta de cómo el pensamiento computacional se implementa en áreas diversas y que en los proyectos de investigación convergen varios campos del conocimiento y no sólo computación. Asimismo, en el caso de los ponentes de proyectos de investigación tienen un primer acercamiento al método científico aplicado. Lo anterior es de gran importancia para la materia de Taller de investigación. Por último, este frente permite que los alumnos interactúen con profesionistas expertos en vivo y ayuda a que vayan mejorando sus habilidades de comunicación.

1.2.3 Proyecto integrador

En este frente, íntimamente relacionado con el anterior, los alumnos identificaron un problema para el cual desarrollaron un proyecto de solución como parte de su estancia en la preparatoria y que sea de su interés. En este nivel académico, la innovación se fomentará, pero el objetivo inicial es que tengan un primer acercamiento a un problema de la realidad y que puedan proponer una estrategia de solución. Asimismo, que los alumnos intenten segmentarlo para identificar los subproblemas involucrados. Con base en ello, que puedan ser capaces de aprovechar soluciones planteadas tanto por ellos como soluciones previas del estado del arte, que les puedan facilitar el logro de su proyecto. También los alumnos serán capaces de identificar los elementos más relevantes para dicha solución, así como los pasos que serían necesarios para llegar a implementarlo. En esta estrategia se buscará que los alumnos desarrollen habilidades del pensamiento crítico para ayudarlos a identificar y entender el problema a abordar con el pensamiento computacional, ir delimitando y analizando la factibilidad de su proyecto, así como poder hacer contrastes entre las soluciones planteadas por ellos y soluciones previas del problema que eligieron.

Una parte que ha sido fundamental para poder guiar a los estudiantes es la realización periódica de charlas para ayudar a delimitar los proyectos, para que puedan ser guiados y desarrollados con las materias por estudiar en la preparatoria. Como resultado y acorde al nivel académico de los estudiantes, se espera un prototipo que ilustre la factibilidad de su proyecto. En aras de la innovación, nuestra hipótesis es que ésta surgirá con base en la experiencia que el alumno vaya teniendo con las versiones de su proyecto, retroalimentación del comité académico de la preparatoria y participación en eventos relacionados con ciencia y tecnología como expociencias ()https://www.expociencias.net/).

Por último, el desarrollo de estos proyectos permite que el alumno también se familiarice con la generación de documentos y presentaciones en las se pueden integrar y/o evaluar lo aprendido en otras materias del semestre como: Taller de Investigación, Inglés, Taller de Lectura y Redacción. Así como otras materias novedosas como Pensamiento Crítico y Caricatura. Por lo tanto, con base en un acuerdo académico hemos llegado a la conclusión de contar con dos etapas: elaboración de un reporte del proyecto y una exposición final altamente centrada en caricaturas hechas por ellos para explicar tanto el problema a abordar como el proyecto con el cual pretenden abordar dicho problema.

1.2.4 Evaluaciones de los frentes

Evaluación del 1er frente

La evaluación de la materia de pensamiento computacional nos ha permitido dar seguimiento paulatino del grado en el que se fueron alcanzando los objetivos planteados. Para ello, en el primer frente, llamado introducción al pensamiento computacional, se diseñaron y asignaron tareas que lograron introducir a los alumnos de forma natural al pensamiento computacional. Se buscó que lograran describir situaciones de la vida diaria, identificar problemas e ir dividiéndolos en partes más pequeñas, de forma ordenada y mediante la jerarquía que presenta el problema. De igual forma, se les asignaron actividades donde lograron reconocer patrones, realizar cálculos y conteos que involucraron el uso de la imaginación de perspectivas, análisis y rapidez mental. Además, diseñaron prácticas que les permitieron reforzar las estructuras de control, mediante el uso de las herramientas: la plataforma electrónica Arduino, HourofCode y Scratch.

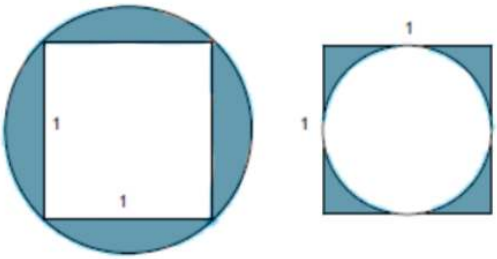
Por último, adicional al conjunto de tareas y prácticas, los alumnos realizaron 2 exámenes bimestrales para supervisar el avance y el apropiamiento de las habilidades de la materia. En la Figura 1.2 se observa dos ejemplos de preguntas de un examen de la materia.

Evaluación del 2º frente

Paralelamente a lo anterior, se comenzó a trabajar en el frente dos, el cual consistió en que los alumnos tuvieron charlas con expertos sobre el uso del pensamiento computacional, como una herramienta fundamental para solucionar problemas reales en diversas áreas. Además durante dichas charlas hubo la oportunidad para que los alumnos aclararan sus dudas, y el profesor hiciera hincapié en qué habilidad del pensamiento computacional estaba siendo usada durante algún momento específico de las exposiciones.

La evaluación de lo aprendido en este frente se realizó al incluir algunas imáge-

2. Describe todos los pasos que seguirías para poder determinar en cuánto excede el área sombreada de la figura X al área sombreada de la figura Y. No es necesario realizar cálculos sólo que todos los pasos estén completos (**valor 10 pts**).



3. Analice su proyecto final e identifique un ejemplo en el que tendrá que utilizar una estructura de selección (Si-entonces o Si-entonces-SiNo), y un ejemplo de alguna estructura de repetición (para, mientras que, hasta que, repetir) para que su sistema (programa, sistema, app, robot, drone, etc) pueda operar correctamente (**valor 10 pts**).

Figura 1.2: Ejemplos de preguntas en un examen de pensamiento computacional.

nes presentadas en las charlas para evaluar en el examen 1 conceptos del pensamiento computacional. Adicionalmente, éstas sirvieron como base para los alumnos para saber cómo proponer, desarrollar y presentar sus propios proyectos.

Evaluación del 3er frente

La evaluación en el tercer frente, proyecto integrador, se comenzó a realizar desde una etapa temprana, con el objetivo de poder guiar a los alumnos; de forma que el avance de sus proyectos fuera constante. Con ello, también se buscó que el conocimiento y las habilidades adquiridas de las materias que complementaron el primer semestre de preparatoria, pudieran converger en el desarrollo del proyecto. Para ello, se elaboró una calendarización de revisiones que fueron incluyendo de manera incremental los contenidos del documento de propuesta de cada proyecto y un borrador de su exposición final. Se programaron y realizaron revisiones cada semana durante un mes. En estas revisiones los alumnos presentaron sus avances frente a un panel de revisores que evaluaron el trabajo realizado e, hicieron sugerencias y observaciones que consideraron necesarias. Cabe mencionar que en dichas reuniones se llegó a invertir desde media hora hasta una hora por alumno. Además, en las materias de pensamiento computacional y pensamiento crítico se realizó la retroalimentación necesaria derivada de cada reunión semanal.

Una vez concluido el periodo de elaboración del documento de propuesta, se

asignó una semana más para que los alumnos trabajaran en la puesta a punto de su presentación final de su proyecto. De igual forma, se realizaron dos sesiones en las cuales se hicieron observaciones y sugerencias para mejorar dicha presentación.

Por último, para la evaluación del proyecto se consideraron el documento de propuesta y el desempeño de la presentación. Para la presentación final se invitó a todos los docentes de la institución (incluyendo a los de educación básica) y a padres de familia. Todo ello, con el objetivo de dar a conocer el trabajo realizado y desarrollar habilidades de comunicación en los alumnos.

El instrumento de evaluación para la presentación consistió de una escala numérica que incluyó siete preguntas estrechamente relacionadas con el desarrollo del proyecto. Se contó con cinco evaluadores y se obtuvo un promedio grupal de 8.76. Mientras que los docentes responsables del grupo evaluaron los documentos de propuesta de cada alumno, obteniendo un promedio grupal de 7.98 (ver anexo 2 para criterios de evaluación del documento). Con ambos resultados se obtuvieron las evaluaciones generales del proyecto alcanzando un promedio general grupal del proyecto de 8.3.

En la Figura 1.3 se puede apreciar una diapositiva del proyecto final de una de las estudiantes cuyo proyecto está relacionado con la limpieza semiautomática de ríos. En esta se puede observar que se integran varias habilidades adicionales al pensamiento computacional como el inglés, la caricatura, el taller de investigación.

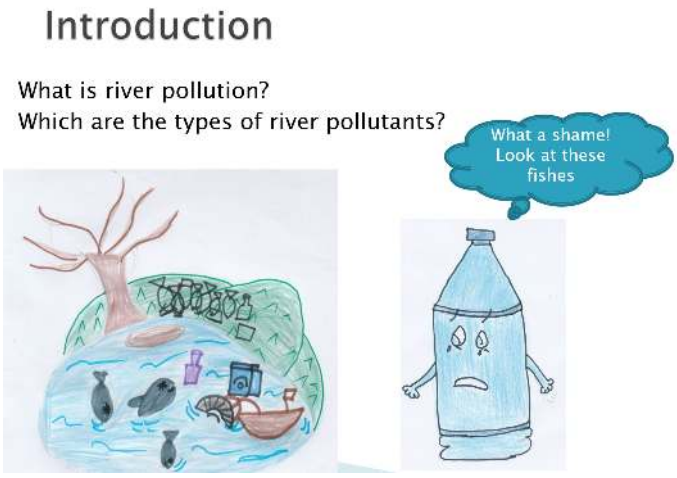


Figura 1.3: Ejemplo de diapositiva de la exposición de un proyecto final.

A continuación, se presentan de manera general los proyectos desarrollados por los alumnos, en esta primera etapa de pensamiento computacional como materia básica de preparatoria. De igual forma se describen los logros y avances de algunos

de los proyectos involucrados.

1.3 Resultados de los proyectos integradores

En el primer semestre de esta estrategia de enseñanza del pensamiento computacional en preparatoria se tiene como objetivo que los alumnos puedan comprender un problema a tratar, y segmentarlo para poder encontrar un nicho de contribución para proponer una solución. Asimismo, los alumnos identificarán los pasos que podrían ser necesarios para llevar a cabo la implementación de sus proyectos y la evaluación de estos proyectos. A continuación, se describe brevemente los proyectos propuestos por los alumnos y algunas gráficas de cómo los visualizan en funcionamiento final.

En este primer curso los estudiantes han identificado que son de su interés los siguientes proyectos

- Aplicación para la verificación de fake news.
- Detección de plagas en cultivos.
- Limpieza semiautomática de ríos.
- Mejora de la enseñanza en primaria (se delimitará a la materia de música).
- Diseño de gimnasio Virtual.
- Seguimiento de la alimentación de personas con diabetes.

En la Figura 1.4 se puede observar una de las pantallas de la propuesta relacionada con hacer una app para la mejora de la enseñanza en el nivel primaria. En esta imagen se ven dos de las actividades que incluirá la app, un memorama musical y una rutina de ritmos. Estas representaciones abstractas de sus soluciones van a incluir sonidos para reforzar el aprendizaje de los alumnos, y una evaluación del progreso de los estudiantes. En cuanto a las rutinas de ritmos se busca que los alumnos mejoren sus habilidades de coordinación e interpretación de símbolos musicales.

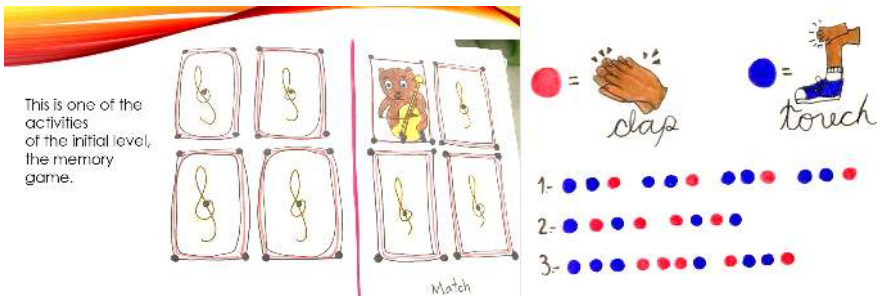


Figura 1.4: Prototipos del proyecto de aplicación para la mejora de la enseñanza en la primaria.

Mientras que en la Figura 1.5 se puede apreciar dos representaciones de cómo

otro estudiante a partir de tener la inquietud de abordar el problema de identificación de plagas en cultivos ha ido segmentando su problema para delimitarlo al problema del barrenador del maíz. Particularmente, la Figura 1.5 muestra cómo el estudiante visualiza tanto el problema del barrenador del maíz como el funcionamiento final de su proyecto. Con esto se refuerza la habilidad de abstracción del alumno.

Figura 1.5: Prototipo del proyecto para abordar el problema de la detección de plagas en cultivos del maíz.

También se puede observar en la Figura 1.5 cómo el estudiante logra identificar que para su proyecto sería relevante analizar las imágenes de las plantaciones, pero no sólo en el rango visible sino con otro tipo de imagen (hiper-espectrales o térmicas) que pudiera darle mayor información con respecto a imágenes comunes.

Por otra parte, en la Figura 1.6 se observa la propuesta de solución de la estudiante que abordó el problema de la limpieza semiautomática de ríos. Como se puede observar, del problema inicial de limpiar de manera semi-autónoma los ríos, ha identificado que de los contaminantes presentes en un río se enfocará en el plástico y, dentro de esta rama de contaminantes, sólo se enfocará en las botellas. Además, se observa que para ella es relevante que dicho prototipo cuente con un sistema de cámaras para ayudar a la navegación y al proceso de identificación de las botellas de plástico. Además de un mecanismo de recolección parecido al de una aspiradora pero con una estructura que le permita ciertos grados de libertad.

En la Figura 1.7, se observa la metodología propuesta por esta misma estudiante para ayudar a limpiar de manera semi-autónoma los ríos. Para lograr cumplir con lo anterior, la estudiante ha identificado los pasos que se observan en la Figura 1.7 para desarrollar un mecanismo que pueda recolectar botellas de plástico respecto de otros residuos en ríos. Con esta metodología la alumna demuestra que es capaz, con apoyo de las supervisiones de los profesores, de dividir el problema global en problemas más pequeños (descomposición del problema) en un primer nivel de detalle.

Figura 1.6: Prototipo del proyecto para abordar el problema de la limpieza semi-automática de ríos.

Por lo que queda pendiente, seguir trabajando en estos subproblemas para seguirlos refinando y eventualmente, llegar a la implementación/diseño de un algoritmo.

La metodología muestra también cómo la estudiante identificó algunos problemas similares (*reconocimiento de patrones*) que están inmersos en su solución como la navegación del mecanismo, el procesamiento de imágenes para poder discriminar las botellas de plástico con respecto a otros residuos, entre otros.

1.4 Conclusiones

En este artículo se aborda la pregunta de *cómo introducir los conceptos fundamentales e impartir la materia de pensamiento computacional en el nivel bachillerato*. Para ello, se presenta una guía preliminar de implementación basada en un enfoque de aprendizaje basado en proyectos. Se presenta una estrategia basada en 3 pilares, enseñanza de las habilidades del pensamiento computacional, presentación del uso del pensamiento computacional en áreas diversas a través de ponentes invitados, y proyecto integrador a desarrollarse en la preparatoria. Esta estrategia ha sido aplicada en la primera generación de la prepa Mind Up e impartida en el 1er. semestre de preparatoria.

Hemos logrado dar evidencia de que con el apoyo del profesor de clase y de los profesores afines, un estudiante de preparatoria es capaz de abordar un problema de la sociedad complejo (real) aplicando estrategias del pensamiento computacional para plantear una solución. Además, hemos logrado observar una mejora en

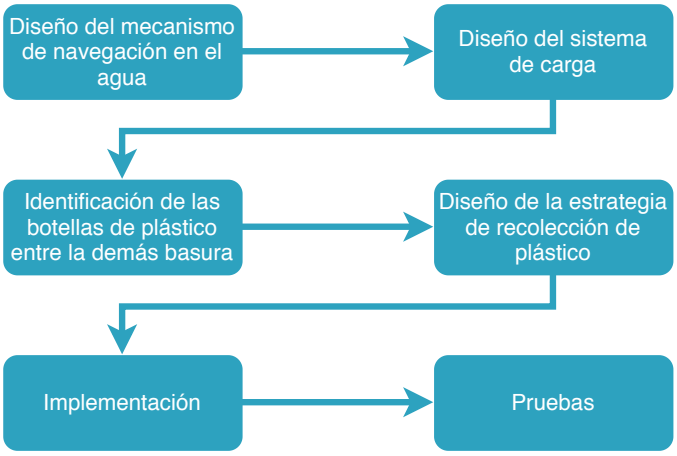


Figura 1.7: Metodología del proyecto para abordar el problema de la limpieza semi-automática de ríos.

los estudiantes reflejada en actitudes como la confianza en ellos, resiliencia, tolerancia, argumentación, discusión, observación, análisis y una mejor comprensión de su entorno. Asimismo, en todo momento, se ha buscado crear un ambiente en el cual los alumnos traigan sus propias ideas y observen que hay diferentes soluciones para abordar un problema o proyecto dado, mitigando el enfoque de respuesta única/correcta y el pensamiento único.

Con lo desarrollado en esta materia es factible poder darles seguimiento a estos proyectos en los siguientes semestres y orientar el contenido de las materias futuras para lograr la implementación de la mayor cantidad de bloques de las metodología propuesta durante la preparatoria. Además, hemos notado que algunos estudiantes tienen sub-problemas comunes, los cuales podrían ser trabajados en equipo.

Otra pregunta relevante de esta investigación es la siguiente: *¿Es el pensamiento computacional una materia integradora de los conocimientos vistos en otras asignaturas del semestre?* Ante la cual se puede concluir que siempre que se siga un enfoque de aprendizaje basado en proyectos y las estrategias de evaluación planteadas, esto es cierto ya que un proyecto da la oportunidad de integrar los conocimientos de materias como taller de investigación, matemáticas, taller de lectura y redacción, e inglés. En el caso de Mind Up, las materias de caricatura y pensamiento crítico también fueron muy relevantes. Además, la materia de pensamiento computacional no sólo favoreció que los estudiantes integraran conocimientos del semestre, sino que también favoreció el trabajo colaborativo de los maestros desarrollando un esquema transdisciplinario, es decir, que los maestros colaboran desde sus disciplinas para un objetivo en común.

Como parte de las actividades futuras en el frente 1, planeamos introducir a los estudiantes en el uso de la herramienta Karel el robot [14], el cual es usado como herramienta en la olimpiada mexicana de informática. Ésta podría ser presentada después de algunos ejercicios básicos en hourofcode. Lo anterior le ofrecería otra ventana a los alumnos para enriquecer su conocimiento mediante la participación en este concurso de programación. Mientras que en el frente 2, se ha considerado que los alumnos realicen un reporte o ensayo acerca de lo aprendido con las charlas con expertos. Finalmente, en el frente 3 se plantea invitar a expertos externos a la presentación final de los proyectos.

Por último, es importante recalcar que los resultados obtenidos podrían verse potenciados si los alumnos llevaran clases de pensamiento computacional o afines durante la educación básica (primaria y/o secundaria), tal como se hace en otros países (como se describe en [1]). Lo anterior favorecería que el tipo de alcance para esta materia en preparatoria sea más práctico que conceptual en este primer semestre. Además, la situación actual relacionada con el COVID-19, ha mostrado que es necesario reforzar el aprendizaje de las TICs en los estudiantes de secundaria y primaria, en cuanto al uso de software de ofimática. Esto con el objetivo de que los estudiantes pudieran cumplir de una forma más pronta las actividades de redacción.

Agradecimientos

Los autores agradecen a la preparatoria "Mind Up Despertando Mentes" de Tuxtla Gutiérrez por las facilidades y la visión para implementar un programa innovador como el actual. Especialmente a los profesores: Roberto Valls Esponda,Vanessa Merlo Bringas, Karen Padilla Padilla, Jacob Pérez Toledo y Josefa Coutiño Megchún. Asimismo, a los estudiantes de la primera generación de la institución: Lucero Margarita Magdaleno García, Isabella Lorenzo Ríos, Diego Escutia Villaseñor, Santiago Trujillo López, Iker Vidaurri Ruiz, y Victor Armando Fernández Corzo. Finalmente, extendemos nuestro agradecimientos al grupo de expertos que compartieron sus experiencias con los estudiantes: Erick González Castañeda, Eréndira Vázquez Palacios, Ingrid Hernández Valencia, Omar Carreño Sánchez y Guadalupe Martínez.

Referencias

[1] Peter Seown y col. Educational Policy and Implementation of ComputationalThinking and Programming: Case Study of Singapore. En: *Computational Thinking Education*. Ed. por Siu-Cheung Kong y Harold Abelson. Springer Nature, 2019, págs. 345-361. ISBN: 978-981-13-6528. DOI: 10.1007/978-981-13-6528-7.

[2] Jorge Luis Zapotecatl López. *Introducción al Pensamiento Computacional: Conceptos Básicos Para Todos*. Ciudad de México: Academia Mexicana de Computación, 2018. ISBN: 978-607-97357-2-2.

[3] Fernando Raúl Alfredo Bordignon y Alejandro Adrián Iglesias. *Introducción al Pensamiento Computacional*. Universidad Pedagógica Nacional y Educar, 2020. ISBN: 978-987-3805-49-3.

[4] Gobierno del Estado de Sonora. *Plan de Estudios*. Colegio de Bachilleres del Estado de Sonora. 2021. URL: http://www.cobachsonora.edu.mx/plan-de-estudios.

[5] José Raymundo Muñoz Islas. *Nuestros Planes y Programas de Estudio*. Centro de Bachillerato Tecnológico Industrial y de Servicios No. 179. 2014. URL: http://www.cbtis179.edu.mx/portal/index.php/informacion-general/oferta-educativa/planes-y-programas.

[6] Colegio de Bachilleres del Estado de Sonora. *Informática I. Formación Básica*. Módulo de Aprendizaje. Hermosillo: Grupo de Servicios Gráficos, 2020. ISBN: 978-607-730-056-4.

[7] Subsecretaría de Educación Media Superior. *Informática II. Programa de Estudios. Segundo Semestre*. Jun. de 2017.

[8] Eduardo Rodríguez-Sandoval, Édgar Mauricio Vargas-Solano y Janeth Luna-Cortés. Evaluación de La Estrategia "Aprendizaje Basado En Proyectos". En: *Educación y educadores* 13.1 (abr. de 2010), págs. 13-25. ISSN: 0123-129.

[9] Phyllis C. Blumenfeld y col. Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. En: *Educational psychologist* 26.3-4 (1991), págs. 369-398. DOI: 10.1080/00461520.1991.9653139.

[10] O. Carreño-Sánchez y col. Desarrollo de Objetos de Aprendizaje Para Motivar El Aprendizaje de Las Matemáticas En Niños de Primer Grado de Primaria Usando Un Sensor Kinect. En: *Research in Computer Science* 60 (oct. de 2012), págs. 75-79.

[11] I. Valencia-Hernandez y col. Description of Breast Density Based on a Homogeneity Representation. En: *2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC 2019) Proceedings*. Auckland, New Zealand: IEEE, mayo de 2019, págs. 1-5. DOI: 10.1109/I2MTC.2019.8826913.

[12] Erick F. González-Castañeda y col. Sonification and Textification: Proposing Methods for Classifying Unspoken Words from EEG Signals. En: *Biomedical Signal Processing and Control* 37 (2017), págs. 82-91. DOI: 10.1016/j.bspc.2016.10.012.

[13] Eréndira Palacios Vázquez. Análisis de La Similitud Molecular Entre Fármacos y Plantas Medicinales Aplicando Espectroscopía Raman. Tesis de mtría. México: Instituto Nacional de Astrofísica, Óptica y Electrónica, 2019.

[14] Richard E. Pattis. *Karel the Robot: A Gentle Introduction to the Art of Programming*. 1st edition. New York: Wiley, jul. de 1994. ISBN: 978-0-471-08928-5.

Capítulo 2

Diseño de intervención del pensamiento computacional para el fortalecimiento de rendimiento académico en los recién ingresantes a educación superior

Ronald Paucar Curasma [1] y *Arturo Rojas López* [2]

[1] Universidad Nacional Autónoma de Tayacaja Daniel Hernández Morillo. Perú.
rpaucarc@unat.edu.pe

[2] Universidad Tecnológica de Puebla.
arturo.rojas@utpuebla.edu.mx

Resumen. Los autores proponen el diseño de intervención del pensamiento computacional para la mejora del rendimiento académico en los recién ingresantes a educación superior. Se han definido cuatro categorías para el desarrollo del pensamiento computacional en relación con los contenidos temáticos del curso Gestión de la información. Las habilidades de abstracción, generalización, descomposición, evaluación y diseño algorítmico forman parte de una evaluación diagnóstica para determinar a qué categoría pertenece. La intervención está basada en la consideración del pensamiento computacional como un proceso que involucra la resolución de problemas dentro de los diferentes temas de estudio relacionados a las problemáticas reales de la zona Huancavelica-Perú. Esta forma de trabajo motivará al estudiante, porque se sentirán parte del equipo para resolver la problemática real de la zona y como consecuencia el incremento del rendimiento académico en los estudiantes.

Palabras clave. Pensamiento computacional, rendimiento académico, educación superior, categorías, tema de estudio.

2.1 Introducción

El pensamiento computacional (PC) "involucra la solución de problemas, diseño de sistemas y la comprensión de la conducta humana, basándose en los conceptos fundamentales de las ciencias de la computación" [1]. En semejanza con las aportaciones que realiza la lectura, la escritura y la aritmética en la formación académica inicial de una persona, aporta capacidad analítica cuando se trata de resolver un problema de la vida real. Un componente principal, con respecto a otros pensamientos, es el valor de la abstracción, es decir, la habilidad para discernir qué detalles o información de un problema son importantes y cuáles se pueden omitir [2]. En consecuencia, fortalece el planteamiento de los problemas y cultiva la capacidad para resolverlos. Tales soluciones, se pueden representar como algoritmos computacionales complejos o una colección de soluciones de menor complejidad a partir de una división del problema [3]. En recientes definiciones, se afirma que el PC no sólo se caracteriza por habilidades, sino también por el desarrollo de competencias en las personas; tales como la competencia para lidiar con problemas abiertos y difíciles, persistencia en el trabajo y confianza en el tratamiento de la complejidad [4]. Por lo anterior, es clave en el quehacer de resolver problemas no estructurados, así como comprender e interpretar datos, y finalmente comunicar información a otros mediante el uso de la computadora o por otros agentes [5]. Finalmente, el pensamiento computacional es aplicado en diferentes áreas del conocimiento como las ciencias, el periodismo, la geografía, los negocios, el medio ambiente, la ingeniería y particularmente la investigación; siendo más aplicado en unas áreas que en otras.

Algunos autores [6] se enfocaron en desarrollar las habilidades del PC de los estudiantes universitarios en el curso de ciencias biológicas de la universidad de Tel-Aviv en Israel. Los estudiantes fueron expuestos al "contexto" abstracto, algorítmico y lógico de las ciencias de la computación, y familiarizados con ideas y fundamentos computacionales. La programación contribuyó principalmente como un medio para practicar algunas de las habilidades del pensamiento computacional en la biología, aplicando enfoques computacionales para el diseño de experimentos, así como a la creación, la inclusión, el análisis de datos y al modelado respectivo de fenómenos biológicos. En conclusión, cuando el aprendizaje incluye concretamente experiencias prácticas como en el curso de biologías, las habilidades del pensamiento computacional se adquieren y los conceptos subyacentes se entienden mejor.

En la Facultad de Ciencias de la Educación de la Universidad Nacional de San Agustín, Perú, se vienen implementando actividades dirigidas a la formación en y para la investigación, con elementos tales como flexibilidad, pertenencia, formación ética y humanista, identidad y ejercicio de la ciudadanía; basándose en un perfil profesional que responde a la realidad actual [7]. Por lo anterior, el currículo se convierte en el eje de construcción de las estrategias de enseñanza-aprendizaje,

que junto a la experiencia formativa e investigadora contribuyen al desarrollo de las competencias para la investigación. El presente trabajo expone la planeación de integrar la ejercitación del pensamiento computacional dentro de un curso de gestión de la información, debido a la fuerte convicción de su impacto positivo aplicado a diferentes áreas o disciplinas.

La propuesta se establece como un componente del proyecto titulado Investigación formativa y pensamiento computacional en el fortalecimiento de las competencias investigativas y el rendimiento académico de los alumnos de la Universidad Nacional Autónoma de Tayacaja (UNAT) – Perú. La pregunta de investigación involucrada dentro del proyecto mencionado es: ¿Cómo influye la combinación de la investigación formativa y el pensamiento computacional en el fortalecimiento de las competencias investigativas y rendimiento académico de los estudiantes de la Universidad Nacional de Tayacaja?

El artículo está organizado de la siguiente forma con el objetivo de responder a la pregunta de investigación una vez que sea posible realizar la intervención del diseño experimental. En la sección de metodología, se establecen las condiciones de la investigación que permitan obtener los datos para su análisis. La sección diseño de intervención contiene la propuesta a partir de establecer la relación de las habilidades del pensamiento computacional con las competencias del curso gestión de la información. Lo anterior a partir de establecer qué conceptos del PC influyen con el contenido del curso y podrán impactar en el rendimiento académico. Finalmente, se exponen las conclusiones del artículo destacando el trabajo a futuro por realizar, la ventaja y propuesta establecida con el proyecto.

2.2 Metodología

Durante el desarrollo de la investigación, se utilizará el enfoque mixto cuantitativo y cualitativo. Se utilizarán técnicas cuantitativas de recogida de datos, y el cualitativo [8] que plantea que los hechos son investigados después de que hayan ocurrido [9]. Para la recolección de datos, se utilizará un instrumento basado en encuestas; así como entrevistas. Para la etapa de evaluación se procesarán los datos utilizando estadística inferencial correlacional entre el pensamiento computacional y el rendimiento académico.

La evaluación del rendimiento académico se realizará en base a las calificaciones que los alumnos obtendrán en el curso de Gestión de la Información del II ciclo de las 4 carreras profesionales (Ingeniería Industrial, ingeniería de industrias alimentarias, ingeniería forestal y ambiental, y enfermería) con una población de 156 estudiantes aproximadamente por semestre.

Con respecto al diseño de investigación, se considerarán grupos de control y experimentales; donde, se verificará que el rendimiento académico obtenido estará

asociado con el desarrollo de habilidades del pensamiento computacional (grupo experimental) y los que carecieron del desarrollo de pensamiento computacional (grupo de control); donde, como resultado se contará con datos comparativos que permitirán encontrar la relación a través de cálculos estadísticos.

A continuación, se proponen los componentes y actividades a desarrollarse durante la ejecución del proyecto de investigación.

2.3 Diseño de intervención

El desarrollo del pensamiento computacional, por medio de un taller donde se ejerciten las habilidades a partir de la comprensión de los conceptos y su impacto en la resolución de problemas, permitirá compensar el proceso de enseñanza-aprendizaje y las competencias investigativas en los estudiantes del grupo experimental en el curso gestión de la información del II ciclo. A la fecha, existen diversas experiencias en el desarrollo del pensamiento computacional en los tres niveles de la educación (primaria, secundaria y superior) [10]. Para el caso de estudio y de acuerdo a las variables de pensamiento computacional y el rendimiento académico, se utilizará la definición de pensamiento computacional propuesta de Selby [11], que incluye las habilidades de abstracción, descomposición, diseño algorítmico, generalización y evaluación.

El primer resultado de la propuesta de intervención está determinado por la relación de habilidades del pensamiento computacional con competencias del curso gestión de la información. La Tabla 2.1 contiene la información indicando además el reactivo que evalúa la habilidad respectiva del pensamiento computacional.

Como una actividad diagnóstica, se realizará una evaluación del pensamiento computacional a través del uso de los reactivos propuestos de los resultados de investigación reportados previamente [12] para las preguntas etiquetadas como Móviles, Canguro y Espías [13], para los ejercicios de Castores y Salto de charcos [14] lo cual es aplicable a otras disciplinas, tales como el área de salud, así como el área de humanidades.

Posterior a la evaluación diagnóstica, se ofertarán los proyectos del curso gestión de la información con base a los reactivos correctos que obtengan los estudiantes, en consecuencia, se establecen las siguientes cuatro categorías.

Categoría 1: 5 reactivos correctos

Los estudiantes ubicados en esta categoría contestaron correctamente todas las preguntas. Los estudiantes que obtengan este resultado de la evaluación del pensamiento computacional, indican una alta capacidad de resolución de problemas. En esta categoría se fortalecerán más las 5 habilidades mediante los temas de estudio

Tabla 2.1: Relación de habilidades, reactivo y competencias.

		Competencia	
Habilidad	Reactivo	General	Específica
Abstracción	Canguro		Discriminar entre diversos gestores de la información.
Diseño algorítmico	Castores	Realizar aprendizaje autónomo de manera eficaz y eficiente.	Gestionar información para su comunicación.
Evaluación	Salta charcos	Aplicar pensamiento crítico en la indagación, análisis e interpretación de temas de su formación profesional. Demostrar las etapas del flujo de la información en casos propios de su campo profesional.	
Descomposición	Móviles	Realizar aprendizaje autónomo de manera eficaz y eficiente.	Investigar y acceder a bibliotecas virtuales y bases de datos de informaciones nacionales y extranjeras
Generalización	Espías	Utilizar diversos gestores de la información.	Utilizar diversos gestores de la información. Experimentar y comparar diversos sistemas de almacenamiento de resultados. Establecer un patrón de búsqueda para su reutilización en diferentes contextos.

con cierto nivel de complejidad en su ejecución; el tema propuesto es el "estudio de los colegios secundarios tecnológico en la provincia de Tayacaja".

Categoría 2: Ningún reactivo correcto

Cuando los estudiantes no obtuvieron algún reactivo correcto, el proyecto correspondiente debe ser básico, pero con el cumplimiento total de las competencias del curso gestión de la información, además de considerar una atención cuidadosa de las actividades del taller de ejercitación del pensamiento computacional.

Categoría 3: Incorrecto el ejercicio de móviles o canguro o castores

En esta categoría se reforzará las habilidades de descomposición, el diseño algorítmico y la evaluación. Como, en esta categoría se estima mayor cantidad de estudiantes, se considerarán varios temas de estudio: "representación de la malla curricular o plan de estudios de la UNAT", "representación de las rutas a las provincias la región Huancavelica (provincia y distrito)" y "representación de los procesos de contrata de servicios de internet en Pampas"

Categoría 4: Incorrecto el ejercicio de espías o salto charcos

En esta categoría se reforzarán las habilidades de generalización y evaluación. El tema de estudio propuesto es "evaluación de empresas importantes de la región de Huancavelica"

Los resultados de incluir el pensamiento computacional en el curso de gestión de la información, son el desarrollo de las siguientes capacidades: Diseñar soluciones a los problemas (aplicación de la abstracción, capacidad de automatización de procesos, creación de algoritmos, recopilación y análisis de datos); implementar diseños (programación según corresponda); probar y depurar; modelar, ejecutar simulaciones, hacer análisis de sistemas, reflexionar sobre la práctica y la comunicación [15].

El post-test del pensamiento computacional será de nivel similar del pre-test (evaluación diagnóstica), tanto en nivel de complejidad para evitar obtener puntuaciones sesgadas y estarán relacionados con el estándar de los estudiantes universitarios del ciclo II.

2.4 Conclusiones

Se han definido cuatro categorías para el desarrollo del pensamiento computacional a partir de una evaluación diagnóstica. Se considera dicho desarrollo como un

proceso de resolución de problemas [16, 17] mediante los diferentes temas de estudio relacionados a las problemáticas reales de la zona, lo cual representa una aportación al establecer una forma de revelar las habilidades necesarias para el pensamiento computacional, en términos de tareas dentro del curso Gestión de la información.

Las categorías diseñadas, generarán motivación y trabajo en equipo, por tratar de resolver problemas reales de su zona, y los resultados repercutirán en el fortalecimiento de las habilidades del pensamiento computacional y como consecuencia la mejora en el rendimiento académico de los estudiantes. Lo anterior expone la ventaja principal del trabajo de investigación.

La evaluación del rendimiento académico se basa en los indicadores planteados en el silabo del curso de Gestión de la Información, donde, la evaluación se realiza en cada una de las cuatro unidades didácticas.

Este proyecto, es la primera experiencia en la zona, donde, se ejecutará el proyecto de investigación y única en la región de Huancavelica del Perú.

Referencias

[1] Jeannette M Wing. Computational Thinking. En: *Communications of the ACM* 49.3 (2006), págs. 33-35.

[2] Shuchi Grover y Roy Pea. Computational Thinking in K-12: A Review of the State of the Field. En: *Educational Researcher* 42.1 (2013), págs. 38-43. ISSN: 0013-189X. DOI: 10.3102/0013189X12463051.

[3] V. Alfred Aho. Computation and Computational Thinking. En: *Computer Journal* 55.7 (2012), págs. 833-835. ISSN: 0010-4620. DOI: 10.1093/comjnl/bxs074.

[4] David Weintrop y col. Defining Computational Thinking for Mathematics and Science Classrooms. En: *Journal of Science Education and Technology* 25.1 (2016), págs. 127-147. ISSN: 1059-0145. DOI: 10.1007/s10956-015-9581-5.

[5] Irene Lee, Fred Martin y Katie Apone. Integrating Computational Thinking Across the K–8 Curriculum. En: *ACM Inroads* 5.4 (2014), págs. 64-71. ISSN: 21532184. DOI: 10.1145/2684721.2684736.

[6] Amir Rubinstein y Benny Chor. Computational Thinking in Life Science Education. En: *PLoS Computational Biology* 10.11 (2014). DOI: 10.1371/journal.pcbi.1003897.

[7] Osbaldo Turpo-Gebera y col. Formative Research at the University: Meanings Conferred by Faculty at an Education Department. En: *Educacao e Pesquisa* 46 (2020), págs. 1-18. ISSN: 0000000175. DOI: 10.1590/S1678-4634202046215876.

[8] Rocío Gordillo y col. *Metodología de La Investigación Educativa: Investigación Ex Post Facto*. Inf. téc. Madrid, España: Universidad Autónoma de Madrid, 2010.

[9] Manuel Álvarez. Metodología de La Investigación Educativa. En: *Revista Mexicana de Investigación Educativa* 10.25 (2005), págs. 593-596.

[10] Karen Brennan y Mitchel Resnick. New Frameworks for Studying and Assessing the Development of Computational Thinking. En: *Proceedings of the 2012 Annual Meeting of the American Educational Research Association*. Vancouver: American Educational Research Association, 2012, págs. 1-25.

[11] Cynthia C. Selby. Relationships: Computational Thinking, Pedagogy of Programming, And Bloom's Taxonomy. En: *WiPSCE '15: Proceedings of the Workshop in Primary and Secondary Computing Education*. Proceedings of the Workshop in Primary and Secondary Computing Education. London, United Kingdom: Association for Computing Machinery, 2015, págs. 80-87. DOI: 10.1145/2818314.2818315.

[12] Brebas. *What Is Bebras*. Brebas: International Challenge on Informatics and Computational Thinking. 2015. URL: https://www.bebras.org/.

[13] Computer Olympiad South Africa. *Talent Search*. Computer Olympiad – Computer Olympiad South Africa. 2017. URL: https://olympiad.org.za/talent-search/past-papers/online-trial/.

[14] A. Rojas-López y F. J. García-Peñalvo. Learning Scenarios for the Subject Methodology of Programming From Evaluating the Computational Thinking of New Students. En: *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje* 13.1 (feb. de 2018), págs. 30-36. ISSN: 1932-8540. DOI: 10.1109/RITA.2018.2809941.

[15] Valerie Barr y Chris Stephenson. Bringing Computational Thinking to K-12. En: *ACM Inroads* 2.1 (2011), pág. 48. ISSN: 2153-2184. DOI: 10.1145/1929887.1929905.

[16] Filiz Kalelİoğlu, Yasemin Gülbahar y Volkan Kukul. A Framework for Computational Thinking Based on a Systematic Research Review. En: *Baltic Journal of Modern Computing* 4.3 (2016), págs. 583-596.

[17] Gloria Cecilia Rios. SCRATCH + ABP, Como Estrategia Para El Desarrollo Del Pensamiento Computacional. Master Thesis. Medellin, Colombia: Universidad EAFIT, 2015.

Capítulo 3

Alfabetismos digitales y pensamiento computacional

Rafael Morales Gamboa [1] y *Alberto Pacheco-González* [2]

[1] Universidad de Guadalajara.
rmorales@suv.udg.mx

[2] TecNM campus Chihuahua.
alberto.pg@chihuahua.tecnm.mx

Resumen. En este artículo se presenta una definición de alfabetismo en término de tres componentes: materiales, competencias y procesos de desarrollo de éstas. Sobre esta base, se presenta una categorización de los alfabetismos digitales que se derivan de las características fundamentales de las tecnologías digitales, la selección de competencias necesarias para el uso adecuado de las mismas y de los métodos para su desarrollo. Finalmente, se discute el rol que juega el pensamiento computacional en cada categoría de alfabetismo digital, considerando una definición del primero en términos de tres componentes: resolución de problemas, abstracción y automatización.

Palabras clave. Tecnología digital, alfabetismo digital, pensamiento computacional, abstracción, automatización.

3.1 Introducción

De acuerdo con el Diccionario de la Lengua Española de la Real Academia Española, el término 'alfabetismo' hace referencia a las capacidades básicas de leer y escribir [1], la educación, como medio, y la comunicación, como fin, con una acepción propia de una era en la que el lenguaje escrito se había convertido en medio clave para la comunicación, más allá de la expresión oral y distinto a la expresión artística, la primera de uso generalizado y la segunda restringida a un segmento de población

mucho menor. Más recientemente, su significado se ha ampliado y diversificado a través de la integración de la colección de habilidades que permiten 'expresar, explorar, cuestionar, comunicar y entender el flujo de ideas' entre personas, en un contexto dado, o bien 'interpretar, reflexionar, interrogar, teorizar, investigar, explorar, investigar y preguntar... actuar sobre y transformar dialógicamente... el mundo social', en un sentido más Freireano, siendo las condiciones del contexto las que determinan las habilidades propias del alfabetismo [2].

Por otra parte, nuestro entorno de vida tiene ahora un componente nuevo que hace extremadamente fácil almacenar información y transmitirla de manera instantánea, masiva y global: las llamadas Tecnologías de Información y Comunicación (TIC). Consecuencia de ello, nuestro entorno de vida nos ofrece hoy en día facilidades de creación e intercambio de información, de telecomunicación y teleinteracción a escala global nunca antes vistas, que hemos aprovechado —de manera desigual, dadas las condiciones económicas, sociales y culturales de cada población— para construir una sociedad global de abundancia de información. Se puede hablar entonces de la necesidad de una nueva alfabetización, adaptada a las condiciones del nuevo entorno digitalizado, dando lugar a la construcción del concepto de *alfabetismo digital*.

El concepto que nos convoca, sin embargo, es el de *pensamiento computacional*, planteado inicialmente por Jeannette Wing en el ámbito de las ciencias computacionales como la competencia de aplicar sus conceptos, métodos y estrategias en diversos ámbitos de la vida [3], de modo que este artículo busca clarificar los puentes entre ambos conceptos, alfabetismo digital y pensamiento computacional, bosquejados inicialmente por Wing.

El artículo comienza con un breve análisis de las tecnologías digitales, a fin de establecer sus características fundamentales, que a su vez permiten desglosar el concepto de alfabetismo digital en cuatro categorías relativamente independientes que organizan las tendencias en su desarrollo teórico y práctico. El artículo cierra con el establecimiento de las conexiones, o falta de ellas, entre las distintas categorías de alfabetismo digital y el pensamiento computacional, y la discusión sobre sus implicaciones.

3.2 Tecnología digital

Subyacentes a las facilidades que ofrecen las TIC se encuentran sus tres características distintivas:

1. Toda la información se codifica usando un esquema de *representación simbólica* mínima, que corresponde a agrupar cantidades varias de los dígitos binarios 0 y 1.

2. El esquema de representación de información se puede usar para la *representación de procesos*.
3. Incorpora mecanismos capaces de interpretar la representación de los procesos y realizar las operaciones en la información *sin* la intervención de seres humanos.

La primera computadora de propósito general que representaba y operaba información en formato digital —esto es, usando un número finito de símbolos interpretados como valores enteros consecutivos empezando por cero, o *dígitos*, generalmente dos, o y 1, los cuales suelen ser interpretados también como *falso* y *verdadero*— fue construida y presentada al mundo en 1936 por el matemático inglés Alan Turing [4] pero, a diferencia de las computadoras que vemos todos los días, se trata de una construcción matemática abstracta. No obstante, la teoría acumulada desde entonces nos dice que no hay nada que una computadora física, "de verdad" (de cualquier marca, modelo o capacidad), haya podido o pueda hacer que no pueda ser llevado a cabo también por una máquina como la de Turing.

Lo que esto nos dice es que las computadoras en general (ej. servidores, computadoras de escritorio, portátiles, tabletas, teléfonos inteligentes y sistemas embebidos) tienen dos características fundamentales íntimamente relacionadas pero claramente distintas: son digitales y son electrónicas.

1. Lo *digital* es una propiedad simbólica o abstracta que comparten con las máquinas de Turing, porque representan la información usando símbolos que se combinan y organizan para formar arreglos finitos usando reglas bien definidas.
2. Lo *electrónico* se refiere a la manera actual en que estos símbolos son representados y transmitidos en un medio físico: mediante cambios de estado y flujos de energía que se mantienen y fluyen por medios diversos tales como electromagnéticos, ópticos y, más recientemente, cuánticos[1].

La naturaleza digital de las llamadas tecnologías de información y comunicación (TIC) define algunas de sus características más sobresalientes. La representación digital de información y procesos es simbólica; esto es, discreta (en piezas claramente identificables y contables), abstracta (los símbolos no reflejan necesariamente las propiedades de sus significados), exacta o tan precisa como sea necesario, lo cual hace que se pueda reproducir sin pérdida (el original y la copia son indistinguibles), modificar y ejecutar con precisión absoluta. Por otra parte, la naturaleza electrónica de las TIC las provee de su extrema maleabilidad, tanto en términos de flexibilidad como de velocidad de transformación. La información representada en estados de energía se opera (modifica, copia, transfiere) sin mucho esfuerzo y a gran veloci-

[1] Más específicamente, estaríamos hablando de lo electrónico, optoelectrónico y cuántico, respectivamente. En el contexto de este documento se usa el término genérico 'electrónico' para hacer referencia a todas y cada una de estas formas de materializar lo digital.

dad. Montada en corrientes eléctricas, haces de luz, radiación electromagnética o estados cuánticos, la información, invisible e intangible a los sentidos sensoriales humanos, viaja a velocidades millones de veces superiores a las alcanzadas por aparatos como aviones ultrasónicos o sondas espaciales. Consecuentemente, podemos decir que la información representada como estados de energía es extremadamente flexible: se puede manipular, copiar y transferir automáticamente, a gran velocidad y sin pérdida de datos —de la misma manera que puede ser interferida y alterada, se proporcionan mecanismos de corrección de errores.

Mientras las tecnologías anteriores —como los libros en papel— dependían exclusivamente de la capacidad cognitiva de los seres humanos para procesar información, la tecnología digital nos permite extender esa capacidad cognitiva hacia el entorno en que vivimos, el cual es capaz de procesar información por nosotros de manera automática, veloz y relativamente autónoma. Ello nos ha permitido construir robots que arman autos o revisan la Web mientras dormimos, procesadores de texto que nos revisan la ortografía mientras escribimos y sistemas de recomendación que nos dan sugerencias de textos, música o video, a partir de lo que hemos leído, escuchado o visto, entre muchos otros procesos automatizados.

Desafortunadamente, las características diferenciadoras de la tecnología digital las convierten en fuente de una nueva brecha entre los seres humanos, de modo que el mundo se divide entre quienes pueden hacer uso de ellas para beneficio propio y de su contexto social, y los que no. Asimismo, la llamada *brecha digital* hace más grandes las diferencias producidas por brechas existentes previamente (económicas, de salud, de educación, de participación en la toma de decisiones), lo cual ha llevado a incluir las competencias asociadas al buen uso de las tecnologías digitales como parte de un nuevo y necesario *alfabetismo digital*.

3.3 Alfabetismos digitales

Las cuatro características de la tecnología digital, identificadas en la sección anterior, nos permite establecer un marco de referencia para el análisis de diversas tendencias en la construcción de la noción de alfabetismo digital que gradualmente han terminado por establecer distintas categorías de alfabetismos digitales claramente identificables en la literatura especializada (Tabla 3.1). Lo anterior no quiere decir que un alfabetismo digital atienda únicamente a una característica de la tecnología digital; solamente que se enfoca en ella.

Por otra parte, la UNESCO [2] hace referencia a cuatro aproximaciones a todo alfabetismo que, si bien da a entender que han surgido y pueden atenderse de manera un tanto independiente, en conjunto ofrecen una visión más completa de la estructura de un alfabetismo:

- colección de habilidades,

Tabla 3.1: Marco para el análisis de alfabetismos digitales.

Característica	Alfabetismo
Toda la información se codifica usando un esquema de representación simbólica mínima, que corresponde a agrupar cantidades varias de los dígitos binarios.	Alfabetismos en relación a la información y los medios digitales.
El esquema de representación de información se puede usar para la representación de procesos.	Alfabetismos en relación a la programación de computadoras.
Incorpora mecanismos capaces de interpretar la representación de los procesos y realizar las operaciones en la información sin la intervención de seres humanos.	Alfabetismos en relación a la automatización de procesos.
Los símbolos son representados y transmitidos en un medio físico mediante estados y flujos de energía.	Alfabetismos en relación a los dispositivos y las redes digitales.

- práctica situada,
- materiales,
- proceso de aprendizaje.

Las dos primeras, complementadas con el bagaje conceptual de sobre la tercera, conforman *competencias*, entendidas como la integración de conocimientos, habilidades, actitudes y valores que, en conjunto, habilitan a un individuo para realizar una acción pertinente de manera efectiva y eficaz en cierto tipo de contextos [5]. La cuarta aproximación hace énfasis en las teorías, metodologías, situaciones, contextos y procesos de desarrollo de las competencias.

La tercera aproximación atiende a los recursos que operan las competencias, dependientes de la tecnología o concomitantes a ella, con sus reglas de operación, convenciones y modos de interpretación [6], como ha sido el caso del lenguaje matemático y las tablillas de arcilla de los babilonios, usadas para llevar cuentas de operaciones comerciales; el lenguaje escrito y los papiros, la pluma de ganso entintada, el papel, el lápiz y los libros y, más recientemente, la noción de computadora, tecnología digital y lo que la ha acompañado y que con ella se ha producido.

Se propone entonces integrar las cuatro aproximaciones distintas a todo alfabetismo que identifica la UNESCO en una visión más amplia de todo alfabetismo,

que incluya los materiales, o *recursos* internos y externos, a ser operados por las competencias que lo conforman, así como los procesos de aprendizaje que se envisionan para desarrollarlas (Figura 3.1). A continuación, se bosquejan los cuatro tipos de alfabetismos presentados en la Tabla 3.1 en términos de sus tres componentes.

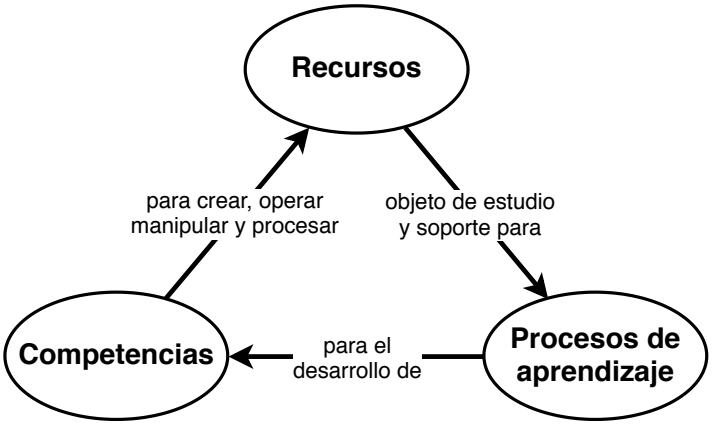


Figura 3.1: Componentes de un alfabetismo.

3.3.1 Alfabetismos de dispositivos y redes

La noción de *nativo digital* [7], que acaparó la atención por más de una década, popularizó este tipo de habilidad operativa relacionada con el manejo y uso de dispositivos, medios, aplicaciones y herramientas digitales, identificada de manera imprecisa como una de las habilidades del siglo XXI y como uno de los rasgo distintivo de las generaciones expuestas desde su nacimiento a la tecnología digital. Su fácil identificación y popularidad ha derivado, desafortunadamente, en que sean también foco de atención en la escuela, donde tampoco se desarrollan ni se explotan de manera sistémica e integral, mucho menos transcurricular, pues su tratamiento se limita generalmente a un taller o curso independiente que poco aporta al desarrollo de habilidades cognitivas superiores y de aplicación a la resolución de problemas de forma transdisciplinaria, situación por la cual se ha llegado incluso a poner en cuestionamiento la utilidad de ser incluída curricularmente en diversos países [8].

A continuación se presenta una propuesta de componentes típicos para este tipo de alfabetismos.

Recursos

- Dispositivos como computadoras de escritorio y portátiles, tabletas, teléfonos inteligentes, relojes inteligentes, termómetros sin contacto y medidores

de presión arterial.
- Periféricos y accesorios como teclado, ratón, cámara (web, de vigilancia), lectores ópticos y terminales para tarjetas.
- Aplicaciones del tipo de redes sociales, chat, ofimática, lectura de códigos QR, captura de audio, imágenes o video.

Competencias

Operación de dispositivos, periféricos, accesorios y aplicaciones para lograr objetivos inmediatos y directamente vinculados con las funcionalidades de los instrumentos.

Procesos de aprendizajes

Formal: talleres o cursos aislados, enfocados en la operación de dispositivos, periféricos, accesorios y aplicaciones, y uso (limitado) de los mismos en otras actividades de educación formal.

Informal: aprendizaje de "trucos" para usar dispositivos, periféricos, accesorios y aplicaciones, para consumir contenidos y para obtener ciertos resultados.

3.3.2 Alfabetismos informacionales y mediacionales

La adopción de las tecnologías digitales se amplía rápidamente a un sector más amplio de la sociedad. Asimismo, se observa un uso creciente de una Internet cada vez más rápida que alcanza ya a la mitad de la población mundial [9], a la que se suma el Internet de las Cosas y la incorporación de sistemas inteligentes artificiales, lo cual ha propiciado un flujo cada vez más rápido y cuantioso de información digital en un sinfín de formatos (ej. texto, sonido, imagen, animación, video) y de multitudes de fuentes, humanas y no humanas, que han hecho más urgente la necesidad de desarrollar las competencias para administrar la vastedad y complejidad de la información y los medios digitales.

Desde una perspectiva semiótica social [10], esta categoría de alfabetismo digital corresponde al proceso mismo de "dar sentido" a la información digital que recibimos y producimos constantemente, en grandes cantidades y por medios diversos, lo cual implica ser consciente de la diversidad de dimensiones de acción y análisis [11], así como de la variedad de opciones en cada una de ellas; de sus interrelaciones, implicaciones y consecuencias para el individuo y la sociedad —por ejemplo, los dilemas éticos que está desatando el desarrollo de la Inteligencia Artificial y la Analítica de Datos— a fin de tomar decisiones de manera crítica en lo referente a la generación, almacenamiento, transmisión, transformación y uso de información.

Existen aspectos técnicos asociados al alfabetismo de información y medios digitales, tales como la distinción entre diversos formatos para representación de información digital (por ejemplo, texto plano, hipermedia, imágenes, audio y video) y sus especializaciones. Otros aspectos técnicos importantes incluyen la distinción entre los datos y los metadatos —esto es, entre la información contenida en un paquete de información, como un documento, y la información acerca del paquete mismo, que en el caso del documento incluirían el título, los autores, la fecha de producción, palabras clave, codificación, aplicación con la que se construye, entre otros—, diversas formas de organizar información almacenada, tales como un sistema de archivos o una base de datos, así como conceptos más avanzados como los de compresión, codificación y encriptamiento de datos. Claramente, la ejecución de las competencias asociadas a este tipo de alfabetismo digital requiere la operación de dispositivos electrónicos digitales, tales como computadoras y dispositivos móviles, empero el énfasis del mismo no está en ello, sino en la gestión y procesamiento de información digital que ellos habilitan, ya sea de manera directa o a través de intermediarios. Por ejemplo, adultos mayores competentes en la gestión y procesamiento de información digital pueden ejecutar las acciones correspondientes sin hacer uso directo de los dispositivos digitales, a través de la mediación de personas más hábiles en ello.

A continuación se presenta una propuesta de componentes típicos para este tipo de alfabetismos.

Recursos

- Información en sus distintas representaciones y medios.
- Aplicaciones para gestionar información.
- Contenidos digitales multimedia (texto, páginas web, audio, video, realidad virtual).

Competencias

- Acceder, buscar, filtrar, interpretar, analizar, recopilar, integrar, simbolizar, evaluar, representar y sintetizar información con el apoyo de tecnología digital.
- Elaborar contenidos multimedia digitales.
- Usar software de autoría para la generación (automática) de representaciones gráficas de información digital en diversos medios.

Procesos de aprendizajes

Formal: cursos de formación en el uso de ciertas aplicaciones para la gestión de información (por ejemplo, ofimática, bases de datos de publicaciones, buscadores, gestores de referencias, aplicaciones de estadística, de visualización de información o de análisis cualitativo). Uso de esas herramientas como parte de otras actividades de educación formal.

Informal: sobre la marcha, para atender las necesidades de acceso a información y medios digitales en la educación formal, las actividades laborales y la vida diaria.

3.3.3 Alfabetismos de la programación

La programación de computadoras está implícita en todos los procesos informáticos de la sociedad digital que se conforma día a día, a través de las instrucciones que ejecutan los miles de millones de dispositivos operando en nuestro planeta y algunos más que hemos lanzado a explorar el Universo. Actualmente, el simple hecho de encender la televisión o hacer una llamada telefónica desde un teléfono celular implica la operación automática de colecciones de instrucciones organizadas de tal modo que se puedan ejecutar de manera física y mecánica, las cuales han sido elaboradas por programadores (especialistas en la organización de instrucciones) para lograr que los dispositivos digitales se comporten de la manera esperada. Es por ello que, con cierta frecuencia, la programación se ha considerado un alfabetismo [12], a pesar de que la gran mayoría de los ciudadanos no saben leer ni escribir programas.

Durante lo que va del siglo, el alfabetismo de la programación ha sido el foco de atención y promoción en diversas iniciativas de gran escala tales como la Hora del Código —que establece como su misión introducir el aprendizaje de la programación en educación básica de manera similar a como se aprenden las matemáticas y las ciencias, así como promover el aprendizaje de la programación entre sectores minoritarios de la población y las mujeres— CS Education Week, EU Code Week y Code First: Girls.

Entre los años sesenta y setenta del siglo pasado hubo varias iniciativas hacia la generación de lenguajes de programación con énfasis en su uso educativo, a fin de promover el aprendizaje de la programación entre los jóvenes o desarrollar buenas prácticas en la misma que condujeran a la producción de software más confiable y eficiente. Se produjeron lenguajes tales como Logo [13], BASIC [14], Pascal [15] y Scheme [16]. Más recientemente, en lo que va de este siglo ha habido una erupción de lenguajes de programación enfocados hacia la formación en programación en educación básica e incluso preescolar, los cuales se caracterizan por su representación y manipulación gráfica, en vez de textual, así como la visualización de su operación y resultados mediante animación y otras formas de generación de

multimedia. Entre ellos destaca Scratch [17], un entorno de programación visual disponible primero como una aplicación de escritorio y más recientemente como una aplicación en línea, como elemento central de un entorno orientado hacia el aprendizaje colectivo de la programación mediante el acceso libre y gratuito a los programas construidos por los usuarios del sistema. El proyecto que ha producido y mantenido a Scratch se orienta principalmente a niños en educación básica y adolescentes, pero se ha desarrollado una versión para niños en edad preescolar (cinco a siete años), llamado ScratchJr, el cual está disponible como una aplicación para dispositivos móviles e incluye un lenguaje de programación visual basado también en la metáfora del rompecabezas, pero con un número menor de piezas, con un lenguaje más gráfico y menos textual y maneras más sencillas de armar programas. Existen otras iniciativas como Blockly y el uso de dispositivos como BBC Micro:bit, entre otros.

A continuación se presenta una propuesta de componentes típicos para este tipo de alfabetismos.

Recursos

- Lenguajes de programación.
- Infraestructura para el desarrollo (*frameworks*) de programas y aplicaciones.
- Entornos de programación.

Competencias

- Análisis, diseño e implementación de algoritmos.
- Resolución de problemas mediante la creación de programas.
- Diseño de arquitecturas y modelo de sistemas.
- Depuración de código.
- Trabajo colaborativo para construir sistemas y programas complejos.

Procesos de aprendizajes

Formal: programas de formación en ingeniería de software y similares. Cursos de programación en ciertos lenguajes de programación. Introducción a la programación en educación preuniversitaria mediante lenguajes gráficos o robótica.

Informal: mentoría, MOOC, documentación en línea, foros y videos.

3.3.4 Alfabetismo de automatización

Como se ha comentado previamente, una diferencia fundamental de las tecnologías digitales con respecto a tecnologías previas es su capacidad para realizar procesos

computacionales —operaciones aritméticas, ordenamiento de datos, búsqueda de información, entre otros— fuera de las mentes de los seres humanos. Sin esta característica sería imposible la existencia de Internet, porque sería imposible realizar automáticamente el enrutamiento de datos a través de la red, desde el origen hasta el destino; la existencia de las funcionalidades de los dispositivos que usamos todos los días y de la Inteligencia Artificial que ha dado origen a la noción de la Cuarta Revolución Industrial [18], caracterizada por la automatización masiva de procesos de generación de productos, procesamiento de datos y prestación de servicios.

Vivir en un entorno con estas características demanda el establecimiento de una relación con las tecnologías digitales que adquiere, al menos, uno de tres matices. O bien somos manipulados por un entorno digital que toma decisiones por nosotros [19], o bien podemos establecer un diálogo que nos permita cooperar con sistemas inteligentes tales como Siri, Cortana, Alexa o Assistant, o incluso robots [20], o bien podemos hacer uso de las tecnologías digitales para automatizar procesos y ser más productivos. A fin de cuentas, la tecnología va a estar ahí, y quienes puedan sacarle provecho tendrán la ventaja [20], [21].

A continuación se presenta una propuesta de componentes típicos para este tipo de alfabetismos.

Recursos

- Funcionalidades de automatización de dispositivos, aplicaciones y redes.
- Oportunidades de automatización que ofrecen ciertos procesos individuales, colectivos u organizacionales.

Competencias

- Comprensión y automatización de procesos informáticos, mecánicos y otro tipo de procesos en distintos sectores.
- Integración de sistemas basados en *hardware* (como lectores de código) y/o *software* (como Excel o aplicaciones de bases de datos).
- Automatización vía el manejo de nuevas tecnologías del Internet de las cosas, casas o edificios inteligentes, análisis de grandes cantidades de datos, la nube y elementos de inteligencia artificial.

Procesos de aprendizajes

Formal: la escuela en educación preuniversitaria se queda corta, por lo que se cubre hasta nivel profesional muchas veces con capacitaciones y certificaciones especializadas.

Informal: pocos incursionan por esta vía, pero lo hacen de maneras diversas (cursos, documentación en línea, videos).

3.4 Pensamiento computacional

Jeannette M. Wing [22] define el pensamiento computacional como

> 'Los procesos de pensamiento involucrados en la formulación de un problema y la expresión de su(s) solución(es) de tal manera que una computadora —ser humano o máquina— pueda llevarlo a cabo de manera efectiva'.

Añade que las ciencias de la computación tienen como objetivo central la automatización de abstracciones, de modo que bien podríamos decir que computar es automatizar abstracciones. De modo que, asumiendo que resolver un problema implica su expresión o modelación acorde con las herramientas que se van a usar para buscar soluciones, tendría sentido decir que

> el pensamiento computacional consiste en los procesos de pensamiento involucrados en la resolución de problemas de manera que sus soluciones sean automatizaciones de abstracciones.

De modo que es posible identificar en el pensamiento computacional tres componentes clave: resolución de problemas, abstracción y automatización —entre los cuales Wing resalta la abstracción como 'el proceso de pensamiento de alto nivel más importante del pensamiento computacional'. Tendría sentido entonces observar la presencia de estos tres elementos del pensamiento computacional en los cuatro alfabetismos digitales desglosados en la sección anterior, como se muestra en la Tabla 3.2.

Todos los alfabetismos, incluyendo los digitales, tienen un sentido práctico [6], en el sentido de que nos permiten estar mejor capacitados para atender nuestras necesidades en el entorno correspondiente, tomar mejores decisiones y, en ese sentido general, resolver los problemas que enfrentamos día a día. Sin embargo, la atención de problemas difíciles y complejos demanda un proceso de abstracción que modifique su descripción y solución, a fin de que sean comprensibles para los seres humanos por lo menos en su etapa de diseño.

En el caso de los alfabetismos de dispositivos y redes, la abstracción y automatización se limitan a la funcionalidad de la herramienta o dispositivo —un procesador de texto abstrae y automatiza muchas de las operaciones que antes se realizaban manualmente en una imprenta, en tanto que un modem lo hace con las propias de diseño lógico de cierto tipo de circuitos electrónicos. Se sabe usar cierto tipo de

Tabla 3.2: Alfabetismos digitales y componentes del pensamiento computacional.

	Componentes del pensamiento computacional		
Alfabetismos digitales	Resolución de problemas	Abstracción	Automatización
Alfabetismos de dispositivos y redes	Ocasional (a nivel operativo)	Baja	Baja
Alfabetismos informacionales y mediacionales	Sí	Media	Media
Alfabetismos de programación	Sí	Alta	Alta
Alfabetismos de automatización	Sí	Alta	Alta

aparatos y ciertas aplicaciones para resolver problemas particulares, y las competencias no son fácilmente transferibles a otros contextos —un ejemplo típico son aquellas personas que solamente saben usar Microsoft Word y Excel, no procesadores de texto y hojas de cálculos. La abstracción puede ser mayor en los alfabetismos orientados a la información, donde se aplican conceptos más generales, como los de espacios informativos de exhibición, producción e interacción y comunicación en las plataformas para la educación en línea [23], aunque no necesariamente implica su transferencia a otros ámbitos. Sin embargo, puede ser todavía mayor en los alfabetismos orientados a la programación y los alfabetismos orientados a la automatización. En cierto sentido, se podría decir que los alfabetismos de automatización hacen uso de lenguajes de muy alto nivel —como es el caso, por ejemplo, cuando alguien se comunica con su Google Home Assistant y le dice '¡Ey, Google! Establece un temporizador de diez minutos'

Como ya se comentó, la automatización es baja cuando se limita a las funcionalidades específicas de aplicaciones y dispositivos, puede ser mayor cuando se gestiona información (por ejemplo, programación de alertas de publicaciones, búsquedas avanzadas en buscadores web o bases de datos de referencias, uso de sistemas gestores de referencias y de archivos de notas), es todavía mayor en los alfabetismos de programación y automatización, en los cuales las competencias se centran específicamente en la automatización de procesos. Hasta aquí, se han analizado las cuatro categorías propuestas de alfabetismo digital desde la perspectiva del pensamiento computacional.

Ahora bien, la relación puede plantearse también en sentido inverso y realizar

un análisis del pensamiento computacional como alfabetismo, lo cual nos lleva a la modificación de la propuesta presentada en la Tabla 3.1 para proponer la integración de los últimos dos renglones en un solo alfabetismo:

- Alfabetismos de dispositivos y redes.
- Alfabetismos informacionales y mediacionales.
- Alfabetismos de pensamiento computacional.

Desde esta perspectiva, los alfabetismos de pensamiento computacional serían una suerte de generalización de los alfabetismos de programación y de automatización en términos de las abstracciones de las ciencias computacionales, cuyas competencias estarían directamente relacionadas con la automatización, y cuyo objetivo y proceso central de aprendizaje y es la resolución de problemas (Figura 3.2).

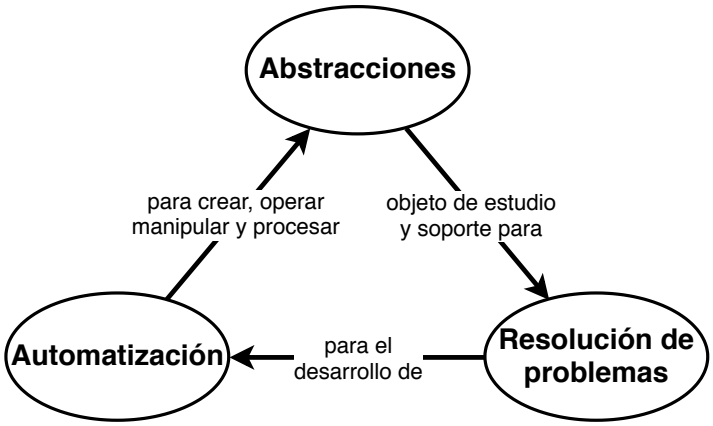


Figura 3.2: Pensamiento computacional como alfabetismo.

3.4.1 Comparación con otras definiciones

La concepción de pensamiento computacional que aquí se construye parece ir en sentido contrario al análisis detallado de sus componentes, su extensión y su vinculación con otros tipos de pensamiento, notoria en el caso de la definición ofrecida por Miguel Zapata Ros [24], quien propone 'con ánimo de exhaustividad' los siguientes componentes del pensamiento computacional:

- Análisis ascendente.
- Análisis descendente.
- Heurística.
- Pensamiento divergente.
- Creatividad.
- Resolución de problemas.

- Pensamiento abstracto.
- Iteración.
- Métodos por aproximaciones sucesivas.
- Ensayo-error.
- Métodos colaborativos.
- Patrones.
- Sinéctica.
- Metacognición.
- Cinestesia.

A los cuales añade posteriormente elementos de lo que llama pensamiento bayesiano [25]. El mismo autor argumenta sobre la conveniencia de este tipo de descripciones detalladas de los componentes del pensamiento computacional para operacionalizar la formación en pensamiento computacional como parte del currículo de la educación formal, particularmente en comparación con definiciones tan abstractas que resultan mucho menos útiles, como es el caso de la definición original de Wing [3].

Desde nuestro punto de vista, la concepción de pensamiento computacional como alfabetismo digital que aquí se ofrece se ubica en un punto intermedio entre definiciones abstractas como la Wing, a la que desglosa, y definiciones detalladas como la de Zapata-Ros, a las que organiza. La primera parte se ha presentado ya en la sección anterior, en tanto que la segunda parte se ilustra en la Figura 3.3.

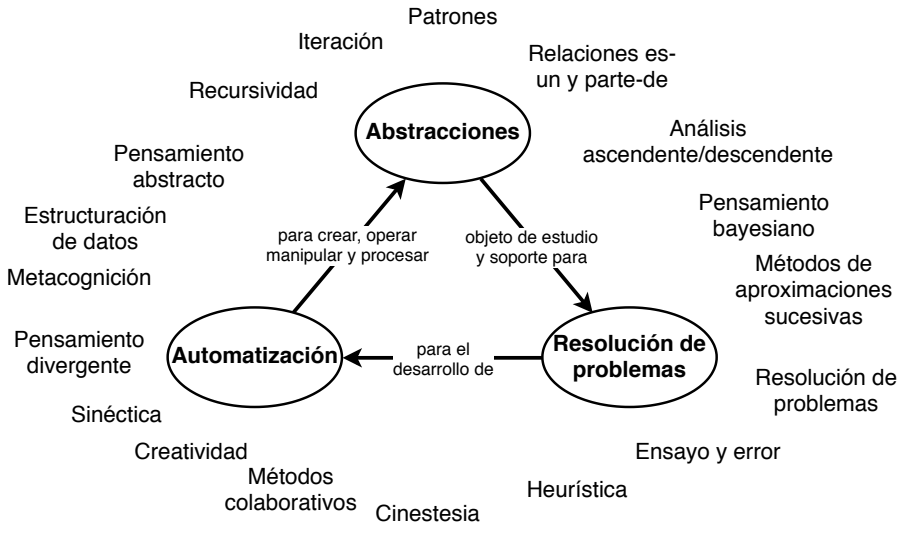


Figura 3.3: Integración de elementos de otras definiciones de pensamiento computacional.

Se pueden identificar así componentes del pensamiento computacional relacionados más directamente con el proceso de solución de problemas, con la competencia de automatización o con las abstracciones como los recursos operados por dichas competencias para la resolución de problemas.

Un siguiente nivel de análisis consistiría en discernir entre aquellos elementos directamente relacionados con la competencia de automatización de abstracciones para la solución de problemas y aquellos que establecen vínculos con otras formas de pensamiento útiles para construir ideas y analizarlas, representar información, "visualizar" procesos y resolver problemas en general, que resultan útiles para el pensamiento computacional.

3.5 Conclusiones

En este artículo se desarrollaron de inicio cuatro categorías de alfabetismos orientados a las tecnologías digitales atendiendo a las características básicas de estas últimas: alfabetismos de dispositivos y redes, alfabetismos informacionales y mediacionales, alfabetismos de programación y alfabetismos de automatización. Posteriormente, se ha generado una definición de pensamiento computacional con tres componentes esenciales, resolución de problemas, abstracción y automatización, y se ha bosquejado una relación de dos vías entre el pensamiento computacional y las categorías de alfabetismos digitales planteadas previamente, mediante la identificación de los componentes del pensamiento computacional en cada una de ellas y mediante la concepción del pensamiento computacional como un alfabetismo. En consecuencia, se ha ofrecido una generalización de los alfabetismos de programamción y de automatización como alfabetismos de pensamiento computacional.

Todo el proceso ha sido esencialmente de generalización, dado que los distintos alfabetismos que se describen en la literatura no son "puros", en el sentido de que no se inscriben necesariamente a una sola categoría. Sin embargo, se espera haber delineado un esquema básico que permita aclarar lo que es el pensamiento computacional y su relación con los distintos tipos de alfabetismos digitales, lo cual a su vez deja claro que la formación en tecnologías digitales en educación básica, e incluso bachillerato, orientada generalmente al alfabetismo de dispositivos y, menos frecuentemente, al alfabetismo de programación, no está cubriendo las necesidades de la formación del pensamiento computacional en las nuevas generaciones.

Finalmente, la perspectiva del pensamiento computacional como alfabetismo coloca a la resolución de problemas tanto como área de aplicación como proceso de aprendizaje del mismo, resaltando la naturaleza práctica del pensamiento computacional, en conjunción con las abstracciones propias de las ciencias computacionales y las competencia de automatización de abstracciones.

Referencias

[1] Real Academia Española. Alfabetismo. En: *Diccionario de la lengua española* (2014).

[2] UNESCO. *Education for All: Literacy for Life*. Global Monitoring Report. Paris: United Nations Educational, Scientific and Cultural Organization, 2006.

[3] Jeannette M Wing. Computational Thinking. En: *Communications of the ACM* 49.3 (2006), págs. 33-35.

[4] Alan M. Turing. Computing Machinery and Intelligence. En: *Mind* 49 (1950), págs. 433-460.

[5] Philippe Perrenoud. Construir competencias ¿es darle la espalda a los saberes? En: *Revista de Docencia Universitaria* Número monográfico II: Formación centrada en competencias (II) (2008). ISSN: 1887-4592.

[6] Andrea A. diSessa. *Changing Minds: Computers, Learning, and Literacy*. A Bradford Book. The MIT Press, 2001. ISBN: 978-0-262-54132-9.

[7] Marc Prensky. Digital Natives, Digital Immigrants Part 2: Do They Really Think Differently? En: *On the Horizon* 9.6 (nov. de 2001), págs. 1-6. ISSN: 1074-8121. DOI: 10.1108/10748120110424843.

[8] Zsuzsanna Szalayné Tahy. Guess the Code of Conditional Summation. En: *Proceedings of the 10th International Conference on Applied Informatics*. Heger, Hungary, 2017, págs. 279-284. DOI: 10.14794/ICAI.10.2017.279.

[9] Miniwatts Marketing Group. *World Internet Users Statistics and 2020 World Population Stats*. Internet World Stats: Usage and Population Statistics. Oct. de 2020. URL: http://www.internetworldstats.com/stats.htm.

[10] Jay L. Lemke. Multimedia Semiotics: Genres for Science Education and Scientific Literacy. En: *Developing Advanced Literacy in First and Second Languages: Meaning with Power*. Ed. por M. J. Schleppegrell y M. C. Colombi. Lawrence Erlbaum Associates, 2002, págs. 21-44.

[11] SooHwan Kim, SeonKwan Han y HyeonCheol Kim. How Can We Teach Computational Literacy to All Levels of Students? En: *2009 Fifth International Joint Conference on INC, IMS and IDC*. IEEE, ago. de 2009, págs. 1395-1400. DOI: 10.1109/NCM.2009.192.

[12] Annette Vee. *Coding Literacy: How Computer Programming Is Changing Writing*. Software Studies. The MIT Press, 2017. ISBN: 978-0-262-03624-5.

[13] Seymour Papert. Teaching Children Thinking. En: *Journal of Structural Language* (1975).

[14] Bennet P. Lientz. A Comparative Evaluation of Versions of BASIC. En: *Communications of the ACM* 19.4 (1976), págs. 175-181.

[15] Kathleen Jensen y Niklaus Wirth. *PASCAL – User Manual and Report. ISO Pascal Standard*. Springer Verlag, 1974. ISBN: 0-387-97649-3.

[16] Revised5 Report on the Algorithmic Language Scheme. En: *Higher-Order and Symbolic Computation* 11.1 (1998). Ed. por Richard Kelsey, William Clinger y Jonathan Rees.

[17] Mitchel Resnick y col. Scratch – Programming for All. En: *Communications of the ACM* 52.11 (2009), págs. 60-67.

[18] Klaus Schwab. *The Fourth Industrial Revolution*. New York: Crown Business, 2017. ISBN: 978-1-5247-5887-5.

[19] Wu Youyou, Michal Kosinski y David Stillwell. Computer-Based Personality Judgments Are More Accurate than Those Made by Humans. En: *Proceedings of the National Academy of Sciences of the United States of America* 112.4 (ene. de 2015), págs. 1036-1040. DOI: 10.1073/pnas.1418680112.

[20] Anca Dragan. Putting Humans in the AI Equation. En: *Possible Minds: Twenty-Five Ways of Looking at AI*. Ed. por John Brockman. New York: Penguin Press, 2019.

[21] David Deutsch. Beyond Reward And Punishment. En: *Possible Minds: Twenty-Five Ways of Looking at AI*. Ed. por John Brockman. New York: Penguin Press, 2019.

[22] Jeannette M Wing. Computational Thinking's Influence on Research and Education for All. En: *Italian Journal of Educational Technology* 25.2 (2017), págs. 7-14. DOI: 10.17471/2499-4324/922.

[23] María Elena Chan Núñez. Tendencias En El Diseño Educativo Para Entornos de Aprendizaje Digitales. En: *Revista Digital Universitaria* 5.10 (2004).

[24] Miguel Zapata-Ros. El Pensamiento Computacional, Una Cuarta Competencia Clave Planteada Por La Nueva Alfabetización. En: *Las Tecnologías En (y Para) La Educación*. Ed. por José Miguel García y Sofía García Cabeza. Montevideo, Uruguay: FLACSO Editorial, 2020, págs. 171-209. ISBN: 978-9915-9329-0.

[25] Miguel Zapata-Ros. *Pensamiento Bayesiano Como Pensamiento Computacional Por Todas Partes*. RED: El aprendizaje en la sociedad del conocimiento. Nov. de 2020. URL: https://red.hypotheses.org/tag/pensamiento-bayesiano.

Parte II

Experiencias en la enseñanza de pensamiento computacional

Capítulo 4

Análisis de la competencia de resolución de problemas algorítmicos en estudiantes de educación superior

María Luisa Velasco Ramírez [1] y *Guillermo Leonel Sánchez Hernández* [1]

[1] Universidad Veracruzana.
lvelasco@uv.mx, gusanchez2@uv.mx

Resumen. Esta investigación plantea el análisis de la competencia de resolución de problemas algorítmicos en los estudiantes de la carrera de Licenciado en Sistemas Computacionales de una Universidad pública del estado de Veracruz. La investigación se ha realizado bajo un enfoque cuantitativo, transversal y descriptivo. La prueba se aplicó a 122 estudiantes que cursaban la experiencia educativa de Algorítmica. Los resultados obtenidos muestran que existen deficiencias en las habilidades cognitivas que deben desarrollar los estudiantes para resolver problemas algorítmicos, como: interpretar, abstraer, modelar, identificar, analizar y validar la información que ofrece la situación problemática, lo que revela la necesidad de introducir nuevas propuestas que permitan perfeccionar el proceso de enseñanza-aprendizaje de la competencia. Se concluye que los estudiantes muestran un nivel inicial-receptivo básico en el que aún no planifican, ni logran diagnosticar adecuadamente un problema algorítmico.

Palabras clave. Análisis, competencias, resolución de problemas, algoritmo, educación superior.

4.1 Introducción

Un gran desafío para los estudiantes universitarios en el área de Ciencias de la Computación es el aprendizaje de la disciplina Algorítmica, una rama de la informática, la cual proporciona una base para la apropiación de todos los lenguajes de programación [1]. De la disciplina Algorítmica, la actividad cognitiva más importante es la resolución de problemas a través de un algoritmo, un tema abstracto, de dificultad para los estudiantes, la cual se basa principalmente en los siguientes elementos:

- La incomprensión de diferentes conceptos a menudo vistos como difusos y difíciles de comprender, muchos estudiantes de pregrado todavía carecen de conocimiento semántico de lo que sucede dentro de la computadora para conceptos como variable, su declaración y asignación.
- Una sobrecarga cognitiva intrínseca debido a los flujos de información que deben aprenderse, los estudiantes son forzados a aprender simultáneamente el conocimiento de cálculo aritmético, conceptos algorítmicos y la sintaxis.
- Una sobrecarga cognitiva extrínseca debido a la manera en que se presenta la información. Los cursos se basan generalmente en una breve explicación de la estructura de los elementos algorítmicos seguidos por una serie de ejemplos que dejan poco espacio para las fases de análisis y diseño.
- Falta de estrategias para descomponer los problemas en subproblemas.

El nivel abstracto del tema y la falta de vinculación con problemas prácticos y significativos son factores que afectan la motivación en los estudiantes al enfrentar la resolución de problemas en el tema de desarrollo de algoritmo [2]. Hablar de resolución de problemas algorítmicos es hablar del pensamiento algorítmico. El pensamiento algorítmico de acuerdo con Futschek [3] se compone del siguiente conjunto de habilidades conectadas con la construcción y comprensión de algoritmos:

- 'La capacidad de analizar problemas dados.
- La capacidad de especificar un problema con precisión
- La capacidad de encontrar las acciones básicas que son adecuadas para el problema dado.
- La capacidad de construir un algoritmo correcto para un problema dado usando las acciones básicas.
- La capacidad de pensar en todos los posibles casos especiales y normales de un problema.
- La capacidad de mejorar la eficiencia de un algoritmo'.

Para Futschek [3] 'el pensamiento algorítmico tiene un fuerte aspecto creativo: la construcción de nuevos algoritmos que resuelven problemas dados. Si alguien quiere hacer esto, necesita la capacidad de pensamiento algorítmico'. El objetivo de este estudio es conocer en qué nivel de dominio se encuentran los estudiantes al iniciar sus estudios en Algorítmica. Hernández [4] describe las características de los distin-

tos niveles de dominio o desempeño en la Tabla 4.1.

Tabla 4.1: Nivel de dominio de las competencias. Tomada literalmente de [5] (citado por [4]).

Niveles	Características principales
Preformal	Aprendizajes de nociones muy generales sin organización. Hay aprendizaje de algunos conocimientos, pero sin manejo de procedimientos ni de actividades de la competencia. Hay baja motivación y compromiso.
Inicial-receptivo • Identifica • Reconoce • Registra • Se concentra • Describe • Define	Hay recepción y comprensión general de la información. El desempeño es muy básico y operativo. Hay baja autonomía. Se tienen nociones sobre el conocer y el hacer. Hay motivación frente a la tarea.
Básico-resolutivo • Comprende • Resuelve • Ejecuta • Planifica • Elabora • Diagnostica • Implementa • Realiza	Se resuelven problemas sencillos del contexto. Se tienen elementos técnicos de los procesos implicados en la competencia. Se poseen algunos conceptos básicos. Realiza las actividades asignadas.
Autónomo • Argumenta • Explica • Autorregula • Mejora • Formula • Critica • Analiza • Articula	Hay autonomía en el desempeño (no se requiere de asesoría de otras personas o de supervisión constante). Se gestionan proyectos y recursos. Hay argumentación científica. Se resuelven problemas de diversa índole con los elementos necesarios. Se actúa en la realidad con criterio propio.

Tabla 4.1: Continuación.

Niveles	Características principales
Estratégico	Se plantean estrategias de cambio en la realidad.
• Crea	Hay creatividad e innovación.
• Innova	Hay desempeños intuitivos de calidad.
• Vincula	Hay altos niveles de impacto en la realidad.
• Transversaliza	Hay análisis prospectivo y sistémico de los
• Sinergia	problemas.
• Adapta	Se tiene un alto compromiso con el bienestar
• Teoriza	propio y de los demás.

La licenciatura en Sistemas Computacionales Administrativos de una Universidad pública del estado de Veracruz [6] define como perfil de egreso, que un egresado será capaz de 'integrar y crear soluciones de sistematización y automatización de información, para el desarrollo estratégico de las organizaciones en un contexto de competencia global'. También deberá contar con 'una visión integral de la administración que le permitirá diseñar e implementar acciones de negocio óptimas aplicando las Tecnologías de la Información y las Comunicaciones de manera responsable'; así como conocer 'los procesos administrativos y financieros en las organizaciones para facilitar su sistematización y automatización'.

La licenciatura adopta un modelo educativo integral y flexible basado en competencias a partir del año 2011, incluyendo como parte de su plan de estudios la experiencia educativa de Algorítmica, la cual se encuentra en el nivel de iniciación de la disciplina y es considerada base para la materia de Programación, experiencia de nivel disciplinar para la formación de los estudiantes. Se ha observado que los estudiantes carecen de habilidades en la resolución de problemas algorítmicos, debido a la falta de solidez en los conocimientos previos como manipulación algebraica, comprensión lectora. Además, definir e identificar los datos necesarios de entrada y resultados a alcanzar para poder analizar el problema es una tarea complicada para ellos.

Otro problema que se ha identificado es una actitud poco alentadora al cursar la experiencia educativa de programación, por lo que es necesario preguntarnos si los estudiantes cuentan con un nivel apropiado de competencia de resolución de problemas algorítmicos. La presente investigación tuvo por objetivo analizar el nivel de competencia de resolución de problemas algorítmicos de los estudiantes de la Licenciatura en Sistemas Computacionales Administrativos, para aportar información válida, que pueda servir de base a una futura intervención tecno-pedagógica con el fin de favorecer su desarrollo.

En el contexto de la presente investigación, el constructo "competencia de resolución de problemas algorítmicos" se define de la siguiente manera:

> Como el conjunto de conocimientos, habilidades y actitudes necesarios para implementar algoritmos que permitan resolver problemas de un tipo específico. Involucra las capacidades de comprensión de lectura, identificación del problema, manipulación algebraica, planificación paso a paso y análisis del proceso ([7], citado por [8]). Para lo cual el estudiante debe ser capaz de poner en práctica habilidades cognitivas como: asociar, inferir, comprender, abstraer, resolver, planificar, modelar y analizar durante el proceso de resolución de problemas algorítmicos. (Adaptado de [8])

En la Tabla 4.2 se describen las dimensiones del constructo resolución de problemas algorítmicos, las habilidades de cada una y sus indicadores, adaptadas de [8].

4.2 Metodología

4.2.1 Muestra

Este estudio se llevó a cabo en la Universidad Veracruzana, en la Facultad de Contaduría y Administración, campus Xalapa, Veracruz, México, durante el período febrero-mayo de 2019; específicamente en el programa educativo de Licenciado en Sistemas Computacionales Administrativos. Se seleccionó una muestra por conveniencia de 122 estudiantes de segundo semestre, que cursaban la experiencia educativa de Algorítmica. Los estudiantes contaban ya con conocimientos básicos de álgebra, matemáticas administrativas, tecnologías de las computadoras; experiencias educativas que cursaron durante el primer semestre; así como, lectura y redacción, como parte del área básica del programa educativo. Estas experiencias educativas se consideran necesarias para cursar Algorítmica. La muestra estuvo conformada por un 34% de mujeres y 66% de hombres, de edades comprendidas entre 18 y más años, sin establecerse un límite de edad, debido a que en ocasiones se inscriben alumnos de 24 años o más en los primeros semestres.

4.2.2 Desarrollo del instrumento

Se seleccionó y adaptó un test para medir la competencia de resolución de problemas algorítmicos a partir de la disertación doctoral de [8][1], el cual se encuentra conformado de 11 ítems (el primer ítem consta de 14 opciones de falso o verdadero) distribuidos en cinco dimensiones:

[1] Previa autorización del autor [Comunicación personal]. 4 de marzo de 2019.

Tabla 4.2: Dimensiones del constructo resolución de problemas algorítmicos.

Dimensiones	Indicadores	Habilidades
Comprensión de lectura.	Asociar Inferir	Identificar términos clave. Identificar cuantificadores. Distinguir detalles. Deducir ideas preconcebidas.
Identificación del problema.	Comprender Abstraer	Resumir y organizar la información. Interpretar la información. Traducir el problema a un contexto diferente.
Manipulación algebraica.	Resolver	Resolver operaciones aritméticas simples representación numérica o algebraica.
Planificación por pasos.	Planificar Modelar	Observar el panorama general del problema. Descomponer el problema en subproblemas. Diseñar un plan. Modelar tareas simples de manera algorítmica.
Análisis.	Analizar	Determinar el propósito de las instrucciones. Rastrear manualmente errores. Depurar manualmente errores.

Comprensión lectora. Esta dimensión busca probar la capacidad de leer textos generales y problemas verbales de manera objetiva.

Identificación del Problema. En esta dimensión se enlistan preguntas para probar la capacidad del estudiante para resumir y organizar la información, y para identificar problemas en un contexto. Además, también verifica la generalización de reglas, generalmente expresadas en forma de relaciones algebraicas.

Manipulación algebraica. Las preguntas dentro de esta categoría buscan examinar la capacidad del estudiante para realizar operaciones aritméticas simples, ya sea numéricamente o en representación algebraica, para prever el resultado del algoritmo e interpretar problemas algebraicos.

Planificación paso a paso. Esta dimensión busca determinar la capacidad del estudiante para describir tareas simples de manera algorítmica.

Análisis de proceso. Esta sección busca explorar la capacidad del estudiante para trabajar con secuencias cortas de simples instrucciones: comprender el propósito de las instrucciones, el rastreo manual de las instrucciones y la depuración de errores. Las preguntas están dirigidas a determinar la consistencia en la aplicación de la lógica, los conceptos erróneos en los algoritmos, las suposiciones inadecuadas y la intuición en los bucles y condicionales.

En la tabla Tabla 4.3 se muestran los ítems del test y la dimensión a la que corresponden; así como, los conocimientos y habilidades que busca medir en más de una dimensión. El instrumento se muestra en la Tabla 4.4.

Tabla 4.4: Test para medir la competencia de resolución de problemas algorítmicos.

El siguiente test está compuesto de 11 preguntas y pretende ser una herramienta de diagnóstico para mejorar las clases de Algorítmica y Programación en función de tus habilidades para resolver problemas. Los resultados no afectarán tu calificación final. Por favor sé completamente honesto, no uses ningún libro, notas o calculadora. Si no entiendes la pregunta indícalo.

Tabla 4.4: Continuación.

1. Lee el fragmento de manera cuidadosa y contesta las siguientes preguntas, indicando F o V.

> Los números primos fascinan y frustran a todos los que los estudian. Su definición es tan simple y obvia; Es tan fácil encontrar uno nuevo; La descomposición multiplicativa es una operación tan natural. ¿Por qué, entonces, los números primos se resisten fuertemente a los intentos de ordenarlos y regularlos? ¿No tienen ningún orden o estamos demasiado ciegos para verlo? Hay, por supuesto, algún orden oculto en los números primos. El Tamiz de Eratóstenes obtiene los números primos de los enteros. Primero 2 es un primo. Ahora elimine todos los enteros pares más altos (que deben ser divisibles entre 2). El siguiente entero más alto que sobrevive, 3, también debe ser primo. Quita todos sus múltiplos, y 5 sobrevive. Elimina los múltiplos de 5, y quedan 7. Continúe de esta manera y cada entero que caiga a través del tamiz es un primo. Este procedimiento ordenado, aunque lento encontrará todos los primos. Además, como n va a infinito, sabemos que la proporción de números primos a no primos entre los primeros n enteros se aproxima a $(\log n)/n$. Desafortunadamente, el límite es solo estadístico y en realidad no ayuda a encontrar números primos.

a) Todos los números primos tienen un orden oculto.
b) El tamiz aprovecha la técnica de Euclides.
c) La entrada requerida por el tamiz es números.
d) Al usar el tamiz, la salida resultante son números impares.
e) El pasaje describe el significado de los números primos.
f) El tamiz detecta cualquier número primo excepto 0.
g) La entrada requerida por el tamiz puede ser números negativos.
h) En un tamiz de números $m \times n$, al menos n^2 son primos
i) Al usar el tamiz, los valores de salida resultantes son primos.
j) El tamiz aprovecha la técnica de Eratóstenes.
k) La palabra “ciegos”, en la sexta línea, significa “falta de vista”.
l) La palabra “ciegos”, en la sexta línea, significa “despistados”.
m) El pasaje describe la técnica para encontrar todos los números primos.
n) El pasaje describe una técnica para encontrar todos los números primos.

Tabla 4.4: Continuación.

2. ¿Qué calcula esta secuencia de instrucciones?

Paso 1: Multiplica el precio por 0.07.
Paso 2: Añade esa respuesta al precio.

a) Calcula un impuesto a las ventas del 7%.
b) Calcula una reducción de precio del 7%.
c) Calcula un precio total que incluye un 7% de utilidad.
d) Calcula un margen de beneficio que eleva el precio 107%.

3. Una lata de pintura tiene una etiqueta que dice que un galón cubre x pies cuadrados. Tienes que pintar una pared de bloques de cemento en su parte frontal (sólo) con 2 capas de pintura (es necesario pintarlo dos veces). La pared es de l pies de largo y h pies de alto y t pies de ancho. ¿Qué le dirías a la persona que te pregunta cómo "calcular" la cantidad correcta de galones a comprar?

a) Multiplica l por h por t y divide por 2.
b) Multiplica l por h, h por t, suma esos números y divide la última respuesta por x.
c) Multiplica 2 por l, esa respuesta por x, y luego divide por 2.
d) Multiplica 2 por h, esa respuesta por l y luego divide por x.

4. ¿Cómo le dirías a una persona que encuentre el costo total de la gasolina para un viaje de X millas con un auto que obtiene Y millas por galón, si la gasolina cuesta Z dólares por galón?

a) Divide X por Y, luego divide el resultado por Z.
b) Multiplica X por Y, luego divide el resultado por Z.
c) Multiplica X por Y, y luego multiplica el resultado por Z.
d) Divide X por Y, luego multiplica el resultado por Z.

5. ¿Qué se calcula con esta secuencia de instrucciones?

Paso 1: Divide 100 por 24.
Paso 2: Redondea el resultado del paso 1 hasta el siguiente número entero más grande.

a) Calcula cuántos galones de gasolina se utilizan para recorrer 100 millas.
b) Calcula cuántos vehículos se necesitan para transportar a 100 personas si cada vehículo lleva 24 personas.
c) Calcula cuántas cajas se llenarán completamente con manzanas si son 100 manzanas para poner en 24 cajas.
d) Todas las anteriores.

Tabla 4.4: Continuación.

6. Identifica el procedimiento para calcular la siguiente secuencia de hasta ocho términos: 0,1, 1, 2, 3, 5, 8, 13...

a) **Procedimiento 1:**
   - Sea a el número 0.
   - Sea b el número 1.
   - siguiente $= a + b$.
   - $a = b$.
   - $b = a$.
   - Repite los pasos (2) a (5) cinco veces más.

b) **Procedimiento 2:**
   - Sea a el número 0.
   - Sea b el número 1.
   - siguiente $= a + b$.
   - $a = b$.
   - $b =$ siguiente.
   - Repite los pasos (2) a (5) cinco veces más.

c) **Procedimiento 3:**
   - Sea a el número 0.
   - Sea b el número 1.
   - siguiente $= a + b$.
   - $a = b$.
   - $b =$ siguiente.
   - Repite los pasos (3) a (5) cinco veces más.

d) **Procedimiento 4:**
   - Sea a el número 0.
   - Sea b el número 1.
   - siguiente $= a + b$.
   - $a = b$.
   - $b =$ siguiente.
   - Repite los pasos (3) a (5) seis veces más.

Tabla 4.4: Continuación.

7. Utilizando las siguientes instrucciones, planifica la secuencia de instrucciones correctas de un algoritmo que permita calcular la suma de los números impares desde 1 hasta antes de un número entero positivo proporcionado por el usuario:
   1) $i = i + 2$
   2) Mientras ($i <$ número
   3) $i = 1$
   4) Leer número
   5) Escribir i
   6) Fin mientras

   a) 4, 1, 2, 5, 3, 6.
   b) 4, 3, 1, 2, 5, 6.
   c) 4, 5, 1, 2, 6, 3.
   d) 4, 3, 2, 5, 1, 6.

8. Tu hermano menor está planeando una pijamada con 5 amigos. Tu madre le dijo que comprará 2 hot dogs, 3 barras de dulce y algo para leer, para él y para cada invitado. También necesita un refresco y sabe que 1 litro de refresco es suficiente para 3 niños. ¿Cuánta comida deberá comprar en la tienda?
   a) 2 hot dogs, 3 caramelos, 1 litro de refresco, 5 cómics.
   b) 10 hot dogs, 15 dulces, 2 litros de refresco, 5 cómics.
   c) 10 hot dogs, 15 caramelos, 1 litro de refresco, 6 cómics.
   d) 12 hot dogs, 18 dulces, 2 litros de refresco, 6 cómics.

9. ¿Cuál es el resultado de seguir las siguientes instrucciones?

Paso 1: Piensa en un número (X), pero mantenlo en silencio en tu mente.
Paso 2: Toma el número y multiplícalo por 2.
Paso 3: Suma 8 al resultado anterior.
Paso 4: Toma el resultado del paso 3 y resta el número con el que comenzaste.
   a) X.
   b) $2X$.
   c) $X + 8$.
   d) $2X +$.

Tabla 4.4: Continuación.

10. Las instrucciones en pseudocódigo que se listan a continuación deben mostrar valores decimales de $1/1$, $1/2$, $1/3$, $1/4$, $1/5$. Sin embargo, no funcionan correctamente. Identifica la instrucción incorrecta.

```
Mientras num < 5
        dec = 1/num
        Desplegar en pantalla dec
        num = num + 1
Fin Mientras
```

a) La instrucción num $= 0$.
b) La instrucción Mientras num < 5.
c) La Instrucción dec $= 1/$num.
d) La instrucción num $=$ num $+ 1$.

11. Las instrucciones en pseudocódigo que se listan a continuación deben mostrar los números pares entre 1 y 10 inclusive, en orden descendente. Sin embargo, el pseudocódigo es incorrecto. Identifica la(s) instrucción (es) incorrecta(s).
Nota: res, es la operación de obtener el residuo de una división.

```
num = 10
Mientras (num  > 1 y  num < 10)
        Si res(num/2) = 0 entonces
                Print ("El número", num, es "par")
        num = num − 2
Fin Mientras
```

a) num $= 10$
b) Mientras (num > 1 y num < 10)
c) Si res$(\text{num}/2) = 0$ entonces
d) num $=$ num $- 2$

4.2.3 Validez del instrumento

Para su validación se recurrió al juicio de expertos en el tema de resolución de problemas algorítmicos. Es importante señalar que, para estimar la confiabilidad de un juicio de expertos, es necesario conocer el grado de acuerdo entre ellos [9]; para lo cual se utilizó el Coeficiente de Kappa de Fleiss [10] (citado en [11]), quien generalizó la aplicación del índice Kappa de Cohen para medir el acuerdo entre más de dos

Tabla 4.3: Dimensiones del Test de resolución de problemas algorítmicos.

Dimensión	Ítem
Comprensión de lectura	1
Identificación del problema	4
Manipulación algebraica	5
Manipulación algebraica	3
Análisis	8
	9
Planeación	6
	7
Análisis	2
	10
	11

codificadores u observadores para datos de escala nominal y ordinal.

Los valores obtenidos en el coeficiente de Kappa de Fleiss para cada criterio evaluado por los expertos se muestran en la Tabla 4.5.

Tabla 4.5: Coeficiente de Kappa de Fleiss obtenido.

Suficiencia	Coherencia	Relevancia	Claridad
$K = 0.47$	$K = 0.81$	$K = 0.76$	$K = 0.77$

Para la interpretación del índice de Kappa de Fleiss, se tomó la clasificación propuesta por Altman□ [12] (citado en [11]), el cual indica que los coeficientes registran valores que van desde 0 a 1, siendo 0 el valor donde hay mayor desacuerdo entre investigadores y 1 el punto donde se encuentra mayor acuerdo. Su clasificación indica que los índice Kappa pueden ir de *Pobres* a *Muy buenos*, según se observa en la Tabla 4.6. Si se toman los rangos de valores proporcionados en la Tabla 4.5, se puede concluir que la concordancia entre los criterios valorados por los jueces resulta: moderada para el criterio de Suficiencia, buena para los criterios de Relevancia y Claridad y muy buena para el criterio de Coherencia. Algunas sugerencias fueron acerca de mejorar la redacción en algunos ítems.

Tabla 4.6: Interpretación del índice de Kappa de Fleiss [12].

Valor de K	Concordancia
< 0.20	Pobre
0.21–0.40	Débil
0.41–0.60	Moderada
0.61–0.80	Buena
0.81–1.00	Muy buena

4.2.4 Confiabilidad del instrumento

Después de realizar los cambios propuestos por los expertos se aplicó el instrumento para verificar su nivel de confiabilidad. Una vez aplicado el test, se cargaron los datos en Excel para su codificación. El análisis de confiabilidad se realizó utilizando el coeficiente de Kuder Richardson KR-20, obteniendo un valor de 0.6144.

La fórmula KR-20 considera (a) el número de elementos de la prueba, (b) el rendimiento del estudiante y (c) la varianza entre los elementos de la prueba [13]. El autor indica que una puntuación KR-20 de 0.60 o mayor es deseable.

En las pruebas multidimensionales; es decir, aquellas que miden diferentes aspectos o dimensiones de un constructo, tienden a presentar coeficientes de confiabilidad de consistencia interna de magnitud moderada, indicando que no son totalmente homogéneas. La homogeneidad está relacionada con la característica de unidimensionalidad de una prueba [14].

4.3 Análisis y resultados

El desarrollo de la competencia para resolver problemas algorítmicos puede ser poco efectivo por una serie de factores, tales como conceptos erróneos: conocimiento o ideas que debilitan el proceso real de resolución de problemas, o un inconsistente conocimiento, conceptos que la persona sabe, pero no aplica de manera efectiva.

Todos los datos obtenidos a partir de la administración de la prueba de resolución de problemas algorítmicos fueron analizados y procesados en Excel para su análisis. Como se puede observar en la Figura 4.1, los resultados muestran un bajo nivel de competencia en las cinco dimensiones que conforman la competencia de resolución de problemas algorítmicos; en la Tabla 4.7, se describen los ítems, dimensión a la que corresponden y las habilidades no detectadas.

Las opciones (1a) y (1d) son puntos de control para observar si el estudiante ha estado atento tanto a la lectura como a las respuestas. El significado de los números primos, la opción (1e), no se puede inferir de la lectura, y la diferencia entre las dos

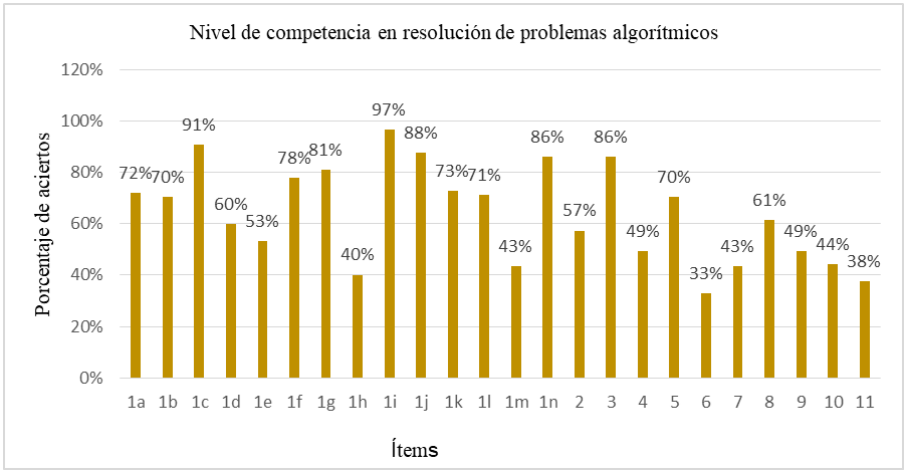


Figura 4.1: Nivel de competencia en resolución de problemas algorítmicos.

Tabla 4.7: Descripción de ítems, dimensión a la que corresponden y habilidades no detectadas.

Número de ítems	Dimensión	Habilidades no detectadas
1d.	Comprensión de lectura.	Atención a la lectura y a las respuestas.
1e.	Comprensión de lectura.	Distinguir detalles del texto.
1h.	Comprensión de lectura.	Distinguir del contexto del texto.
1m.	Comprensión de lectura.	Identificar cuantificador.
2	Análisis.	Determinar el propósito de las instrucciones.
4	Identificación del problema con manipulación algebraica.	Interpretar el problema y resolverlo utilizando una expresión aritmética.
9	Manipulación algebraica.	Resolver operaciones con representación algebraica .
6,7	Planificación por pasos.	Planificar una secuencia de instrucciones.
10,11	Análisis	Determinar el propósito de las instrucciones Rastrear manualmente errores Depurar manualmente errores.

últimas afirmaciones (1m) y (1n) es el cuantificador (es decir, el tamiz de Eratóstenes no es la única manera de encontrar números primos).

El nivel de detalle durante la lectura de un problema algorítmico es muy importante para su resolución; desafortunadamente la falta de atención y comprensión de la lectura es un problema recurrente en el proceso de resolución de problemas señalado por especialistas, tal como se puede observar en los ítems con calificación reprobatoria que se muestra en la Figura 4.1. Se puede destacar que los únicos ítems con porcentaje aprobatorio corresponden a elementos explícitos en el texto que facilitaban la comprensión de éste.

Los ítems (4) y (9) incluyen la dimensión de manipulación algebraica, los resultados observados en la gráfica 1 muestran la falta de capacidad del estudiante para realizar operaciones aritméticas simples, ya sea numéricamente o en representación algebraica. Específicamente, el ítem (9) muestra acciones simples en cada paso, incluso pueden parecerse a algunos trucos matemáticos que comienzan con un número, solicitan realizar algunas operaciones y llevan a un valor específico; sin embargo, el procedimiento no termina con un valor concreto sino simbólico. La capacidad del estudiante para describir tareas simples de manera algorítmica también se observa en un bajo nivel en el porcentaje de aciertos obtenidos en los ítems (6) y (7).

Por último, la dimensión de análisis evaluada a través de los ítems (2), (10) y (11) como se observa en la gráfica 1, tampoco muestra resultados satisfactorios; se debe recordar que esta dimensión se refiere a la capacidad del estudiante para trabajar con secuencias cortas de simples instrucciones: comprender el propósito de las instrucciones, el rastreo manual de las instrucciones y la depuración de errores.

Por otra parte, es interesante el resultado obtenido en el ítem (5), aunque no en un alto porcentaje, este fue aprobatorio; en este ítem la respuesta correcta era la opción (b); sin embargo, cualquier opción proporciona información sobre las habilidades de los estudiantes, desde adivinación afortunada hasta comprensión parcial o lectura deficiente. La última opción, "todas las anteriores", puede indicar una lectura superficial o una falta de comprensión de los detalles involucrados en cada opción. La primera opción podría atribuirse a descuidar el paso de redondeo (el consumo de gasolina generalmente se da sin redondeo); mientras que la opción (c) podría indicar que no se prestó atención a la palabra clave "completamente", tal como lo indica Vasconcelos-Santillán [8].

Ha sido constante exponer que la asimilación de un curso algorítmico es un problema persistente para muchos estudiantes universitarios del área de Ciencias de la Computación [1]. La mayor dificultad a la que se enfrentan los estudiantes es la falta de capacidad y flexibilidad en la resolución de problemas, lo que implica establecer una sucesión de pasos elementales, cada uno de los cuales genera un conocimiento nuevo, que se obtiene como inferencia lógica a partir de los conocimientos y experiencias del estudiante y de las condiciones del problema [15].

Los resultados de nuestro estudio muestran que existen deficiencias en las habilidades cognitivas que deben desarrollar los estudiantes para resolver problemas algorítmicos, como: interpretar, abstraer, modelar, identificar, analizar y validar la información que ofrece la situación problemática, reflejadas en las dimensiones de comprensión de lectura, identificación del problema, manipulación algebraica, planificación por pasos y análisis; lo que revela la necesidad de introducir nuevas propuestas que permitan perfeccionar el proceso de enseñanza-aprendizaje de la competencia en resolución de problemas algorítmicos. Esto coincide con la postura de [16].

La experiencia obtenida a través de la aplicación del test de resolución de problemas algorítmicos permitió observar que: las respuestas de los alumnos podrían proporcionar información sobre las habilidades de resolución de problemas en los estudiantes, observando cómo podían influir (positiva o negativamente) en la viabilidad de algunas preguntas al desencadenar una actitud hacia ellas, como el exceso o falta de confianza; tal como lo expone Vasconcelos-Santillán [8].

Algorítmica y Programación son materias de vital importancia en la formación de estudiantes de las carreras de Informática o Ciencias de la Computación. El éxito o fracaso en ellas influye en las posibilidades de seguir avanzando en la carrera si no logran alcanzar las competencias necesarias. Esta investigación en particular aporta datos para identificar las deficiencias en la competencia de resolución de problemas algorítmicos de los estudiantes de la Licenciatura en Sistemas Computacionales Administrativos, pero también la necesidad de buscar estrategias que ayuden a su desarrollo, tal como lo manifiestan Rosanigo y Paur [17].

4.4 Conclusiones

Resolver problemas es un aspecto recurrente en programación, que requiere la capacidad de integrar y aplicar una serie de conceptos fundamentales, habilidades cognitivas y estilos de pensamiento. Desarrollar habilidades para resolver problemas de manera metodológica es crucial para el desarrollo de algoritmos como un paso anterior a la creación de programas de computadora.

Los resultados obtenidos a lo largo del estudio reafirman la necesidad de transformar el proceso de enseñanza-aprendizaje de Algorítmica de manera que se logren desarrollar las habilidades necesarias, a través de la combinación de enfoques pedagógicos; así como el uso de recursos tecnológicos con el fin de estructurar actividades didácticas para el logro de objetivos de aprendizaje. Se puede concluir que los estudiantes muestran un nivel inicial-receptivo bajo en el que aún no planifican ni logran diagnosticar adecuadamente un problema algorítmico.

Finalmente, es importante resaltar que la competencia de resolución de problemas es fundamental en un licenciado en Sistemas Computacionales Adminis-

trativos como en cualquier profesional del área de Computación, lo que destaca la importancia de continuar con investigaciones en las que se apliquen actividades y estrategias que promueven la adquisición de destrezas en Comprensión lectora, Identificación del problema, Manipulación algebraica, Planificación por pasos y Análisis, capacidades elementales en el aprendizaje de Algorítmica y Programación.

Referencias

[1] A. Babori y col. An E-Learning Environment for Algorithmic: Toward an Active Construction of Skills. En: *World Journal on Educational Technology: Current Issues* 8.2 (2016), págs. 82-90.

[2] C. M. Chezzi y col. Estrategia de Motivación Para El Razonamiento de Algoritmos Computacionales Mediante Juegos. En: *Ría* 5 (2017).

[3] Gerald Futschek. Algorithmic Thinking: The Key for Understanding Computer Science. En: *Informatics Education – The Bridge between Using and Understanding Computers*. Ed. por Roland T. Mittermeir. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, págs. 159-168. ISBN: 978-3-540-48227-7. DOI: 10.1007/11915355_15.

[4] José Silvano Hernández Mosqueda. Procesos de Evaluación de Las Competencias Desde La Socioformación. En: *Ra Ximhai* 4 (2013). ISSN: 1665-0441.

[5] Sergio Tobón, Julio Herminio Pimienta Pedro y Juan Antonio García Fraile. *Secuencias Didácticas: Aprendizaje y Evaluación de Competencias*. México: Pearson Education, 2010. ISBN: 978-607-442-909.

[6] Universidad Veracruzana. *Licenciatura En Sistemas Computacionales Administrativos: Plan de Estudio 2011*.

[7] J. Vasconcelos y J. Houlahan. *Development and Interpretation of a Survey on Problem Solving Skills*. Personal Communication. Ago. de 2003.

[8] Jorge Vasconcelos-Santillan. Development of a Web-Based Surveying Instrument to Identify Problem-Solving Abilities Related to Effective Instruction in Computer Programming. Tesis doct. USA: Johns Hopkins University, 2008.

[9] L. Aiken. *Test Psicológicos y Evaluación*. México: Pearson Education, 2003.

[10] J.L. Fleiss. *Statistical Methods for Rates and Proportions*. New York: John Wiley and Sons, 1981.

[11] J.J. Torres y V.H. Perera. Cálculo de La Fiabilidad y Concordancia Entre Codificadores de Un Sistema de Categorías Para El Estudio Del Foro Online En E-Learning. En: *Revista de Investigación Educativa* 27 (2009), págs. 89-104.

[12] D.G. Altman. *Practical Statistics for Medical Research*. New York: Chapman and Hall, 1991.

[13] Nesrin Oruç Ertürk. Testing Your Tests: Reliability Issues of Academic English Exams. En: *International Journal of Psychology and Educational Studies* 2.2 (2015), págs. 47-52. DOI: 10.17220/ijpes.2015.02.005.

[14] Carlos Ruiz Bolívar. *Confiabilidad. Programa de Doctorado En Educación*.

[15] A. Salgado y col. Didáctica de La Resolución de Problemas de Programación Computacional. En: *Pedagogía Universitaria* 18 (2013), págs. 62-75.

[16] A. Blanco-Hamad, A. Salgado-Castillo e I. Alonso-Berenguer. Habilidades Para La Algoritmización Computacional En La Licenciatura En Educación: Especialidad Educación Laboral-Informática. Maestro y Sociedad. En: *Revista Electrónica para Maestros y Profesores* 13.1 (2016), págs. 18-30.

[17] Z.B. Rosanigo y A.B. Paur. Estrategias Para La Enseñanza de Algorítmica y Programación. En: *I Congreso de Tecnología En Educación y Educación En Tecnología*. 2006, págs. 117-124.

Capítulo 5

Aprende ciencias de la computación jugando: experiencias y resultados de un proyecto de servicio social en Baja California

Eloísa García-Canseco [1], *Francisco Juárez García* [1], *Adrián Enciso Almanza* [1], *Alejandro González Sarabia* [1], *José Angel González Fraga* [1] y *Verónica Luna Hernández* [1]

[1] Universidad Autónoma de Baja California.
eloisa.garcia@uabc.edu.mx

Resumen. Presentamos las experiencias y resultados que hemos obtenido durante los últimos cinco años con el proyecto de servicio social "Aprende Ciencias de la Computación Jugando (ACCJ)" en el Estado de Baja California. Dicho proyecto ha beneficiado a la fecha a más de mil estudiantes (desde primaria hasta preparatoria) así como a más de un centenar de profesores que se han capacitado en los talleres que hemos impartido en alianza con diversas organizaciones. En este trabajo damos a conocer el impacto de este programa a nivel estatal, así como nuestra perspectiva sobre la enseñanza y aprendizaje del pensamiento computacional en educación básica a partir de las experiencias que hemos adquirido con el proyecto.

Palabras clave. Pensamiento computacional, olimpiada de informática.

5.1 Introducción

Si bien es cierto que la enseñanza de las ciencias computacionales ha comenzado a implementarse en diferentes países, tanto en planes de estudio obligatorios como

optativos [1], el enfoque y los contenidos temáticos varían enormemente en todo el mundo [2]. Inclusive la terminología es diferente, en algunos países suelen llamarle "computación", "informática", o "ciencias computacionales", por mencionar algunos. Para complicarlo un poco más, la interpretación de estos conceptos puede variar de acuerdo al contexto educativo, es decir, el término "ciencias computacionales" puede referirse a las habilidades técnicas del hardware o a diferentes especializaciones en software [2].

Es importante mencionar que aunque en los planes y programas de estudio para la Educación Básica de la Secretaría de Educación Pública (SEP) a nivel nacional, se promueve el desarrollo de habilidades y competencias para el uso responsable de las Tecnologías de la Información y Computación (TICs), no se contempla actualmente el estudio de las ciencias computacionales ni alguna asignatura relacionada con la programación [3]. En México, aunque se ha reconocido la computación como una área de investigación, en lo que respecta a la enseñanza de esta disciplina científica se siguen generando consumidores de tecnología y no creadores de ella. No obstante la SEP ha dado un primer paso, reconociendo la importancia de desarrollar el pensamiento crítico, el pensamiento creativo, el manejo de la información, el uso de la tecnología, la ciudadanía digital e incluso el pensamiento computacional con la creación del marco referencial para la enseñanza del pensamiento computacional en la educación básica [4]. En lo que se refiere al estado de Baja California, todavía son muy pocas las escuelas de secundaria y bachillerato (y menos aún las de primaria) que ofrecen actualmente cursos de programación. Siendo este hecho uno de los mayores obstáculos al que nos hemos enfrentado al momento de comenzar en 2012 con la Olimpiada Mexicana de Informática en Baja California [5, 6], ya que son pocos los estudiantes (y todavía menos los profesores asesores) que llegan con conocimientos de lógica, algoritmos y programación a los primeros exámenes de la olimpiada. Es por ello que desde el 2016 comenzamos con las actividades del programa de servicio social "Aprende Ciencias de la Computación Jugando", el cual describimos en la siguiente sección.

5.2 Descripción del programa

El programa "Aprende Ciencias de la Computación Jugando" tiene como función primordial introducir a estudiantes y profesores de educación básica y media superior a los fundamentos de las ciencias de la computación, a través de la impartición de talleres didácticos y lúdicos en los cuales participan activamente como prestadores de servicio social, estudiantes universitarios de programas de estudio afines a las ciencias de la computación.

Entre las actividades que realizamos a lo largo del año podemos destacar las siguientes:

- *Cursos de capacitación para los instructores/talleristas:* Con estos cursos formamos recursos humanos a nivel licenciatura en la enseñanza de ciencias computacionales en los diferentes niveles educativos. En cada edición del programa participan aproximadamente 30 estudiantes universitarios que estudian ciencias e ingeniería.
- *Cursos de capacitación para docentes:* Estos talleres están enfocados en docentes de primaria, secundaria y bachillerato, tanto de las áreas de matemáticas como de informática o tecnologías de la información. Ver Figura 5.1.
- *Talleres para estudiantes de primaria, secundaria y bachillerato:* En estos cursos participan niños a partir de tercer grado de primaria y hasta tercer semestre de bachillerato. Los talleres están separados en cuatro categorías: primaria para niños de tercero y cuarto grado de primaria, primaria para niños de quinto y sexto grado de primaria, secundaria y preparatoria.

Figura 5.1: Actividades con el juego de mesa Robot Turtles [7]. (Izquierda) Taller para alumnos de secundaria. (Derecha) Taller para docentes de informática y matemáticas de nivel secundaria.

En el caso de estudiantes en los primeros años de educación primaria, la mayoría de las actividades se basan en juegos y temáticas que no requieren el uso de las computadoras (actividades desconectadas), ya que desafortunadamente la mayoría de las escuelas de educación primaria no cuentan con salas de cómputo. Sin embargo, siempre que es posible incorporamos también la programación por bloques (Scratch [8], OMIBOT [5], entre otros.). En cuanto a los útlimos años de primaria y secundaria se refiere, además de algunas actividades desconectadas, la herramienta principal es Karel el Robot [9] por ser la plataforma oficial de la Olimpiada Mexicana de Informática para Primaria y Secundaria (OMIPS). En nivel preparatoria, el lenguaje de programación oficial es C++.

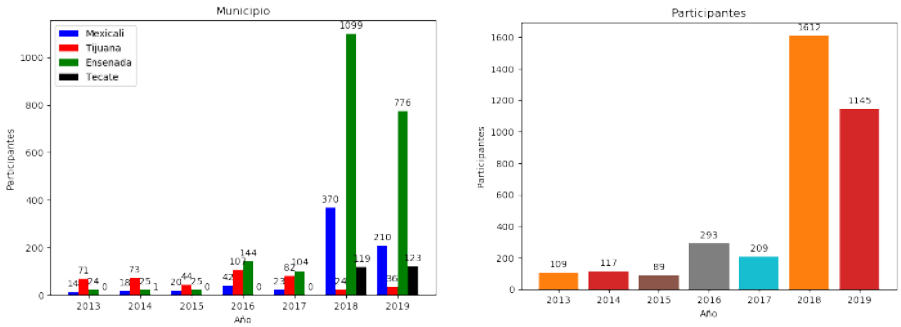


Figura 5.2: (Izquierda) Participación histórica por municipio. (Derecha) Número total de participantes.

5.3 Resultados e impacto del programa

Como se puede observar en la Figura 5.2 y la Figura 5.3, cada año hemos tenido un incremento en el número de participantes a nivel estatal, con una ligera disminución en 2019 debido a factores externos como huelgas de trabajadores en algunos subsistemas educativos. En 2020, debido a la contingencia sanitaria actual no fue posible tener los talleres presenciales, sin embargo actualmente estamos analizando diversas estrategias para poder realizar los talleres de forma virtual. La permanencia e interés año con año de los participantes, tanto estudiantes como profesores, ha sido fundamental para ir observando el avance del proyecto. El año pasado, la primera generación de primaria de nuestro proyecto se incorporó a la categoría secundaria, con mucha experiencia y más herramientas que quienes participan por primera vez, dando muestra del trabajo constante e intenso a lo largo de todos estos años de los profesores y prestadores de servicio social que participan en el programa. Durante el año 2019, gracias a una colaboración con OmegaUp [10], impartimos un curso de certificación a más de cincuenta docentes de informática del estado, evento al que se le dió difusión y que puede ser consultado en el siguiente enlace: https://youtu.be/PVlkgbYYzNU. Nuestros talleres de programación para niños también se presentan cada año durante la Noche de Ciencias, la Semana Nacional de Ciencia y Tecnología, la Expo Ciencia y Tecnología UABC, el Museo El Trompo y en las ferias científicas de las escuelas que nos invitan a participar. En febrero de 2020 fuimos invitados a participar en el 1er. Foro Internacional de Altas Capacidades, organizado por la Facultad de Ciencias Administrativas y Sociales (FCAyS) de la Universidad Autónoma de Baja California, en el cual tuvimos la oportunidad de impartir un taller de programación creativa a treinta niños de altas capacidades.

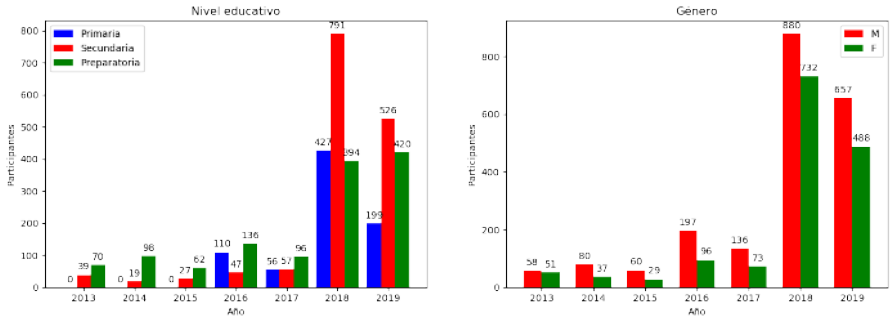


Figura 5.3: (Izquierda) Participación histórica por nivel educativo. (Derecha) Participación por género.

5.4 Perspectivas sobre la enseñanza del pensamiento computacional

De acuerdo con los trabajos de [11, 12] podemos decir que el pensamiento computacional (PC) es un proceso mental que seguimos para definir o formular un problema, construir una solución y expresarla adecuadamente, de tal forma que pueda ser realizada por una computadora. Podemos agregar que la tarea la debería también poder realizar otra persona. Para construir soluciones a los problemas que formulamos recurrimos a: la lógica, la descomposición de un problema en partes más pequeñas, el modelado y abstracción, el reconocimiento de tipos de problemas, la construcción de algoritmos, así como la evaluación de la corrección y eficiencia de la solución.

En el desarrollo del programa "Aprende Ciencias de la Computación Jugando" la enseñanza del PC es considerada como una etapa primordial en el proceso formativo tanto de la población infantil como de los docentes que han participado en el programa. Debido a que al inicio del proceso, el público tiene formación, experiencia y conocimiento diverso, debemos escoger recursos accesibles para todos, por lo que en esta etapa del programa la mayoría de las actividades que realizamos son desconectadas, es decir, que no requieren la utilización de computadoras. A continuación presentamos algunos ejemplos de los recursos que utilizamos en los talleres, y que de alguna forma permiten a los participantes que inician en los primeros niveles, seguir un proceso progresivo en el aprendizaje del pensamiento computacional y la programación.

Figura 5.4: (Izquierda) Actividades con el juego de mesa Robot Turtles [7] en Pre-escolar. (Derecha) Actividades con Makey Makey [13] en Primaria.

5.4.1 Pre-escolar y primeros años de primaria

Las actividades para trabajar con niños desde tercer grado de pre-escolar hasta tercer grado de primaria consisten principalmente en actividades desconectadas. Los juegos de mesa son una fuente inagotable de ideas para introducir no solo el pensamiento computacional sino las primeras instrucciones de programación (o pseudocódigo) tanto para alumnos y maestros que no saben programar. En el proyecto hemos utilizado el juego de mesa Robot Turtles [7], un juego que permite a partir de la pregunta: ¿qué camino debe seguir la tortuga para llegar a su piedra preciosa? introducir a los niños a la solución de problemas en ciencias computacionales, mientras aprenden las primeras instrucciones de programación. Con niños un poco más grande hemos observado que es bastante intuitivo pasar del juego de mesa, a ambientes de programación por bloques tipo Scratch. Una herramienta que hemos encontrado útil son los kits de programación de Makey Makey [13], la cual además de reforzar los conceptos de programación, también permiten a los niños conocer cómo sus programas pueden interactuar con el mundo físico aprendiendo nociones básicas de circuitos eléctricos (Figura 5.4).

5.4.2 Primaria

De manera adicional a las actividades mencionadas en la sección anterior, a partir del tercer grado de primaria, podemos comenzar a trabajar la noción de iteración y eficiencia utilizando problemas como: Calcula 8×5 o 5×8. Utilizando el procedimiento que se emplea en las escuelas primarias para explicar el concepto de multiplicación tenemos que: $8 \times 5 = 8+8+8+8+8$ y $5 \times 8 = 5+5+5+5+5+5+5+5$. ¿En cuál de las dos multiplicaciones se realizan menos operaciones de suma? Este problema permite introducir la noción de abstracción de iteración, esto es, cómo

indicar que se realice la acción de sumar repetidamente una cantidad arbitraria de veces. Podemos generalizar si, en lugar de decir "realice la acción de sumar repetidamente", decimos "realice la acción sobre los elementos de una colección repetidamente", sin preocuparnos de donde surgen estos objetos.

Trabajar con los números figurados también permite introducir a los estudiantes a los fundamentos de la teoría de números y reforzar la iteración, en particular con el ambiente de programación de Karel [9], el cual trabajamos con los alumnos que ya tienen un dominio de la programación en bloques. En Karel, se itera así:

```
1 iterate (n)
2 giraDerecha ();
```

Por ejemplo, en la Figura 5.5 la abstracción del procedimiento en Karel nos permite definir nuevas operaciones, sin decir como es que estas operaciones se realizan, i.e.: $9 = 1 + 3 + 5$, $16 = 1 + 3 + 5 + 7$, y podemos entonces discutir con los alumnos la pregunta $25 = ¿?$

5.4.3 Secundaria

El el nivel secundaria, un ejemplo de recurso para enseñar pensamiento computacional puede ser una película como "Misión rescate (*The Martian*)" [14]. Utilizar este recurso muestra que enseñar PC no es enseñar ciencias de la computación, pero es una habilidad fundamental que debe desarrollar un estudiante ya sea en el área de ciencias de la computación o incluso en otras áreas del conocimiento.

A partir de la temática de la película, los estudiantes deben definir o formular un problema principal, como por ejemplo: sobrevivir en Marte, utilizando los recursos de que dispone. Una vez formulado el problema, surgen las tareas (descomposición

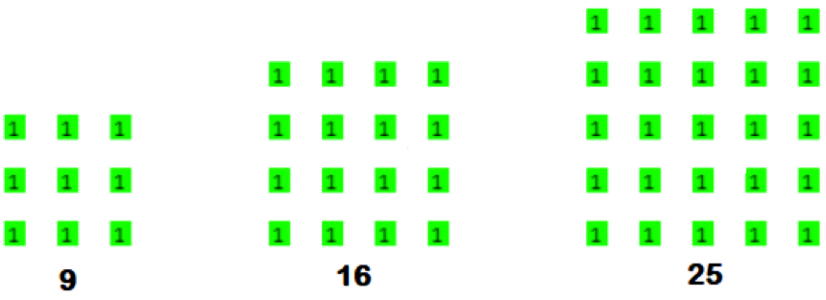


Figura 5.5: Numeros figurados con Karel.

del problema) como: incrementar la cantidad de los alimentos, producir agua, crear un sistema de comunicaciones, mejorar el sistema de transporte, entre otras. Los algoritmos se introducen explicando que para cada tarea hay un proceso que se debe realizar, e.g.: producir alimentos, producir agua, comunicarse con la Tierra, mejorar el transporte, etc. Al construir cada algoritmo, también abordamos la necesidad de reconocer lo que es importante y necesario para cada tarea, esto es, empezamos a abstraer y a modelar. También es importante hacer énfasis en la diversidad de conocimientos que se requieren para resolver cada tarea. La complejidad de un algoritmo se puede también tratar discutiendo cómo se intercambian los mensajes entre la Tierra y Marte.

La parte de programación en nivel secundaria se realiza utilizando Karel [9] a través de la plataforma de OmegaUp [10] ya que como se mencionó anteriormente, es el lenguaje de programación oficial de la Olimpiada Mexicana de Informática para Primaria y Secundaria.

5.4.4 Preparatoria

En nivel Preparatoria, antes de introducir el lenguaje C++, es importante trabajar con los estudiantes actividades que les permitan adquirir el pensamiento computacional, planteándoles algunas actividades desconectadas como la siguiente [15, 16]: ¿Cómo calcular las raíces cuadradas de los números naturales 2, 3 y 4 utilizando una tira rectangular de papel de ancho 1 unidad? (ver Figura 5.6). El procedimiento a discutir con los estudiantes sería el siguiente: Llevamos el vértice A hacia el borde denotado por a. Marcamos entonces A'. El segmento OA' mide 1. Entonces OB mide $\sqrt{2}$. Ahora llevemos el vértice B hacia el borde a. Marcamos C y tenemos que OC=$\sqrt{2}$. Enseguida doblamos girando sobre el vértice C de tal manera que el lado a cae sobre si mismo. Marcamos el punto D y tenemos que OD=$\sqrt{1 + \text{OC}^2} = \sqrt{3}$. La pregunta para los estudiantes sería: ¿Cómo calculamos $\sqrt{4}$?

5.5 Conclusiones y trabajo futuro

Hasta el momento, la experiencia que hemos adquirido con el programa de servicio social "Aprende Ciencias de la Computación Jugando" ha sido de forma empírica. Sin embargo, identificamos la necesidad de ir formalizando la metodología de trabajo de tal forma que podamos a futuro, tener herramientas que nos permitan medir de forma cuantitativa y cualitativa el impacto real del proyecto. En este sentido, estamos iniciando una colaboración con investigadores del área de educación para realizar estudios a nivel estatal que nos permitan:

- Entender los procesos y estrategias que utilizan actualmente los docentes de informática y tecnologías para la enseñanza de la programación.

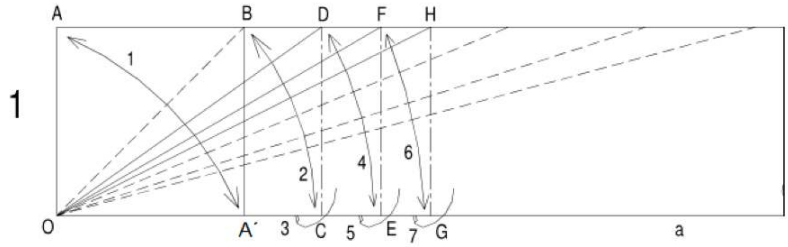


Figura 5.6: Actividad de papiroflexia para calcular la raíz cuadrada de números naturales. Figura adaptada de [15]

- Identificar las competencias y habilidades de los docentes de informática y tecnologías para enseñar el pensamiento computacional.
- Diseñar estrategias para la enseñanza del pensamiento computacional, principalmente en la educación básica.

Agradecimientos

Agradecemos a la Universidad Autónoma de Baja California a través de la Convocatoria de Apoyo a Proyectos de Servicio Social por el financiamiento para realizar las actividades del proyecto desde el año 2016.

Referencias

[1] Paulo Blikstein y Sepi Hejazi Moghadam. *Pre-College Computer Science Education: A Survey of the Field*. Inf. téc. Google, 2018.

[2] Sally A. Fincher y Anthony V. Robins, eds. *The Cambridge Handbook of Computing Education Research*. Cambridge Handbooks in Psychology. Cambridge: Cambridge University Press, 2019. ISBN: 978-1-108-49673-5. DOI: 10.1017/9781108654555.

[3] Secretaría de Educación Pública. *Aprendizajes clave para la educación integral: plan y programas de estudio para la educación básica*. First. Ciudad de Mëxico: Secretaría de Eduación Pública, 2017. ISBN: 978-607-97644-0-1.

[4] María Cristina Cárdenas Peralta, ed. *Marco Referencial Para El Pensamiento Computacional En La Educación Básica*. Ciudad de México: Secretaría de Educación Pública, 2018.

[5] Olimpiada Mexicana de Informática. *Olimpiada Mexicana de Informática*. Nov. de 2020. URL: http://www.olimpiadadeinformatica.org.mx.

[6] Facultad de Ciencias, UABC. *Olimpiada Mexicana de Informática En Baja California*. URL: http://omibc.org.

[7] Robot Turtles. *Robot Turtles – The Board Game That Teaches Programming to Kids*. 2014. URL: http://www.robotturtles.com/.

[8] John Maloney y col. The Scratch Programming Language and Environment. En: *ACM Transactions on Computing Education (TOCE)* 10.4 (2010), págs. 1-15.

[9] Joseph Bergin y col. *Karel J. Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java*. Dream Songs Press, 2005. ISBN: 978-0-9705795-1-5.

[10] OmegaUp. *OmegaUp*. 2020.

[11] Jeannette M Wing. Computational Thinking. En: *Communications of the ACM* 49.3 (2006), págs. 33-35.

[12] Jeannette M Wing. *Computational Thinking Benefits Society*. Social Issues in Computing. Ene. de 2014. URL: http://socialissues.cs.toronto.edu/index.html%3Fp=279.html.

[13] Makey Makey. *Makey Makey*. Joylabz Official Makey Makey Store. 2019. URL: https://makeymakey.com/.

[14] Ridley Scott. *The Martian*. Adventure,Drama,Science Fiction. 2015.

[15] Jesús de la Peña Hernández. *Matemáticas y Papiroflexia*. Asociación Española de Papiroflexia, mar. de 2001. ISBN: 978-84-607-2169-7.

[16] Alfred V Aho y Jeffrey D Ullman. *Foundations of Computer Science*. New York: Computer Science Press, Inc., 1992.

Capítulo 6

Análisis de un caso de estudio de transferencia y consolidación de competencias aplicando el pensamiento computacional

Alberto Pacheco-González [1] y *Rafael Morales Gamboa* [2]

[1] TecNM campus Chihuahua.
alberto.pg@chihuahua.tecnm.mx

[2] Universidad de Guadalajara.
rmorales@suv.udg.mx

Resumen. En este artículo se propone identificar procesos de transferencia y consolidación de competencias asociadas al pensamiento computacional presentes al momento de resolver problemas de programación. Se concluye que, efectivamente, el pensamiento computacional puede considerarse como un conjunto de competencias, susceptibles de ser fomentadas, transferidas y consolidadas de una forma gradual. Para el análisis del caso de estudio se propone un modelo de transferencia basado en la consolidación de competencias del pensamiento computacional usando un método *define-explora-aplica-evalúa* por niveles y etapas. Se demuestra que este abordaje permite identificar y mapear de forma explícita las competencias involucradas al momento de programar, logrando en la práctica contrastar las competencias consolidadas dentro del nivel principiante con respecto al nivel experto donde, en este último, se requiere un mayor esfuerzo mental, pensamiento estratégico y metacognitivo.

Palabras clave. Transferencia del aprendizaje, pensamiento computacional, metacognición, competencias, resolución de problemas.

6.1 Introducción

Desde 2006, Wing caracterizó el pensamiento computacional, estableciendo que involucra la resolución de problemas aplicando conceptos fundamentales de Computación para resolver problemas de forma tal que puedan operar dentro de un sistema computacional [1]. Desde entonces, diversos campos dentro de la Ciencia han considerado la perspectiva computacional como 'un nuevo lenguaje para describir hipótesis y teorías' [2, 3]. La Sociedad Internacional de Tecnología para la Educación (ISTE, por sus siglas en inglés) y la Asociación de Profesores de Ciencias Computacionales (CSTA, por sus siglas en inglés) presentan como definición operativa del pensamiento computacional 'proceso de *resolución de problemas* [énfasis propio] que incluye, pero no está limitado a:

- formular problemas de forma tal que sea posible usar computadoras para resolverlos;
- analizar y organizar datos de forma lógica;
- representar modelos abstractos y simulaciones informáticas;
- automatizar procedimientos aplicando el pensamiento algorítmico;
- identificar, analizar e implementar soluciones haciendo uso eficiente y efectivo de todos los recursos involucrados;
- generalizar y transferir el proceso de resolución de problemas a otras disciplinas o contextos;
- programar, aunque no se limita a la actividad de programación' [4].

El reporte publicado en 2010 del Consejo Nacional de Investigación (National Research Council, NRC) menciona que los conceptos y prácticas del pensamiento computacional son adquiridos de manera más efectiva si son estudiados dentro de las propias materias de cada disciplina en vez de enseñarse solamente dentro del contexto de cursos aislados de programación, factor que puede dificultar la transferencia de competencias del pensamiento computacional a otras disciplinas y contextos [5].

Diversos autores y estudios [1-7], han demostrado que las competencias del pensamiento computacional pueden ser *transferidas* a otros contextos de aprendizaje y resolución de problemas. Sin embargo, el pensamiento computacional requiere una comprensión profunda del proceso de resolución de problemas, a diferencia del proceso de aprendizaje por acumulación de conocimiento y memorización [1]. Si bien es evidente la importancia de la *transferencia* del pensamiento computacional, en la enseñanza es común dar por hecho que el proceso de transferencia de obtiene de manera implícita y automática omitiendo, además, el abordaje de problemas complejos en otros contextos y disciplinas, lo cual no garantiza resultados exitosos y puede incluso terminar cuestionando la utilidad de desarrollar las competencias del pensamiento computacional.

Por lo tanto, la habilidad para resolver problemas es parte del DNA del pensamiento computacional que se espera juegue además un rol clave dentro de los empleos del futuro [8]. Sin embargo, el problema de la *transferencia* de competencias del pensamiento computacional para la resolución de problemas no ha sido operacionalizado por completo [9], por lo que es de especial interés en el presente estudio comprender mejor este problema, para lo cual se recurre a un caso de estudio para identificar la consolidación de cada competencia del pensamiento computacional conforme se implementan algoritmos, es decir, su programación, para obtener la solución del problema por etapas.

6.2 Definiciones y contexto

Existen en la literatura diversas definiciones y tipos de transferencia del conocimiento o aprendizaje [9-12]. En el presente análisis interesa un subconjunto de dicha literatura, en particular, aquella que aborda la transferencia orientada a procesos de resolución de problemas [13-15]. Si bien existen algunos trabajos aún sobre la transferencia del pensamiento computacional a otras disciplinas, según [9], hay cierta ausencia de trabajos que aborden el tema específico de la operacionalización de los procesos de transferencia y menos aún sobre el tema del presente trabajo, específicamente sobre la operacionalización de procesos de transferencia presentes al momento de aplicar el pensamiento computacional dentro de la práctica misma de la programación.

6.2.1 Transferencia

Aunque no existe consenso al respecto, algunas fuentes [10-12] coinciden en que la transferencia es un proceso socio-cognitivo o actividad mental donde las experiencias pasadas (*transfer source*) pueden influir y/o interferir en el desempeño al afrontar una nueva situación (*transfer target*), considerando que emerge una representación mental (esquema o estado mental) a partir de la identificación, selección e integración de contenidos mentales anteriores que sean apercibidos como relevantes o afines (*concordance*) para alcanzar un objetivo determinado (*goal*). La transferencia de conocimiento y habilidades a otros contextos es evidencia de una comprensión y un pensamiento cognitivo de orden superior, que desafortunadamente es pocas veces valorado dentro de los programas educativos [16]. La síntesis entre hechos concretos (percepciones, experiencias) y un saber conceptual (teorías, principios) debe cultivarse (ejemplos y casos) para mejorar la calidad y el desempeño de una competencia y robustecer la creatividad [16, pp. 13-14]. Para el presente trabajo relacionado con las competencias del pensamiento computacional, se delimitó a la transferencia cognitiva enfocada a la resolución de problemas, ya que

existen otras perspectivas, tales como la transferencia socio-emocional [10].

El mecanismo de transferencia no es un fenómeno tipo 'todo-o-nada', que sea claramente medible más allá de una mejora en el desempeño o comportamiento [10]. Además, en ocasiones, no se centra en una sola fuente de transferencia, sino más bien es de naturaleza inestable y dinámica; es decir, requiere de esfuerzo mental y probar de manera ágil diversas fuentes que continuamente compiten, siendo además este proceso muy susceptible al estado de ánimo, la situación, el contexto, las experiencias previas y la motivación individual. Es también una experiencia de aprendizaje situado, muy susceptible al sesgo (*bias*), dado que quién determina qué elementos percibidos (*perceptions*) tienen un mayor sentido o importancia dependerá en buena medida de las experiencias y conceptos previos (*conceptions*) de cada individuo. Algunos autores [10, 16], identifican dicho proceso de integración como apercepción (*apperception*). Por otro lado, según algunos estudios neurocientíficos [17], lo novedoso, emotivo y específico atrae nuestra atención y la significancia de un contenido aumenta si nos resulta familiar y se explora de forma reflexiva y activa dentro de un contexto determinado.

6.2.2 Tipos de transferencia enfocados a la resolución de problemas

Se sugiere a continuación una taxonomía de trabajo, no exhaustiva, más orientada y acotada a la aplicación de las competencias del pensamiento computacional para la resolución de problemas aplicando la programación computacional [10, 18].

- *Referentes al contexto:* Transferencia cercana/lejana (*near/far*), específica/general, dentro/fuera del contexto o tarea (*within/across context or task*).
- *Referentes al grado de esfuerzo cognitivo:* Automática/consciente; superficial/profunda; problemas débiles/fuertes (*weak/strong*); problemas simples/-complejos; conocimiento declarativo/procedimental; basada en conocimiento/habilidad/estrategia; espontánea/intencional/metacognitiva; trayecto bajo/alto *(low/high road*).
- *Referente al abordaje del problema:* Positiva/negativa, creativa, análoga, inversa, lateral.

De interés particular para el presente trabajo fueron elegidos los siguientes tipos de transferencias, mismos que son definidos a continuación.

- *Transferencia análoga o figurativa:* Diversos autores, mencionados en [10], consideran que usar ejemplos, analogías y metáforas es un camino válido para abordar un problema y facilitar un mecanismo de transferencia de competencias.
- *Transferencia de trayecto alto/bajo (high/low road):* Hay instancias de transferencia donde se llevan a cabo procesos mentales de recuperación, mapeo e inferencia de conocimientos, en contraposición a aquellas instancias que

ocurren de manera bastante espontánea o automática. La transferencia por trayecto bajo involucra representaciones mentales que se emplean de forma automática o frecuente, por lo general de tipo procedural o para el caso de competencias previamente consolidadas. Por el contrario, la transferencia por trayecto largo está más impulsada por la creatividad y requiere un mayor esfuerzo cognitivo o metacognitivo y un trayecto de exploración más elaborado y prolongado [13].

- *Transferencia lateral:* Es cuando un aprendizaje o competencia previamente activada se transfiere de nuevo, facilitando su consolidación, pero con un alcance limitado a un nivel de complejidad u operatividad muy similar, como pudiera ser adaptar un procedimiento previamente establecido a otra tarea parecida [10].
- *Transferencia inversa:* A veces llamada transferencia de retroceso, que ocurre cuando un conocimiento se transfiere en la dirección opuesta al proceso de resolución esperado, con el propósito de anotar, ajustar, extender o revisar nuestra experiencia o conocimiento en función de sus similitudes con lo aprendido en una nueva situación [11].

6.2.3 Competencias

De acuerdo con [19], la especificación de una competencia incluye conocimientos, habilidades y actitudes o disposiciones [20], mismas que son observables al momento de efectuar una tarea que prescriba un propósito dentro de un determinado contexto de trabajo. Las habilidades introducen la capacidad de aplicar conocimiento para lograr de forma activa la realización de una tarea. Además, el desarrollo de una habilidad requiere tiempo y práctica [20]. Las tareas también establecen un contexto de acción de una competencia especificando su meta, objetivos, rol y restricciones [20].

6.2.4 Proceso de consolidación de competencias

Dentro del proceso de desarrollo y maduración de una competencia en una persona, se asume que aquellos conocimientos, habilidades y actitudes que se activen y reutilicen de manera más prolongada a lo largo de la resolución de un problema o tarea, terminan siendo 'consolidadas' y se integran como nuevas competencias adquiridas dentro del repertorio personal, mismas que vendrán siendo más susceptibles de ser perfeccionadas y transferidas al momento de ejecutar tareas similares o resolver problemas en otros contextos. De forma análoga, en las etapas tempranas, las competencias incipientes (proto-competencias) que dejen de activarse, no serán 'consolidadas' y, por tanto, estarán ausentes dentro del repertorio de competencias de dicha persona.

6.2.5 Niveles de competencia

De acuerdo al grado de experiencia y rendimiento esperado de una competencia computacional podemos tener los niveles de principiante, intermedio y avanzado. La taxonomía revisada de Bloom ofrece una dimensión provista de los siguientes niveles de competencia cognitivos: recordar, comprender, aplicar, analizar, evaluar y crear [21].

6.3 Objetivo

En el presente estudio es de particular interés explorar el problema con la operacionalización del proceso de transferencia y consolidación de competencias durante la resolución de problemas complejos aplicando las competencias básicas del pensamiento computacional y programación, descomponiendo el proceso en sus componentes referenciales intermedios (etapas) utilizando un caso de estudio con diversos niveles de competencia y dificultad, para identificar procesos de transferencia y consolidación de competencias.

6.4 Hipótesis

Para validar la relación que puede existir entre las nociones de competencia y transferencia, se definen las siguientes hipótesis de trabajo:

- *H1. Activación de una competencia:* En la resolución de un problema o tarea, al inicio del desarrollo de una competencia, el aprendiz elige conocimientos, habilidades, disposiciones y experiencias previas, es decir, activa de forma selectiva protocompetencias, misma que al no estar consolidadas, derivan en un rendimiento observable muy limitado.
- *H2. Consolidación de una competencia:* Con el tiempo y la práctica, reactivando y aplicando de manera reiterativa una protocompetencia bajo distintos contextos (problemas y tareas relacionados), se logra consolidar dicha competencia, es decir, ser integrada de forma efectiva dentro del repertorio de competencias de una persona, logrando, con el tiempo y sucesivas activaciones (reuso de una competencia consolidada) alcanzar niveles más avanzados y un mayor rendimiento en el desempeño de dicha competencia.

6.4.1 Preguntas de investigación

Este trabajo plantea además varias preguntas de investigación sobre el rol que juega la transferencia durante el proceso de aplicación (activación y consolidación) de las competencias del pensamiento computacional y en particular, orientadas al proce-

so de resolución de problemas complejos. A continuación, se detallan las principales preguntas de investigación:

- *Pregunta 1:* ¿Cómo se transfieren?, es decir, ¿cómo se aplican y consolidan las competencias del pensamiento computacional al momento de resolver problemas aplicando la programación?
- *Pregunta 2:* ¿Qué tipos de transferencia se presentan al momento de programar y resolver problemas complejos aplicando las competencias del pensamiento computacional?
- *Pregunta 3:* ¿Se transfieren las mismas competencias del pensamiento computacional para un problema sencillo que uno complejo? ¿Para un programador principiante que uno más avanzado?

6.5 Metodología

Para el presente estudio se integran diversos métodos referidos en [9, 14, 15, 22], tales como (a) el método del pensamiento computacional para resolver problemas aplicando las competencias de abstracción, descomposición, pensamiento algorítmico, evaluación y generalización; (b) el método de resolución de problemas matemáticos que incluye identificar el problema y su contexto, conceptualizar un modelo, explorar, elegir y ejecutar una estrategia de resolución e interpretar los resultados; (c) la teoría de aprendizaje de Polya: comprender el problema, idear plan o hipótesis, experimentar o ejecutar plan y evaluar resultados (*look back*). En síntesis, el proceso que denominaremos como 'define-explora-aplica-evalúa' generaliza e integra las fases más comunes mencionadas en los métodos anteriores, incorporando las competencias básicas del pensamiento computacional [3-5], como sigue:

1. *Define.* Plantear y comprender el problema.
2. *Explora.* Identificar patrones y conocimientos previos, descomponer, abstraer, simbolizar, conceptualizar una hipótesis o modelo.
3. *Aplica.* Seleccionar e implementar el plan, estrategias y algoritmos;
4. *Evalúa.* Reflexionar, revisar y generalizar el conocimiento aplicado o aprendido durante la implementación en la etapa actual (*source*) para identificar posibles patrones, abstracciones y estrategias que puedan ser transferidos a la siguiente etapa (*target*), hasta lograr resolver el problema (*goal*).

6.6 El caso de estudio

6.6.1 Planteamiento del problema

Para el presente caso de estudio se consideró un problema hipotético tomado de un curso a nivel universitario de introducción a la programación.

> Dado un conjunto de cantidades expresadas en pulgadas, obtener las tres cantidades más altas ordenadas de mayor a menor expresadas en centímetros dentro de un rango específico. Por ejemplo, partiendo de un conjunto de lecturas expresadas en pulgadas, convertirlas a centímetros y elegir las tres lecturas más altas dentro del rango de 10 a 40 cm.

6.6.2 Descomposición y cambio de contexto

El abordaje propuesto plantea tres niveles de dificultad o profundidad: nivel principiante, intermedio y avanzado. Cada nivel se descompone en subproblemas (etapas) y se soluciona aplicando las competencias del pensamiento computacional hasta resolver el problema. Al pasar de una etapa a otra se revisa y evalúa de forma introspectiva qué competencias han sido transferidas, observando si se activaron por única ocasión o de manera reiterativa (consolidación). Adicionalmente, se plantea la 'migración' de la implementación a otro lenguaje de programación como ejemplo de transferencia lejana con cambio de contexto y sobre todo, para valorar si son de utilidad las competencias consolidadas anteriormente o si acaso se activan otras competencias al resolver el nuevo problema.

6.7 Etapas del caso de estudio

A continuación, se describe de forma progresiva para cada nivel, cómo fue llevada a cabo la resolución de cada subproblema hasta obtener la implementación final de la solución.

6.7.1 Nivel principiante

Partiendo de la metáfora de un algoritmo secuencial como si fuese una 'calculadora' podemos transferir de manera análoga, cercana y directa (transferencia de bajo trayecto) varios conceptos básicos de Computación (tipo de dato, variables, constantes, operadores, expresiones y operación de salida con formato) para iniciar el abordaje del primer subproblema, lo cual ofrece una excelente oportunidad para reactivar, integrar y poner en práctica dichos conocimientos básicos.

Subproblema 1

Conversión de pulgadas a centímetros aplicando metáfora de la 'calculadora'.[1]
Entrada: Cantidad numérica para pulgadas (Código 6.1:1).

[1] Se hace referencia a las líneas de código usando la notación Código *N*:Línea. Por ejemplo, la Código 6.1:4-7 se refiere a las líneas 4 a 7 dentro de Código 6.1.

Salida: Cantidad numérica para centímetros (Código 6.1:2).
Proceso:

1. Convertir usando una expresión aritmética (Código 6.1:2).
2. Imprimir resultado con formato (Código 6.1:3)

Código 6.1: Conversión de unidades usando como 'calculadora' elementos básicos de programación.

```
1 let pulg = 5.2
2 let cm = pulg * 2.54
3 print ("\(pulg) in = \(cm) cm")
```

```
5.2 in = 13.208 cm
```

Subproblema 2

Partiendo de la solución anterior, el proceso de transferencia de bajo trayecto se enfoca a identificar y encapsular las operaciones (descomposición), declarándolas como funciones con nombres descriptivos (abstracción) (Código 6.2:3-13). Una transferencia sencilla y específica, pero de mayor trayecto es la que corresponde a identificar y declarar estas cantidades que pertenecen a un dominio de aplicación propio (sistema métrico para distancia) como un tipo de dato abstracto Distancia (Código 6.2:1). Por otra parte, la nueva función dec2() refina el formato de salida limitando la impresión hasta dos decimales, presentándose aquí una transferencia de bajo trayecto aplicando operaciones aritméticas básicas (Código 6.2:8). Al generalizar el código en la función p(), se aplicó una transferencia lateral que permite consolidar la competencia específica para formatear datos, activada anteriormente (Código 6.2:3). La solución parcial que aparece en la última sentencia (Código 6.2:15) expresa de manera más compacta, genérica y simbólica el procesamiento (abstracción), es decir, la secuencia como algoritmo de conversión a centímetros, ajuste de decimales e impresión con formato (Código 6.2:15).

6.7.2 Nivel intermedio

En este nivel, al cambiar el nivel de complejidad y abstracción, se amplia y refina el planteamiento del problema, en este caso, consiste en convertir cualquier número de cantidades de pulgadas a centímetros realizando un esfuerzo consciente de transferencia para efectuar la implementación y el aprendizaje obtenido en la etapa anterior, proceso que en la práctica puede llevar a su vez a revisar y modificar en caso necesario la solución anterior (transferencia inversa).

Código 6.2: Solución parcial para el nivel principiante (Swift).

```
1  typealias Distancia = Double
2
3  func in2cm(_ pulg: Distancia) -> Distancia {
4      return pulg * 2.54
5  }
6
7  func dec2(_ num: Distancia) -> Distancia {
8      return (num * 100.0).rounded() / 100.0
9  }
10
11 func p(_ x: Distancia, _ y: Distancia) {
12     print("\t\(x) in = \(y) cm")
13 }
14
15 p( pulg, dec2(in2cm(pulg)) )
```

```
5.2 in = 13.21 cm
```

Subproblema 3

El uso de un *playground* en Swift o *notebook* en Python permite agregar nuevas celdas reutilizando las declaraciones anteriores (Código 6.1 y Código 6.2). Por lo tanto, es posible enfocarnos en definir exclusivamente las nuevas declaraciones. Bajo el nuevo contexto, que requiere manejar más datos, es posible transferir por completo la solución anterior y solo basta incorporar una nueva estructura de datos: *arreglos* en Swift o *listas* en Python (Código 6.3:1) y mejorar el algoritmo incorporando un ciclo para procesar cada dato. Este proceso puede catalogarse como transferencia de trayecto bajo o intermedio porque si bien involucra nuevos conceptos, son muy básicos (Código 6.3:3-5).

Subproblema 4

Para el contexto específico del lenguaje de programación Swift es posible representar las operaciones creadas como propiedades extendiendo el tipo de dato abstracto Distancia (Código 6.4:1-13). Este cambio estructural requiere efectuar una transferencia más profunda, de trayecto alto y más lejana (*far transfer*). Además del uso de metáforas, fue necesario incorporar técnicas, principio y modelos más formales de Computación, tales como: la extensión de clases para incorporar nuevos métodos y atributos (programación orientada a objetos y basada en protocolos) (Código 6.4:1-13). Podemos tomar este caso como buen ejemplo de un concepto que por ser muy

Código 6.3: Arreglo y ciclo para convertir datos usando operación transferidas del paso anterior.

```
1  let datos = [5.2, 19.6, 3.7, 12.1, 16.5, 9.2]
2
3  for x in datos {
4      p(x, dec2(in2cm(x)))
5  }
```

```
 5.2 in = 13.21 cm
19.6 in = 49.78 cm
 3.7 in = 9.4 cm
12.1 in = 30.73 cm
16.5 in = 41.91 cm
 9.2 in = 23.37 cm
```

distinto a lo que se acostumbra a usar en otros lenguajes, que en caso de no ser aplicado de nuevo (reactivado), terminará por ser confuso y probablemente no sea fácil de asimilar, pero que, si se aplica de forma reiterativa en la programación de las siguientes etapas, existirán más posibilidades de que se convierta en una nueva competencia que permita elevar a un nivel más avanzado a un programador, de ahí la importancia del repetir este proceso de transferencia para convertir un concepto meramente teórico en una experiencia práctica y una habilidad, es decir, que se consolide como una nueva competencia en el repertorio del programador aprendiz.

También para el caso específico del lenguaje Swift, la notación simbólica puede mejorar si se usan propiedades calculadas (*computed properties*), tal como se aplican en las propiedades in2cm y dec2 (Código 6.4:2-7), eso permite expresar de manera similar a como se manejan las propiedades en los tipos de datos primitivos (transferencia análoga y metacognitiva de alto trayecto), por ejemplo, en x.in2cm se obtiene la representación del valor de x expresado en centímetros y asimismo se habilita un patrón de encadenamiento (*serialization*) que puede ser de gran utilidad para expresar una secuencia de transformaciones, como es el caso de la secuencia dec2.pad(6) tanto para x como y dentro de la función genérica r() (Código 6.4:16). Esta transferencia lateral, comienza a consolidar una competencia que puede ser de gran utilidad en la versión final más avanzada del código.

6.7.3 Nivel avanzado

En este nivel se incorporan (transfieren) algoritmos y nociones más avanzadas y aprendizajes de las etapas previas, para respaldar la consolidación de un repertorio de competencias más apropiado para abordar la parte más compleja del problema.

Código 6.4: Solución parcial implementada en el nivel intermedio (Swift).

```
1  extension Distancia {
2      var dec2 : Distancia {
3          return (self * 100.0).rounded() / 100.0
4      }
5      var in2cm : Distancia {
6          return self * 2.54
7      }
8      func pad(_ len: Int) -> String {
9          let s = String(self)
10         let tam = s.count
11         return String(repeating:" ", count: len - tam) + s
12     }
13 }
14
15 func r(_ x: Distancia, _ y: Distancia) {
16     print("\t\(x.dec2.pad(6)) in =\(y.dec2.pad(6)) cm")
17 }
18
19 for x in datos {
20     r(x, x.in2cm)
21 }
```

```
 5.2 in = 13.21 cm
19.6 in = 49.78 cm
 3.7 in =  9.4 cm
12.1 in = 30.73 cm
16.5 in = 41.91 cm
 9.2 in = 23.37 cm
```

Retomando el planteamiento del problema completo, donde a partir de un conjunto de lecturas en pulgadas, se seleccionan valores dentro de un rango, las tres cantidades más grandes se deben reportar ordenadas de mayor a menor y expresarlas en centímetros. Adoptando la metáfora de la 'tubería' (*pipes*), i. e. proceso gradual en secuencia de un flujo de datos de manera similar a como se manejan comandos pipe dentro de *scripts* para el *shell* de Unix. Analizando modelos y alternativas (transferencia de alto trayecto) se elige un enfoque basado en programación funcional. Se transfieren sin cambios, las operaciones definidas de las etapas anteriores para convertir unidades (in2cm) y expresar cantidades bajo un formato de dos decimales de precisión (dec2). Sin embargo, es necesario implementar ahora las transformaciones restantes para obtener una solución completa (esfuerzo de alto trayecto) que puede simplificarse usando primero un prototipo rápido y después transferir lo aprendido, si fue apropiadamente asimilado y consolidado, permitirá obtener la solución final del problema.

Subproblema 5

Usando como estrategia la noción de cerradura (*closure*) y 'algoritmo' para arreglos del lenguaje Swift (`map`, `filter` , `sorted`), aplicando además la metáfora de la tubería, es posible idear un prototipo rápido concatenando operaciones usando la estrategia *objeto.propiedad1.propiedad2...* Partiendo del arreglo de datos inicial (Código 6.5:1), se aplica el siguiente algoritmo: convertir las unidades a centímetros *mapeando* cada elemento con la función `in2cm` (Código 6.5:2); *filtrar* los datos ubicados dentro del rango de 10 a 40 cm (Código 6.5:3); *ordenar* los datos de mayor a menor (Código 6.5:4); *recortar* los primeros tres elementos del arreglo (Código 6.5:5); finalmente, *mapear* a cantidades con dos decimales reusando la función `dec2` (Código 6.5:6).

Al ejecutar dicha celda del *playground* de Swift (Código 6.5) es posible corroborar que efectivamente se obtienen los resultados correctos (sin formato), por lo que es factible transferir correctamente lo aprendido en esta etapa (autoeficacia) para obtener la implementación final, donde aún es posible aplicar y transferir abstracciones, algoritmos y patrones similares que permitan mejorar la representación simbólica del código de la etapa final.

Subproblema 6

Transfiriendo lo aprendido en la etapa anterior, se busca consolidar las competencias ya 'activadas' y mejorar la representación abstracta y simbólica de la solución a un nivel cercano al seudocódigo. Para ello se recurre de nuevo al patrón basado en tipos datos abstractos: usando seudotipos como estrategia para representar un arreglo de distancias `ListDist` y un rango para dichas distancias `Rango` (Código 6.6:1-2). La extensión del tipo `ListDist` incorpora las operaciones de transforma-

Código 6.5: Primer prototipo aplicando programación funcional (Swift).

```
datos
    .map    { $0.in2cm }
    . filter  {  (10...40). contains($0) }
    .sorted  ( by: > )
    . prefix  ( upTo: 3 )
    .map    { $0.dec2 }
```

```
3 elements
— 0 : 30.73
— 1 : 23.37
— 2 : 13.21
```

ción de flujos de datos (Código 6.6:4-21). Con ello se logra obtener como algoritmo una secuencia de transformaciones simbólicamente más clara y parametrizada (Código 6.6:24-29). Dicho algoritmo se 'oculta' dentro de la función top() (Código 6.6:23-30), para generalizarlo agregando parámetros. En esta etapa se aplicaron todas las competencias del pensamiento computacional y tipos de transferencias, predominado la *transferencia de largo trayecto*, es decir, la metacognición, esfuerzo mental, revisión estratégica del conocimiento y experiencia adquirida que en la práctica puede potenciar la consolidación de las competencias desarrolladas a lo largo del proceso para ser aplicadas en un futuro en la resolución de nuevos y diferentes problemas, incluso para otras disciplinas.

Por último, se imprimen con formato los resultados obtenidos. Esto se logra de manera directa transfiriendo la estrategia manejada en la implementación de etapas anteriores (transferencia lateral). Para ello, la estrategia y competencia 'consolidada' basada en extender un tipo de dato, en este caso Distancia permite convertir centímetros a pulgadas mediante la nueva propiedad cm2in (Código 6.7:1-3). Mediante un simple algoritmo compuesto por un ciclo es posible reportar los resultados generados por la función top(), misma que simboliza los datos y parámetros que aparecen en el planteamiento del problema (Código 6.7:6).

6.8 Transferencia con cambio de contexto

El reto para esta etapa consiste en *transferir* parte de lo aprendido en las etapas previas, tales como principios, abstracciones, patrones y sobre todo competencias consolidadas, ya que, si no se activaron y reusaron de manera regular y ejercitando un creciente nivel de dificultad, es decir, si no se consolidaron, será poco exitosa la labor de resolver un problema similar bajo otro contexto, en este caso, en otro lenguaje

Código 6.6: Mejorando la representación simbólica y aportando una solución más genérica (Swift).

```
typealias ListDist = [Distancia]
typealias Rango = (min: Distancia, max: Distancia)

extension ListDist {
    var in_2_cm: ListDist {
        return self.map { $0.in2cm }
    }
    func en_rango(_ r: Rango) -> ListDist {
        let (mn, mx) = r
        return self. filter { (mn...mx).contains($0) }
    }
    var ord_may: ListDist {
        return self.sorted(). reversed()
    }
    func top_n(_ n: Int) -> ListDist {
        return Array(self.prefix(upTo: n))
    }
    var decim_2: ListDist {
        return self.map { $0.dec2 }
    }
}

func top( lista : ListDist , rango: Rango, tam: Int) -> ListDist {
      return lista
        .in_2_cm
        .en_rango(rango)
        .ord_may
        .top_n(tam)
        .decim_2
}

top( lista :datos, rango:(min:10, max:40), tam:3)
```

```
3 elements
- 0 : 30.73
- 1 : 23.37
- 2 : 13.21
```

Código 6.7: Solución final reusando el mismo patrón para convertir a pulgadas (Swift).

```
extension Distancia {
    var cm2in: Distancia { return self / 2.54 }
}

for cm in top( lista :datos, rango:(min:10, max:40), tam:3) {
    r(cm.cm2in, cm)
}
```

```
12.1 in = 30.73 cm
 9.2 in = 23.37 cm
 5.2 in = 13.21 cm
```

de programación, que es usando Python.

Subproblema 7

Como primer experimento de transferencia, se creó un prototipo rápido, que descrito de forma sumaria (los pasos de las etapas anteriores aparecen detallados en [23, 24]), aplica funciones *lambda* para definir las operaciones básicas in2cm y dec2 correspondientes a las primeras etapas (Código 6.8:1-2). Se usa una lista para los datos de entrada (Código 6.8:4), donde se inicia la conversión a centímetros de cada lectura aplicando la función lambda in2cm (Código 6.8:10). Posteriormente, se eligen los datos que se ubican dentro de un rango usando un filtro (Código 6.8:9). El resultado es ordenado en orden inverso, de mayor a menor (Código 6.8:8), para posteriormente reducir la precisión a dos dígitos decimales (Código 6.8:7) y al final tomar los primeros tres elementos de la lista (Código 6.8:12) que fueron los mismos resultados obtenidos en el caso anterior. Bajo este nuevo contexto, algoritmicamente resulta menos entendible la representación de una tubería en Python debido a la forma de anidar la ejecución de funciones dentro de funciones que dan por resultado un orden inverso (de derecha a izquierda o abajo hacia arriba) que puede resultar confuso y plagado de paréntesis. Por tanto, el reto es, para la última etapa, mejorar y acercar la representación simbólica a la tubería implementada en lenguaje Swift.

Subproblema 8

A diferencia de Swift, Python no soporta la implementación de tipos de datos abstractos con el mismo status que un tipo de dato nativo (*first-citizen class*), ni ofre-

Código 6.8: Primer prototipo de la transferencia a Python basada en programación funcional.

```
in2cm = lambda x: x * 2.54
dec2  = lambda x: round(x * 100.0) / 100.0

datos = [5.2,  19.6,  3.7,  12.1,  16.5,  9.2]

 list (
     map(dec2,
         reversed(sorted(
             filter (lambda x: 10 <= x <= 40,
                         map(in2cm, datos)
 )))))[:3]
```

```
[30.73,  23.37,  13.21]
```

ce la posibilidad de extender tipos de datos primitivos —aunque una opción sería usar clases para emular datos primitivos. Sin embargo, podemos *extender* Python por medio de librerías. Como una alternativa para hacer más legible el algoritmo usando programación funcional se usó la librería PyToolz [25]. La implementación final en este lenguaje incorpora dicha librería para lograr una tubería semejante al manejo de propiedades en Swift. De hecho, es precisamente la función `pipe()` la que nos ayudará a transferir la secuencia de transformaciones (algoritmo), expresadas como funciones lambda (Código 6.9:3-6), de forma similar a como se logró con el primer caso (Código 6.9:17-23). Con la excepción de que, si sus operaciones requieren parámetros, estas se declaran como *funciones parcializadas* mediante la anotación `@curry` para permitir un manejo similar a una función lambda (Código 6.9:9-14). Dado que la mayor parte de estas declaraciones pueden ocultarse dentro de una librería, lo importante para el usuario final es que en ambos casos se resuelve el problema ejecutando prácticamente las mismas instrucciones tanto en Swift (Código 6.7:5-7), como en Python (Código 6.9:25-26).

6.9 Análisis de los resultados

Tradicionalmente se conoce como 'migrar' o 'portar' al proceso de *transferir* código implementado en un lenguaje de programación a otro. Dicho proceso se llevó a cabo de Swift a Python, sin embargo, para el presente estudio es de mayor interés analizar la forma de abordar el problema, particularmente, el proceso metacognitivo aplicado en la resolución *gradual* del problema, entrelazando el pensamiento computacional y la programación, observando y analizando la forma en que se van

Código 6.9: Solución final en Python aplicando metáfora de la tubería.

```
from toolz import *

in_2_cm = lambda L: map(in2cm, L)
decim_2 = lambda L: map(dec2, L)
ord_may = lambda L: reversed(sorted(L))
cm2in   = lambda x: x / 2.54
def r(x, y): print(f"{x:6.2f} in ={y:6.2f} cm")

@curry
def en_rango(r, L):
    return filter (lambda x: r[0] <= x <= r[1], L)

@curry
def top_n(n, L): return list (take(n, L))

def top(L, rango, tam):
    return list (pipe(
        L,
        in_2_cm,
        en_rango(rango),
        ord_may,
        top_n(tam),
        decim_2))

for cm in top(datos, rango=(10,40), tam=3):
    r(cm2in(cm), cm)
```

```
12.10 in = 30.73 cm
 9.20 in = 23.37 cm
 5.20 in = 13.21 cm
```

activando y consolidando las competencias necesarias para resolver con éxito toda una familia o dominio de problemas lo más amplio posible.

En la Tabla 6.1 se resume el análisis comparativo para ambas implementaciones. Ambos lenguajes soportan un entorno interactivo (*notebooks*), programación funcional y un conjunto de operaciones a nivel de listas capaces de integrar funciones parcializadas, mismas que facilitaron la implementación tipo 'tubería' (*pipes*) mediante algoritmos con *closures* en Swift y funciones *lambda* en Python. En Swift fue posible extender un tipo de dato primitivo o abstracto, lo cual permitió que de forma nativa fuera posible emular tuberías usando propiedades y métodos de clases extendidas. En Python fue necesario usar una librería externa para lograr una notación más natural para el uso de tuberías. En cuanto al formato de impresión de salida, las cadenas con formato (*f-strings*) de Python 3 ofrecen un mecanismo más avanzado que el disponible en Swift. Por otra parte, Swift ofreció un mejor entorno para depurar errores por ser un lenguaje compilado con reglas más estrictas para el manejo de tipos de datos (*strongly typed*), sin embargo, Python es más flexible y permite lograr una implementación más directa y compacta, con la desventaja de que es más difícil encontrar el origen de una falla y corregirla comparado con un compilador más preciso, robusto y confiable de Swift, como sucede en la declaración de constantes y estructuras de datos inmutables, como en el uso de **let** (Código 6.1:1-2) y la gran conveniencia de extender y crear sinónimos de tipos, e.g. **typealias** (Código 6.6:1-2) que permiten acercar la representación de objetos al dominio del problema. Como observación final, en ambos casos no se declararon clases para evitar imponer un nivel adicional de abstracción en un nivel intermedio de programación.

Tabla 6.1: Análisis comparativo de ambas implementaciones.

Funcionalidad nativa del lenguaje	Swift	Python
Entorno interactivo en línea	Sí	Sí
Programación funcional y funciones parcializadas	*Closures*	*Lambda*
Extender tipos de datos (abstracción)	Sí	No
Algoritmos, transformaciones e iteradores a nivel de listas o arreglos	Sí	Sí
Notación secuencial para tuberías	Sí	No
Formato avanzado de impresión	No	Sí
Verificación fuerte de tipos de datos	Sí	No

La Tabla 6.2 recopila un resumen sobre el análisis de los tipos de transferencias de aprendizaje identificados en los diferentes niveles y etapas (subproblemas) aplicando el pensamiento computacional y la programación, destacando las que se

describen a continuación.

- *La transferencia analógica* prevalece en los niveles principiante e intermedio donde el uso de metáforas facilita y enmarca de manera más accesible tanto el proceso de resolución como la valoración de los avances logrados, pero a su vez, se limita a activar competencias muy elementales, corriéndose el riesgo de limitar y estancar el alcance y nivel de resolución de problemas, a menos de que una vez consolidadas las competencias básicas, mediante múltiples oportunidades de transferencias cada vez de más largo trayecto, se logre consolidar la comprensión y aplicación de un conocimiento más profundo y técnico, como fue en este caso con las nociones de tipos de datos abstractos y funciones parcializadas propias del paradigma de programación funcional.
- *La transferencia de bajo trayecto* es necesaria para facilitar la consolidación de las competencias involucradas y desde luego, predomina en los niveles menos avanzados, limitándose al principio a conceptos muy básicos de Computación (variables, constantes, operadores, expresiones, funciones, listas y ciclos), mismos que son reutilizados de forma más rutinaria y mecánica en el nivel avanzado. Sin embargo, es también relativa, ya que cuando las competencias más avanzadas o complejas se consolidan, terminarán en convertirse en transferencias laterales (rutinarias) y de bajo trayecto (esfuerzo) para sus practicantes expertos.
- *La transferencia de alto trayecto* se presenta principalmente en los saltos de nivel, de principiante a intermedio y de intermedio a avanzado, y también su intensidad aumenta sobre todo en las últimas etapas de los niveles más avanzados, al requerir un esfuerzo mental más consciente, i.e. auto cognitivo, para revisar y generalizar las estructura y algoritmos que serán transferidos en la siguiente etapa, como puede ser la identificación de patrones y operaciones para un mismo tipo de datos que son particulares y específicos para el problema en cuestión, que 'detonan' la necesidad de transferir nociones más abstractas, i.e. tipo de dato abstracto. En el nivel avanzado es donde juega un rol preponderante este tipo de trasferencia, principalmente en las fases explora y evalúa del proceso, al momento de elegir y aplicar esquemas de solución cada vez más complejos y abstractos, tales como las transiciones de Código 6.5 a Código 6.6 y de Código 6.8 a Código 6.9. Por último, para 'portar' la solución a Python fue sin duda clave este tipo de transferencia.
- *La transferencia lateral* corresponde al proceso rutinario de programación que permite ir consolidando competencias que mejoran significativamente el rendimiento del programador, que son competencias de repertorio que terminan en gran medida siendo automatizadas o efectuadas cada vez con menor esfuerzo, y por ello a veces se ignoran o pasan por alto al tratar de explicar o enseñar a terceros los pasos a seguir. Hace tal vez un uso muy efec-

tivo del reconocimiento de patrones para saber cuándo y dónde deben aplicarse (transferirse) determinados patrones. En Código 6.7 del nivel avanzado puede apreciarse este tipo de transferencia, que cuando se convierte en una competencia consolidada puede parecer cosa de magia para los demás, por ejemplo, cómo identificar y reproducir rápidamente el mismo esquema de extensión de tipos vía propiedades para resolver el problema en cuestión usando menos código.

- *La transferencia inversa* es también un caso al que se presta poca atención pero que es una característica inherente del proceso gradual de desarrollo de soluciones complejas en programación y en el pensamiento computacional, donde lo que se 'descubre' en una etapa posterior o más avanzada puede ayudar a reescribir o mejorar las etapas anteriores, lo cual puede ser una valiosa oportunidad para reajustar y perfeccionar una competencia en vías de consolidación. Desde luego, no hay evidencia de la misma en las figuras presentadas, pero durante el proceso de resolución, en varias ocasiones se encontraron aspectos nuevos dentro del destino de la transferencia que fueron transferidos a etapas previas (origen). Como ejemplos, en la Tabla 2 se mencionan desde el cambio de nombre de variables y funciones, hasta la reorganización completa del código en el nivel más avanzado. Faltaría destacar, que la migración de la solución a Python aportó también transferencias inversas, como fue el caso de los iteradores y las funciones lambda.

Tabla 6.2: Tipos de transferencias identificados durante el caso de estudio.

Tipo de transferencia	Nivel		
	Principiante	Intermedio	Avanzado
Analógica	Metáfora de la calculadora	Metáfora de la calculadora	Metáfora de la tubería
Bajo trayecto	Definir funciones	Listas y ciclos	Reutilizar código de etapas previas
Alto trayecto	Tipo de dato abstracto	Extensión de tipo	Migrar a otro lenguaje
Lateral	Dar formato	Declaraciones	Proceso de parcialización
Inversa	Empatar declaraciones	Reestructuración	Portar funcionalidad

Finalmente, las tablas 6.3, 6.4 y 6.5 reportan los distintos procesos de transfe-

rencia, activación y consolidación de competencias específicas de programación y competencias generales de pensamiento computacional para cada uno de los niveles de competencia, Principiante (P), Intermedio (M) y Avanzando (A) y etapas del caso de estudio (subproblemas 1 al 8, referidos usando la notación S1–S8), ubicados en la última columna de cada tabla. Se han cuantificado las oportunidades de activación de competencias y se sugiere un modelo de transferencia que inicia con la identificación de conocimientos y habilidades de programación, fundamentos de programación, paradigmas de programación (funcional) y cambio de lenguaje y entorno (contexto), que se espera sean transferidos y aplicados mediante algún tipo de transferencia (primera columna), activando así determinadas competencias de programación (segunda columna) y competencias del pensamiento computacional (tercera columna, subdividida de acuerdo con los componentes descomposición (D), abstracción (A), algoritmos (G) y reconocimiento de patrones(P)).

Se analizan entonces los niveles de activaciones de las competencias del pensamiento computacional como un posible instrumento para mostrar el grado de oportunidades de consolidación de dichas competencias. Para ejemplificar como se llevó a cabo el conteo de las oportunidades de activación de competencias, se observa que en segunda etapa del nivel Principiante (S2) destaca la activación en tres ocasiones de la competencia de descomposición, en la declaración de tres funciones. Dichas activaciones pueden contribuir de manera positiva a consolidar esa competencia del pensamiento computacional. Asimismo, se identifica la activación de dos mecanismos distintos de abstracción: definir seudotipos (alias) y funciones. También se identifica la aplicación de un tipo de algoritmo (un ciclo, en S2), dos tipos de patrones repetitivos: expresiones tipo calculadora (S1) o conversiones y una tubería (S2). En este tipo de análisis es posible evaluar en parte el mérito didáctico del ejercicio: si promueve o no la reactivación, en las etapas posteriores, de la competencia de abstracción mediante la definición y uso de tipos de datos abstractos. De no ser así dicha competencia difícilmente tendrá oportunidad de consolidarse y, por lo tanto, no será susceptible de ser transferida con éxito para resolver otros problemas. Afortunadamente, en una etapa posterior (S6) se reitera en dos ocasiones su activación y aplicación, aumentando así las probabilidades de que se consolide como una competencia que llegue a formar parte del repertorio de competencias transferibles del aprendiz de programador. De igual forma se fueron identificando y contabilizando las activaciones para cada competencia de pensamiento computacional y los resultados se presentan en las tablas. Es importante recordar que en las últimas etapas (S5-S8) se involucraron todos los requerimientos del problema planteado, lo cual incrementa de manera significativa la cantidad de activaciones para ciertas competencias en dichas etapas. Por ello, resulta conveniente comparar primero los niveles Principiante e Intermedio (Tabla 6.3) y analizar por separado el nivel Avanzado (Tabla 6.4 y Tabla 6.5).

Para los primeros dos niveles, se involucran nociones básicas de programación y hay un bajo nivel de activaciones, lo cual limita las oportunidades de consolidar tanto las competencias de programación como de pensamiento computacional, sobre todo en lo que respecta a las competencias que apliquen algoritmos y capacidades de abstracción. Se observa además un alto riesgo de que algunos aprendices tengan dificultades en el nivel Intermedio (S4) para asimilar la noción de extensión de tipos. Afortunadamente, este tipo de análisis permite detectar dichos riesgos y, por tanto, tomar las previsiones necesarias, como sería explicar con mayor detenimiento esos conceptos antes de llevar la competencia a la práctica, para evitar frustraciones, confusión y un mal rendimiento en el ejercicio de la competencia, que es otro factor que puede inhibir su consolidación y transferencia.

Por último, respecto al nivel Avanzado (S5-S8), es importante recordar que S5 y S6 corresponden a la solución en lenguaje Swift, en tanto que S7 y S8 son su equivalente en Python, pues al parecer existe un patrón en ambos conjuntos en el que destaca un menor índice de activaciones en las etapas iniciales (S5 y S7), como etapas de transición desde el nivel intermedio, en las cuales se produce solamente un prototipo que, sin embargo, permite reducir la complejidad del código y comprender mejor las estrategias y técnicas de solución del problema para, en la siguiente etapa, abordar con éxito la solución final. En la última etapa existen más activaciones de las competencias del pensamiento computacional debido a la extensión misma de la implementación final, destacando las activaciones para la descomposición (sobre todo aplicando funciones lambda) y para algoritmos de funciones lambda. Se observa además que la competencia genérica de abstracción se incrementa y diversifica, lo cual implica una mayor complejidad (más transferencias de alto trayecto) y la aplicación de un mayor esfuerzo mental, estratégico y metacognitivo. Este alto índice de activaciones hace posible también que en este nivel se logre una mejor consolidación de las competencias del pensamiento computacional y alcanzar así los niveles cognitivos superiores de la taxonomía de Bloom, que caracterizan al nivel avanzado de programación.

6.10 Hallazgos, conclusiones y trabajo a futuro

El método define-explora-aplica-evalúa por niveles y etapas, permitió identificar de forma explícita tanto la activación, consolidación y transferencia de competencias. Además, dicho método y el análisis del caso de estudio permitieron validar de forma parcial las hipótesis planteadas previamente, referentes a la activación (H1) y consolidación (H2) de las competencias de programación y del pensamiento computacional en la resolución de problemas por etapas y niveles.

En este caso de estudio encontramos además que: (1) al resolver problemas, es conveniente el uso de metáforas para facilitar el proceso inicial de transferencia de

Tabla 6.3: Proceso de transferencia y consolidación de competencias en cada etapa y nivel para conocimientos y habilidades de *fundamentos de programación*. Las competencias del pensamiento computacional (CPC) son descomposición (D), abstracción (A), algoritmos (G) y reconocimiento de patrones(P); los niveles (N) corresponde a Principiante (P), Intermedio (M) y Avanzado (A); los subproblemas son S1 a S8.

Tipo de transferencia	Competencias de programación	Activaciones de CPC				
		D	A	G	P	N
Analógica	Formato de impresión					P
Bajo trayecto	Declarar variables	3	2	1	2	S1
Lateral	Expresiones y funciones					S2
Analógica	Listas y ciclos					M
Bajo trayecto	Declarar variables	4	3	1	4	S1
Lateral	Expresiones y funciones					S2

Tabla 6.4: Proceso de transferencia y consolidación de competencias en cada etapa y nivel para conocimientos y habilidades de *paradigmas de programación (funcional)*. Las competencias del pensamiento computacional (CPC) son descomposición (D), abstracción (A), algoritmos (G) y reconocimiento de patrones(P); los niveles (N) corresponde a Principiante (P), Intermedio (M) y Avanzado (A); los subproblemas son S1 a S8.

Tipo de transferencia	Competencias de programación	Activaciones de CPC				
		D	A	G	P	N
Analógica Alto trayecto Lateral	Funciones lambda Secuención de lambdas (tuberías) Algoritmos lambda	5	4	5	5	A S5
Analógica Alto trayecto Lateral	Pseudotipos Extensiones de tipos Tuberías y algoritmos lambda	4	3	1	4	A S6

Tabla 6.5: Proceso de transferencia y consolidación de competencias en cada etapa y nivel para conocimientos y habilidades de *cambio de lenguaje y entorno (contexto)*. Las competencias del pensamiento computacional (CPC) son descomposición (D), abstracción (A), algoritmos (G) y reconocimiento de patrones(P); los niveles (N) corresponde a Principiante (P), Intermedio (M) y Avanzado (A); los subproblemas son S1 a S8.

Tipo de transferencia	Competencias de programación	Activaciones de CPC¨				
		D	A	G	P	N
Alto trayecto Lateral Inversa	Migrar solución de lenguaje Funciones, pipes y algoritmos lambda	2	3	8	3	A S7
Alto trayecto Lateral Inversa	Formato avanzado cadenas Programación funcional avanzada Librería funcional avanzada	14	5	18	8	A S8

bajo trayecto, ya que permiten consolidar de forma más temprana las competencias básicas; (2) para todos los niveles es importante proveer tiempo y espacio suficiente para ejercitar y consolidar las competencias más adecuadas para cada nivel; (3) para consolidar una competencia, al cambiar de etapa, nivel o contexto, es importante promover la reactivación de las competencias activadas en las etapas previas para lograr consolidarlas; (4) las fases aplica y evalúa son particularmente valiosas para promover la metacognición, plantear estrategias e identificar las competencias que han sido transferidas o están en proceso de consolidación; (5) bajo un enfoque de aprendizaje centrado en la consolidación gradual de competencias, la transferencia de bajo trayecto propia del nivel principiante no se equipara al rendimiento que puede alcanzar una competencia determinada que ha sido consolidada de forma más sólida, reiterada y diversificada, por lo que resulta conveniente, durante su proceso de consolidación, elevar de forma gradual la complejidad y brindar suficientes oportunidades y una gran diversidad de problemas y contextos. ´ Finalmente, con lo anteriormente expuesto, es posible responder de forma parcial a las preguntas de investigación planteadas y proponer posibles trabajos a futuro.

- *Pregunta 1:* La fase metacognitiva *evalúa* fue fundamental para generalizar e identificar procesos de transferencia en las fases *explora* y *aplica*. Dado el planteamiento del problema, la solución de la etapa previa (fuente de la transferencia) y los cambios realizados en la etapa subsecuente (destino de la transferencia), fue posible identificar los procesos de consolidación y transferencia de competencias (activaciones), y monitorear además su impacto en las etapas subsecuentes.
  Trabajo futuro: Es posible complementar este enfoque con diversas teorías sobre metacognición [27], para operacionalizar con mayor precisión a nivel de procesos de (micro)transferencia subdividiendo cada etapa en tareas.
- *Pregunta 2:* Para el presente caso de estudio se identificaron cinco procesos diferentes de transferencia (Tabla 6.2), algunos de los cuales son transversales (transferencia analógica, lateral e inversa) y otros dependen del nivel de complejidad.
  Trabajo futuro: En la práctica, el presente análisis permitió integrar varios temas aislados dentro de un curso de programación básico [26]. La más reciente Curricula de Computación de la ACM/IEEE (CC2020) [21] incorpora un enfoque basado en competencias, que puede facilitar la incorporación del esquema propuesto para auxiliar en la consolidación de dichas competencias, por lo que a futuro se propone su integración y validación experimental con estudiantes.
- *Pregunta 3:* Los procesos de transferencia pueden diferir con respecto a varios criterios, dimensiones y alcances, para el presente caso de estudio, en los niveles menos avanzados, predominó la transferencia de bajo trayecto y un

bajo índice de activación para competencias con nivel de abstracción bajo. En cambio, para el nivel avanzado, predominó la transferencia de alto trayecto y un mayor índice de activaciones que permitieron consolidar de forma más sólida las competencias debido en parte, al esfuerzo mental, la metacognición, uso de estrategias y conocimientos más abstractos involucrados al momento de resolver problemas.
Trabajo futuro: Las competencias consolidadas en el presente caso de estudio permitirían en un futuro, para fines didácticos, incorporar contextos más avanzados de programación (dominios), tales como microservicios con funciones lambda, programación reactiva, Combine de Swift y MapReduce para minería de datos [26-29].

Referencias

[1] Jeannette M Wing. Computational Thinking. En: *Communications of the ACM* 49.3 (2006), págs. 33-35.

[2] Alan Bundy. Computational Thinking Is Pervasive. En: *Journal of Scientific and Practical Computing* 1.2 (2007), págs. 67-69. ISSN: 1936-5020.

[3] Siu-Cheung Kong y Harold Abelson. *Computational Thinking Education*. Springer, 2019, pág. 26. ISBN: 978-981-13-6528.

[4] ISTE y CSTA. *Computational Thinking in K–12 Education Leadership Toolkit*. ISTE, 2011.

[5] National Research Council. *Report of a Workshop on the Scope and Nature of Computational Thinking*. National Academies Press, abr. de 2010. ISBN: 978-0-309-15372-0.

[6] Ashok Basawapatna y col. Recognizing Computational Thinking Patterns. En: *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*. SIGCSE '11. New York, NY, USA: Association for Computing Machinery, mar. de 2011, págs. 245-250. ISBN: 978-1-4503-0500-6. DOI: 10.1145/1953163.1953241.

[7] Suchi Grover, Roy Pea y Steve Cooper. "Systems of Assessments" for Deeper Learning of Computational Thinking in K-12. En: *Proceedings of the Annual Meeting of the American Educational Research Association*. Chicago: American Educational Research Association, 2015, págs. 15-20.

[8] World Economic Forum. *The Future of Jobs Report: 2020*. Inf. téc. World Economic Forum, oct. de 2020.

[9] M. Pedaste y col. Complex Problem Solving as a Construct of Inquiry, Computational Thinking and Mathematical Problem Solving. En: *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*. Vol. 2161-377X. Jul. de 2019, págs. 227-231. DOI: 10.1109/ICALT.2019.00071.

[10] Sacha Helfenstein. *Transfer : Review, Reconstruction, and Resolution*. Jyväskylä Studies in Computing 59. Finland: University of Jyväskylä, 2005. ISBN: 951-39-2386-X.

[11] Henry C. Ellis. *The Transfer of Learning*. New York: The Macmillan Company, 1965.

[12] R.S. Woodworth. *Experimental Psychology*. New York: Holt, Rinehart and Winston, 1938.

[13] Richard E. Mayer y Merlin C. Wittrock. Problem-Solving Transfer. En: *Handbook of Educational Psychology*. Ed. por David C. Berliner y Robert C. Calfee. Routledge, 1996. ISBN: 978-0-8058-5080-2.

[14] Van Bien Nguyen, Eduard Krause y Cam Tho Chu. Problem Solving. En: *Comparison of Mathematics and Physics Education I : Theoretical Foundations for Interdisciplinary Collaboration*. Ed. por Simon Friedrich Kraus y Eduard Krause. MINTUS – Beiträge Zur Mathematisch-Naturwissenschaftlichen Bildung. Wiesbaden: Springer Fachmedien, 2020, págs. 345-368. ISBN: 978-3-658-29880-7. DOI: 10.1007/978-3-658-29880-7_14.

[15] Y. Afiyati, K. Warniasih y N. W. Utami. Problem-Solving with Guided Inquiry Learning: An Analysis of Student's Problem-Solving Ability. En: *Journal of Physics: Conference Series* 1581 (jul. de 2020). ISSN: 1742-6596. DOI: 10.1088/1742-6596/1581/1/012035.

[16] H.Lynn Erickson. *Concept-Based Curriculum and Instruction for the Thinking Classroom*. Corwin Press, 2007. ISBN: 978-1-4129-1700-1.

[17] Eric Jensen. *Teaching with the Brain in Mind*. Revised 2nd edition. Alexandria, Va: ASCD, jun. de 2005. ISBN: 978-1-4166-0030-5.

[18] D.H. Schunk. *Learning Theories: An Educational Perspective*. Fourth. Upper Saddle River, New Jersey: Pearson Merrill Prentice Hall, 2004, págs. 217-224.

[19] Leslie Waguespack y col. Adopting Competency Mindful of Professionalism in Baccalaureate Computing Curricula. En: *2019 Proceedings of the EDSIG Conferen*. Cleveland, Ohio, EUA: Information Systems and Academic Profession, 2019, págs. 1-17.

[20] Association for Computing Machinery y IEEE Computer Society. *The Computing Curricula 2020*. Inf. téc. Draft v44. Association for Computing Machinery (ACM) and IEEE Computer Society, nov. de 2020.

[21] Lorin W. Anderson y David R. Krathwohl. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. White Plains, NY: Longman, 2001. ISBN: 0-8013-1903-X.

[22] K. R. Daulay e I. Ruhaimah. Polya Theory to Improve Problem-Solving Skills. En: *Journal of Physics: Conference Series* 1188 (2019). ISSN: 1742-6596. DOI: 10.1088/1742-6596/1188/1/012070.

[23] Alberto Pacheco. *Caso de Estudio: Transferencia Por Niveles En Swift*. GitHub. 2021. URL: https://github.com/AlbertoTecNM/transfer-swift.

[24] Alberto Pacheco. *Caso de Estudio: Transferencia Por Niveles En Python*. GitHub. 2021. URL: https://github.com/AlbertoTecNM/transfer-python.

[25] Matthew Rocklin y John Jacobsen. *PyToolz API Documentation*. Toolz 0.10.0 documentation. 2013. URL: https://toolz.readthedocs.io/en/latest.

[26] Amazon. *AWS Lambda Documentation*. 2021.

[27] Apple. *Combine*. Apple Developer Documentation. 2021. URL: https://developer.apple.com/documentation/combine.

[28] Luke Lee. *First Steps with PySpark and Big Data Processing*. Real Python. 2021. URL: https://realpython.com/pyspark-intro.

[29] Greg Watson. *Big Data with PySpark: MapReduce Primer*. Big Data with PySpark. New York University. 2017. URL: https://nyu-cds.github.io/python-bigdata/02-mapreduce.

Parte III

Técnicas para desarrollar el pensamiento computacional

Capítulo 7

El Pensamiento Computacional y el aprendizaje de la algorítmica y de la programación de computadoras

Guillermo de Jesús Hoyos Rivera [1]

[1] Universidad Veracruzana.
ghoyos@uv.mx

Resumen. El concepto de Pensamiento Computacional es relativamente reciente que tiene que ver con el reconocimiento formal de la existencia de un conjunto de habilidades que sería altamente deseable que todo individuo adquiriese y desarrollase con al menos un cierto nivel de profundidad. Si bien es cierto que suele relacionársele principalmente con la programación de computadoras, su ámbito de aplicación va mas allá, pues puede usarse para plantear soluciones para resolver problemas de prácticamente cualquier índole, al permitir abordarlos desde una estructura analítica, lógica y robusta al mismo tiempo. Una de las posibles formas de abordar el problema la adquisición de las habilidades relacionadas con el Pensamiento Computacional es a través de la algorítmica, la cual puede considerarse como un recurso valioso del que vale la pena echar mano, pues promueve el uso de estructuras claras y bien definidas, a través de las cuales se pueden definir detalladamente los pasos a seguir para implementar alguna solución. Desafortunadamente no es sino hasta recientemente que se toma realmente consciencia de la importancia de este tipo de competencias, y aunque en el sentido estricto la algorítmica no es más que un conjunto de símbolos, y un conjunto de reglas para organizarlos, suele existir un cierto grado de dificultad por parte de quienes se inician en su aprendizaje.

El objetivo es, pues, promover tan ampliamente como sea posible la adquisición de las habilidades relacionadas con el Pensamiento Computacional, y para esto echar mano de los recursos de la algorítmica, y eventualmente la programación de computadoras. Como se pretende ilustrar en las próximas páginas, el punto principal radica en promover el razonamiento estructurado, y para lograrlo, hacer uso de los recursos apropiados, pero sobre

todo, abordándolos en un orden que motive al aprendiz, y promueva el uso de la imaginación creativa para tratar de proponer soluciones a problemas específicos.

Palabras clave. Pensamiento computacional, programación, computadora, aprendizaje, habilidad.

7.1 Introducción

Tradicionalmente, la enseñanza del Pensamiento Computacional ha sido relegada a un segundo plano, sin dársele la importancia que realmente reviste. Por otro lado, cuando ha habido quienes deciden enseñarlo, ha sido a través de un enfoque sustentado en la programación de computadoras, lo que ha dado como resultado que se vuelva una tarea ardua, pues el enfoque es demasiado tecnológico, y no se ha dado suficiente importancia al hecho de que no son lo mismo, y del carácter más amplio del primero sobre el segundo.

En tiempos recientes mucho se ha dicho y escrito con relación a este tema, y si bien es cierto que hay trabajos relevantes, como es el caso planteado por Zapotecatl López [1], obra vanguardista a todas luces en el área, y propuestas como las planteadas en [2] y [3], en el fondo la mayoría de los recursos disponibles carecen de un enfoque didáctico que facilite la labor, y que permita llevar a cabo un proceso de enseñanza-aprendizaje de los conceptos de manera sencilla, como es el caso de [4-9].

En cierta forma resulta irónico que, a pesar de que desde muy temprana edad conocemos el concepto de recetas, que no son otra cosa que algoritmos, suele costarnos trabajo asociar ambos conceptos, y aunque en general somos capaces de seguir de manera suficientemente exitosa prácticamente cualquier receta, escribir una resulta no ser una tarea del todo trivial.

Esto puede deberse principalmente al hecho de que no estamos habituados a pensar de manera estructurada, desligando nuestras explicaciones cualquier contexto específico, sine-qua-non conditio para poder abocarse en algún momento a crear programas para computadoras. En la vida cotidiana, sobra mencionarlo, dependemos del contexto y de los sobreentendidos en nuestras comunicaciones con quienes nos relacionamos. Es parte de la naturaleza humana.

De lo anterior se desprende el hecho de que solemos encontrarnos con dificultades cuando nos vemos en la necesidad de explicar un procedimiento de manera detallada, y sin ambigüedad, para que éste sea comprendido y aplicado por otra persona, sobre todo cuando no hay contacto directo con ésta, y por ende, no se puede establecer una interacción en la que sea posible preguntar por detalles cuando no se comprenda con suficiente claridad el significado de alguno de los pasos a seguir. Peor aún es cuando tenemos la necesidad de explicar cómo hacer ciertas tareas a un nivel fino de granularidad, más que simplemente explicar, grosso modo, qué hacer. La selección el nivel de detalle de las explicaciones dentro de una receta suelen tener

una flexibilidad en cuanto al contexto que se asume sobre éstas.

Finalmente, cuando el Pensamiento Computacional se aborda desde una perspectiva basada en la programación de computadoras, el problema se vuelve realmente complejo pues, aparte de las vicisitudes expuestas anteriormente, nos topamos con la necesidad de expresar esto en un lenguaje ajeno al nuestro, ciertamente con muchas limitaciones en cuanto a su expresividad, y poder comunicarnos con una computadora, la cual, cuando no entiende algo de lo que pretendemos decirle a través de un programa, se comunica con nosotros a través de mensajes que tienden a ser ininteligibles, aún por parte de hablantes nativos de la lengua en que estos mensajes son expresados. Sí, porque los lenguajes de programación, tanto a la ida como a la vuelta, suelen ser expresados en inglés.

En resumen, pretender la enseñanza del Pensamiento Computacional a través de la programación de computadoras ha sido un craso error, el cual se ha repetido una vez tras otra a lo largo de muchos años, produciendo generaciones de personas en las que se ha creado un rechazo casi sistemático a todo lo que tenga que ver con el razonamiento estructurado. Pero esto no es lo más grave, sino que asociado a este hecho, se ha perdido una oportunidad histórica de fomentar el Pensamiento Computacional como un recurso profundamente valioso.

El objetivo debería ser, entonces, partir de la comprensión de los conceptos y estructuras asociados al Pensamiento Computacional, lo que bien puede hacerse a través del aprendizaje de la algorítmica, y posteriormente ligar estos conocimientos con la noción de leguaje de programación, pero sin hacer énfasis en este último. De hecho, la selección del lenguaje de programación en que se podrían poner en práctica los conceptos debería ser el aspecto menos relevante.

Es importante mencionar que este trabajo surge como producto de la experiencia laboral enseñando estos conceptos a nivel licenciatura, razón por la cual existirán referencias a hechos escolares a lo largo del presente documento.

7.2 El Pensamiento Computacional y su relación con esta propuesta

El concepto de Pensamiento Computacional, si bien fue inicialmente planteado de manera general por Seymour Pappert, no es sino hasta 2006 en que la Dra. Jeannette Wing, profesora de la Carnegie-Mellon University, en Pittsburgh, Estados Unidos, lo definió con precisión en [10] como:

> ‘[El Pensamiento Computacional]... implica resolver problemas, diseñar sistemas y comprender el comportamiento humano, basándose en los conceptos fundamentales de la ciencia de la computación.

> El Pensamiento Computacional incluye una amplia variedad de herramientas mentales que reflejan la amplitud del campo de la computación... [además] representa una actitud y unas habilidades universales que todos los individuos, no sólo los científicos computacionales, deberían aprender y usar'

Si analizamos este concepto, podemos ver que comprender la algorítmica, y posteriormente, la programación de computadoras es, claramente, un forma de Pensamiento Computacional, y esta forma de pensamiento tiene que ser adquirida y crecer independientemente de la noción de lenguaje de programación, o incluso, de computadora. En otras palabras, necesitamos que el estudiante adquiera las habilidades para expresar computacionalmente las recetas que resuelvan los problemas a los que debe dar una solución, sin que necesariamente tenga asociado un lenguaje de programación específico.

Evidentemente, si en un momento dado se pueden poner en práctica los conceptos relacionados con el Pensamiento Computacional a través de un lenguaje de programación, y en consecuencia, en un programa de computadora, se puede llegar a un punto en el cual se puede comprobar, en términos prácticos, la orquestación de los conceptos aprendidos, pudiendo hacer tangibles los resultados.

Sin embargo, es importante hacer nuevamente hincapié en el hecho de que el enfoque no debe ser basándose en un lenguaje de programación, sino en los conceptos generales. Esto no obsta para poder afirmar que, si se logra un nivel suficiente de dominio de la algorítmica, su puesta en operación en cualquier lenguaje de programación será una labor relativamente simple.

Como lo menciona Miguel Zapata-Ros [11], ésta es la nueva formación que, a la par de las habilidades de lecto-escritura, y las habilidades matemáticas, conformará el nuevo conjunto de competencias deseables para el mundo laboral del futuro a corto, mediano y largo plazo.

7.3 Algorítmica: la palabra clave

El estudio de este tema parte de la comprensión de dos conceptos fundamentales: la naturaleza de las computadoras, y otros dispositivos electrónicos, y los alcances de la noción de algoritmo. Sobre el primer punto, lo que se pretende es aclarar el hecho de que la computadora es, simple y sencillamente, una herramienta mas. Eso sí, no es una herramienta como cualquier otra, sino que se trata de una herramienta universal. No es como un atornillador, o una llave de tuercas, que sirven para un fin específico muy claro, y a las que probablemente se les pueden asociar algunos otros usos adicionales, que no tienen que ver necesariamente con el fin para el cual fue diseñada, los cuales están en función de sus características propias.

La computadora, por el contrario, es una herramienta capaz de ejecutar una gran gama de tareas diferentes, en función del programa que se ponga a ejecutar dentro de ésta. Eso sí, las computadoras, a menos que tenga integrados sensores y actuadores, convirtiéndolas así en una especie de robots, actúan sobre el mundo lógico, el mundo de los datos.

De esta manera, se ilustra el hecho de que un teléfono celular, de los injustamente llamados "inteligentes", es una de las posibles formas que puede tomar una computadora, la cual puede operar como calculadora, agenda, cliente de correo electrónico, consola de videojuegos, navegador Web, y por supuesto, como teléfono, dependiendo del programa que se encuentre en ejecución en un momento dado.

Una vez comprendidos estos conceptos, lo que procede es trabajar en la introducción de los principales componentes estructurales de los algoritmos, que a su vez forman parte de prácticamente cualquier lenguaje de programación imperativo. El hecho importante es que en esta etapa se trabaja únicamente y exclusivamente en papel, elaborando ejercicios que ilustren cada uno de los conceptos asociados con éstos. Partiendo de un curso de un semestre a nivel de licenciatura no especializada en Ciencias de la Computación, esta parte tomaría, grosso modo, la mitad del tiempo del mismo.

Para efecto de llevar a cabo la práctica relativa a la creación de algoritmos, es cierto que existen diferentes recursos, tales como los diagramas de flujo, los diagramas de caja, y el español estructurado, variante castellanizada del inglés estructurado, también conocido como pseudocódigo. Mucha ha sido la reflexión realizada sobre el mejor medio para introducir a quien busca aprender a programar computadoras, y la selección hecha fue el español estructurado. La principal razón para esto es que resulta práctico y, en caso de que se opte por aprender algún lenguaje de programación posteriormente, es directo el paso de una representación en éste, a su equivalente en lenguaje de programación.

Reiterando un poco lo que ya se ha afirmado anteriormente, la idea subyacente a esta iniciativa consiste en la construcción de conceptos cada vez más complejos, sustentándose en el andamiaje de conceptos de naturaleza más simple. Con la finalidad de dar claridad a la secuencia misma a través de la cual se aborda la enseñanza de la algorítmica, en lo subsecuente de la presente sección se comentará cada uno de los temas bajo la forma de una sub-sección.

7.3.1 La secuencia de instrucciones

El primer concepto a ilustrar es el de secuencia de instrucciones, siguiendo el principio de que una secuencia simple debe ser seguida (ejecutada) desde la primera hasta la última, en el orden de aparición de éstas. Tal vez suene banal tal afirmación, pero

el seguimiento con rigor de este principio a veces puede ser difícil de aplicar, sobre todo cuando no se ha fomentado previamente el razonamiento estructurado en el estudiante. Así, el primer encuentro con la algorítmica consiste en la realización de varios algoritmos simples que sólo consisten de secuencias de instrucciones, hasta que se logre reafirmar esta noción.

7.3.2 Las variables, la asignación, y los operadores aritméticos

Enseguida, lo que procede es explicar el concepto del recurso fundamental para trabajar con datos, que es la variable, haciendo hincapié, por supuesto, en el hecho de que no se trata de una variable como las que se suelen usarse en matemáticas, sino que tiene una acepción y operación diferentes.

Aunado a esto, se plantea el concepto de asignación de valores a las variables, haciendo énfasis en el hecho de que la asignación es una operación destructiva, pues elimina, o destruye, cualquier valor almacenado previamente en la variable, sin que quede rastro del mismo. Dicho de otra manera, una variable no tiene memoria histórica, esto es, no es capaz de "recordar" los valores que en el pasado ha almacenado, y únicamente conoce el valor almacenado en el momento actual, siendo éste uno de los elementos que constituyen el concepto de estado de ejecución de un algoritmo.

Sobra mencionar que también se trabaja sobre el hecho de que toda variable debe ser inicializada, pues de lo contrario no se tiene certeza del valor que tendrá almacenado al momento de su primera utilización. Inmediatamente después se procede a introducir dos conceptos adicionales, que no tienen que ver directamente con los algoritmos en sí, y que corresponden a la existencia de los operadores aritméticos, a saber, adición ($+$), sustracción ($-$), producto (*), cociente (/) y residuo (%), indicando claramente que son precisamente estos símbolos los que se deben utilizar, y que el producto no puede obviarse, como a veces se hace en matemáticas (en ocasiones se opta por escribir ab para denotar el producto de a por b). También se reitera sobre la noción de precedencia en la ejecución de los diferentes operadores, y también son presentados los paréntesis en tanto que símbolos de agrupación, que permiten crear expresiones en las cuales la precedencia de ejecución de los operadores puede ser modificada.

Aquí es donde se plantea la importancia de los símbolos, y su significado asociado, así como la rigidez que suele estar asociada con la programación de computadoras, y que es un recurso para evitar la ambigüedad en la expresión de los algoritmos.

Una vez comprendidos estos conceptos, se aprovecha para presentar y explicar las operaciones de entrada y salida de datos, como un recurso alternativo a través del cual los programas de computadora pueden comunicarse con el usuario, haciendo hincapié en el hecho de que no necesariamente los roles de creador y de usuario de un algoritmo recaerán en la misma persona.

7.3.3 La prueba de escritorio

Posterior a la explicación y puesta en práctica de las nociones anteriormente citadas, así como al planteamiento de ejemplos de simpleza extrema, utilizados para ilustrar los conceptos básicos de variable, asignación, operador aritmético, y operaciones de entrada y salida, la siguiente pregunta que se debe formular es aquella relacionada con el correcto funcionamiento del algoritmo, en función de lo planteado como problema a resolver. Para abordar este problema se presenta la noción de prueba de escritorio, como un recurso a través del cual se puede hacer el análisis de ejecución de algoritmos pequeños y de complejidad limitada. Éste es un recurso principalmente didáctico, a través del cual se busca que el estudiante sea capaz de incorporar el concepto de ejecución de un algoritmo, identificando a cada momento de ésta el estado del mismo, el cual está dado por el apuntador que indica la próxima instrucción a ser ejecutada en el algoritmo, y los valores en ese mismo momento de todas y cada una de las variables del mismo.

La prueba de escritorio tiene la forma de una tabla, en la cual las columnas representan las variables, y los renglones representan las acciones que se van ejecutando a lo largo del tiempo, el cual corre hacia abajo. Adicionalmente hay tres columnas que no representan variables, sino que denotan la presencia de las vías a través de las cuales llegan las entradas, y hacia donde son enviadas las salidas, y una columna que indica el apuntador, o número de instrucción, que se encuentra en ejecución en un momento dado, para efecto de que se pueda establece una relación entre el algoritmo y la prueba de escritorio misma. El aspecto general es el que se presenta en la Tabla 7.1.

Tabla 7.1: Forma general de la prueba de escritorio (la primera columna se incluye para señalar la dirección del tiempo).

t	Ap	Entrada	var_1	var_2	var_3	Salida
⇓						

Con el uso de esta herramienta, se plantea ilustrar los conceptos de atomicidad de la ejecución de las instrucciones, y cómo éstas son ejecutadas a lo largo del tiempo, buscando hacer hincapié en la comprensión de los efectos de cada una de las operaciones en el estado del algoritmo, en particular, lo concerniente al reemplazo de los valores de las variables a través de los procesos de asignación o lectura de valores. Para efecto de evitar confusiones, se parte de la premisa de que, para cada operación realizada, se debe ocupar un sólo renglón de la prueba de escritorio. De esta manera, para el algoritmo de la Figura 7.1, la prueba de escritorio correspondiente es la que se presenta en la Tabla 7.2.

```
1 ALG {
2         ESCRIBE "Introduzca la base del triángulo"
3         LEE base
4         ESCRIBE "Introduzca la altura del triángulo"
5         LEE altura
6         area = base * altura / 2
7         ESCRIBE "El área del triángulo es "
8         ESCRIBE area
9 }
```

Figura 7.1: Algoritmo para el cálculo del área de un triángulo.

Tabla 7.2: Prueba de escritorio para el algoritmo de cálculo del área de un triángulo.

Ap	Entrada	base	altura	área	Salida
2					Introduzca la base del triángulo
3	12	12			
4					Introduzca la altura del triángulo
5	3		3		
6				18	
7					El área del triángulo es
8					18

Si ponemos atención, podremos observar algunos hechos relevantes. El primero de ellos es que la sintaxis de español estructurado utilizada se aproxima a aquella de un lenguaje de programación. El segundo es que, en la columna correspondiente al apuntador a la instrucción en ejecución, se omite presentar las líneas de los símbolos de agrupación de la secuencia de instrucciones, esto es, las correspondientes a los caracteres '{' y '}', puesto que, en sí, no son instrucciones. Un tercero es que, si ponemos atención al renglón correspondiente a la lectura, se repite el valor 12. La razón de ser de este hecho es que se busca hacer evidente el paso del valor que se colocará en la variable asociada a la lectura, a través de una vía de entrada.

Algo en que se hace énfasis constantemente, es que la prueba de escritorio sólo sirve para verificar si un algoritmo es correcto, con base en lo especificado en el enunciado que describe al problema que se pretende resolver. Dicho en otras palabras, se pretende determinar si el algoritmo resuelve, efectivamente, el problema para el que fue creado. Un aspecto diferente es el correspondiente a la optimalidad del mismo, que tiene que ver con la cantidad de procesamiento y de espacio de almacenamiento necesario para lograr la solución de un problema, planteando la máxima siguiente:

> "En un algoritmo se pueden usar tantas variables y tantas instrucciones como sea necesario, pero se debe usar tan pocas como sea posible".

Con base en estos conceptos, y con la finalidad de reafirmar los conocimientos, se establece una dinámica en la cual los estudiantes presentan diversas propuestas de solución a problemas selectos sencillos, haciendo sucesivas comparaciones de éstas, y discutiendo acerca de las ventajas y desventajas de cada una de ellas, tanto a si son soluciones correctas, como si son soluciones que tienden a la optimalidad, dejando claro que ésta no siempre puede ser alcanzada, y que en muchas ocasiones bastará con elaborar soluciones plausiblemente eficientes. Evidentemente, varias de esas propuestas serán similares en cuanto a la forma en que se hace uso de los recursos de cómputo, dejando a opción del creador del algoritmo seleccionar aquella que más adecuada le parezca, o aquella que le resulte más fácil de analizar y comprender.

7.3.4 Las estructuras de control

El siguiente paso dentro del proceso de adquisición de conocimientos cada vez más complejos, consiste en introducir los conceptos de estructuras de control, las cuales se dividen en dos grandes secciones: las sentencias condicionales, y las sentencias de repetición.

Sentencias condicionales

En lo concerniente a las sentencias condicionales, éstas son presentadas de manera incremental en cuanto a su complejidad, partiendo de las condiciones sin complemento (`SI`), enseguida las que tienen un complemento (`SI-DOM`), las condiciones encadenadas (`SI-(DOM-SI)-DOM`), y las condiciones de tipo caso (`ELIGE-CASO`), como formas complementarias para la formulación de condiciones.

Sobra mencionar que cada una de estas alterativas va siendo presentada junto con ejemplos de uso correspondientes, asociándolos con situaciones de la vida cotidiana, y haciendo hincapié en las ventajas y desventajas que tienen cada una de ellas en diferentes circunstancias. De hecho, la forma de abordar el tema es presentando ejemplos en los que la estructura anterior resulta insuficiente, inadecuada, o ineficiente para efectos de expresar una condición, y proponiendo como alternativa de solución el uso de una estructura más compleja, pero que tiene como resultado que el algoritmo sea más eficiente en términos de la cantidad de operaciones que debe ejecutar. Así, por ejemplo, se confrontan y comparan soluciones con complemento de condición, usando tanto la sentencia `SI`, como la `SI-DOM`, y se analizan las ventajas de ésta última con relación a la primera para ciertos supuestos. Sobra mencionar que se trabajan aquí conceptos más elaborados, como lo son en encadenamiento y el anidamiento de condiciones, como recursos para lograr hacer evaluaciones más elaboradas.

Es en este punto donde se hace un nuevo esfuerzo por mejorar el proceso de enseñanza-aprendizaje de la algorítmica, aprovechando el hecho de que es necesario establecer condiciones, y que esto se hace a través de la comparación de valores. Es así que se introducen los operadores relacionales, a saber, mayor que (`>`), mayor o igual que (`>=`), menor que (`<`), menor o igual que (`<=`), igual (`==`) y diferente (`!=`), así como los operadores lógicos, a saber, Y lógico (`&&`), O lógico (`||`) y negación (`!`).

La razón de ser de esta decisión es que, nuevamente, se aproveche la necesidad de uso de tales operadores, para efecto de justificar su existencia, y a partir de ahí ilustrar a través de ejemplos sucesivos las situaciones en las que pueden ser utilizados.

Aunado a la presentación de las condicionales, se introduce el concepto de legibilidad de los algoritmos, y posteriormente de los programas, acudiendo al uso de sangrías para denotar las secuencias de instrucciones que se han de ejecutar en caso de que una condición se cumpla, o en caso de que no. Si bien es cierto que sólo en algunos lenguajes de programación, como es el caso de Python, se hace uso formal de las sangrías como elemento sintáctico indispensable para definir el código contenido dentro de determinadas instrucciones, se hace la observación insistente en el hecho de que un algoritmo debidamente sangrado será más fácil, o al menos, menos difícil, de entender, que un algoritmo escrito sin hacer uso de este recurso,

y que esta facilidad estará dada también en función de la extensión del algoritmo. En otras palabras, escribir con orden y elegancia, facilita posteriormente el proceso de lectura. Como un recurso visual para comprender este concepto, se propone el uso de líneas verticales para unir instrucciones que se encuentran dentro del mismo margen, y que por ende, son contenidas dentro de otras. Esta noción es ilustrada en la Figura 7.2.

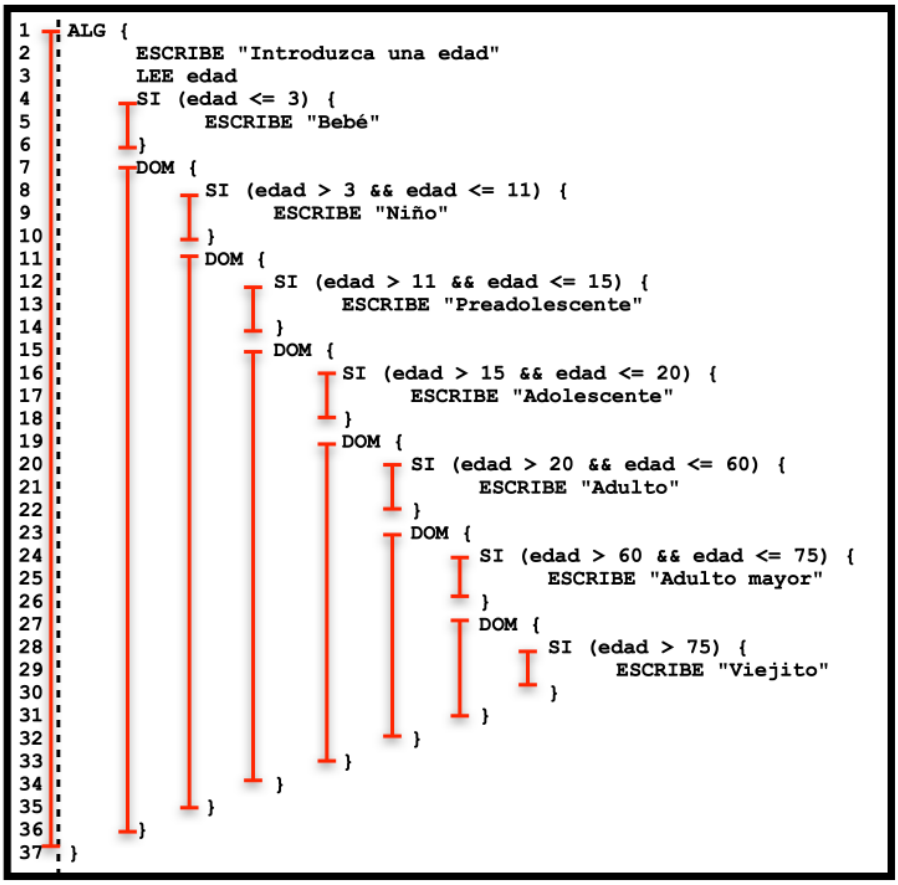


Figura 7.2: Identificación de los niveles de sangría a través del uso de líneas verticales.

En lo referente a las pruebas de escritorio, se procede a identificar claramente el momento de la ejecución de una instrucción condicional mediante una linea en blanco que abarca todo el ancho de la tabla, presentando el resultado de verdad inmediatamente a la pregunta formulada, para efecto de justificar la acción que corresponderá que sea ejecutada inmediatamente después. Esto se ilustra en la Tabla 7.3.

Sobra mencionar que, como se podrá apreciar si se presta atención a los deta-

Tabla 7.3: Prueba de escritorio del algoritmo de la Figura 7.2.

Ap	Entrada	edad	Salida
2			Introduzca una edad
3	2	2	
4	¿Es 2 menor o igual que 3? $\rightarrow V$		
5			Bebé

lles, la propuesta de algoritmo de la Figura 7.2 es una versión incompleta, de ésas que se analizan con la finalidad de ilustrar los diversos casos posibles de solución, hasta llegar a una que sea suficientemente satisfactoria, pues clasifica en la primera categoría cuando se introducen valores negativos, lo cual no aplica para el caso de las edades de personas. El uso de este tipo de recursos didácticos es de gran utilidad para efecto de la correcta transmisión de las ideas.

Sentencias de repetición

La segunda parte relacionada con las estructuras de control, corresponde a la especificación de repeticiones, como un recurso que justifica su existencia en el hecho de que frecuentemente nos vemos en la necesidad, a la hora de elaborar algoritmos, de especificar acciones que se han de ejecutar repetidamente, y que el término de tales repeticiones está definido por el cumplimiento, o falta de cumplimiento, de una condición determinada. Evidentemente la mejor forma de mostrar esa eficiencia es a través de la repetición de secuencias de instrucciones mediante el uso de las sentencias `MIENTRAS`, `HACER-MIENTRAS` y `PARA`, nombres definidos parafraseando los ciclos existentes en la mayoría de los lenguajes de programación imperativos estructurados.

En ésta se sigue la misma metodología ya explicada anteriormente, y sólo se acude a algunas ilustraciones que permiten comprender, de manera visual, cómo operan los ciclos, mismas que son presentadas en la Figura 7.3.

Evidentemente, en el texto se acompaña a estas imágenes con las explicaciones correspondientes asociadas a cada una de las transiciones indicadas con las flechas numeradas, y se llevan a cabo una buena cantidad de ejercicios interactivos, a través de los cuales se hace la comparación de diversas propuestas de solución, confrontando e ilustrando las características inherentes a cada una de las soluciones, buscando identificar aquellas que caracterizan a las mejores de éstas.

Finalmente, como una forma de dar una noción de la forma en que se pueden establecer equivalencias operacionales entre las sentencias de repetición `MIENTRAS` y `PARA`, se acude nuevamente a un recurso visual, mismo que se presenta en la Fi-

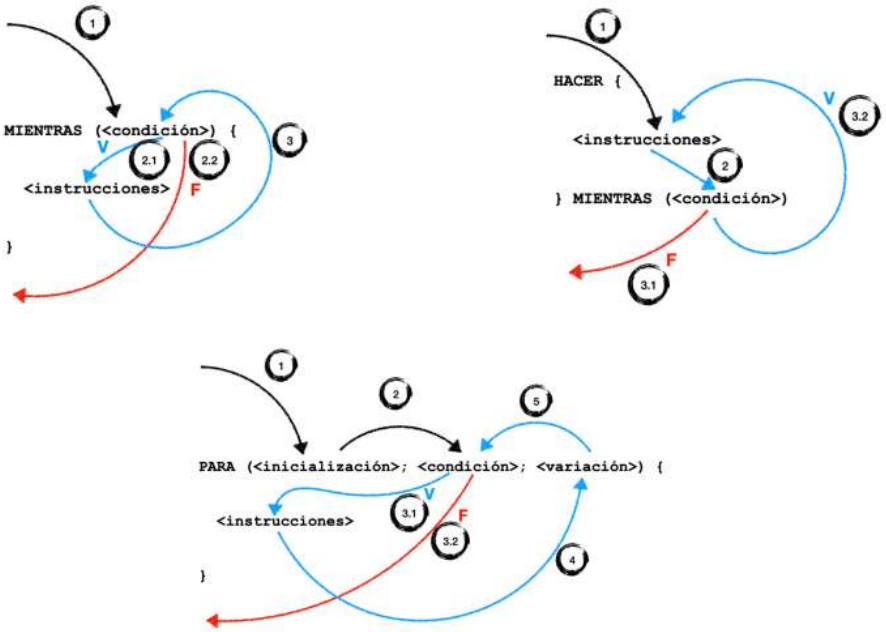


Figura 7.3: Ilustración de la operación de las sentencias de repetición.

gura 7.4.

7.3.5 Conceptos adicionales

Como último componente relacionado con la parte de la algorítmica, se presentan tres nociones que también son fundamentales. Primero, la noción de colecciones de datos, como un medio para sustentar la noción de arreglos y matrices, como un recurso para almacenar datos en estructuras de naturaleza específica y bien definida. La segunda es la noción de sub-rutinas, como un recurso para facilitar la escritura de los algoritmos, a través del aislamiento de ciertas secuencias de operaciones, las cuales son identificadas a través de un nombre, por medio del cual se puede llamar a ejecución a dichas secuencias de instrucciones. La tercera, es la noción de almacenamiento externo, como un recurso útil para almacenar datos en medios de almacenamiento no volátiles, como discos duros o memorias USB.

Si bien es cierto que la mención de estos elementos aquí se hace de manera somera, en el texto que da nacimiento a este planteamiento de postura se abunda en detalles que permiten comprender los conceptos asociados, ilustrando nuevamente su existencia y uso a través de analogías de la vida cotidiana. En todo caso, y así lo he planteado ante mis colegas de la facultad, debería haber formaciones subsecuentes a

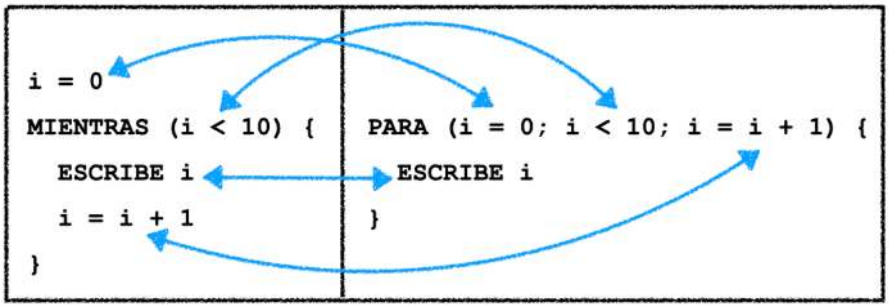


Figura 7.4: Ilustración de la equivalencia entre las sentencias de repetición `MIENTRAS` y `PARA`.

la presentada en este documento, en las que se aborden conceptos como paradigmas alternativos de programación, como lo es la Programación Orientada a Objetos y la Programación Funcional, y otra más concerniente a las Estructuras de Datos, elemento vital para realmente estar en capacidad de crear programas de computadora sofisticados, pues las Ciencias de la Computación se están evidenciando como un recurso de amplia utilidad para los profesionistas de las Matemáticas.

Sin embargo, un paso a la vez. Como bien dijo el profesor John Keating, interpretado magistralmente por Robin Williams, en el film La Sociedad de los Poetas Muertos: "Hay que chuparle el tuétano a la vida, pero no atragantarse con el hueso". El Pensamiento Computacional y la algorítmica no son algo que se pueda aprender de la noche a la mañana, y requieren de un proceso de maduración.

7.4 Puesta en práctica del Pensamiento Computacional usando un lenguaje de programación

La afirmación reiterada que se ha venido haciendo a lo largo de las reflexiones vertidas en el presente documento es que lo realmente importante de aprender para fomentar el Pensamiento Computacional es, esencialmente, la algorítmica, dejando de lado los lenguajes de programación. Sin embargo, esto no quiere decir que los lenguajes de programación no sean importantes, sino simplemente que el enfoque principal no debe estar orientado al aprendizaje de éstos, sino de los rudimentos mentales que permitirán, en algún momento, abordar el aprendizaje de prácticamente cualquier lenguaje de programación.

Por otro lado, hay que ponen en realce la importancia de poder obtener alguna retroalimentación en cuanto a los aprendizajes logrados, y cuando lo que se está estudiando es la algorítmica, el medio natural a través del cual poner en práctica,

y lograr hacer tangibles a través de una pantalla de computadora los resultados, es precisamente a través del uso de algún lenguaje de programación.

En este orden de ideas, la selección de un lenguaje de programación específico se vuelve una simple cuestión de preferencia, siempre y cuando la selección se halle dentro del espectro de los lenguajes de programación imperativos, procedimentales y estructurados, que es el ámbito que cubre el tipo de algorítmica planteada en este documento.

Opciones hay muchas, que se encuentran en una gama muy amplia, y que van desde las alternativas didácticas de naturaleza gráfica, como lo son Scratch [12], Alice [13], Logo [14] o Karel [15], pasando por alternativas didácticas como Racket [16], hasta lenguajes de programación de alto nivel en el sentido estricto de la expresión, como lo son el tan de moda Python [17], el ampliamente utilizado Java [18], o el nunca bien ponderado y casi omnipresente lenguaje de programación C [7] junto con todas sus variantes, que no serán mencionadas aquí. ¿Cuál seleccionar? Depende del público objetivo, su nivel de formación, sus necesidades mediatas e inmediatas, el contexto de aprendizaje en el que se encuentra, entre otras posibles situaciones.

Una característica común a todas estas alternativas es que, salvo algunas implementaciones de Logo y de Karel, es que todas ellas están en lengua inglesa, salvo algunas implementaciones de Logo o Karel, y hay un lenguaje de programación en español de reciente aparición: Latino [19]. Esto en algunas ocasiones puede volverse una limitante cuando quien está aprendiendo el Pensamiento Computacional a través de la aplicación de los conceptos de la algorítmica a través de un lenguaje de programación de computadoras no es angloparlante, pues el intercambio en una lengua que resulta ajena, puede volverse una fuente de frustración, y que dificulta, en consecuencia, el proceso de enseñanza-aprendizaje. En el caso particular del planteamiento hecho en el libro Algorítmica Computacional, de próxima aparición, el enfoque se hace sobre el lenguaje de programación C.

La pregunta que naturalmente saltará a la mente en este momento seguramente es, ¿y por qué tal selección, si se trata de una alternativa árida y que no facilita el aprendizaje? Ésta es una buena cuestión, la cual ha llevado muchas horas de reflexión en cuanto a lo que resultaría más conveniente para realmente aprender a crear programas para computadoras.

En todo caso, la justificación para esta selección puede o no ser compartida por el lector, sin embargo el principal motivo tiene que ver con la capacidad de formar las estructuras mentales formales que permitan al estudiante, poner en práctica los conocimiento estructurados aprendidos anteriormente, principalmente por el contexto dentro del cual se gestó este conjunto de reflexiones, que es el de una licenciatura en Matemáticas, dentro de la cual hay un curso obligatorio y dos opcionales de programación.

Un cuestionamiento que se me ha planteado frecuentemente mis colegas de la

facultad es, ¿por qué no enseñar mejor a programar en R o en Python? La justificación es simple, desde mi punto de vista personal. En lo concerniente a R, no es un lenguaje de programación en el sentido estricto, sino un paquete con muy interesantes recursos matemáticos y estadísticos, con la peculiaridad de que incorpora algunos recursos para crear alguna suerte de programas. Por otro lado, si hablamos de Python, efectivamente es un lenguaje de programación muy versátil, y sobre todo, ampliamente utilizado en la actualidad, pero tiene un solo defecto: hay tal cantidad de bibliotecas disponibles en la Internet que se pueden incorporar a un programa en Python, que el estudiante hábil es capaz de localizarlas con relativa facilidad, y simplemente utilizarlas, perdiéndose así la necesidad de aprender realmente a crear programas para computadoras.

He hecho intentos de usar Python como uno de los lenguajes de programación candidatos en el pasado, y me ha resultado difícil evitar que los estudiantes resuelvan los ejercicios sin el uso de las citadas bibliotecas, razón por la que he decidido descartarlo, pues aunque la instrucción sea clara en el sentido de no utilizarlas, lo hacen, lo que da como resultado una cierta cantidad de tiempo perdido, que se vuelve crucial para el avance del semestre, que resulta ser muy corto.

Mi defensa en cuanto a la selección del lenguaje de programación radica en el argumento de que, primero que nada, lo que es realmente fundamental es aprender a aplicar en papel los conceptos relacionados con la algorítmica, y enseguida, ponerlos en práctica, viéndose en la necesidad de hacer algún esfuerzo para efecto de realmente crear programas de computadoras. El estudiante que logre hacer esto, como se ha mencionado anteriormente, será capaz de programar en prácticamente cualquier lenguaje de programación imperativo, pues bastará con aprender la sintaxis del nuevo lenguaje de programación para poder, en poco tiempo, comenzar a crear programas de computadora.

Las explicaciones y ejemplos del libro están basados en el uso del compilador GCC y, por ende, en un ambiente de trabajo basado en alguna distribución de Linux (Ubuntu), como ambiente de trabajo a través del cual se ilustran los ejemplos. Es por esto que el libro contiene elementos que presentan una introducción al shell de Unix, y sus principales comandos. También se dan alternativas de editor de textos, tales como Gedit o Atom, un ambiente integrado de desarrollo, como lo es Code::Blocks, y un intérprete de C, tal como lo son Clling e IGCC, y que en un momento dado pueden utilizarse para hacer algunas pruebas de concepto del uso de algunos recursos del lenguaje de programación C. Cabe mencionar que todos los recursos de software seleccionados son de acceso libre, pues así se evita que el estudiante tenga que hacer erogaciones para adquirir licencias, o que incurra en la práctica indeseable de la piratería.

Para el anecdotario, hace ya varios años uno de mis antiguos estudiantes regresó de hacer su doctorado en una universidad de los Estados Unidos, y al preguntarle

sobre su estancia doctoral, me comentó que si había algo que le había causado dificultad era que allá todo el ambiente de trabajo estaba basado en Unix, mientras que yo a los alumnos de su generación, y generaciones contemporáneas, les había dado yo clases en ambiente Windows. Esto me llevó a hacer una profunda reflexión, y rechazar en lo subsecuente cualquier tipo de plataforma que no fuese gratuita, y privilegiando, por supuesto, algún ambiente basado en Unix pues, al ser futuros egresados de una carrera con orientación fundamentalmente científica, seguramente les sería de más utilidad.

Regresando al tema de la enseñanza del lenguaje de programación, acto seguido lo que corresponde es explicar los principales recursos de este lenguaje de programación, tales como su estructura, la existencia de los archivos de encabezado, los tipos de datos y sus características, la declaración de variables, y por supuesto, estableciendo relación directa con los componentes explicados en la parte de algorítmica, como las condicionales, las estructuras de repetición, las colecciones de datos, etc.

Si bien es cierto que no se trata de un compendio exhaustivo del lenguaje de programación C, para eso están libros como [7], sí se analizan y estudian sus principales componentes, y la manera en que estos se implementan dentro de un programa.

Un recurso adicional del texto, es una sección consagrada a presentar un compendio de los principales mensajes de error del lenguaje de programación C, o al menos los más comunes, pues la experiencia dicta que la interpretación de éstos no resulta del todo banal para el principiante, y es una frecuente fuente de frustración, al no comprender lo que el compilador pretende comunicarnos cuando hemos cometido un error a la hora de escribir un programa.

7.5 Conclusión

Parafraseando a Antón Ego, el personaje antagonista de la película de Pixar Animation Studios® Ratatouille: "No cualquiera puede llegar a ser un gran cocinero, pero un gran cocinero puede provenir del origen más humilde".

Esto mismo aplica para la programación de computadoras. Son falsos profetas aquellos que afirman que se puede aprender a programar computadoras en una semana, o en seis meses, o incluso en un año. Es necesario caer muchas veces, y cometer errores, antes de poder considerarse a sí mismo un programador con habilidades suficientes, pero con suficiente empeño y motivación se puede lograr.

Como suelo decir a mis estudiantes, la única forma de aprender a montar en bicicleta, es justamente montándose en una bicicleta, y comenzando a pedalear. Seguramente perderemos el equilibrio y caeremos más de una vez antes de lograr comprender los misterios del equilibrio en dos ruedas basado en el efecto giroscópico de éstas, y el uso del contramanillar como recurso para mantener el equilibrio y la trayectoria correcta. Si bien puede haber alguien que le explique a uno la teo-

ría, es sólo eso, teoría, y nunca podrá ser reemplazada por la práctica persistente y tenaz. En todo caso, tanto la programación de computadoras, como el montar en bicicleta, una vez que se aprende, ya no se olvida, y vale mucho la pena el tiempo y esfuerzo invertidos.

Sí, es cierto que el planteamiento aquí hecho defiende la postura de primer proceder a comprender la algorítmica como base fundamental, tanto del Pensamiento Computacional, como de la programación de computadoras, y he sido cuestionado sobre este hecho, contraponiendo la afirmación aquí hecha sobre el ciclismo. Pero es que aprender la algorítmica también es un aprendizaje que sólo puede ser adquirido a través de la práctica tenaz y persistente, en la cual también es necesario caer muchas veces antes de dominarla. En fin, que el libro que contiene la propuesta sobre la que se escribe este artículo no pretende ser mas que una propuesta más, dentro de las muchas que ya existen, y solo busca aportar un grano de arena más a la reflexión relacionada con esta gran empresa que es la formación en Pensamiento Computacional. Evidentemente, un texto no es suficiente si no se cuenta con el acompañamiento y asesoría adecuados, y la práctica constante, comparando las ideas propias con las de otros.

Finalmente, en lo concerniente a los beneficios del uso de la metodología aquí planteada, puedo afirmar que los resultados han sido alentadores. Si bien es cierto que no tengo análisis numéricos que me permitan demostrarlo, lo que sí he logrado ver es que mis estudiantes logran proseguir con éxito estudios que están relacionados con la programación de computadoras, o que hacen uso de estos recursos para la solución de problemas de sus respectivos ámbitos de trabajo, como lo son los posgrados en Inteligencia Artificial, lo que me hace pensar que el camino no ha sido errado.

Referencias

[1] Jorge Luis Zapotecatl López. *Introducción al Pensamiento Computacional: Conceptos Básicos Para Todos*. Ciudad de México: Academia Mexicana de Computación, 2018. ISBN: 978-607-97357-2-2.

[2] Computer Science Education Research Group. *Informática Sin Un Ordenador*. CS Unplugged. University of Canterbury. Feb. de 2021. URL: https://www.csunplugged.org/es/.

[3] Computer Science Education Research Group. *Computer Science Field Guide*. CSFG. University of Canterbury. Feb. de 2021. URL: https://csfieldguide.org.nz/es/.

[4] Mihaela Juganaru Mathieu. *Introducción a La Programación. Primera Edición Ebook*. México: Grupo Editorial Patria, 2014. ISBN: 978-607-438-920-3.

[5] *Tutorial Para La Asignatura Introducción a La Programación*. Ciudad Univesitaria: Fondo Editorial FCA, 2003.

[6] Jordi Álvarez Canal y Josep Vilaplana Pastó. *Introducción a La Algorítmica*. Universitat Oberta de Catalunya, 2010.

[7] Brian Kernighan y Dennis Ritchie. *El Lenguaje de Programación C*. Second. Prentice Hall, 1991. ISBN: 978-968-880-205-2.

[8] Héctor Tejeda Villela. *Manual de C*. Mayo de 2018.

[9] Brian Gough. *Una Introducción a GCC – Para Los Compiladores GNU Gcc y G++*. Trad. por David Arroyo Menéndez y Luis Palomo de Onís Gutiérrez. Revisado y actualizado. Bristol: Network Theory Limited, ago. de 2011. ISBN: -9541617-9-3.

[10] Jeannette M Wing. Computational Thinking. En: *Communications of the ACM* 49.3 (2006), págs. 33-35.

[11] Zapata-Ros. *¿Cómo El Pensamiento Computacional Se Ha Convertido En Una Competencia Clave Para Los Tiempos Que Corren?* Pensamiento computacional y alfabetización digital. Dic. de 2020. URL: http://computational-think.blogspot.com/2020/12/como-el-pensamiento-computacional-se-ha.html.

[12] Lifelong Kindergarten Group. *Scratch – Imagine, Program, Share*. URL: https://scratch.mit.edu/.

[13] Carnegie Mellon University. *Alice – Tell Stories, Build Games, Learn to Program*. 2020. URL: https://www.alice.org/.

[14] Logo Foundation. *Logo Foundation*. 2021. URL: https://el.media.mit.edu/logo-foundation/.

[15] Joseph Bergin. *Karel Download Page*. Joseph Bergin. Sep. de 2017. URL: http://csis.pace.edu/%5Ctextasciitilde%20bergin/KarelJava2ed/downloads.html.

[16] The Racket Ecosystem. *Racket*. URL: https://racket-lang.org/.

[17] Python Software Foundation. *Welcome to Python.Org*. Python. 2021. URL: https://www.python.org/.

[18] Oracle. *Java*. URL: https://www.java.com/es/.

[19] Melvin Guerrero. *Lenguaje Latino*. 2021. URL: https://www.lenguajelatino.org/.

Capítulo 8

Método de enseñanza y aprendizaje del pensamiento computacional basado en el desarrollo de simulaciones por computadora para probar hipótesis

Jorge L. Zapotecatl [1]

[1] Instituto Nacional de Astrofísica, Óptica y Electrónica.
jzapotecatl@gmail.com

Resumen. Usualmente los cursos que se imparten en las aulas de clases consisten en exponer conceptos y resolver ejemplos estructurados de los tópicos relacionados con un tema donde el estudiante participa como un receptor del conocimiento. El objetivo de este trabajo es proponer un método de enseñanza y aprendizaje del pensamiento computacional que se fundamenta en la integración del método científico y de simulaciones por computadora como medio para probar hipótesis. El método presentado en este trabajo pretende desarrollar en el estudiante las competencias enfocadas en resolver problemas abiertos y actitudes de autocrítica y aprendizaje por exploración. El método engloba el diseño e implementación de simulaciones por computadora a fin de propiciar el desarrollo del pensamiento computacional en los estudiantes al ejercitar los conceptos y habilidades clave: abstracción, descomposición de problemas, algoritmos y simulación. El método propuesto en este trabajo se ejemplifica con una simulación por computadora, en el lenguaje Scratch, donde se desea obtener el tiempo de transmisión de un virus en un grupo de personas para probar una hipótesis.

Palabras clave. pensamiento computacional, simulación, método de aprendizaje, método científico, aprendizaje exploratorio, hipótesis.

8.1 Introducción

"Cuéntame y olvidaré; muéstrame y quizás recuerde; involúcrame y entenderé"

Confucio

Las computadoras cambiaron la manera en que nos comunicamos, entretenemos o hacemos labores administrativas o de oficina. Las actividades que antes demoraban días o semanas, por ejemplo, la entrega de un mensaje, actualmente se ejecuta en cuestión de segundos. Además de los beneficios que ofrecen las computadoras en las actividades de nuestra vida diaria, las computadoras sean vuelto un instrumento científico importante debido a sus capacidades de procesamiento. Dichas capacidades de procesamiento permiten estudiar determinados fenómenos por medio de simulaciones por computadora. La definición de simulación por computadora utilizada en este trabajo se presenta a continuación.

Definición 8.1.1 (simulación por computadora) *La simulación por computadora es un modelo matemático que se ejecuta en una computadora a fin de probar hipótesis y predecir el comportamiento de un fenómeno físico o del mundo real.*

Las computadoras permiten ejecutar simulaciones por computadora compuestas por miles o millones de entidades que interactúan entre sí donde resolver este tipo de problemas mediante una solución analítica puede resultar complejo. Además, hay ciertos fenómenos que no pueden ser probados en un laboratorio, que son muy raros que sucedan en la realidad o que son muy costosos de reproducir, donde es conveniente implementar simulaciones por computadora. Por ejemplo, el estudio de cómo explota una estrella.

La simulación por computadora se considera como el tercer pilar de la ciencia, junto con la teoría y la experimentación clásica [1]. En consecuencia, la simulación por computadora contribuye a resolver los retos presentes y futuros a nivel mundial. Además de las aportaciones de la simulación por computadora en la ciencia, el desarrollo y la experimentación con simulaciones por computadora puede aplicarse como una estrategia de enseñanza y aprendizaje del pensamiento computacional.

La idea esencial de este trabajo consiste en aplicar el método científico para probar hipótesis mediante el desarrollo de simulaciones por computadora. El diccionario de Oxford [2] define el método científico como:

> El método científico es una metodología para obtener nuevos conocimientos que consiste en la observación sistemática, medición, experimentación y la formulación, análisis y modificación de hipótesis.

Si el método científico y el desarrollo de simulaciones por computadora son correctamente adaptados en el aula de clases, entonces el aprendizaje se da basado en la exploración y el descubrimiento. Los estudiantes pueden aprender de un determinado tema al resolver un problema o fenómeno por medio del desarrollo y experimentación de simulaciones por computadora, en lugar de resolver algunos ejemplos y considerar la posibilidad de aplicar los conocimientos adquiridos en algún problema de la vida real. El diseño e implementación de simulaciones por computadora implica el desarrollo de habilidades orientadas a resolver problemas, como lo son: la abstracción, la descomposición de problemas y los algoritmos.

8.2 Estado del arte

En la literatura se aborda el aprendizaje por exploración únicamente por medio de la experimentación con simulaciones por computadora previamente desarrolladas para probar hipótesis. Es decir, a diferencia del presente trabajo, en los trabajos relacionados no se aborda el desarrollo de la simulaciones como el medio de aprendizaje.

En el artículo [3] se estudia la efectividad del aprendizaje por descubrimiento en las aulas de clase donde los estudiantes experimentan con simulaciones por computadora para resolver problemas. Además, se discute cómo las simulaciones pueden combinarse con guías de apoyo para poder resolver dichos problemas. La simulación por computadora se adapta al aprendizaje por descubrimiento porque la tarea principal del alumno es inferir, a través de experimentación, las características del modelo subyacente de la simulación.

En el trabajo [4] se examinaron los resultados de aplicar el aprendizaje basado en la experimentación con simulaciones por computadora en el desarrollo de un proyecto. El objetivo del proyecto es que los estudiantes desarrollen la capacidad de lidiar con problemas complejos y abiertos. El proyecto consistió en que los estudiantes tenían que realizar una investigación acerca de desechos peligrosos utilizando un software que simula procesos físicos y procesos de ingeniería. La evaluación de los resultados de los estudiantes se basó en el uso de mapas de conocimiento y en exámenes para medir el aprendizaje de los estudiantes.

En el trabajo [5] se propone un esquema triple de aprendizaje por descubrimiento integrado. El esquema incluye: a) apoyo interpretativo para ayudar a los estudiantes a adquirir comprensiones significativas e integradoras; b) apoyo experimental para apoyar a los alumnos en actividades experimentales sistemáticas; y c) apoyo reflexivo que aumenta la autoconciencia de los descubrimientos e impulsa su abstracción. Se realizaron dos experimentos con estudiantes de octavo grado para examinar los efectos del esquema de aprendizaje mediante un programa de simulación sobre flotación y hundimiento. Los resultados sugieren que si se aplica el

esquema triple en un entorno de simulación, entonces se logra un aprendizaje por descubrimiento significativo, sistemático y reflexivo.

En el trabajo [6] se investiga acerca de los efectos de aplicar la estrategia de aprendizaje basada en simulación con tres modelos: experimentos, hipótesis y orientación por pasos. Se implementó un sistema de aprendizaje por simulación basado en estos tres modelos, y se exploraron las diferencias entre el aprendizaje basado en simulación y el aprendizaje de laboratorio tradicional en el contexto de los estudios de física. En estudiantes de secundaria, el aprendizaje de las características de una lente óptica fue significativamente mejor para el aprendizaje basado en simulación que para el aprendizaje de laboratorio tradicional.

En este trabajo [7] se investigó los efectos de proporcionar guías a los estudiantes para elaborar hipótesis. Cincuenta y dos estudiantes trabajaron en una tarea de investigación sobre conceptos de movimiento en relación con la primera ley de Newton. Un grupo de estudiantes recibió una hipótesis parcial de ayuda para proponer hipótesis y otro grupo de estudiantes no la recibió. Los resultados mostraron que los estudiantes que recibieron hipótesis parciales propusieron hipótesis más complejas, se desempeñaron mejor en la recolección de datos y adquirieron más conocimientos del tema a investigar que los estudiantes que no recibieron hipótesis parciales.

8.3 Método de enseñanza y aprendizaje del pensamiento computacional

En este trabajo se propone un método de enseñanza y aprendizaje para el desarrollo de las habilidades del pensamiento computacional mediante la investigación de hipótesis aplicando el método científico y el desarrollo de simulaciones por computadora. Para dicho método se requiere que el profesor muestre al estudiante un conjunto de fenómenos o sistemas a través de vídeos, imágenes, guías, entre otros tipos de recursos. El estudiante seleccionará uno de los fenómenos o sistemas de acuerdo con sus intereses y aplicará el método de enseñanza y aprendizaje del pensamiento computacional que se presenta a continuación (ver Figura 8.1).

Método de enseñanza y aprendizaje del PC

1. El estudiante observa y se hace preguntas sobre el fenómeno o sistema de su interés.
2. El estudiante propone una hipótesis acerca del sistema seleccionado.
3. El estudiante elabora una simulación por computadora que permite experimentar y probar su hipótesis. Las etapas que el estudiante aplicará en el desarrollo de su simulación por computadora son las siguientes:
   (a) Formulación del problema: descripción del fenómeno a investigar.

(b) Formulación del modelo: abstracción de las entidades relevantes del sistema con sus atributos y funciones.
(c) Diseño e implementación: elaboración de algoritmos que son codificados y ejecutados en un lenguaje de programación.
(d) Experimentación: ejecución de la simulación y obtención de información.

4. El estudiante obtiene una conclusión mediante la extracción de conocimiento de la información obtenida con su simulación y reporta sus resultados.

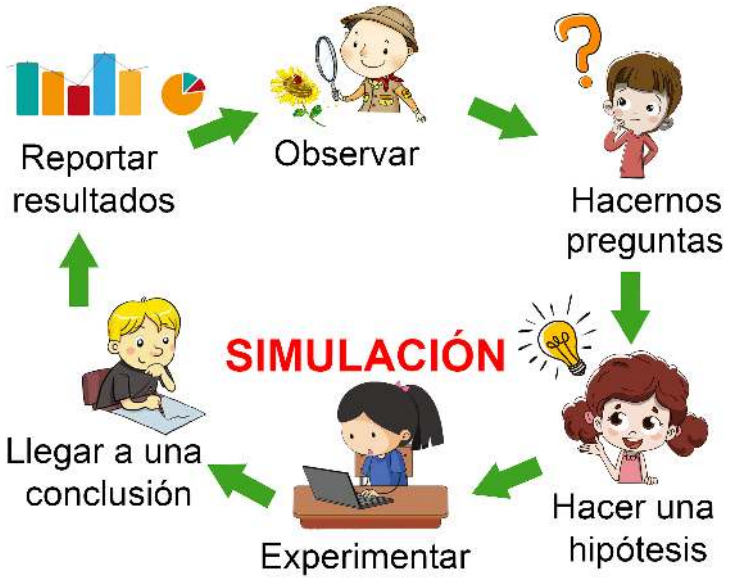


Figura 8.1: Método de enseñanza y aprendizaje basado en la prueba de hipótesis.

El objetivo del método de enseñanza es que, por un lado, la aplicación del método científico desarrolle en el estudiante un pensamiento crítico al someter sus ideas o hipótesis a su comprobación por medio de la observación y experimentación sistemática. Los estudiantes pueden fortalecer su aprendizaje a través de preguntas de su interés. Además, se promueve que los estudiantes desarrollen la capacidad de auto cuestionarse por medio de la retroalimentación de la experimentación con las simulaciones. Por otro lado, el desarrollo de simulaciones por computadora puede contribuir al desarrollo del pensamiento computacional porque la creación de mundos virtuales ejercita las técnicas y conceptos clave de abstracción, información, descomposición de problemas, algoritmos y simulación.

Los sistemas o fenómenos que el instructor indique como alternativas para simularse y analizarse deben ser cuidadosamente elaborados. El nivel de dificultad del fenómeno a simular tiene que ser acorde al nivel académico del estudiante, de la misma manera que ocurre en la enseñanza de otras disciplinas. Por ejemplo, en la enseñanza de las matemáticas para los grados de primaria se comienza con la enseñanza de la aritmética: suma, resta, multiplicación y división. Posteriormente, para los grados de bachillerato se enseña cálculo: concepto de limite, derivación e integración.

El instructor debe proporcionar guías a los estudiantes que les permita enfocarse en el problema que deben resolver. El método no se enfoca en la creación de simulaciones aplicadas en algún problema de la vida real. Por dicha razón, la sobre simplificación de los fenómenos o sistemas es conveniente en beneficio de realizar proyectos simples y didácticos que permitan al estudiante desarrollar la capacidad de resolver problemas mediante el pensamiento computacional.

8.4 Descripción del método con una simulación

En esta sección se muestra la aplicación del método de enseñanza del pensamiento computacional propuesto mediante un ejemplo. El estudiante selecciona un sistema natural o artificial de interés dentro de un conjunto de sistemas bien definidos y cuidadosamente elaborados por el instructor. En este caso suponemos que el estudiante seleccionó el fenómeno: transmisión de un virus en un grupo de personas.

El fenómeno consiste en simular cómo se propaga un virus en un conjunto de personas si se toman o no se toman las medidas de distanciamiento social. El primer escenario del fenómeno consiste en un grupo de personas donde la movilidad es total e inicialmente hay un infectado, ver Figura 8.2. El segundo escenario del fenómeno consiste en un grupo de personas que permanecen en su sitio e inicialmente hay un infectado que se desplaza por todo el escenario, ver Figura 8.3.

8.4.1 El estudiante observa el fenómeno

En esta etapa el estudiante observa y se hace preguntas sobre el fenómeno o sistema de su interés. Por ejemplo:

- ¿Qué tiempo pasará para que todas las personas del primer escenario sean contagiadas?
- ¿Qué tiempo pasará para que todas las personas del segundo escenario sean contagiadas?

Figura 8.2: Movilidad total. Todos los individuos se mueven dentro del escenario.

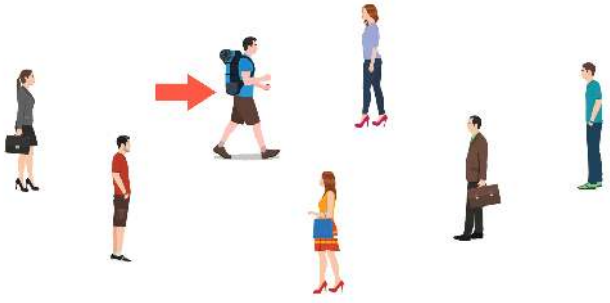


Figura 8.3: Permanecen en su sitio. Solo se mueve la persona infectada.

8.4.2 El estudiante propone una hipótesis

En esta etapa el estudiante propone una hipótesis acerca del sistema seleccionado. Después de un periodo de reflexión y análisis el estudiante redacta una hipótesis que es susceptible de poder probarse mediante una simulación por computadora. Por ejemplo:

- Hipótesis: *Permanecer aislado de las personas aumenta el tiempo en el que se propaga la transmisión del virus en todo el grupo.*

8.4.3 El estudiante elabora una simulación por computadora

En esta etapa el estudiante elabora una simulación por computadora que permite experimentar y probar su hipótesis. Las etapas que el estudiante aplica en el desarrollo de su simulación por computadora son:

1. Formulación del problema.
2. Formulación del modelo.
3. Diseño e implementación.
4. Experimentación.

En las siguientes subsecciones se describen las etapas sugeridas para el desarrollo de la simulación. Cabe señalar que actualmente se cuenta con lenguajes de programación como Scratch donde no es necesario aprender sintaxis complicadas para poder elaborar una simulación por computadora. El estudiante se concentra en la lógica del problema. Las tareas de incorporar imágenes, animaciones y sonidos en lenguajes como Scratch son sencillas.

Formulación del problema

La etapa de formulación del problema consiste en la descripción del fenómeno a investigar. La descripción de problema puede realizarse mediante texto, diagramas, definiciones matemáticas, entre otras. Por ejemplo, el estudiante se hace la siguiente pregunta:

- ¿Cómo describirías el fenómeno que deseas simular?

Una posible descripción del problema se muestra a continuación.

Descripción del problema: *La simulación consistirá en situar n individuos saludables y 1 individuo enfermo en un escenario rectangular. Los individuos se desplazarán o no en el escenario. Si un individuo saludable es tocado por una persona enferma, entonces el individuo saludable pasa al estado de enfermo. El resultado que se desea obtener es el tiempo transcurrido cuando todos los individuos están enfermos. En un primer experimento las personas se moverán y en un segundo experimento las*

personas no se moverán.

Otra manera de describir el problema se realiza mediante su definición desde una una perspectiva computacional donde se indican: entradas, salidas y relación entre entradas y salidas, ver Tabla 8.1.

Tabla 8.1: Definición del problema de la transmisión de un virus.

Componentes	Definición del problema
Entradas	Las instancias de entrada son el número de personas contagiadas $c_t = 1$ y el número de personas sanas $s_t = n$.
Salidas	Las instancias de salida cuando todos los individuos se contagiaron son el tiempo t y el número c_t de personas contagiadas.
Relación	La relación de instancias de entrada y salida se puede aproximar con una función exponencial del tipo: $c_t = c_1 \cdot r^{t-1}$, donde c_1 es el número de contagiados inicialmente y r es la razón de cambio con respecto al tiempo.

Formulación del modelo

La etapa de formulación del modelo consiste en la abstracción de las entidades relevantes del sistema con sus atributos y funciones. Además de la selección de modelos conceptuales, gráficos o matemáticos. El estudiante, por ejemplo, se haría las siguientes preguntas:

- ¿Qué entidades son esenciales para representar el proceso de transmisión del virus?
- ¿Qué atributos y funciones tienen las entidades?

Después de cierto proceso de abstracción puede concluir que las entidades que conforman el fenómeno son individuos. La Tabla 8.2 ilustra a la entidad individuo con sus respectivos atributos y funciones.

El estudiante ahora tiene una abstracción de la entidad que representa el movimiento de las personas de manera similar al modelo del movimiento de partículas de la mecánica clásica. Cada individuo tendrá los atributos: posición, rapidez, dirección y estado de salud. La representación de cada individuo puede hacerse con un círculo donde un círculo verde representa un individuo saludable y un círcu-

Tabla 8.2: Abstracción de atributos y funciones de la entidad principal del fenómeno.

Individuo	
Atributos	Funciones
Posición Rapidez Dirección Estado de salud	Caminar Contagiar Virar

lo rojo un individuo enfermo. Los individuos se desplazan dentro de un escenario rectangular (ver Figura 8.4).

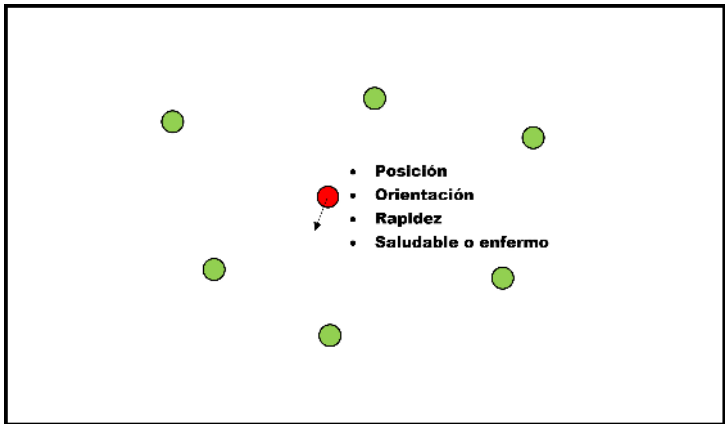


Figura 8.4: Escenario donde se desplazan los individuos, la cantidad de individuos permanece constante.

Diseño e implementación

La etapa de diseño e implementación consiste en la elaboración de algoritmos que son codificados y ejecutados en un lenguaje de programación. El estudiante ahora debe implementar el modelo, por consiguiente, tiene que desarrollar un algoritmo.

- ¿Qué pasos o conjunto de instrucciones sigue cada individuo en el proceso?
- ¿Cuales son los subalgortimos que ejecuta cada entidad?

Cada individuo en el escenario de movilidad total ejecutaría los Algoritmos 8.1, 8.2, 8.3 y 8.4. El algoritmo para el escenario de permanecer en su sitio es el mismo que el Algoritmo 8.1 sin la invocación de los algoritmos 8.2 y 8.3 (funciones caminar y virar).

Algoritmo 8.1 Algoritmo que ejecutaría cada individuo con Movilidad total.

```
1: Situar individuo en una posición aleatoria del escenario.
2: Orientar individuo en una dirección aleatoria.
3: mientras No termina simulación hacer
4:     Caminar( )
5:     Virar( )
6:     Contagiarse( )
7: fin mientras
```

Algoritmo 8.2 Caminar.

```
1: procedimiento Caminar( )
2:     Desplazar individuo con una determinada rapidez.
3: fin procedimiento
```

Algoritmo 8.3 Virar.

```
1: procedimiento Virar( )
2:     si Individuo toca un borde del escenario entonces
3:         Dirigir individuo con una orientación que lo mantenga en el escenario.
4:     fin si
5: fin procedimiento
```

Algoritmo 8.4 Contagiarse.

```
1: procedimiento CONTAGIARSE( )
2:     si Es un individuo saludable entonces
3:         si Individuo toca a un individuo enfermo entonces
4:             Individuo saludable cambia al estado enfermo.
5:         fin si
6:     fin si
7: fin procedimiento
```

La implementación de los algoritmos en el lenguaje de programación Scratch se muestra en la Figura 8.5. El lector es invitado a consultar la implementación y ejecutar el programa de transmisión del virus en el siguiente enlace: `https://scratch.mit.edu/projects/417667583/`

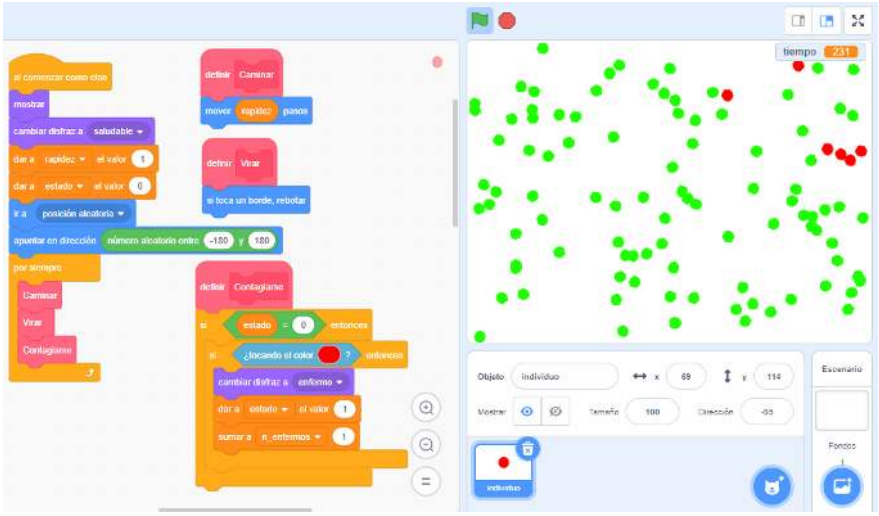


Figura 8.5: En la parte izquierda se muestran los bloques de instrucciones correspondientes al algoritmo. En la parte derecha se muestra la salida del algoritmo donde se mueven los individuos.

Experimentación

La etapa de experimentación consiste en la ejecución de la simulación y obtención de información. El estudiante experimenta con dos escenarios: donde la movilidad de los individuos es total y donde las personas, excepto el individuo contagiado, permanecen en su sitio (ver Figura 8.6).

- En el primer experimento relacionado al escenario de movilidad total el tiempo promedio transcurrido para que el virus se propague en todas las personas es de 619 segundos.
- En el segundo experimento donde los individuos permanecen en su sitio el tiempo transcurrido para que todos los individuos se contagien es de 12, 103 segundos en promedio .

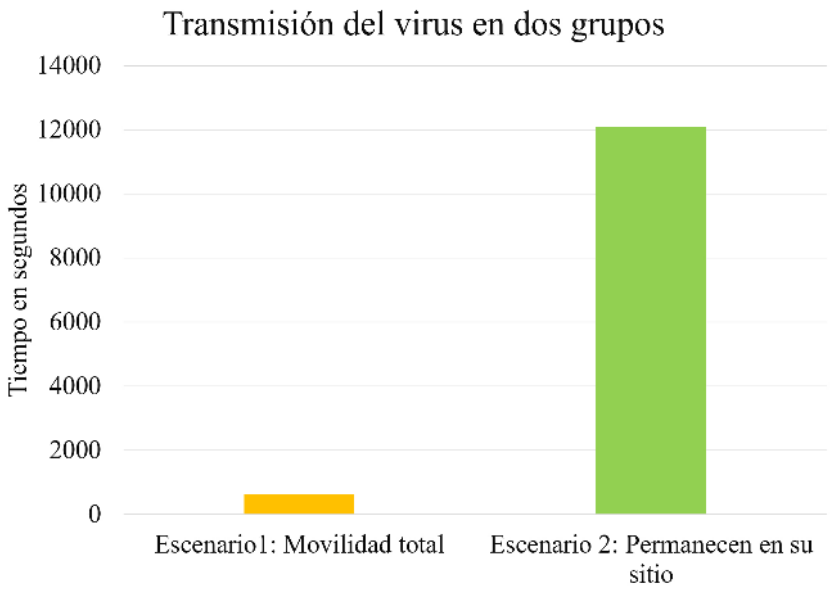


Figura 8.6: Tiempos de transmisión de un virus.

8.4.4 El estudiante obtiene una conclusión

En esta etapa el estudiante extrae conocimiento de los resultados obtenidos en la simulación. El estudiante puede constatar con la gráfica que la diferencia en el tiempo de propagación de la transmisión es notable. Cuando las personas permanecieron en su sitio el tiempo que transcurrió para que todos se contagiaran fue más del 1900% (12,103 segundos) en comparación con la movilidad total (619 segundos).

Con base en los resultados de la simulación, el estudiante determina que si las

personas permanecen aisladas, entonces la propagación del virus se transmite lentamente. En consecuencia el estudiante obtiene una conclusión. La hipótesis del estudiante se respalda con los resultados obtenidos y puede concluir los siguiente:

- *permanecer aislado de las personas aumenta el tiempo en el que se propaga la transmisión del virus en todo el grupo*.

Cabe señalar, que la simulación de la transmisión del virus presentada anteriormente es un modelo demasiado simplificado del fenómeno porque no incluye la probabilidad de contagio, distancias de separación, y muchas otras propiedades que son significativas y que lo asemejan al fenómeno real. No obstante, para propósitos académicos y de aprendizaje exploratorio, la simulación es conveniente.

8.5 Conclusión y trabajo futuro

La elaboración de hipótesis y el desarrollo de simulaciones por computadora es un método que puede propiciar la enseñanza y el aprendizaje del pensamiento computacional porque permite ejercitar: la abstracción, la descomposición de problemas, el pensamiento algorítmico y el análisis de información. Además, presentar problemas del interés para el estudiante junto con la incorporación de multimedia posibilita que el estudiante sea motivado a aprender acerca del fenómeno.

Los sistemas o fenómenos que se presenten como alternativas para simular y analizar deben ser cuidadosamente elaborados mediante modelos simples a fin de no abrumar a los estudiantes. Además, el instructor debe proporcionar guías a los estudiantes que les permita enfocarse en el problema que deben resolver. La finalidad del método es el desarrollo de las habilidades del pensamiento computacional para la resolución de problemas. En ese sentido, el método no se enfoca en la creación de simulaciones aplicadas en algún problema de la vida real. Por dicha razón, la sobre simplificación de los fenómenos o sistemas es conveniente en beneficio de realizar proyectos simples y didácticos.

Como trabajo futuro se realizará la aplicación del método de enseñanza propuesto en este trabajo en grupos de estudiantes para evaluar de manera cuantitativa y cualitativa el aprendizaje y desarrollo de las habilidades del pensamiento computacional.

Referencias

[1] M. Riedel y col. Classification of Different Approaches for E-Science Applications in next Generation Computing Infrastructures. En: *2008 IEEE Fourth International Conference on eScience*. 2008, págs. 198-205. DOI: 10.1109/eScience.2008.56.

[2] Oxford Dictionaries. *Definition of Scientific Method by Oxford Dictionary*. en. https://www.lexico.com/definition/scientific_method. 2021.

[3] Ton De Jong y Wouter R. Van Joolingen. Scientific Discovery Learning with Computer Simulations of Conceptual Domains. En: *Review of Educational Research* 68.2 (1998), págs. 179-201. DOI: 10.3102/00346543068002179. eprint: https://doi.org/10.3102/00346543068002179.

[4] G. K. W. K. Chung, T. C. Harmon y E. L. Baker. The Impact of a Simulation-Based Learning Design Project on Student Learning. En: *IEEE Transactions on Education* 44.4 (2001), págs. 390-398. DOI: 10.1109/13.965789.

[5] Jianwei Zhang y col. Triple Scheme of Learning Support Design for Scientific Discovery Learning Based on Computer Simulation: Experimental Research. En: *Journal of Computer Assisted Learning* 20.4 (2004), págs. 269-282. DOI: 10.1111/j.1365-2729.2004.00062.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2729.2004.00062.x.

[6] Kuo-En Chang Sung y col. Effects of Learning Support in Simulation-Based Physics Learning. En: *Computersand Education* 51.4 (2008), págs. 1486-1498. ISSN: 0360-1315. DOI: 10.1016/j.compedu.2008.01.007.

[7] Xiulin Kuang, Tessa H.S. Eysink y Ton de Jong. Effects of Providing Partial Hypotheses as a Support for Simulation-Based Inquiry Learning. En: *Journal of Computer Assisted Learning* 36.4 (2020), págs. 487-501. DOI: 10.1111/jcal.12415. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jcal.12415.

Capítulo 9

¿Cómo resolver problemas de Optimización Combinatoria ayuda a desarrollar el Pensamiento Computacional?

David Martínez-Galicia [1], *Judith Agueda Roldán Ahumada* [1] y *Marcela Quiroz-Castellanos* [1]

[1] Universidad Veracruzana.
DavidGalicia@outlook.es, jara2678@gmail.com, maquiroz@uv.mx

Resumen. La Optimización Combinatoria es una rama de las Matemáticas Aplicadas y de las Ciencias de la Computación, relacionada con encontrar combinaciones de elementos simples para formar soluciones complejas. La combinatoria es una tarea innata para el ser humano, y se encuentra inmersa en casi todas las actividades de la vida diaria. La enseñanza de estrategias para generar posibles combinaciones, validar las soluciones correctas y encontrar la solución óptima son los objetivos de estudio en asignaturas de Optimización Combinatoria. En los últimos años, el pensamiento computacional ha formado parte del debate sobre las asignaturas de la educación obligatoria de numerosos países. Se ha planteado que el desarrollo de habilidades de pensamiento computacional en niños y jóvenes debe fortalecer y facilitar su pensamiento creativo, sus destrezas de comunicación a través de una variedad de medios, sus competencias para resolver problemas del mundo real y sus capacidades para analizar temas cotidianos desde una perspectiva diferente. En este artículo se plantean algunos de los beneficios de estudiar Optimización Combinatoria con la finalidad de que el lector visualice la importancia de considerar esta disciplina como parte de los contenidos en un curso de pensamiento computacional, pues permite potencializar habilidades para la solución de problemas y desarrollar un pensamiento computacional más profundo, por medio de abstracciones más inteligentes o sofisticadas, y el análisis de pro-

blemas de gran escala y sus posibles soluciones.

Palabras clave. Pensamiento computacional, optimización combinatoria, modelado de problemas, resolución de problemas.

9.1 Optimización Combinatoria como potenciador del Pensamiento Computacional

El pensamiento computacional se puede definir como un conjunto de habilidades que permiten tanto formular problemas específicos, como plantear sus soluciones [1]. Entre este conjunto de habilidades se pueden reconocer cuatro piedras angulares: la capacidad de descomposición, el reconocimiento de patrones, la abstracción de subproblemas y el diseño de algoritmos. Cada una de estas habilidades ayuda a entender los requerimientos demandados por problemas complejos, y a su vez, permite evaluar las posibles vías de solución tratando de encontrar la mejor posible. Si bien, se dice que las personas que se desempeñan en el ámbito de las Ciencias Computacionales y Tecnológicas tienen más desarrollado este tipo de pensamiento por su práctica constante, esto no quiere decir que una persona ajena a este campo o con poco conocimiento sobre computadoras sea incapaz de refinar estas habilidades. El pensamiento computacional no solo se orienta a métodos de resolución de problemas, sino también se enfoca en el perfeccionamiento de técnicas de modelado, con el propósito de entender cada uno de los requerimientos y las restricciones que plantea una problemática.

El presente artículo tiene como objetivos: (1) resaltar las ventajas que brinda el estudio de la Optimización Combinatoria al desarrollo del pensamiento computacional, y (2) plantear un ejemplo de las técnicas empleadas dentro de este campo. La Optimización Combinatoria es una disciplina que busca la mejor solución dentro de un conjunto definido de soluciones para un problema [2]. Generalmente, estas soluciones se encuentran compuestas por un grupo finito de elementos que pueden ser objetos, recursos materiales e inclusive humanos [3]. La diferencia que puede indicar que una solución sea mejor que otra, puede ser una asignación de recursos, tal vez una agrupación de elementos, o un simple ordenamiento de objetos, todo esto dependerá tanto de la definición del problema como de las restricciones que tenga el mismo [4]. De esta forma, buscar la mejor configuración que satisfaga las condiciones de un problema es la tarea principal de la Optimización Combinatoria.

Aunque, actualmente esta disciplina se ha inclinado por el uso de programas computacionales, también se requiere de nociones matemáticas para poder comprender los problemas y las soluciones propuestas. Si bien, esta disciplina podría parecer difícil, la formación acertada dentro de la misma puede resultar en un desa-

rrollo más profundo del pensamiento computacional, involucrando a los estudiantes en actividades que les permitan mejorar sus habilidades de análisis, descomposición, abstracción, solución de problemas, reconocimiento de patrones, creatividad, y comunicación de ideas en diferentes contextos.

El estudio de la Optimización Combinatoria podría brindar un puente entre los problemas de la vida diaria y el pensamiento computacional de forma que, los alumnos sean capaces de relacionar las técnicas ocupadas en una situación cotidiana, con métodos robustos para problemas complejos. A continuación, se exponen tanto los principales beneficios que brindaría el estudio de la Optimización Combinatoria, como un pequeño ejemplo práctico que será desarrollado para resaltar estos beneficios.

9.2 Beneficios del estudio de la Optimización Combinatoria

9.2.1 Las aplicaciones prácticas son intuitivas

Uno de los aspectos que se considera clave en esta disciplina, es generar técnicas que permitan solucionar problemas en la vida real, y es por eso por lo que, al tomar un curso de Optimización Combinatoria los temas siempre estarán relacionados con un problema en específico o con un conjunto de problemas parecidos. Entre los problemas más comunes que pueden estudiarse, se distinguen: encontrar la ruta más corta para realizar un recorrido, seleccionar un subconjunto de productos que sean baratos pero que a la vez sean útiles, planear de forma adecuada una lista de actividades, entre muchas otras aplicaciones. Cada uno de estos problemas pueden estar relacionados a la industria, a la docencia e inclusive a la vida cotidiana de una persona; y al ser problemas que se encuentran en el día a día es necesario abstraerlos, reconocer patrones, diseñar algoritmos y algunas veces descomponerlos en problemas que, en la literatura, se sabe como solucionar.

9.2.2 Desarrollo de habilidades para la resolución de problemas

Para resolver un problema es indispensable primero entenderlo, es decir, cuál es su objetivo, qué tipos de condiciones tiene, y qué factores hacen que una solución sea buena o inservible. Una de las habilidades de mayor relevancia en esta área, es aprender a modelar problemas esta tarea va de la mano con abstraer el problema planteado, para estas actividades es necesario seleccionar la forma adecuada la representación de una solución, definir qué tipos de soluciones se buscan y proveer formas adecuadas de evaluar las mismas que permitan diferenciar qué tipo de soluciones son mejores que otras. Si bien estas tareas prácticamente definen una primera etapa en la resolución de problemas en cualquier disciplina, específicamente en

Optimización Combinatoria estas tareas fortalecen de forma concreta la descomposición de las tareas necesarias en el proceso de resolución, el reconocimiento de métodos y estrategias que facilitan la búsqueda de soluciones, la abstracción de métodos que sirven en un dominio determinado para ser usados de forma general, y finalmente, la descripción y estructuración de forma algorítmica de cada uno de los recursos considerados en las habilidades previamente mencionadas

9.2.3 Aprender nuevas formas de resolver un problema

Cuando una persona se enfrenta a un problema, muchas veces obvia lo difícil que puede ser encontrar la mejor solución a ese problema en específico. Esta dificultad puede depender de diferentes factores, por ejemplo, porque se trata de un problema nuevo y diferente a los solucionados anteriormente, por falta de experiencia en ese contexto, porque la complejidad del problema es grande, por el gran número de soluciones para resolver ese problema, o bien, porque la estrategia que se está considerando no es la adecuada. El campo de la Optimización Combinatoria estudia el desarrollo de algoritmos heurísticos, y en este sentido, una heurística es un procedimiento que se basa en la experiencia y que facilita la búsqueda de soluciones a un problema. Este tipo de algoritmos tiene dos beneficios prácticos: (1) el primer beneficio es que puede aproximar soluciones óptimas buscando sólo en un subconjunto de estas, dicho de otra forma, evita el análisis de cualquier posible combinación que dé como resultado una solución a un problema; (2) el segundo beneficio es que los algoritmos heurísticos proveen diversos métodos para la resolución de los problemas, esta ventaja será abordada en el siguiente punto. Es importante mencionar que muchos de los problemas que trata de resolver la Optimización Combinatoria son realmente complejos, donde el número de posibles soluciones puede llegar a exceder cualquier número que la mente humana pueda imaginar; en el intento de resolver estos problemas podría ser necesario poner en práctica la habilidad de descomposición, para llevar un problema a dos o más problemas que sean más manejables que del que se partió.

9.2.4 Adquisición de distintas perspectivas para analizar un problema

Como se mencionó en la sección anterior (9.2.3), las técnicas heurísticas incorporan distintos métodos para resolver un problema, y estos métodos pueden estar basados en procesos de enfriamientos de metales, la evolución de especies, la búsqueda de un punto más alto en una superficie con relieve, y la lista sigue. Este tipo de heurísticas invita a pensar de qué otras formas se puede resolver un problema o qué se puede hacer para mejorar las estrategias de búsqueda existentes. Asimismo, este punto en general no solo obliga a entender una estrategia, también provee una visión amplía de cómo han ido evolucionando los mecanismos para resolver problemas, cuáles

eran las técnicas que se empleaban hace 20 años y cuáles son las técnicas que se emplean actualmente. Con este panorama la persona que se adentre a la Optimización Combinatoria no solo tiene más herramientas para resolver un problema, sino que desarrolla o fortalece su habilidad para lidiar con la frustración, dicho de otra forma, esta persona perseverará, sin perder el ánimo, en la búsqueda de opciones de qué camino tomar si una solución a un problema no es tan adecuada como ella la imaginaba.

9.2.5 Aportaciones a la ciencia, al conocimiento y el desarrollo personal

Al desarrollarse en este campo de la ciencia, una persona no sólo obtendrá un beneficio personal como lo es el conocimiento o el desarrollo del pensamiento computacional, también puede generar nuevas investigaciones, mejores técnicas y experimentos más completos, mejorando la forma en la que se resuelven los problemas, y la forma en la que se hace investigación en este campo. Ejemplos de estas técnicas, que suelen dar mejores resultados en optimización combinatoria, son las relacionadas a Cómputo Evolutivo e Inteligencia Colectiva. Este tipo de métodos surgieron tiempo después de los algoritmos clásicos para búsqueda de soluciones en problemas combinatorios, no obstante, son unas de las técnicas más empleadas en la actualidad para encontrar soluciones óptimas a problemas combinatorios complejos. Los fundamentos y motivaciones de estas herramientas no solo muestran ingenio, sino una gran capacidad de abstracción donde personas que las ocupan o las estudian, adquieren la habilidad de hacer analogías entre eventos de la naturaleza y métodos puntuales para la búsqueda de soluciones. Así mismo, poder interpretar los resultados que ofrecen estos métodos y brindar explicaciones de su desempeño forman parte de las ventajas del desarrollo del pensamiento computacional. El perfeccionamiento de este tipo de habilidades, además de brindar una ventaja a la persona que las practica, representa un beneficio para sociedad en general si, con las habilidades y conocimientos adquiridos, se generan investigaciones y aplicaciones en contextos y problemáticas relevantes.

9.3 Ejemplo práctico de un problema combinatorio de la vida cotidiana

El siguiente ejemplo plantea una situación común donde será desarrollado cada uno de los beneficios mencionados en la lista de la sección anterior. El problema que se plantea es un caso específico del Problema de la Mochila, un problema clásico de Optimización Combinatoria [5].

9.3.1 Las aplicaciones prácticas son intuitivas

Ejemplo: La hermanas Distancia, Sus y Ana Distancia, fueron a visitar un centro de atención a clientes de un proveedor de telefonía debido a que Sus necesitaba un nuevo teléfono para su trabajo. Al llegar al centro de atención se encontraron con un gran anuncio con la siguiente promoción: "En la compra de dos celulares te regalamos una tarjeta con $500.00 para comprar lo que quieras de los artículos marcados de la tienda" (Tabla 9.1).

Las hermanas Distancia decidieron aprovechar la promoción y comprar también un celular para Ana con el fin de obtener la tarjeta de regalo y compartir las cosas que compraran con ella. Además de la condición que la suma de los precios de los artículos seleccionados no sea mayor a $500.00, también necesitaban que los productos no fueran tan pesados porque todavía tenían un día muy largo y no querían llevar tanto peso con ellas.

Entonces, las hermanas se dieron cuenta que debían tomar una serie de decisiones dado que tenían las siguientes preguntas:

a) ¿Qué artículos podríamos elegir para aprovechar al máximo el monto disponible de la tarjeta?
b) ¿Qué artículos tendríamos que elegir si no queremos cargar tanto y nuestra bolsa ecológica solo soporta a lo más 10 gramos de peso?

Observe que, en este punto, se plantea un problema que puede ser encontrado en la vida cotidiana y podría ser resuelto con herramientas que se brindan al estudiar Optimización combinatoria, las cuales, como se mencionó en la sección anterior, aportan en el fortalecimiento de las habilidades del pensamiento computacional. Una de esas habilidades es la abstracción, la cual ayuda a formular el problema en lenguaje matemático, en el siguiente punto se abordará más al respecto.

Tabla 9.1: Artículos marcados para la promoción.

Artículo	Costo ($)	Peso (g)
Funda de Celular	150.00	3.00
Lapicero Touch	100.00	1.00
Audífonos	250.00	1.50
Cargador	200.00	5.00
Batería Inalámbrica	450.00	10.0

9.3.2 Desarrollo de habilidades para la resolución de problemas

La situación en el ejemplo puede ser planteada matemáticamente a partir de su información más importante:

a) Se cuentan con 5 objetos a elegir (funda de celular, lapicero touch, audífonos, cargador y una batería inalámbrica), de los cuales sólo se podrá elegir a lo más una pieza de cada artículo para tener una lista variada de compras.
b) Las especificaciones de cada artículo contienen su peso, permitiendo crear un vector (una lista) p que represente de forma ordenada los pesos de cada artículo:
$$\boldsymbol{p} = (3.0, 1.0, 1.5, 5.0, 10).$$
c) Al igual que con el vector de pesos, se puede crear un vector c con los costos ordenados de cada artículo:
$$\boldsymbol{c} = (150, 100, 250, 200, 450).$$
d) Para indicar dentro de una solución si un artículo fue elegido, se añadirá otro vector a donde cada posición representa a un artículo; si en determinada posición existe un 0 quiere decir que no se comprará el artículo y si existe un 1 quiere decir que sí se comprará; por ejemplo, para indicar que sólo se comprará el lapicero touch se usará la siguiente representación:
$$\boldsymbol{a} = (0, 1, 0, 0, 0).$$
e) El objetivo para este problema es aprovechar al máximo el monto disponible de la tarjeta, es decir, maximizar la suma de los precios de los artículos elegidos con dos restricciones: la suma de la compra no debe sobrepasar los $500.00, ni su peso debe exceder la cantidad de 10 gramos. Este problema puede ser representado matemáticamente de la siguiente forma:
$$\max \sum_{i=1}^{5} a_i \cdot c_i,$$
sujeto a
$$\sum_{i=1}^{5} a_i \cdot c_i \leq 500$$
$$\sum_{i=1}^{5} a_i \cdot p_i \leq 10.$$

Entonces, las hermanas Distancia notaron que, a pesar de tener el problema y el objetivo ya planteados, no se ha obtenido una solución. ¿Cómo se podrían elegir los productos? ¿Qué tipo de estrategia se podría ocupar para considerar artículos que les ayuden a cumplir su objetivo? En este punto se practicaron

dos habilidades del pensamiento computacional: (1) la capacidad de descomposición, al identificar la información relevante que se describe en los incisos (a)-(e). En optimización combinatoria es importante identificar los parámetros (incisos (b) y (c)), definir la representación de la solución (inciso (d)), las restricciones (inciso (a)) y el objetivo (inciso (e)), y (2) la abstracción, al representar en vectores la información y al formular el modelo matemático.

9.3.3 Aprender nuevas formas de resolver un problema

Retomando el ejemplo, una estrategia que las hermanas Distancia podrían elegir para maximizar el número de objetos y el costo total es concentrarse en los artículos más baratos, siempre y cuando la suma de sus costos no sobrepase los $500.00 ni el peso máximo de 10 gramos. Esta estrategia las llevaría a la siguiente solución:

$$\boldsymbol{a} = (1, 1, 0, 1, 0).$$

Para este caso la funda de celular, el lapicero *touch* y el cargador son seleccionados. Sin embargo, se percatan que la suma de sus costos es de $450.00 y el peso total de las compras es de 9 gramos. A pesar de cumplir con las restricciones del problema, las hermanas se preguntan: ¿podrán existir mejores soluciones que aprovechen todo el monto de la tarjeta y todo el espacio disponible?

Para seguir buscando buenas soluciones, a Sus se le ocurre la siguiente idea: '¿Por qué no nos enfocamos en los productos más caros? Tal vez, así sea más fácil ocupar toda la tarjeta de regalo'. A partir de esta estrategia, las hermanas obtienen la siguiente solución:

$$\boldsymbol{a} = (0, 0, 0, 0, 1);$$

es decir, elegir solo una batería inalámbrica. En este caso también se ha seleccionado una solución cuyo precio y peso satisfacen las restricciones. No obstante, a diferencia de la solución previa se ha reducido el número de artículos seleccionados. A partir de esta observación surgen las siguientes preguntas: ¿esta es la mejor solución? ¿Es mejor o peor que la anterior?

Tabla 9.2: Artículos marcados para la promoción y su razón Costo/Peso.

Artículo	Costo ($)	Peso (g)	Costo/Peso
Funda de Celular	150.00	3.00	50.0
Lapicero Touch	100.00	1.00	100.0
Audífonos	250.00	1.50	166.7
Cargador	200.00	5.00	40.0
Batería Inalámbrica	450.00	10.0	45.0

Ana, revisando qué estrategias se han planteado, se dio cuenta que sólo han tomado en consideración el costo de los artículos y las restricciones planteadas. A Ana se le ocurre la siguiente idea: ¿Por qué no dividimos el precio de cada objeto entre su peso? Y de esta forma podremos saber cuáles son los productos que ofrecen más barato cada gramo de su peso, y tal vez si seleccionamos los más baratos podríamos obtener una buena solución. A partir de estrategia, obtuvieron la siguiente solución:

$$\boldsymbol{a} = (1, 0, 0, 1, 1).$$

Sin embargo, Sus y Ana se percatan que el costo total de los artículos seleccionados es $800.00, que la suma de sus pesos es 18 gramos y, por lo tanto, la solución no es válida. Ana al ver decepcionada a Sus, le dice 'No te preocupes, tengo otra idea. ¿Qué te parece si a esta solución invalida le hacemos un cambio para volverla válida?'. A Sus se le ocurrió seleccionar el segundo y tercer artículo con menor razón costo peso obtenido y eliminar la batería inalámbrica, pues es el artículo que tiene mayor costo y limita comprar más objetos, de esta manera se obtuvo la siguiente solución,

$$\boldsymbol{a} = (1, 1, 0, 1, 0).$$

Notaron con sorpresa que esta solución era la misma que obtuvieron al principio, a pesar de buscarla con diferentes estrategias. Sus y Ana Distancia no tuvieron más opción que pensar en métodos más complejos para tratar de buscar soluciones mejores.

9.3.4 Adquisición de distintas perspectivas para analizar un problema

Como cierre del ejemplo, Sus y Ana Distancia no tuvieron más opción que aceptar la mejor solución que encontraron en su búsqueda, esta solución estaba compuesta de 3 artículos que costaban $450.00 y pesaban 9 gramos. Sin embargo, las hermanas se quedaron con la incertidumbre si esta era la mejor solución que podrían encontrar para el problema.

Una posible forma de abordar el problema, haciendo uso de las herramientas que se pueden adquirir en un curso de optimización combinaría, se presentará a continuación. Sea

$$\boldsymbol{a} = (0, 0, 0, 0, 0)$$

una solución inicial, la cual indica que hasta el momento ningún artículo ha sido seleccionado. Como heurística se ocupará la razón del costo entre el peso, pero en esta ocasión en vez de seleccionar los objetos con razones pequeñas, se tomará en cuenta los objetos que maximicen la misma. Ahora bien, para generar una nueva solución se propone un método un poco más elaborado que consiste en generar un

conjunto de vecinos de nuestra solución actual. Un vecino es la solución resultante al aplicar una transformación a otra solución. Para este caso podría ser cambiar un digito a la vez, es decir, seleccionar una posición, si en esta hay un 0 cambiarlo por un 1 y viceversa.

Las Tablas 9.3, 4, y 5 mostrarán en detalle cómo funciona el método propuesto.

Tabla 9.3: Primer vecindario.

Solución	Costo ($)	Peso (g)	Costo/Peso
$(0, 0, 0, 0, 0)$	0.00	0.0	0.0
Vecinos	Costo ($)	Peso (g)	Costo/Peso
$(1, 0, 0, 0, 0)$	150.00	3.0	50.0
$(0, 1, 0, 0, 0)$	100.00	1.0	100.0
$(0, 0, 1, 0, 0)$	250.00	1.5	166.7
$(0, 0, 0, 1, 0)$	200.00	5.0	40.0
$(0, 0, 0, 0, 1)$	450.00	10gr.	45.0

Del primer vecindario, la mejor solución es $\boldsymbol{a} = (0, 0, 1, 0, 0)$, cuya razón es de 166.7 y representaría el primer paso del método en el segundo paso, se fija la solución inicial como la mejor solución del primer vecindario y se busca sus vecinos.

Tabla 9.4: Segundo vecindario.

Solución	Costo ($)	Peso (g)	Costo/Peso
$(0, 0, 1, 0, 0)$	250.00	1.5	166.7
Vecinos	Costo ($)	Peso (g)	Costo/Peso
$(1, 0, 1, 0, 0)$	400.00	4.5	216.7
$(0, 1, 1, 0, 0)$	350.00	2.5	266.7
$(0, 0, 0, 0, 0)$	0.00	0.0	0.0
$(0, 0, 1, 1, 0)$	450.00	6.5	206.7
$(0, 0, 1, 0, 1)$	700.00	11.5	211.7

Del segundo vecindario, la mejor solución es $\boldsymbol{a} = (0, 1, 1, 0, 0)$ cuya razón es de 266.7, esta solución se caracteriza por costar $350.00 y pesar 2.5 gramos. Como aún hay dinero y espacio disponible, se procede a generar el tercer vecindario.

La mejor solución del tercer vecindario tiene un peso total de 5.5 gramos y cuesta $500.00, el monto exacto de la tarjeta de regalo. Intuitivamente, representa una mejor solución que la encontrada por las hermanas Distancia porque a pesar de que

Tabla 9.5: Tercer vecindario.

Solución	Costo ($)	Peso (g)	Costo/Peso
(0, 1, 1, 0, 0)	350.00	2.5	266.7
Vecinos	Costo ($)	Peso (g)	Costo/Peso
(1, 1, 1, 0, 0)	500.00	5.5	316.7
(0, 0, 1, 0, 0)	250.00	1.5	166.7
(0, 1, 0, 0, 0)	100.00	1.0	100.0
(0, 1, 1, 1, 0)	550.00	7.5	306.7
(0, 1, 1, 0, 1)	800.00	16.0	311.7

tiene el mismo número de productos, reúne artículos de mayor valor, logra ocupar el monto total de la tarjeta de regalo y satisface el límite de peso.

El procedimiento descrito en el método propuesto se clasifica como una búsqueda local dentro de la Optimización Combinatoria debido a que busca el mejor vecino de la solución actual. Si bien, se logró encontrar una solución, este es un problema y un procedimiento sencillo, en comparación con los que se pueden encontrar dentro de la Optimización Combinatoria. Otro tipo de estrategias pueden surgir si se decide de vez en cuando considerar soluciones no tan buenas o considerar más de una solución a la vez. Dentro de la Optimización Combinatoria existen diversas técnicas que se basan en fenómenos intuitivos, naturales y sociales que proponen estrategias para la resolución de problemas.

En este punto se plantea, una forma de resolver el problema haciendo uso de las herramientas que se pueden aprender en optimización combinatoria y se plantean una serie de pasos a realizar que ayudan al diseño del algoritmo.

9.3.5 Aportaciones a la ciencia, al conocimiento y el desarrollo personal

Sus y Ana aplicaron el método científico para resolver un problema de la vida cotidiana. Su método de solución se basó en la observación, la predicción, la experimentación planificada, las reglas del razonamiento y la evaluación de resultados. Cada vez que ellas se enfrenten a algún problema parecido el del ejemplo buscarán soluciones apropiadas y cuando estas soluciones se establezcan como un principio general que se puede reproducir, se estará aplicando el método científico.

El ejemplo estudiado es sólo un caso particular del Problema de la Mochila. Este es un problema de Optimización Combinatoria de formulación sencilla, aunque su resolución puede llegar a ser muy compleja. El problema de la mochila aparece, directamente o como un subproblema en una gran variedad de aplicaciones, inclu-

yendo planificación de la producción, modelización financiera, muestreo estratificado, planificación de la capacidad de instalaciones, entre otras [5].

9.4 Conclusiones

El mundo que nos rodea está lleno de opciones, problemas y situaciones que enfrentar. Los seres humanos, y muchas especies animales, han resuelto problemas a lo largo de la historia de manera intuitiva. Si bien, algunas veces pareciera que se llega a la resolución de los problemas por mera suerte, lo cierto es que, sin darse cuenta, se emplean métodos muy sofisticados basados en la percepción del entorno y en la experiencia adquirida al solucionar problemas similares.

Los problemas de combinatoria se presentan en diferentes áreas prácticas, como la industria y la logística. Dentro de estas áreas se desea continuamente optimizar objetivos, por ejemplo, disminuir tiempos, maximizar ganancias, reducir costos, entre otros. Con muchos de estos problemas, ocurre un fenómeno interesante, el número de soluciones posibles crece de forma exponencial cuando se consideran más factores de un problema. Dicho fenómeno se denomina 'explosión combinatoria'. La Optimización Combinatoria permite analizar dichos problemas, abstraerlos, descomponerlos, modelarlos, y así como aplicar diferentes estrategias de solución con el objetivo de identificar las más apropiadas según las características de un problema particular; además, brinda herramientas para plantear, de forma ordenada, las intuiciones que surgen al buscar maneras de resolver un problema evitando la explosión combinatoria. El desarrollo de estas habilidades permite ampliar, fortalecer y llevar el pensamiento computacional a un nivel más profundo.

Los conocimientos y habilidades que ofrece el estudio de la Optimización Combinatoria ayudan a tomar mejores decisiones cuando, en la vida cotidiana, se encuentra con un problema combinatorio, de elección, agrupación u orden de elementos, como el planteado en el ejemplo. Los métodos intuitivos que proponen Sus y Ana Distancia para solucionar el problema, si bien, brindan opciones válidas, no consiguen obtener una elección de elementos con los que pudieran gastar todo el dinero de la promoción. Abordar el problema con un método más elaborado, permitió encontrar una solución donde se gastaron en total los $500.00. Las hermanas Distancia, analizaban las soluciones que generaban con sus métodos, se preguntaban cómo encontrar una mejor solución y planteaban nuevas formas de mejorar sus métodos, estas actividades se hacen, en mayor escala, al estudiar Optimización Combinatoria fortaleciendo la observación, el entendimiento del método planteado y la habilidad para lidiar con la frustración. De esta forma, es importante considerar tanto las metodologías como los métodos empleados en la Optimización Combinatoria como parte de las enseñanzas o estrategias para fomentar el pensamiento computacional.

Referencias

[1] Jeannette M Wing. *Computational Thinking Benefits Society*. Social Issues in Computing. Ene. de 2014. URL: http://socialissues.cs.toronto.edu/index.html%3Fp=279.html.

[2] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Algorithms and Combinatorics. Berlin Heidelberg: Springer-Verlag, 2003. ISBN: 978-3-540-44389-6.

[3] Francisco Ramón Fernández García. La Combinatoria: ¿un Arte? ¿una Forma de Pensar? ¿un Juego? En: *Las Matemáticas Del Mundo y El Mundo de Las Matemáticas*. Barcelon: Universitat de Barcelona, 2002.

[4] Miguel Sánchez García. Optimización Combinatoria. En: *Números: Revista de didáctica matemática* 43-44 (2000), págs. 115-120. ISSN: 0212-3096.

[5] Fernando Sandoya. El Problema de La Mochila, Complejidad, Cotas y Métodos de Búsqueda Eficientes. En: *Matemática* 12.2 (oct. de 2014), págs. 43-51. ISSN: 2661-6890.

Pensamiento Computacional en México
se imprimió en diciembre de 2021 en el taller Agys Alevín S.C.
Plásticos 84 local 2 Ala Sur, Fracc. Industrial Alce Blanco,
Naucalpan de Juárez, Estado de México CP 53370.
En su composición se utilizó tipo Garamond.
Impreso en papel couché mate de 115 grs.
La edición consta de 50 ejemplares.