Introducción al pensamiento computacional: conceptos básicos para todos

Jorge Luis Zapotecatl López

ACADEMIA MEXICANA DE COMPUTACIÓN, A, C.

Introducción al pensamiento computacional: conceptos básicos para todos.
Autor: Jorge Luis Zapotecatl López.

En colaboración con la Academia Mexicana de Computación:
Coordinador: Luis Enrique Sucar Succar.
Colaboradores capítulos 1 y 4: Yasmín Hernández Pérez y Alfredo Sánchez Huitrón.
Colaboradores capítulo 2: Eduardo Morales Manzanares y Eduardo Gómez Ramírez.
Colaboradores capítulo 3: Julieta Noguez Monroy y Luis Enrique Sucar Succar.
Colaboradores capítulos 5 y 6: Laura Cruz Reyes y Karina Figueroa Mora.
Colaboradores capítulo 7: Julieta Noguez Monroy y Eduardo Gómez Ramírez.
Colaborador capítulo 8: Eduardo Morales Manzanares.

Primera edición: 2018
Academia Mexicana de Computación, A. C.

ISBN:

Corrección de estilo: Laura Cruz Reyes, Miguel Antonio Lupián Soto y Luis Enrique Sucar Succar.
Diseño de portada: Mario Alberto Vélez Sánchez.
Cuidado de la edición: Luis Enrique Sucar Succar.

Este libro se realizó con el apoyo del CONACyT, Proyecto 279550.

Impreso en México.
Printed in Mexico.

Introducción al pensamiento computacional: conceptos básicos para todos.

Jorge Luis Zapotecatl López

Con la colaboración de

Luis Enrique Sucar Succar
Julieta Noguez Monroy
Laura Cruz Reyes
Yasmín Hernández Pérez
Karina Figueroa Mora
Alfredo Sánchez Huitrón
Eduardo Morales Manzanares
Eduardo Gómez Ramírez

Dedicatoria

Dedico este libro a mi familia y a mis amigos.

Agradecimientos

Agradezco a la Academia Mexicana de Computación, en especial a los integrantes que colaboraron con el desarrollo del libro y confiaron en mi trabajo: Luis Enrique Sucar Succar, Eduardo Morales Manzanares, Julieta Noguez Monroy, Laura Cruz Reyes, Yasmín Hernández Pérez, Karina Figueroa Mora, Alfredo Sánchez Huitrón y Eduardo Gómez Ramírez. A Luis Alberto Pineda Cortés por sus valiosos comentarios y sugerencias sobre este libro.

Agradezco a la Coordinación de Ciencias de la Computación del INAOE, y al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas de la UNAM por las facilidades que me han dado para elaboración de este libro.

Agradezco a Carlos Gershenson García, David Arturo Rosenblueth Laguette, Angélica Muñoz Meléndez, Jesús Ariel Carrasco Ochoa y José Francisco Martínez Trinidad por todas sus enseñanzas.

A mis amigos José Roberto Pérez Cruz, Gustavo Carreón Vázquez, Julio César Pérez Sansalvador, Luis Enrique Gómez Vázquez y Atlántida Sánchez Vivar por sus valiosos consejos.

Agradezco al CONACyT por su apoyo para la creación de esta obra.

Suplementos en línea

El sitio web `http://www.pensamientocomputacional.org/` incluye las respuestas de los ejercicios planteados en este libro.

Prólogo

Este libro se publica a casi 60 años de la llegada de la primera computadora a México. Desde entonces, muchas cosas han cambiado y hemos sido testigos de una enorme transformación del mundo y la sociedad, que ha adoptado la tecnología y aceptado a la computación como elemento indispensable para realizar las más diversas actividades, desde resolver problemas científicos hasta actividades lúdicas, como jugar o socializar virtualmente.

El mundo en el que vivimos es dinámico y cada día se desarrollan nuevas herramientas tecnológicas que nos permiten enfrentar los problemas que aquejan a la sociedad. Por ello, resulta necesario aprender a plantear la solución de problemas en términos de procesos computacionales, con el fin de encontrar mejores soluciones a los problemas actuales y futuros. Se han realizado estudios que evidencian que el desarrollo del pensamiento computacional fortalece un conjunto de habilidades necesarias para resolver problemas complejos y multidisciplinarios de forma creativa, por lo que su enseñanza propiciará que las personas adopten un rol más activo para reconocer problemas de interés público o personal, y proponer soluciones innovadoras basadas en sus principios.

El pensamiento computacional es un conjunto de habilidades que permiten formular modelos mentales necesarios para plantear la solución de problemas, usando o no computadoras; además, ayudan a desarrollar la interpretación de

las representaciones. En general, mantiene un enfoque orientado a los procesos y métodos de resolución de problemas y a la creación de soluciones computables.

Resulta necesario que la sociedad en general adquiera estas habilidades para propiciar que todas las personas sean capaces de resolver una gran variedad de problemas utilizando diversas estrategias, las cuales podrán ser implementadas en sistemas computacionales para obtener soluciones más rápidas, confiables y precisas.

Por esta razón, muchos países en Asia, Europa y América están incluyendo el pensamiento computacional como una estrategia para ampliar las competencias en la formación de sus estudiantes desde etapas muy tempranas. En Estados Unidos, por ejemplo, se han unido el gobierno, organizaciones profesionales, investigadores, educadores e industria para patrocinar proyectos enfocados principalmente a estudiantes de educación básica (desde jardín de niños hasta bachillerato). Algunas universidades han actualizado los programas de estudio de cursos introductorios a ciencias de la computación, incorporando conceptos fundamentales de la computación, así como del pensamiento computacional.

La forma en cómo el pensamiento computacional se integra en el currículo escolar varía de un país a otro. En algunos casos, es a través de las áreas temáticas, particularmente en el nivel primario, mientras que en otros es parte de una asignatura informática separada, que generalmente se enseña en el nivel secundario. Además, estos dos enfoques a menudo se combinan. Algunos países como Gales y Austria, consideran que el pensamiento computacional y sus conceptos relacionados forman parte del plan de estudios de competencia digital. Éste también es el caso del Marco Europeo de Competencia Digital para Ciudadanos.

Sin embargo, en nuestro país hemos tardado en reconocer la importancia de educar a la sociedad, en particular a jóvenes y niños, para que desarrollen y apliquen el pensamiento computacional en la resolución de problemas.

Para México es de gran importancia que se haga conciencia del valioso aporte que brinda el pensamiento computacional en todos los niveles educativos. Su incorporación como una estrategia educativa nacional nos puede abrir puertas, permitiendo elevar el nivel académico de nuestros estudiantes, impulsando el progreso y beneficiando al desarrollo del país.

Dada la situación en México de la enseñanza de la computación a nivel medio y medio superior, en el Instituto Nacional de Astrofísica, Óptica y Electrónica, se inició un esfuerzo por traer las ideas de pensamiento computacional a México, y tratar de cambiar la forma de la enseñanza en esta área a nivel secundaria y preparatoria. En este grupo inicial participaron los Doctores Enrique Sucar, Eduardo Morales y Pilar Gómez, y los Maestros Alma Ríos y Jorge Zapotecatl. Dicho esfuerzo continuó con el apoyo de la Red de Tecnologías de Información y Comunicaciones (TICs) de CONACyT bajo el liderazgo del Dr. Eduardo Morales. En este marco de creciente interés hacia el pensamiento computacional, en 2012 el M.C. Jorge Zapotecatl inició una investigación sobre pensamiento computacional, motivado por los Doctores Eduardo Morales y Enrique Sucar, quienes pensaron que era necesario que los estudiantes desarrollaran habilidades críticas al interactuar con las computadoras, además de motivar a más estudiantes a cursar carreras y posgrados en computación. En 2014, el M.C. Jorge Zapotecatl concibió la oportunidad de ofrecer un libro sobre Pensamiento Computacional para estudiantes de secundaria y preparatoria en México, por lo que inició la redacción de la presente obra: *Introducción al Pensamiento Computacional*.

En 2017, la Academia Mexicana de Computación (AMexComp), presidida por el Dr. Luis A. Pineda, advierte el potencial y la necesidad de dicho libro, por lo que decidió impulsar su la elaboración. De esta manera, en la reunión anual 2017 de la AMexComp, los Doctores Luis Pineda y Enrique Sucar convocaron a los miembros a participar en la revisión del libro, recibiendo una entusiasta respuesta, de donde surgió un grupo nutrido de colaboradores: los Doctores Lucía Barrón, Laura Cruz, Karina Figueroa, Eduardo Gómez, Yasmín Hernández, Julieta Noguez y Alfredo Sánchez, además de los Doctores Eduardo Morales, Enrique Sucar y Luis Pineda.

En diciembre 2017 se llevó a cabo una reunión, donde el equipo de colaboradores revisó y redefinió el objetivo, el enfoque y el contenido del libro. De esta reunión surgieron grupos de trabajo para cada capítulo, quienes hicieron una labor importante en la revisión, el contenido técnico y didáctico de cada capítulo. El grupo de trabajo contribuyó con su experiencia en la docencia y en la investigación para mejorar la elaboración de la presente obra. Cabe señalar, siempre se tuvo la intención de mantener la personalidad del libro que el Mtro. Zapotecatl había concebido desde 2012. Así, gracias a la visión y esfuerzo del Mtro. Jorge Zapotecatl, de los Doctores Eduardo Morales, Luis Pineda y Enrique Sucar, así como del apoyo de la AMexComp y CONACYT, este libro ve la luz en 2018.

La obra está compuesta por ocho capítulos que se pueden agrupar, por nivel de complejidad, en tres bloques de estudio. Los capítulos del 1 al 4 proporcionan los elementos básicos para desarrollar el pensamiento computacional. Los del 5 al 7 proporcionan herramientas más avanzadas, que se sugiere estudiar una vez que las básicas ya han sido aprendidas como habilidades creativas. Y el 8 presenta ejercicios que permiten desarrollar, de manera integral y más retadora, habilidades propias del pensamiento computacional. Todos los capítulos incluyen casos

de aplicación, así como ejemplos y ejercicios cuidadosamente dosificados para practicar y mejorar las diferentes habilidades que se exponen en ellos.

El Capítulo 1 inicia con una introducción al pensamiento computacional, destacando la importancia de desarrollar esta habilidad en la sociedad del siglo XXI. El 2 presenta el concepto de abstracción como una de las principales ideas del pensamiento computacional en el proceso de solución de problemas. El 3 trata sobre datos e información: se menciona cómo a partir de datos, se obtiene información y cómo representarlos computacionalmente; finalizando con la descripción de dos técnicas de transformación de mucha aplicación en, por ejemplo, seguridad electrónica. El 4 presenta a los algoritmos como una herramienta de abstracción poderosa para formular la solución de problemas. Este capítulo incluye formas sencillas de representación de algoritmos y los elementos básicos para su construcción: almacenamiento de datos e información, control e integración de acciones que se repiten y refinamiento de algoritmos mediante capas de abstracción.

Una vez adquirida la habilidad de diseñar algoritmos básicos, es tiempo de estudiar formas avanzadas de organizar sus componentes mediante las funciones -básicas y recursivas- introducidas en el Capítulo 5. De suma importancia también es medir la calidad de estos diseños. Así, el Capítulo 6 presenta y ejemplifica el proceso de medición del tiempo de ejecución de un algoritmo. Si bien el diseño de algoritmos eficientes requiere de un pensamiento abstracto, también es necesario desarrollar otras formas de pensamiento para tratar acciones simultáneas e interdependientes presentes en los sistemas naturales y artificiales. Con este fin, el Capítulo 7 introduce la técnica de simulación.

Finalmente, en el Capítulo 8 se describen dos experiencias de aprendizaje en las cuales se integran los conocimientos y habilidades adquiridos en los capítulos

anteriores. El primer ejercicio tratar de escribir tu nombre en maya, y el segundo en tejer un huipil.

Felicitamos al Mtro. Jorge Zapotecatl y a la Academia Mexicana de Computación por el importante esfuerzo para elaborar esta obra que esperamos sea de gran utilidad para docentes y estudiantes de nivel bachillerato. Estamos convencidas que facilitará la incorporación y promoción de competencias asociadas al pensamiento computacional.

Doctoras Lucia Barrón, Laura Cruz, Yasmín Hernández y Julieta Noguez

Personajes

A lo largo de estas páginas encontraras a los siguientes personajes que ayudarán a presentar la información de forma más amena.

Tukkul (pensar): este camaleón es un arqueólogo que le gustan los retos, cada problema lo entiende como una oportunidad para aplicar su conocimiento y creatividad a fin de encontrar soluciones. A veces es un poco solitario, pero se adapta fácilmente al cambio y le gusta cooperar con los demás para alcanza el bien común. A Tukkul le encanta leer las novelas de León Tolstói y jugar video juegos.

Paat (inventar): esta coyota es una científica con mucho entusiasmo para desarrollar constantemente inventos y aplicaciones innovadoras. Para ella, la ciencia cobra sentido cuando se pone al servicio del desarrollo humano. A Paat le gusta el *heavy metal* y le fascinan las matemáticas.

Makool (flojera): este topo es un pescador muy flojo y cree fácilmente todo lo que escucha. Por esta razón, tiene una baja calidad de vida. A Makool le maravillan las puestas de sol, aunque no pueda verlas. Además, le gustan los tacos de lengua con harta salsa.

Kukulkán (serpiente emplumada): este personaje, representa a Quetzatcóatl, es agricultor, músico, pintor, matemático y astrónomo. Se caracteriza por su entrega y amor tanto por la ciencia y el arte. Además, tiene un gran respeto por su cultura, el folclor y la sabiduría popular. Kukulkán solo puede ser contactado en el mundo de los sueños.

Índice general

Capítulo 1

Pensamiento computacional

El hombre se descubre cuando se mide con un obstáculo.
Antoine de Saint-Exupéry

Hace poco tiempo las computadoras de escritorio se volvieron una herramienta esencial para la industria y las actividades diarias de cada persona. En los años 90 del siglo XX, el poder de las computadoras personales aumentó de manera radical. Únicamente las personas más visionarias imaginaron la influencia que la computación tendría en la sociedad.

El desarrollo tecnológico ha permitido que los dispositivos móviles tengan el poder de procesamiento de una computadora de escritorio y puedan conectarse entre sí por medio de *Internet*. En consecuencia, vivimos la era de la computación ubicua (en todas partes), ya que los dispositivos móviles están al alcance de cualquier persona. En nuestra vida diaria y quehacer profesional utilizamos cotidianamente sistemas compuestos de software y hardware. Por un lado, regularmente no tenemos el conocimiento de cómo funcionan internamente y cómo

fueron construidos dichos sistemas. Por otro lado, si tuviéramos el conocimiento, las posibilidades que tenemos para innovar o implementar una idea por medio de dichos sistemas es ilimitada. Específicamente, si tú tienes conocimientos básicos de programación, entonces puedes imaginar una idea y desarrollarla en un lenguaje de programación. Finalmente, por medio de *Internet* poner tu idea a disposición mundial.

Por ejemplo, hasta el año 2000 los científicos de la computación no habían podido prevenir las cuentas de correo electrónico falsas que enviaban correo basura. El guatemalteco Luis Von Ahn (ver Figura 1.1 [1]), con sólo 22 años de edad, inventó el popular sistema de autentificación CAPTCHA, que resuelve dicho problema al determinar si el usuario es un humano por medio de exámenes que los seres humanos son capaces de contestar, pero que para las computadoras es muy complicado (como el reconocimiento de escritura deformada). Posteriormente, Luis Von Ahn inventó el sistema reCAPTCHA, que vendió a Google en millones de dólares. Esencialmente, el sistema CAPTCHA es la prueba Turing ¡conceptualizada por Alan Turing en 1950[2]. Esta es una muestra de cómo, al reorientar un problema de manera creativa, se obtienen soluciones innovadoras.

En la era de la economía del conocimiento, las personas y los países que progresan son los que remplazan el trabajo manual por el trabajo intelectual, ya que el conocimiento se utiliza para generar valor y riqueza, al transformarlo en información. Por ejemplo, el valor de Facebook es mayor que millones de productos manufacturados. Las proezas tecnológicas que contribuyen en prácticamente todas las áreas de estudio y el quehacer humano son creadas por personas creativas que desarrollan las habilidades superiores del pensamiento, como el razona-

[1] Recuperada de `https://es.wikipedia.org/wiki/Luis_von_Ahn`

[2] La llamada Prueba de Turing es básicamente una prueba que ideo el matemático Alan Turing para determinar si una computadora tiene una inteligencia similar a la de un humano.

Figura 1.1: Luis Von Ah, emprendedor y profesor de Carnegie Mellon University, ideó los sistemas de autentificación CAPTCHA y reCAPTCHA con base en los fundamentos de la prueba Turing propuesta en 1950.

miento abstracto, el pensamiento crítico, la resolución de problemas y la toma de decisiones, entre otras habilidades.

La *Fundación Nacional para la Ciencia* (NSF), por medio del *Sociedad Internacional para la Tecnología en la Educación* (ISTE) y la *Asociación de profesores de informática* (CSTA), impulsa activamente un nuevo enfoque de enseñanza para que en todos los niveles de educación se incluya el *Pensamiento Computacional* (*Computational Thinking*). Este nuevo enfoque busca promover en la educación el desarrollo de habilidades de pensamiento que conduzcan a la formación de personas orientadas a la creatividad y a la innovación.

La investigadora Jeannette Wing define el Pensamiento Computacional (PC) 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)".

Jeannette Wing es promotora del PC y su visión es que, al igual que el conocimiento del idioma o la aritmética, el PC "será una habilidad y una actitud de aplicación universal para todas las personas". Jeannette Wing propone que las habilidades de abstracción y las técnicas de resolución de problemas utilizados por los científicos e ingenieros de la computación se enseñen y apliquen en otras disciplinas o actividades de la vida cotidiana (ver Figura 1.2 [3]).

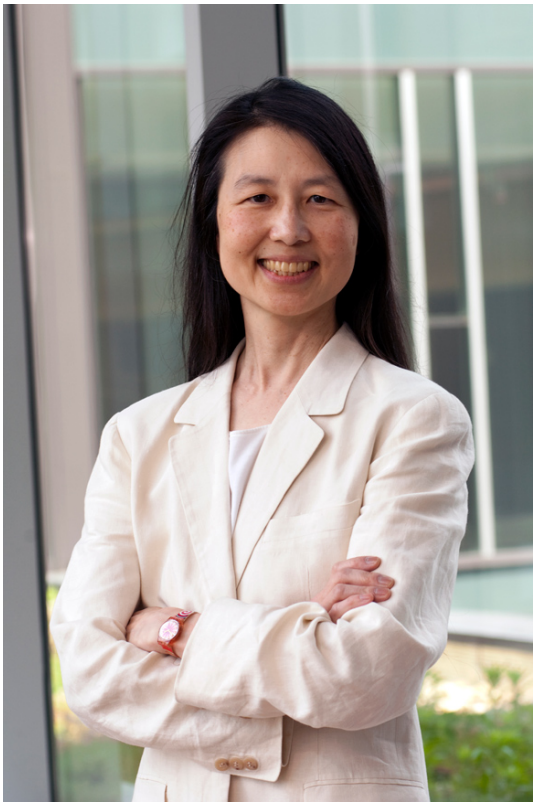


Figura 1.2: Jeannette Wing, investigadora de *Columbia University* e impulsora tenaz del Pensamiento Computacional.

[3] Recuperada de `https://industry.datascience.columbia.edu/profile/jeannette-wing`

El objetivo del PC es desarrollar sistemáticamente las habilidades del pensamiento de orden superior, como el razonamiento abstracto, el pensamiento crítico y la resolución de problemas, con base en los conceptos de la computación. Además, potenciar el aprovechamiento del poder de cálculo que tienen las computadoras actualmente para innovar y volverlo una herramienta científica.

En la actualidad, la capacidad de innovación e imaginación de los *pensadores de la computación* ha permitido la decodificación del genoma humano, el desarrollo de sistemas de información para alertar sobre condiciones climáticas peligrosas y la traducción de lenguajes antiguos, entre otras muchas soluciones a problemas importantes de la sociedad.

Los estudiantes y profesionistas tendrán la necesidad de aprender y practicar las habilidades del PC para poder utilizar las nuevas tecnologías y confrontar los desafíos del siglo XXI. El expresidente de Estados Unidos, Barack Obama, incluye el calentamiento global, la pobreza, y las enfermedades infecciosas entre los principales problemas globales. Desarrollar las habilidades del PC en los estudiantes y profesionistas ayudará a contribuir en la solución de dichos problemas y de otros retos.

1.1. Componentes del pensamiento computacional

Nuestra calidad de vida depende de nuestro pensamiento, porque es la habilidad que nos permite tomar decisiones y resolver problemas. Para alcanzar una buena calidad de vida se debe ejercitar el pensamiento, ya que el pensamiento de mala calidad propicia una mala calidad de vida. Precisamente, el PC ejercita las habilidades de pensamiento crítico y resolución de problemas con base en los

conceptos de la computación. La siguiente fórmula permite recordar e identificar los componentes del PC:

```
PC = pensamiento crítico + poder de la computación
```

El primer componente del PC es el *pensamiento crítico*. La *Fundación para el Pensamiento Crítico* define al pensamiento crítico como: "el modo de pensar (sobre cualquier tema, contenido o problema) en el cual el pensante mejora la calidad de su pensamiento al apoderarse de las estructuras inherentes del acto de pensar y al someterlas a estándares intelectuales".

En el pensamiento computacional, el pensamiento crítico se refuerza y desarrolla mediante los conceptos de la computación, como la abstracción y descomposición de problemas que pueden aplicarse a cualquier área del conocimiento. Cuando una persona utiliza el pensamiento computacional, piensa de manera crítica: tiene un propósito claro; cuestiona de manera constructiva la información, las conclusiones y los puntos de vista; se empeña en ser claro, exacto, preciso y relevante; y busca profundizar con lógica e imparcialidad.

El segundo componente del PC es el *poder de la computación*. Por medio de los conceptos de la computación, es posible entender cuáles aspectos de un problema son susceptibles de resolverse aprovechando el poder de procesamiento de las computadoras actuales. En resumen, el Pensamiento Computacional es un enfoque para la resolución de problemas que enfatiza la integración del pensamiento crítico y los conceptos de la computación. El PC realza las formas de estructurar un problema mediante los conceptos subyacentes de la computación. Además, el PC reenfoca la creatividad humana al permitir considerar las posibilidades de resolver problemas con ayuda del procesamiento de información de las computadoras.

1.1.1. El pensamiento computacional en la ciencia

Algunas personas que saben utilizar una computadora piensan que saben computación porque conocen cómo utilizarla. Sin embargo, como analogía, saber utilizar una calculadora no significa comprender los conceptos de la aritmética. La computadora actualmente es desaprovechada porque la gente desconoce los conceptos de la computación y no es capaz de utilizarla en formas más creativas y explotar su capacidad de apoyo como herramienta para el desarrollo del conocimiento.

El físico Heinz Pagels vislumbró que las computadoras no sólo revolucionarían a la sociedad, sino que también revolucionarían la manera de hacer ciencia. En su libro, *Los sueños de la razón*, menciona que históricamente los grandes avances de la ciencia estuvieron asociados con la invención de un instrumento que permitió una nueva forma de aproximarse a la verdad de la naturaleza.

Heinz Pagels describe el potencial de la computadora como el instrumento científico de la próxima generación de científicos y como la base de una nueva revolución científica. Consideró que instrumentos como el telescopio y el microscopio describían lo que era muy grande o muy pequeño para ser percibido a simple vista y que, en el caso de la computadora, nos proporciona los medios para simular lo que era demasiado complejo para ser entendido únicamente con la mente. Por ejemplo: el fenómeno del clima es difícil de pronosticar debido a que hay una gran cantidad de variables e interacciones que intervienen en su dinámica. Por esta razón, la única manera de aproximarse al fenómeno es simularlo computacionalmente y analizar los resultados. Heinz Pagels estuvo en lo cierto, actualmente la simulación por computadora ha sido designada como el tercer pilar de la ciencia, junto con la teoría y la experimentación clásica.

La computadora por lo general no es más que un instrumento que se utiliza para el trabajo de oficina, en tareas esenciales, pero no aporta un cambio revolucionario en la manera de confrontar los problemas de nuestro siglo. Precisamente, mediante el pensamiento computacional, la computadora puede explotarse de diversas maneras como un nuevo instrumento científico. Jeannette Wing sostiene que la esencia del pensamiento computacional es la abstracción y que las abstracciones para la computación son las herramientas "mentales" , y las computadoras las herramientas "metálicas" que automatizan las abstracciones.

Además, otra aportación importante del pensamiento computacional se relaciona con la manera de enseñar ciencia, que se basa tradicionalmente en *ecuaciones*. Las ecuaciones están en un alto grado de abstracción (un nivel mínimo de detalles), enfocándose en el *qué*. Por un lado, las ecuaciones resaltan la esencia de un problema que se desea representar, sin embargo, su alto grado de abstracción puede dificultar su comprensión. Por otro lado, los *algoritmos* se enfocan en el *cómo*, al descomponer un problema complicado en problemas de menor dificultad de manera estructurada hasta llegar a instrucciones elementales. Los algoritmos son una alternativa de representación que puede ser más comprensible para los humanos que las ecuaciones. En este sentido, si se complementa la enseñanza de las ciencias con los algoritmos, entonces se podría impulsar el desarrollo de la ciencia.

El pensamiento computacional beneficiará a los estudiantes, a las instituciones en todos los niveles de educación y, finalmente, al desarrollo humano, científico y tecnológico a nivel global. Por ejemplo, la computación está revolucionando la estadística; mediante el aprendizaje automático se hace posible identificar patrones y anomalías en enormes conjuntos de datos tan diversos como mapas astronómicos, imágenes obtenidas a partir de resonancias magnéticas

o compras realizadas con tarjeta de crédito. La computación está revolucionando también la biología, ya que el descubrimiento de la secuencia del genoma humano a través del algoritmo *shotgun* ha despertado el interés en los métodos computacionales en dicha área. De manera similar la computación tiene un gran impacto en práticamente todas las ciencias, incluyendo a las ciencias sociales.

1.2. Solución de problemas

El pensamiento computacional se define como los procesos de pensamiento involucrados en la formulación de problemas y representación de sus soluciones. ¿Qué significa la palabra *problema*? Considera el siguiente cuento:

Tukkul observaba el hermoso color turquesa del caribe en el acantilado de las ruinas de Tulum y piensa en cómo demostrarle a Paat la emoción y la alegría que siente cada vez que está con ella. Por esta razón, decidió darle un obsequio. No tiene idea clara de qué regalarle, sin embargo, puede plantear algunas opciones basadas en los gustos de Paat.

Paat siente una gran pasión por la física y la tecnología, de modo que Tukkul cree que los regalos que podrían gustarle son: el DVD de *El placer de descubrir las cosas* del físico Richard Feynman o un circuito Arduino. Si bien las flores no le gustan mucho, las orquídeas también serían una buena opción, porque la relajan. Incluso, Tukkul piensa que podría comprarle varios regalos y no necesariamente uno solo, siempre y cuando el costo total de los regalos no exceda su presupuesto de 1500 pesos.

El problema que Tukkul debe resolver es decidir el regalo o los regalos adecuados para Paat. Considerando los tres regalos (el DVD, el Arduino y las orquídeas), la primera dificultad a la que se enfrenta es a las muchas combinaciones de regalos. 1) Tukkul puede comprar un regalo, lo que implica tres posibilidades; 2) comprar dos regalos, dependiendo del regalo que elija no comprar se obtienen tres posibilidades; 3) comprar los tres regalos, sólo una posibilidad es obtenida y 4) Incluso, una opción sencilla sería simplemente no comprar ningún regalo, lo que implica una posibilidad.

Tukkul se quedó dormido, pensando en cómo solucionar el problema. En el sueño, se le ocurre visitar, en el Castillo de Tulum, a su amigo onírico Kukulkán para pedirle un consejo. Al

entrar, encuentra a Kukulkán tocando alegremente el "Huapango de Moncayo". Tukkul siente que todas las célula de su cuerpo vibran con cada una de las notas; la alegría y el magnetismo que transmite Kukulkán es compartido a todo el ambiente.

Tukkul le explica el problema que debe que resolver. Kukulkán, en lugar de impulsar alguna de las opciones que Tukkul había elegido, le da nuevas ideas: ¿Qué te parece si mejor le regalas algo que revele parte de tu personalidad y sensibilidad, como, un colorido alebrije (artesanía originaria de México) o el libro *Cien años de soledad*? ¡Le encantarían! (ver Figura 1.3).

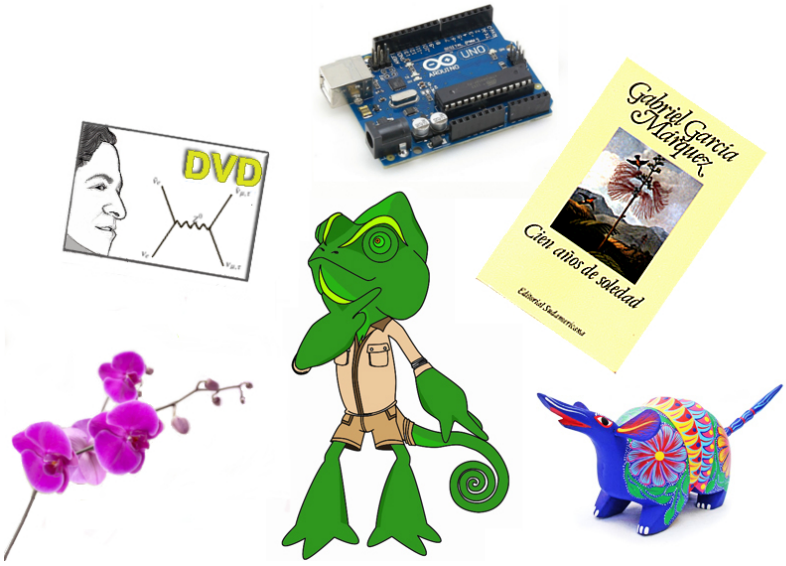


Figura 1.3: Tukkul quiere darle el regalo o los regalos que más le gusten a Paat y que no excedan su presupuesto.

La idea de Kukulkán es buena, sin embargo, ahora la situación es más complicada porque el número de opciones es mayor. Es decir, Tukkul debe verificar un conjunto mayor de posibles soluciones que cumplan con la restricción del presupuesto. Tukkul piensa que, de no encontrar una solución, escribirá una carta que exprese sus sentimientos y que hable desde el fondo de su corazón, que es más importante. Esto lo tranquiliza y, decide que lo primero que debe hacer es comprender cuál es exactamente el problema que debe solucionar.

Un problema es una relación entre un conjunto de instancias y un conjunto de soluciones. Un problema permite establecer formalmente la *relación* deseada entre las instancias de *entrada* y las soluciones de *salida*. El problema se resuelve si se obtiene al menos una solución correspondiente para cada instancia. Por ejemplo:

1. *Elevar un número al cubo*. La entrada es un número x y la salida un número y. La relación entre la entrada x y la salida y es que $y = x^3$.

2. *Encontrar la distancia entre dos puntos*. La entrada son dos puntos (x_1, y_1) y (x_2, y_2) y la salida es la distancia lineal entre los dos puntos. La relación está dada por la fórmula $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

Para exponer de manera ordenada y estructurada el problema de Tukkul, el primer paso es identificar los datos de entrada. Por simplicidad, solo consideraremos el conjunto de tres regalos (DVD, Arduino y orquídeas).

Entrada

1. La cantidad de dinero inicial: 1500 pesos.

2. El conjunto de regalos $R = \{DVD, Arduino, orquídeas\}$. Tukkul supone que el costo y el valor emocional de cada regalo son $\{300, 500, 1000\}$ y $\{10, 7, 5\}$, respectivamente. Los valores emocionales están en el rango de 0, que es indiferente, y de 10, que es emocionante. Por ejemplo: el Arduino cuesta 500 pesos y tiene un valor emocional para Paat de 7.

Una vez especificada la entrada del problema, puede definirse lo que se desea adquirir en la salida.

Salida

1. Las soluciones son los valores emocionales de cada combinación. Las soluciones que impliquen un costo mayor de 1500 pesos tendrán un valor de cero.

Relación entrada/salida

El primer paso es hacer la lista de las ocho combinaciones posibles:

{*DVD, Arduino, orquídeas*}
{*Arduino, orquídeas*}
{*DVD, orquídeas*}
{*DVD, arduino*}
{*DVD*}
{*Arduino*}
{*Orquídeas*}
{} (también se considera la posibilidad de no comprar ningún regalo)

El segundo paso es calcular el costo y el valor emocional de cada combinación (listadas en el mismo orden):

300 + 500 + 1000 = 1800 y 10 + 7 + 5 = 22
500 + 1000 = 1500 y 7 + 5 = 12
300 + 1000 = 1300 y 10 + 5 = 15
300 + 500 = 800 y 10 + 7 = 17
300 = 300 y 10 = 10
500 = 500 y 7 = 7
1000 = 1000 y 5 = 5

0 = 0 y 0 = 0

El tercer paso es relacionar los valores emocionales de cada combinación. Los valores emocionales que impliquen un costo mayor a 1500 adquieren el valor de cero (ver tabla 1.1).

Tabla 1.1: Relación entrada/salida

Entradas (instancias)	Salidas (soluciones)
{DVD, Arduino, orquídeas}	0
{Arduino, orquídeas}	12
{DVD, orquídeas}	15
{DVD, Arduino}	17
{DVD}	10
{Arduino}	7
{Orquídeas}	5
{}	0

Respecto al problema de Tukkul, la solución específica es comprar el DVD y el Arduino, porque tienen el valor emocional más alto (17) y un costo de sólo 800 pesos. El problema fue relativamente sencillo de solucionar. Sin embargo, si Tukkul considera los cinco regalos, entonces tiene que calcular 32 posibilidades. Si después tiene más ideas, como, ocho regalos ¡tendría que calcular 256 posibilidades!

Mediante el pensamiento computacional podemos automatizar nuestras soluciones. El primer paso sería que Tukkul desarrolle una *solución algorítmica*. Una solución algorítmica a un problema consiste de un *algoritmo* que, por cada instancia del problema, calcula al menos una solución correspondiente. Poste-

riormente, Tukkul implementaría su algoritmo en una computadora para calcular las soluciones de manera automatizada. En particular, el cálculo de las 256 posibilidades para las computadoras actuales se lograría en menos de un segundo. A lo largo de este libro se describirá qué son los algoritmos y cómo desarrollarlos para solucionar problemas.

Definir en forma precisa un problema que se desea solucionar es también un problema. Al resolver un problema es necesario, inicialmente, concentrarse en su especificación: definir la entrada, la salida, la relación entrada/salida, y eliminar cualquier restricción no significativa que aparte la atención del objetivo.

Considerando el problema de Tukkul, podría simplemente declararse como entrada al problema el deseo de Tukkul de comprarle un regalo a Paat. Sin embargo, dicha declaración es demasiado ambigua. Cuando el problema no es claro, el pensamiento no tiene una guía precisa y discurre sin tener objetivos relevantes. Pero cuando tomamos el tiempo para clarificar el problema, estamos más capacitados para resolverlo porque tenemos claro qué se requiere para resolverlo.

Una estrategia para especificar el objetivo que se desea lograr es dividir el problema en sus partes. Paso a paso se divide el problema principal en subproblemas de menor complejidad, para ser resueltos por separado. Al subdividir el problema podemos notar las relaciones y dónde se sobreponen. Por ejemplo, Tukkul comenzó por limitar el universo a tres regalos posibles y asignar valores emocionales a los regalos. Tuvo que precisar qué deseaba obtener al especificar la relación de entrada y salida, la cual se define como el deseo de satisfacer los gustos de Paat maximizando los valores emocionales. Además, tuvo que hacer la lista de combinaciones, calcular el costo y el valor emocional de cada combinación y relacionar ambos aspectos para encontrar una solución.

Cabe mencionar que, cuando se presenta un problema, deben considerarse los múltiples significados que pueda tener, dependiendo del contexto y las circunstancias en las que opera. Al hacer el problema más preciso, evitaremos algo que sucede con frecuencia: antes de identificar el problema que se desea resolver, se procede a resolver el problema incorrecto.

Por ejemplo, si Tukkul le hubiera preguntado a Paat qué regalo le gustaría, probablemente ella le hubiera explicado que acababa de comprar un Arduino, y esa opción no tendría que considerarse. Si éste hubiera sido el caso, los valores emocionales de entrada al problema serían incorrectos. La solución no sería incorrecta, lo incorrecto sería la especificación del problema.

1.3. Aplicando el pensamiento computacional

El pensamiento computacional tiene como finalidad desarrollar en los estudiantes y profesionistas el pensamiento crítico en colaboración con los conceptos claves de la computación, como abstracción, algoritmos, programación y simulación. Lo anterior con el fin de que las habilidades y los conceptos de la computación sean difundidos a nivel general, y no únicamente para los ingenieros y especialistas en computación.

Por ejemplo, un estudiante puede adquirir los conceptos de la computación e ingresar a otras profesiones, como la medicina, derecho, negocios, política, o cualquier otro tipo de ciencia o ingeniería e incluso las artes, y aplicar los conceptos de la computación.

Cuando el pensamiento computacional se aplica en situaciones de la vida diaria, las personas empiezan a darse cuenta de la utilidad del pensamiento crí-

tico y de los conceptos de la computación en el proceso de resolución de problemas y se vuelven conscientes de su importancia para mejorar su calidad de vida.

A continuación, se presentan algunos ejemplos de personas aplicando el pensamiento computacional:

- El estudiante que realiza un proyecto universitario, y busca en *Internet* sobre un tema, decide cuál es la información confiable y cuál es la información que debe desechar.

- El escritor que escribe una novela, y la comienza con su hipótesis principal, articula lo que va argumentar, hace un bosquejo y refina repetidamente su trabajo.

- El emprendedor que realiza un estudio de mercado sobre un producto y realiza estadísticas con una hoja de cálculo con base en datos del censo de su país.

- El individuo que expone con claridad, profundidad y amplitud un tema de índole científico, moral, religioso o político en un debate. Por ejemplo: los pros y contras acerca de la legalización de las drogas o el aborto.

- El científico que desarrolla modelos y simulaciones para representar sistemas biológicos o artificiales complejos.

Así como cada persona tiene habilidades de lectura, escritura y aritmética, el pensamiento computacional es una habilidad que puede adquirir cualquier persona, independientemente de su grado de estudio u oficio, y aplicarla en su vida personal y profesional.

1.4. Ejercicios

1. Paat asistirá a un concierto a las 4 de tarde, pero tiene una clase que acaba a las 3 de la tarde. Paat quiere saber en qué horario debe tomar el autobús de tal manera que llegue a tiempo y no tenga que salir temprano de la clase. Si los autobuses salen cada 15 minutos (10:00, 10:15, 10: 30, etc.) y el trayecto es de 30 minutos ¿Cuál es el mejor horario para tomar el autobús? ¿Cuál fue el razonamiento que seguiste? ¿Cuáles fueron las entradas, salidas y el problema por resolver?

2. Paat y Tukkul están organizando una fiesta para festejar el fin de cursos con sus compañeros. Quieren comprar botanas, pastel y refrescos y necesitan saber cuánto dinero deben pedir a sus compañeros. Identifica los componentes del problema: entrada, salidas y relación entrada/salida.

3. Las vacaciones de verano están a punto de terminar, Tukkul se da cuenta de que no tiene una mochila para llevar los libros a la escuela, así que corre a la tienda y compra la mochila que le pareció más bonita, con lo que el problema parecía resuelto. Sin embargo, el primer día de clases, Tukkul se da cuenta de que los libros no caben en la mochila nueva ¿Por qué la manera de resolver el problema no fue la mejor? ¿Qué le faltó a Tukkul? ¿Tú cómo lo hubieras resuelto?

4. Paat y Tukkul están trabajando en equipo en el proyecto final de la materia "Ciencias Naturales" y no saben cómo planear y dividir el trabajo, de tal manera que puedan entregar el proyecto completo y a tiempo, y todos los miembros del equipo trabajen. Piensa en un proyecto que hayas trabajado en equipo, recuerda como planearon y dividieron el trabajo ¿Cómo repartieron el trabajo? ¿El proyecto cumplió con lo que solicitó el

profesor? ¿Lo terminaron a tiempo? ¿Crees que si se hubiera aplicado el pensamiento computacional hubieras tenido mejores resultados? ¿Qué cambiarías?

5. Piensa en un problema en tu vida cotidiana, propón una solución utilizando el pensamiento computacional. Identifica la información disponible para solucionar el problema (entrada), identifica que es lo que quieres obtener como resultado al resolver el problema (salida), identifica el proceso para solucionar el problema (relación entrada/salida).

6. Considera la dinámica de un partido de tenis (o algún otro deporte de competencia). Aplica el pensamiento computacional para definir una estrategia que defina al ganador. ¿En qué consiste el problema? ¿Qué datos se necesitan para resolverlo?

7. Con frecuencia, se considera que el pensamiento computacional es aplicable solamente a problemas de ingeniería o matemáticas; sin embargo, puede aplicarse a prácticamente todas las áreas de actividad humana. Sugiere problemas que podrían surgir en las áreas siguientes y cuya solución podría apoyarse en el pensamiento computacional: derecho, música, economía, poesía, gastronomía.

8. Paat y Tukkul han decidido diseñar un espacio para trabajo y descanso para un grupo de 16 colaboradores. En dicho espacio deben colocar escritorios, sillas y mesas para reuniones, así como sillones, mesas para televisión y mesas de ping-pong. Sugiere cómo aplicar el pensamiento computacional en esta tarea.

9. Paat y Tukkul desean ahorrar para financiar un viaje por el mundo maya. Para ello, deciden aportar cada uno el mismo porcentaje de sus ingresos, los cuales son fijos, pero de montos diferentes. Sugiere cómo usar el pensamiento computacional para determinar el tiempo que se requerirá antes de que Paat y Tukkul puedan realizar el viaje que planean. ¿Cuáles son los datos de entrada? ¿Cómo se procesarán dichos datos para alcanzar el objetivo?

10. En vacaciones, Paat y Tukkul recorren la península de Yucatán. Tukkul fue de visita a a Mérida, mientras que Paat fue a Edzná. Para continuar juntos su recorrido, quieren reunirse en Uxmal, a donde desean llegar al mismo tiempo viajando por carretera; Paat desde el sur, Tukkul desde el norte. ¿Cómo pueden aplicar el pensamiento computacional para determinar la hora en que deben iniciar cada uno su viaje?

Capítulo 2

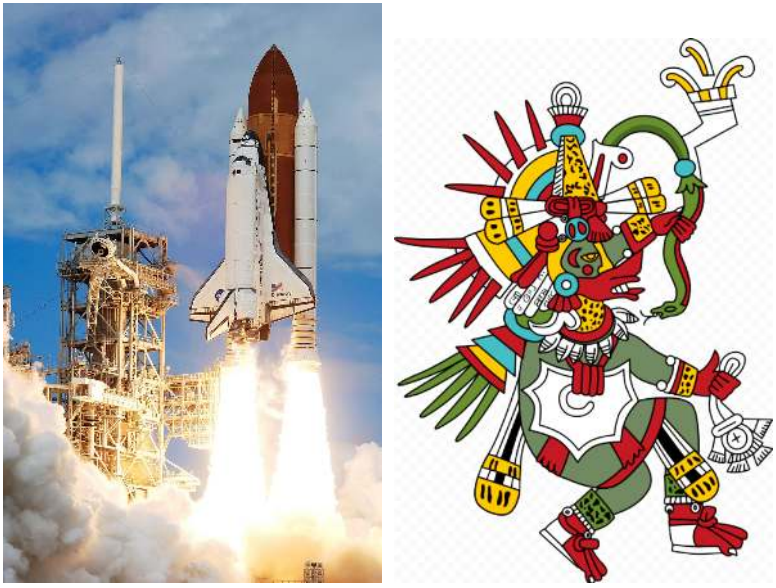
Abstracción

Lo esencial es invisible a los ojos...
Antoine de Saint-Exupéry

El ser humano, desde los comienzos de su existencia, ha luchado por su supervivencia en competencia con otros seres vivos y contra las inclemencias de la naturaleza. Algunos animales poseen habilidades físicas superiores a las del hombre: la fuerza de un oso, la agilidad de un tigre, la vista de una águila o el olfato de un perro. Incluso, el hombre ha soñado con tener la capacidad de volar como las aves. Sin embargo, a pesar de no poseer dichas habilidades físicas, el hombre es el ser vivo dominante en el planeta. La razón obedece a que cuenta con habilidades cognitivas superiores a los demás seres vivos: la memorización, la comprensión, el análisis, la elaboración, etc.

Las habilidades cognitivas han permitido al hombre desde la elaboración de herramientas primitivas para manipular su entorno, hasta alcanzar las grandes proezas científicas y tecnológicas de la actualidad. Además, preguntarse acerca

de su existencia y su devenir en el universo, reflejado en la creación de la religión y la filosofía (ver Figura 2.1[1]).



(a) Transbordador (b) Quetzalcóatl

Figura 2.1: Las habilidades cognitivas han permitido al hombre: a) el lanzamiento de transbordadores espaciales que colocan satélites en la órbita de la tierra y b) la creación de deidades religiosas que simbolizan creencias y prácticas del tipo existencial y moral.

La complejidad es inherente a nuestro mundo, cada entidad en el planeta tiene infinidad de propiedades que se relacionan con las propiedades de otras entidades en infinidad de maneras directas o indirectas. Si se pretende conocer algo acerca del mundo, desde el fenómeno más cotidiano como el estudio de una

[1]Recuperadas de `https://es.wikipedia.org/wiki/Transbordador_STS` y `https://es.wikipedia.org/wiki/Quetzalcóatl`

manzana cayendo de un árbol o un estudio más complejo como la manera de colocar un satélite en órbita, es impensable lograrlo sin la habilidad de abstraer.

El término *abstracción* se entiende popularmente en algunas ocasiones con un significado contrario al que tiene en realidad. Cuando algo parece excesivamente complejo o de difícil entendimiento (por ejemplo, una obra de arte), se suele expresar, "es muy abstracto". Sin embargo, por el contrario, la abstracción es el filtro utilizado para quedarse con lo que se considera esencial, eliminando toda la complejidad innecesaria.

La abstracción es la habilidad que le permite al ser humano combatir la complejidad al considerar sólo lo esencial del objeto o fenómeno que se esté analizando. Esta obra se enfoca en dos aspectos pertinentes de la definición de abstracción del *Webster's Third New International Dictionary*:

- "Un concepto general formado por la extracción de características de ejemplos concretos".

- "El acto de dejar fuera de consideración una o más propiedades de un objeto complejo para atender otras".

La primera definición enfatiza el proceso de *generalización* para identificar la esencia del objeto, y la segunda definición enfatiza el proceso de *eliminación de los detalles* para enfocar la atención en determinadas propiedades del objeto.

Por ejemplo, abstraer de un ahuehuete (especie arbórea típica de México) el concepto general de "árbol", implica extraer la información esencial que lo caracteriza y que se puede aplicar para ser incluido dentro de la categoría general de los árboles, como el tiempo de vida (más de dos años), número de troncos (uno solo), material del tronco (leñoso) y ramificación (a cierta altura).

El conocimiento humano está dividido en diferentes áreas de estudio porque cada una se enfoca en un aspecto especifico de la realidad, cada disciplina tiene sus propias abstracciones. En ese sentido, la abstracción es un concepto clave en toda actividad humana y en cualquier área de estudio, como la matemática, la física, la biología o el arte.

El concepto de abstracción es una de las principales ideas del pensamiento computacional en el proceso de solución de problemas. El pensamiento computacional promueve el desarrollo de las habilidades de abstracción a través del análisis, el diseño y el modelado de soluciones por medio de algoritmos, programas, estructuras de datos abstractas, etc.

Para resolver un problema, normalmente definimos una abstracción con la que podemos resolverlo de acuerdo con nuestros intereses. A veces el nivel de abstracción que se define está restringido por las limitaciones que se tienen en nuestros sistemas. Un ejemplo claro en donde se usa la abstracción es en los video juegos.

En los video juegos cada elemento que se gráfica es una abstracción que el analista diseñó para representar determinados objetos. Los primeros video juegos que se desarrollaron tenían limitaciones importantes en las capacidades computacionales existentes. Por ejemplo, en el juego *Pong* se representaban dos barras que se podían mover hacia arriba o hacia abajo y un cuadro que se movía de izquierda a derecha o de derecha a izquierda. El objetivo era colocar la barra de manera que rebotara el cuadro para que se dirija al otro extremo. El jugador que menos veces dejaba pasar el cuadro ganaba. Con los avances en las computadoras y la introducción de tarjetas gráficas, los simuladores se han acercado cada vez más a escenarios realistas, con juegos más elaborados y en alta resolución (ver Figura 2.2).

Figura 2.2: En Tomb Raider™los objetos pertenecientes al mundo virtual (personajes, animales, plantas, flechas, etc.) son creados a partir de estructuras de datos.

2.1. Generalización

La generalización es el proceso de formular conceptos genéricos a través de la extracción de cualidades comunes de ejemplos concretos. Por ejemplo, en física, una de las abstracciones más trascendentes de la inteligencia humana es la *Ley de la gravitación universal de Newton*. La ley expresa que todo cuerpo atrae a todos los demás cuerpos con una fuerza que, para dos cuerpos cualesquiera, es directamente proporcional al producto de sus masas e inversamente proporcional al cuadrado de la distancia que los separa. La ecuación que representa la ley de la gravitación universal es: $F = G\frac{m_1 m_2}{d^2}$.

La ley de la gravitación universal nos permite constatar la aplicación de la generalización. La cualidad común es que todos los objetos se atraen de acuerdo

a dicha ley independientemente de la instancia concreta. Por ejemplo, la forma en que se atraen la tierra y una hormiga es la misma en que se atraen la tierra y la estrella Aldebarán (Aldebarán tiene un radio 44 veces mayor al del sol) ¡Es asombroso que todas las lunas, los planetas, las estrellas y las galaxias– se rijan por la ley de la gravitación universal!

De manera semejante, el concepto general de "árbol" se establece a partir de las cualidades comunes que lo caracterizan. Por ejemplo, las propiedades comunes de un ahuehuete, un fresno, un ocote y un eucalipto (ver Figura 2.3 [2]) son el tiempo de vida, el número de troncos y la altura a la que se ramifican. Un tejo de fuego (arbusto) no puede generalizarse al concepto de árbol porque se ramifica desde su base; es decir, no tiene altura y, por esta razón, tiene varios troncos.

Si conoces cómo funciona internamente el motor de un automóvil, entonces puedes generalizar los conceptos y reparar el motor de cualquier automóvil que mantenga la misma tecnología, desde un "bochito" hasta un Mastretta MXT. Has concebido el objeto en su esencia o categoría general a partir de objetos concretos. Ahora puedes solucionar problemas concretos que se ajusten al problema general. Por ejemplo, si no enciende un coche y no te da marcha, entonces revisas la batería. Si te da marcha, entonces revisas la cantidad de gasolina. El entender los principales conceptos de un motor de automóvil te puede servir para identificar o reparar fallas, incluso de otros vehículos que tengan principios de funcionamiento parecidos, como camiones o motocicletas.

[2] Recuperadas de `http://www.arboles.org/`

(a) Ahuehuete

(b) Fresno

(c) Ocote

(d) Eucalipto

Figura 2.3: Estos árboles son representativos de la flora de México y los puedes apreciar en el campus de la Universidad Nacional Autónoma de México (excepto el ahuehuete).

2.2. Eliminación de los detalles

La eliminación de los detalles es el proceso de dejar fuera de consideración una o más propiedades de un objeto con la finalidad de enfocarse sólo en algunas propiedades. Es decir, únicamente se capturan las propiedades que son relevantes para un determinado problema o área de estudio.

En función de lo que se pretende representar, la abstracción indica qué debe ser considerado relevante, qué detalles deben eliminarse, cuál es el núcleo o esencia y hasta qué punto debe simplificarse una representación (por ejemplo, el modelo de un fenómeno natural). Dado que la aplicación de la abstracción en un determinado problema puede ser engañosa, el beneficio y valor de una abstracción particular dependen de su finalidad.

Imagina que quieres comprender el movimiento de una manzana cayendo de un árbol. Si tu atención se centra en la densidad del aire, la temperatura ambiente o los fotones que colisionan contra la manzana, entonces no te sería posible deducir qué propiedades son esenciales en el movimiento de la manzana. Las propiedades mencionadas anteriormente influyen en el movimiento de la manzana; sin embargo, descubrir la influencia de cada detalle sobre el fenómeno observado es complejo y en la mayoría de los casos despreciable, es decir, su influencia es insignificante.

La ecuación que representa la ley de la gravitación universal $F = G\frac{m_1 m_2}{d^2}$ nos permite constatar la eliminación de los detalles. Por un lado, solo se enfoca en la masa de los objetos, la distancia que los separa y la gravedad. Por otro lado, infinidad de propiedades que también intervienen en el fenómeno del movimiento en caída libre se han descartado, como la fricción o resistencia ejercida por el aire.

Imagina que debes realizar un informe acerca del estatus social de una persona. Las propiedades razonables a incluir son: nombre, edad, salario y ocupa-

ción. Ahora imagina que debes realizar un informe médico. Las propiedades razonables a incluir son: nombre, edad, peso, tipo de sangre y nivel de glucosa. En cualquier caso, el "objeto" persona cuenta con una infinidad de propiedades (altura, raza, color de pelo, nacionalidad, nivel de educación, etc.). Mediante el proceso de abstracción excluimos todas aquellas que no tiene cabida en nuestro análisis. La abstracción es precisamente la capacidad de aislar los detalles que son relevantes para el problema en cuestión.

2.3. Niveles de abstracción

Un nivel de abstracción se refiera a *el grado de detalle con el que se especifica una representación. Una representación en un alto nivel de abstracción especifica menos detalles que una representación en un bajo nivel de abstracción*. Jeannette Wing resalta como una habilidad esencial del pensamiento computacional el poder pensar en múltiples niveles de abstracción.

Los seres humanos pensamos en términos de objetos. La habilidad de la abstracción nos permite ver una imagen en nuestro celular y reconocer personas, automóviles o árboles, en lugar de sólo puntos coloreados (píxeles). Los seres humanos aprenden a categorizar objetos analizando sus propiedades y sus comportamientos comunes. Un ejemplo natural del uso de los niveles de abstracción se da en la elaboración de un árbol taxonómico, Considera el siguiente cuento:

Estaba Tukkul leyendo con mucha atención el cuento "Martín el zapatero" de León Tolstói, cuando Paat le llama por teléfono:

— ¡Oye Tukkul! ¿Me enteré que compraste una nueva mascota?

— Así es.

— ¿Y qué mascota es?

— !Adivina!

— Mmm... ¿Corre o nada?

— Corre.
— ¿Maúlla o ladra?
— Ladra.
— ¡Es un perro, me encantan los perros! ¿Y qué raza es?
— ¡Adivina!
— Mmm... ¿Es pastor o sabueso?
— No.
— ¿Es Terrier o Labrador?
— No.
— ¡Ah caray, debe ser un perro poco común! ¿Es muy mexicano y no tiene pelo?
—¡Sí!
— ¡Es un Xoloitzcuintle! ¿Cómo se llama?
— Tukkul Junior.
— ¡No, hombre, todo un genio! Hay talento, lo que falta es apoyarlo.

Lo que Paat hizo para poder concluir cuál era la mascota de Tukkul era usar diferentes *niveles de abstracción* desde el objeto general al objeto particular (ver Figura 2.4).

2.4. Modelos

La abstracción es una habilidad esencial para la construcción de modelos y la descomposición de problemas. *Un modelo es una representación abstracta (matemática, declarativa, visual, etc.) de fenómenos, sistemas o procesos*. El ser humano crea modelos que representan la esencia de determinados fenómenos a fin de analizarlos y comprenderlos.

Por ejemplo, el modelo de la gravitación universal (ley de la gravitación universal de Newton) es representado de manera declarativa y matemática como sigue:

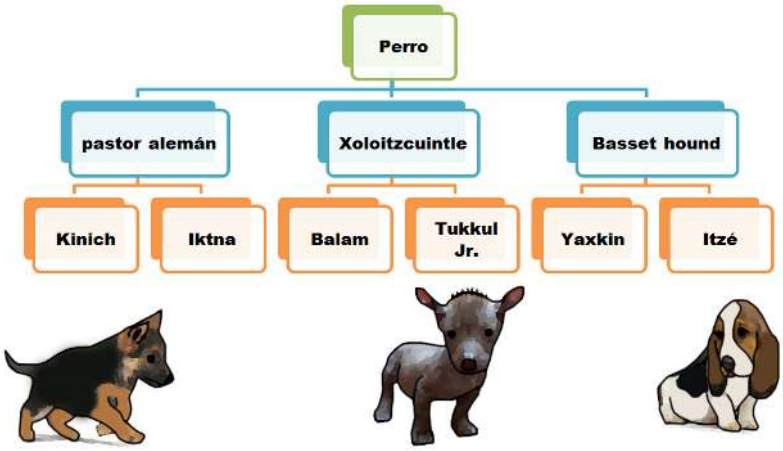


Figura 2.4: Árbol taxonómico de algunos perros.

- Modelo declarativo: *todo cuerpo atrae a todos los demás cuerpos con una fuerza que, para dos cuerpos cualesquiera, es directamente proporcional al producto de sus masas e inversamente proporcional al cuadrado de la distancia que los separa.*

- Modelo matemático: $F = G\frac{m_1 m_2}{d^2}$

El éxito del modelo de la gravitación universal de Newton originó la época de la Ilustración, al demostrar que si se observa y se razona, la humanidad podía descubrir la esencia de la naturaleza física. La formulación de esta regla sencilla es una de las razones principales de los éxitos científicos, porque brindo la confianza para describir también otros fenómenos del mundo mediante modelos.

El modelo de la gravitación universal de Newton ha permitido comprender y explicar el comportamiento del sistema solar. Por ejemplo, el planeta Urano fue descubierto en la década de 1840. En ese entonces los científicos no podían

explicar las desviaciones de la órbita de Urano considerando las perturbaciones ocasionadas por todos los demás planetas conocidos.

Por lo tanto, o el modelo de la gravitación universal fallaba a esa gran distancia del sol o había un octavo planeta desconocido que perturbaba la órbita de Urano. Los astrónomos Adams y Le Verrier, basados en el modelo de Newton, calcularon de manera independiente dónde debería estar el octavo planeta, sugiriendo que se debería buscar un nuevo planeta en determinada zona del cielo ¡La misma noche que se recibieron los cálculos en el observatorio de Berlín fue descubierto el planeta Neptuno!

Newton calculó la rapidez para tener un objeto en órbita circular y, como en aquel tiempo (siglo XVII) era claramente imposible alcanzar esa velocidad inicial, no previó que los seres humanos lanzarían satélites. No obstante, el modelo de la gravitación universal que Newton desarrolló con base en sus abstracciones del movimiento permitió al hombre llegar a la Luna.

A pesar de que el modelo de Newton provee una muy buena aproximación al movimiento de los planetas, no considera ciertos aspectos, por lo

Si bien la ley de la gravitación universal de Newton da una muy buena aproximación para describir el movimiento de un planeta alrededor del Sol, durante el siglo XIX se observaron algunos problemas que no se conseguían resolver. En especial, se encontraba la órbita del planeta Mercurio, la cual en lugar de ser una elipse cerrada, tal y como predecía la teoría de Newton, es una elipse que en cada órbita va rotando, de tal forma que el punto más cercano al Sol (el perihelio) se desplaza ligeramente, unos 43 segundos de arco por siglo. Dicho problema no pudo resolverse, hasta la llegada de la Relatividad General de Einstein. Este es un claro ejemplo de los niveles de abstracción; el modelo de Newton predice el

movimiento de los planetas a cierto nivel; mientras el de Einstein va a un nivel más preciso[3].

2.5. Reconocimiento de patrones

Todos los días reconocemos objetos, fotografías, personas y lugares. Siempre estamos rodeados de una gran cantidad de información que tenemos que diferenciar, asociar y clasificar. Este proceso, tan normal y cotidiano para nosotros, es complicado cuando se quiere que una computadora lo haga. El área de reconocimiento de patrones concentra algoritmos, metodologías, teorías y sistemas que permiten diferenciar, asociar y clasificar datos de forma automatizada. En la vida diaria ya convivimos con tecnología que reconoce nuestra voz, nuestra cara o nuestra huella digital. Además, también se aplica el reconocimiento de patrones cuando buscamos en la red imágenes e información.

Todos estos procesos tienen como base la abstracción, porque se extrae las características más importantes que permiten hacer una comparación, clasificación o inclusive una reconstrucción. Este fue el proceso que usó Patt para reconocer la mascota de Tukkul. Un proceso similar es cuando jugamos a adivinanzas. Tal vez recuerdas algunas que te hacían de niño:

- Se lleva tu sombrero pero no lo ves. ¿Qué es?
- ¿Cuál es el animal que más tarda en quitarse los zapatos?

Ahora te reto, a ver si adivinas las siguientes:

- El cielo y la tierra se van a juntar; la ola y la nube se van a enredar. Vayas donde vayas siempre lo verás, por mucho que andes nunca llegarás.

[3]Ver https://es.wikipedia.org/wiki/Ley_de_gravitación_universal.

- Largo larguero, Martín Caballero, sin brazos ni patas y corre ligero. ¿Qué es?

El proceso interno que realizas es abstraer características que te dice la adivinanza para poder acertar. Seguramente has jugado juegos de adivinar que tienen una lógica similar. Por ejemplo, ¿puedes deducir qué frutas son a partir de las siguiente tabla de características?

¿Qué fruta es?	Forma	Color	Textura
	Curva	Amarillo	lisa
	Cilíndrica	Amarillo	Escamosa
	Ovalada	Verde o roja	Lisa
	Cilíndrica	Verde, amarillo o roja	Lisa

Tabla 2.1: Tabla de características de algunas frutas.

Este sencillo ejemplo te muestra como se podría reconocer frutas en una imagen, con una computadora. Claro que el reto es cómo extraer las características de la imagen, pero esa es otra historia.

Actualmente existen muchas aplicaciones que utilizan reconocimiento de patrones como selección automática de personal, diagnóstico de enfermedades, detección de hábitos de consumo, sistemas de seguridad a partir de cámaras de video, reconocimiento de firmas y de texto, sistemas biométricos, controles de calidad, etc.

2.6. Desarrollando la habilidad de abstraer

La abstracción es una habilidad inherente a todos los seres humanos. No obstante, cada individuo abstrae de forma única debido a que se desarrolla de manera diferente. El psicólogo Jean Piaget, mediante extensos estudios aplicados en niños recién nacidos y personas adultas, derivó cuatro periodos de desarrollo cognitivo: sensorio-motor, preoperatorio, operaciones concretas y operaciones formales. La etapa de las operaciones formales, que se encuentra alrededor de los doce años a la edad adulta, es donde las personas son capaces de pensar de manera abstracta.

La abstracción es fundamental para la ciencia e ingeniería en general, juega un papel crítico en la creación de teorías, modelos, análisis y producción de dispositivos de ingeniería. El pensamiento computacional identifica a la abstracción como una de las grandes ideas de las ciencias computacionales y que las habilidades de abstracción son cruciales para el futuro en el desarrollo científico y tecnológico.

Sin embargo, los programas de estudio de las carreras de ingeniería o ciencias no contienen cursos sobre abstracción, no obstante, todo depende de la habilidad de abstracción para resolver problemas. Por esta razón, la abstracción es una habilidad esencial que se desarrolla indirectamente a través de otros tópicos, como cursos de matemáticas, de programación o ingeniería del *software*.

Las matemáticas son una excelente herramienta para la enseñanza y desarrollo del pensamiento abstracto. Las habilidades de abstracción se pueden mejorar mediante la presentación del formalismo matemático de una manera orientada a los problemas. El investigador Devlin comenta: “La principal ventaja de aprender y hacer matemáticas no es el contenido específico, sino el hecho de que se

desarrolle la capacidad de razonar acerca de estructuras abstractas definidas formalmente con precisión y analíticamente".

La abstracción puede incluir cambios importantes en la formulación de un problema, pero que facilitan su solución o que nos llevan a soluciones novedosas. Por ejemplo, Tukkul decidió jugar un juego sencillo de aritmética con Paat. El juego es el siguiente: hay dos jugadores que tienen que seleccionar alternadamente un dígito entre el 1 y el 9 en tres turnos. Una vez elegido un digito no se puede volver a seleccionar el mismo. El jugador que logra que sus dígitos sumen 15 al tercer intento, gana.

Paat se quedó pensativa un tiempo, tomó papel y lápiz y escribió algo. Después le dijo a Tukkul que estaba lista para jugar. Tukkul había jugado antes y pensó que sería fácil ganarle a Paat. Sin embargo, la mayor parte del tiempo Tukkul perdía o ninguno lograba tener tres dígitos que sumaran 15. Cuando Tukkul le preguntó a Paat qué tenía en el papel que consultaba todo el tiempo durante el juego, Paat le enseñó un *cuadro mágico*. Un cuadro mágico es aquel en donde la suma de todas las columnas, filas y diagonales es lo mismo (ver Figura 2.5).

Paat le explicó que cada vez que él seleccionaba un número le ponía una cruz al cuadro mágico, y cada vez que Paat seleccionaba un número le ponía un círculo al cuadro mágico. Para ganar, le dijo, todo lo que tenía que hacer era jugar "gato" en el *cuadro mágico*. Lo que hizo Paat fue abstraer lo más importante del juego: seleccionar 3 dígitos diferentes que sumaran 15 y pensar en cómo representarlo para que fuera más fácil jugar.

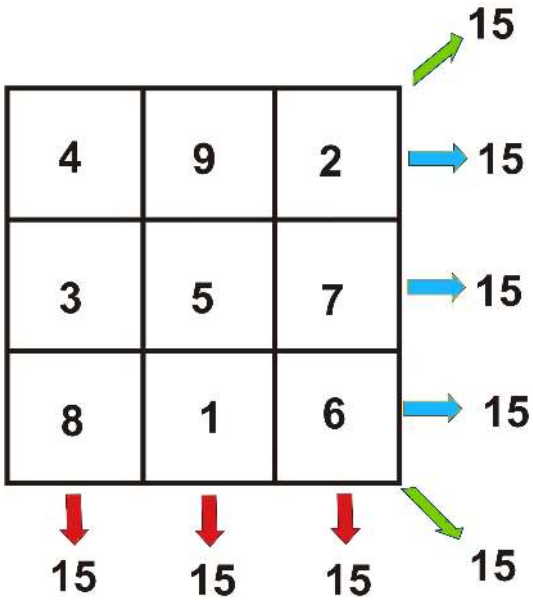


Figura 2.5: Cuadro Mágico de 3×3 que suma 15 para cada fila, columna y las diagonales.

2.7. Aplicando la abstracción

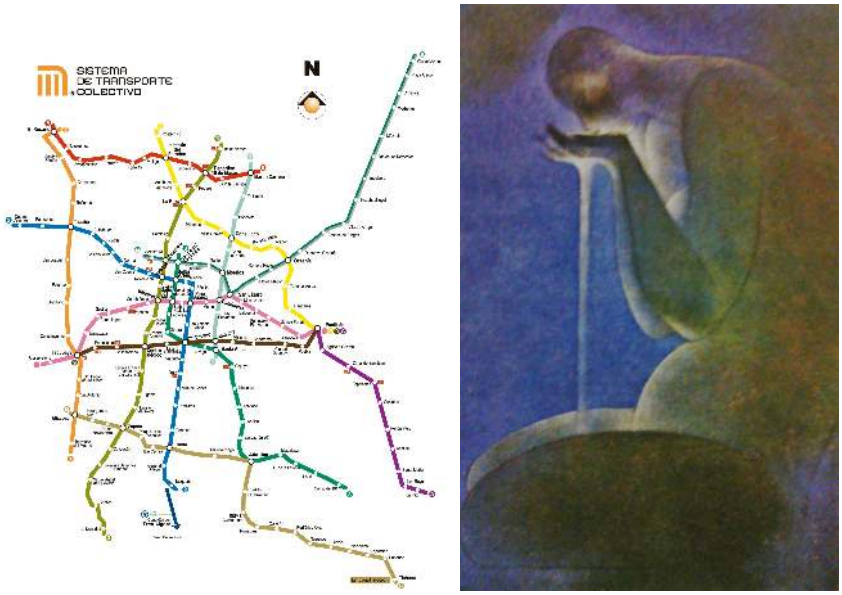
La marcha del progreso en el mundo se acerca cada vez más hacia el desarrollo de la habilidad de abstraer, ya sea investigando nuevas soluciones en diferentes áreas de estudio, creando obras de arte, resolviendo problemas en la vida cotidiana o usando la tecnología.

La habilidad de abstraer se aplica en cualquier actividad de nuestra vida. Por ejemplo, al decidir qué criterios son los más importantes para comprar determinados productos o contratar ciertos servicios. Al escribir un ensayo o preparar una presentación, seleccionamos qué ideas capturan la esencia del tema que deseamos exponer o argumentar. Cuando resolvemos un problema matemático o escribimos un programa por computadora, debemos pensar qué estructuras

matemáticas o computacionales debemos utilizar para representarlo y resolverlo.

Una aplicación de la abstracción es la mostrada por un mapa. La aportación de los mapas es que resaltan únicamente la esencia de la información de interés y se eliminan detalles innecesarios, como árboles, edificios, automóviles, entre muchos otros elementos del entorno. El arte también es un ejemplo de la utilidad de la aplicación de la abstracción. El pintor mexicano Ricardo Martínez representa la esencia de su obra "Mujer con agua" con una pintura que solo tiene líneas sobrias y colores diferentes en su composición; eliminando todo detalle como ojos, nariz y cabello. Al observar la pintura, nuestra mente puede abstraer una mujer tomando agua (ver Figura 2.6).

Otro ejemplo de la utilización de la abstracción en el arte se manifiesta en la literatura, donde el escritor que está redactando una novela comienza con un conjunto de ideas. Posteriormente selecciona las ideas esenciales de su hipótesis, articula qué planea argumentar, hace un bosquejo y refina su trabajo.



(a) Mapa del metro CDMX (b) Mujer con agua

Figura 2.6: Aplicación de la abstracción: a) El mapa permite adquirir la información de manera rápida y simple de las líneas del metro de la Ciudad de México; b) En la obra de Ricardo Martínez "Mujer con agua", tu mente puede abstraer una mujer tomando agua, incluso es probable que te trasmita feminidad, pasividad y fecundidad.

2.8. Ejercicios

1. Imagina que eres un desarrollador de software y tienes que crear un personaje de un videojuego de pelea. ¿Qué propiedades y comportamientos de tu luchador serían relevantes?

2. Diseña un cubo mágico diferente de 3×3. Puedes construir uno de 4×4. ¿Que modificaciones tendrías que hacer en el juego?

3. Tenemos que pronosticar qué tan lejos puede viajar una pelota si se le aplica una fuerza determinada. Si la prueba es en el aire y en el agua ¿qué elementos tenemos que considerar y porqué? ¿Qué pasa si la prueba es en el espacio?

4. Observa la Figura 2.7. Crea una taxonomía de 3 niveles, indicando en cada nivel qué elementos estás considerando. ¿Podrías crear otra taxonomía? ¿Para qué tipo de preguntas te podría servir cada una de ellas?

Figura 2.7: Crea una taxonomía.

5. Selecciona dos objetos e inventa tu propia adivinanza para cada uno.

6. Selecciona varios animales y has tu tabla de características para que puedan ser reconocidos por alguien más.

7. Lee el siguiente cuento, reflexiona sobre el mensaje principal, y determina un título apropiado.

En un día equinoccial, Tukkul se encontraba en Chichén Itzá para observar la descendencia de Kukulcán sobre las escalinatas de la imponente Pirámide de Kukulkán. Después de observar dicho acontecimiento se sintió tan feliz como en aquellos días de su infancia, en los que si le ofrecían un agua de coco, prefería un mango enchilado. En su camino de regreso a su finca, notó un peculiar olor a pescado y siguió el rastro hasta encontrar una modesta pescadería. Al entrar, para su sorpresa, vio a Makool, un amigo de la infancia. Makool se alegró de ver a Tukkul y le pidió que pasara; le contó que ese día había iniciado su modesto changarro de venta de pescados y que deseaba poner un cartel afuera del negocio, pero que no tenía idea de qué escribir. Entonces, le dijo a Tukkul, a ti que te gusta leer tanto y esas cosas ¿Qué anuncio podría escribir?

Tukkul se rascó la cabeza y dijo, pues quizá: "AQUÍ SE VENDE PESCADO FRESCO". Makool le entregó el cartel y un plumón a Tukkul para que escribiera la oración y colocara el cartel. Pasó un vecino y le dijo -es obvio que ese "AQUÍ"no hace falta escribirlo- Makool soltó una carcajada y Tukkul se puso colorado y borró el AQUÍ. El anuncio quedó como "SE VENDE PESCADO FRESCO". Pasó otro vecino y le dijo -es innecesario escribir "SE VENDE", ¿o acaso regala usted el pescado"- Makool soltó otra carcajada hasta el punto que le brotaron las lágrimas, Tukkul dañado en su orgullo, borró el SE VENDE y sólo quedó "PESCADO FRESCO". Al poco rato pasó otro vecino y dijo -¿acaso cree que se podría vender pescado podrido y por eso escribió "FRESCO"?- Makool empezó a rodar por el piso, en un espectáculo verdaderamente incómodo. Tukkul dio un suspiro y borró FRESCO del anuncio, dejando sólo la palabra "PESCADO".

Después de un rato, Makool le dijo a Tukkul que tenía que ir a atender un asunto urgente, que volvería pronto y que le hiciera el favor de atender su negocio mientras regresaba. Aún después de los incómodos hechos ocurridos, Tukkul mantenía su buen talante al recordar el evento en Chichén Itzá, por lo que asintió amistosamente. Transcurrió el tiempo hasta caer la noche y otro vecino pasó y le dijo -¿por qué escribe "PESCADO"? ¿acaso alguien dudaría de que se vende otra cosa que pescado, con el olor que sale de aquí?- Tukkul reflexionó y se dijo a sí mismo, eliminar palabras superfluas, si se excede, se llegará al silencio y Tukkul sólo guardo silencio.

Tukkul se preguntaba dónde estaba su amigo, se puso furioso cuando Makool regresó embriagado, había bebido una buena cantidad de pulque. Tukkul cerró el negocio y

recostó a Makool en una hamaca que encontró entre las desordenadas cosas que habían en el negocio. Al ver a su amigo ebrio y durmiendo, pensó en como su día empeoró y tuvo un sentimiento de desagrado por Makool. Sin embargo, al momento recordó la infancia de Makool, que no tuvo la oportunidad de ir a la escuela por las condiciones de pobreza en que creció o cuando su abuela le puso una paliza porque robó un pan al tener hambre. Recordó que la gente del pueblo decía que Makool era un perdedor, que sólo sabía contar chistes estúpidos. Sin embargo, a Tukkul no le parecía de esa manera, él sabia su historia y se prometió que siempre lo ayudaría cuando estuviera en sus manos. Las lágrimas le brotaron y surgió en él un pensamiento al cual se aferró: "*es mejor enseñar a pescar que regalar un pescado*". Recordó sus deberes en la finca, le echó una rápida ojeada a su amigo y salió a prisa, sin dejar más palabras que el silencio.

8. Representa mediante un dibujo a una persona a diferentes niveles de abstracción, considera al menos tres niveles.

9. Considera que vas a hacer un pastel. Lista los ingredientes requeridos al nivel apropiado de abstracción, de forma que no falte ni sobre informaciń. Verifica que el nivel es el adecuado con un cocinero de pasteles.

10. Los elementos importantes para describir un objeto depende de para qué aplicación se requieren. Especifica los elementos relevantes para describir un automóvil para: (a) un vendedor de autos, (b) un corredor de autos.

Capítulo 3

Información

"Lo que embellece al desierto es que en alguna parte esconde un pozo de agua."
-Antoine de Saint-Exupéry

3.1. Introducción

Actualmente, vivimos inmersos en la era de la información. Las computadoras no cesan de producir cambios en la manera de trabajar de cada individuo y de las organizaciones debido a su capacidad de transformar *datos* en *información*. Los sistemas basados en computadoras son de uso cotidiano para la creación, el almacenamiento y la transferencia de información. Además, gracias a la *Web*, millones de personas tienen acceso fácil e inmediato a una cantidad extensa y diversa de información.

La información es uno de los recursos más valiosos e importantes en cualquier actividad de nuestra vida diaria o en el quehacer profesional. Sin embargo, es común que este término se confunda con el de datos.

Los datos son la representación de realidades concretas en su estado primario. Por ejemplo, el peso de un vehículo o el nombre de un empleado. Para representar esas realidades es posible usar varios tipos de datos como son: números, letras, imágenes, etc. Los datos no tienen capacidad de comunicar un significado por sí mismos.

La utilización de los datos depende del lenguaje de programación que utilices para desarrollar tu aplicación y del *tipo de dato* adecuado para almacenar determinada característica. En la mayoría de los lenguajes de programación hay tipos de datos primitivos, a partir de los cuales se definen las variables a utilizar. Los datos primitivos se clasifican usualmente en tipos de datos: numéricos, lógicos y carácter (ver Figura 3.1).

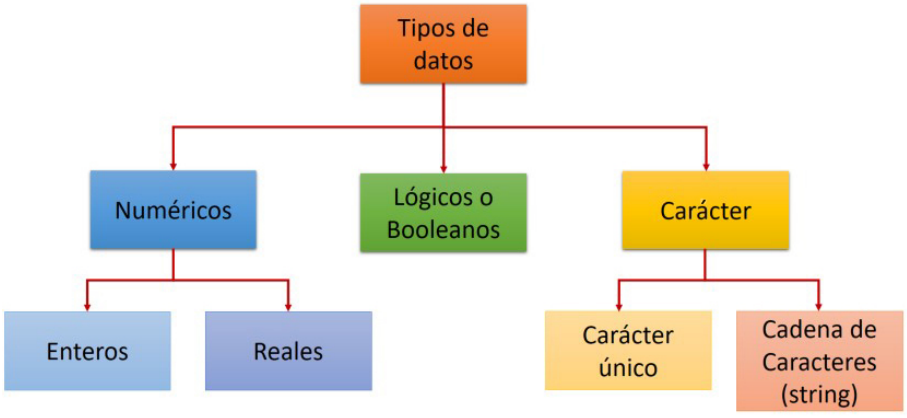


Figura 3.1: Tipos de datos primitivos que soportan algunos lenguajes de programación de una computadora.

Datos numéricos enteros.- Almacenan números sin punto decimal y tienen valores positivos, negativos o el cero. Por ejemplo: La edad de una persona, el número de árboles en una granja o el número de artículos en una tienda.

Datos numéricos reales.- Almacenan números que tienen parte entera y decimal. Por ejemplo, una aproximación del número π. La parte entera es 3 y la parte decimal es .151492. Otros ejemplos de datos numéricos reales son: la presión arterial de una persona, la distancia a la luna o los kilómetros recorridos por una moto.

Datos lógicos o booleanos.- Almacenan valores lógicos que resultan de comparar otros valores entre sí. Una comparación devuelve un resultado lógico (`verdadero` o `falso`). Por ejemplo, "La edad de Juan es mayor que la de Elvira", "La calificación de Enrique es mayor que 9" o "El número de programadores para un videojuego es menor que 10".

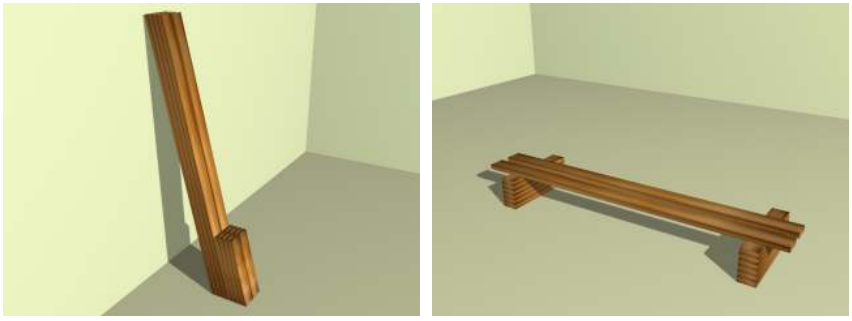
Datos de carácter único.- Almacena un símbolo asociado a un dígito, una letra u otros signos. Por ejemplo: "0", "7", "A", "m", "#", "$" o " %".

Datos de cadena de caracteres o String.- Almacenan una cadena de caracteres. Por ejemplo, el nombre de una persona, su dirección o el nombre de los objetos de un catálogo.

Basados en estos tipos de datos primitivos, se pueden obtener otros tipos de datos compuestos como las estructuras de datos que asocian datos particulares de un objeto. También se pueden obtener representaciones más complejas, tales como voz, música, gráficas, dibujos, fotografías o videos. Además, representaciones de señales físicas como el valor de la contaminación de una ciudad obtenidas por un sensor, o bien representaciones como las que se describen en la sección 3.5 de este capítulo.

Para que los datos puedan ser útiles deben organizarse y relacionarse en forma significativa dentro de un contexto para transformarse en información. *La información es un conjunto de datos organizados de tal modo que adquieren un valor adicional más allá del propio.*

Piensa en los datos como tablones de madera, en cuya condición tienen escaso valor más allá del que inherentemente tienen como objetos concretos (ver Figura 5.2(a)). Mientras que si se define algún tipo de relación entre los tablones adquirirán un valor adicional. Por ejemplo, si los tablones se apilan de cierta manera pueden formar un banco (ver Figura 5.2(b)). El tipo de información creada depende de las relaciones definidas entre los datos existentes.

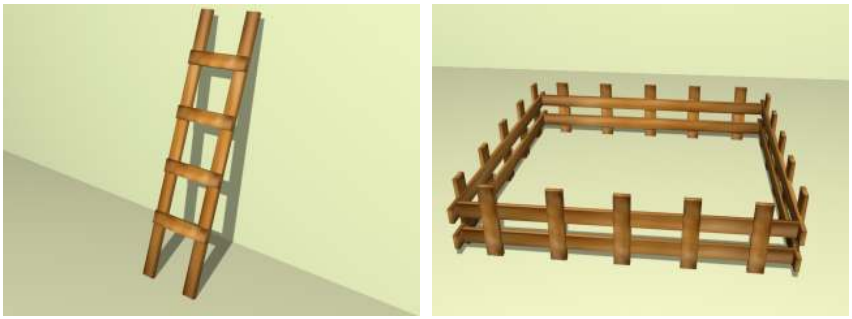


(a) Tablas de madera (b) Banco

Figura 3.2: Las relaciones entre los datos (tablones en el ejemplo) generan información.

Además, la adición de datos nuevos o diferentes significa la posibilidad de redefinir las relaciones y de crear nueva información. Imagina que además de los tablones de madera dispusiéramos de clavos, entonces podríamos incrementar el valor del producto final al construir una escalera o un corral (ver figuras 3.3(a) y 3.3(b)).

En forma similar, pueden establecerse reglas y relaciones para organizar los datos a fin de que provean útil y valiosa información. Por ejemplo, un administrador podría considerar, más acorde con su propósito (es decir, más valioso) conocer las ventas mensuales totales que la cantidad de ventas de cada vendedor individual.



(a) Escalera (b) Corral

Figura 3.3: Las relaciones entre los datos existentes y la adición de datos nuevos le proporciona valor agregado a la información.

La información constituye un mensaje que cambia el estado de *conocimiento* del sujeto o sistema que recibe dicho mensaje. *El conocimiento es la apreciación y comprensión de un conjunto de información y de la utilidad que puede extraerse de ella en beneficio de una tarea específica*. La información es la materia prima fundamental para tomar decisiones porque aumenta el conocimiento.

Por ejemplo, el comprender que mediante la escalera podemos cumplir con la tarea de subir a lugares más altos, es el conocimiento extraído al apreciar que la manera en que fueron dispuestas las tablas nos permite escalar.

Uno de los tópicos principales del pensamiento computacional es la información porque una persona informada, en cualquier ámbito de su vida cotidiana, tiene mayor certeza de desarrollar la solución adecuada a un determinado problema y de tomar decisiones con mayor certidumbre entre un conjunto de alternativas posibles. Por ejemplo, los inversionistas utilizan *sistemas de información* (un conjunto de elementos orientados al tratamiento y administración de datos e información generados para cubrir un objetivo) para tomar decisiones

en las que están en juego miles de millones de pesos o las compañías manufactureras los utilizan para hacer pedidos de suministros y distribuir bienes con mayor rapidez.

3.2. Transformación de datos en información

La transformación de datos en información es un proceso o serie de tareas lógicamente relacionadas entre sí y ejecutadas con el fin de producir un resultado definido. El proceso para definir relaciones entre datos requiere de conocimiento. Por ejemplo, parte del conocimiento necesario para hacer una escalera se fundamenta en la comprensión del hecho de que los peldaños deben colocarse horizontalmente y los postes en forma vertical.

El acto de seleccionar o rechazar datos de acuerdo con lo conveniente que sean para tareas específicas también se basa en el conocimiento empleado en el proceso de conversión de datos en información. En consecuencia, la información puede considerarse como datos provisto de mayor utilidad mediante la aplicación del conocimiento (ver Figura 3.4).

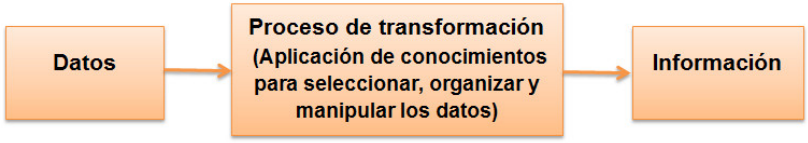


Figura 3.4: Proceso de transformación de datos en información.

3.3. Características de la información

La información debe tener ciertas características para sea valiosa. Si la información no es exacta ni completa, se corre el riesgo de tomar decisiones desacertadas. Por ejemplo, un pronóstico inexacto de demanda futura de un producto que indique un considerable aumento en las ventas; si dado el momento ocurre lo contrario, podría exponer a una organización a invertir en el desarrollo de un producto innecesario.

Además, si la información no es pertinente para la situación tratada, no se hace llegar en forma oportuna o es tan compleja que resulta incomprensible, cabe la posibilidad de que su valor sea mínimo. Las características que la información debe tener son las siguientes:

- *Exacta*: La información exacta carece de errores. La información inexacta se genera cuando se insertan datos inexactos en el proceso de trasformación.

- *Completa*: La información completa contiene todos los datos necesarios. Por ejemplo, un informe de inversión que no incluyera todos los costos estaría incompleto.

- *Simple*: La información debe enfocarse en lo esencial. Un exceso de información hace que la información sea compleja e difícil identificar lo verdaderamente importante.

- *Confiable*: La información confiable tiene un sólido porcentaje de seguridad respecto a su veracidad. Depende de diversos factores como el método de recolección de datos o la fuente de información. Un rumor de fuente

anónima sobre la posibilidad de incremento en los precios del petróleo no sería confiable.

- *Verificable*: La información debe comprobarse si es correcta, quizá mediante la consulta de diversas fuentes al respecto.
- *Oportuna*: La información oportuna es la que se recibe en el momento adecuado o conveniente. Conocer las condiciones climáticas de la semana anterior no servirá de nada para decidir el atuendo de hoy.
- *Segura*: La información debe estar protegida contra el acceso a usuarios no autorizados.
- *Accesible*: La información debe ser accesible para los usuarios autorizados en el formato adecuado y en el momento correcto.
- *Económica*: La información debe tener un costo relativamente bajo. El costo de producir la información debe ser evaluado frente al valor que provee al usuario.
- *Pertinente*: La información pertinente es adecuada en un contexto determinado. Información acerca de la reducción del precio del limón no sería pertinente para una empresa que fabrica automóviles.

La información útil puede variar según el valor de cada uno de estos atributos. Por ejemplo, si se trata de datos de investigación de mercado es aceptable cierto grado de inexactitud y parcialidad, pero la oportunidad es esencial. La investigación de mercado podría alertar sobre una inminente reducción sustancial de los precios de los competidores. El conocimiento de los detalles y del momento preciso en que habrá de ocurrir esa reducción de precios quizá no sea tan importante como el hecho de que se nos advierta con la anticipación suficiente para

planear una respuesta. Por el contrario, la exactitud de la información utilizada para colocar un satélite en órbita es esencial.

3.4. Teoría de la información

La Teoría de la Información formaliza el concepto de información, estableciendo las bases para las comunicaciones, la compresión de datos; la detección y corrección de errores; y la criptografía. Por estas razones, la información es uno de los conceptos centrales de la computación, y en general, prácticamente de cualquier área científica.

La Teoría de la Información surge del área de la comunicaciones, en particular de la pregunta, ¿Cuánta *información* contiene un mensaje? Intuitivamente, si envío cierto mensaje, mientras menos probable sea el contenido del mensaje contiene mayor información, y viceversa. Por ejemplo, si el mensaje es sobre el clima en la Ciudad de México, y dice que estará soleado, este da poca información, ya que es común que este soleado; en cambio si el mensaje dice que va a nevar, contiene mucha información, ya que es muy raro que nieve en la Ciudad de México.

Basado en este principio, Shannon estableció una definición matemática para información:

$$I(m) = log(1/P(m))$$

Donde $I(m)$ es la medida de información del mensaje m y $P(m)$ es la probabilidad de m. La expresión *log* es el logaritmo, que normalmente se utiliza en base 2, de forma que la información se mida en "bits"(más adelante en este capítulo se define lo que es un bit). Básicamente nos dice que hay una relación

inversa entre información y probabilidad. Si la probabilidad es 1, implica que m es algo seguro, entonces la información es 0; en cambio si la probabilidad es muy baja, tendiendo a cero, la información tiende a infinito.

Supongamos que cierta estación meteorológica envía un mensaje diariamente sobre la predicción del clima para el día siguiente. Si el mensaje dice *soleado* y la probabilidad de soleado en ese lugar es de 50 %, entonces el valor de información del mensaje es 1 ($log_2(1/0.5)$); en cambio si dice *lloviendo* y su probabilidad es 1/8, entonces la información es 3. Si la estación envía mensajes durante todos los días de año, ¿Cuál es la información promedio que dan los mensajes?

La información promedio se conoce como *entropía*, y se calcula como el valor esperado o promedio ponderado de la información de cada mensaje:

$$H = E(I) = \sum_i P_i log_2(1/P_i)$$

Donde i representa todos los posibles mensajes, y $P(i)$, la probabilidad de cada mensaje.

El concepto de entropía es muy importante, ya que establece un límite teórico a la compresión de información, que analizaremos más adelante en este capítulo. Es decir, si queremos comprimir cierta información, la entropía o información promedio, establece el mínimo número de "bits" necesarios, así que el mejor algoritmo de compresión no puede lograr menos bits en promedio.

3.5. Representación

A lo largo de la historia, el hombre ha perfeccionando las formas de representar pensamientos y sentimientos para poder expresarlos e interpretarlos de manera más efectiva. En un principio se basó en la denominación de objetos y

después fue creando su propio sistema de signos. Debido a la invención del lenguaje el hombre pudo interpretar el mundo que lo rodea en un mayor nivel de abstracción y comunicarse con otros seres humanos. Las primeras sociedades se toparon muy pronto con el problema de determinar cuántos elementos formaban una colección de cosas o cuál de dos conjuntos de entidades era mayor que otro. Por esta razón, el hombre diseñó una representación del número para dar solución a dicho problema.

El hombre ha creado diversos lenguajes y sistemas de numeración con distintas representaciones. Por ejemplo, los mayas crearon un sistema de numeración en base 20, fueron de las pocas civilizaciones que inventaron el concepto del cero, y desarrollaron un sistema de escritura de los más hermosos que han existido (ver Figura 8.2 para darte una idea de como escribían los mayas [1]) .

El sistema de numeración decimal se usa habitualmente en el mundo y está representado por un conjunto de 10 símbolos que son: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. El sistema binario se usa comúnmente en las ciencias de la computación y está compuesto de dos dígitos como base, el 0 y 1. Por esta razón, se se denomina sistema binario.

El sistema binario se utiliza para representar la información en las computadoras porque es más sencillo de implementarlo en su hardware. Cada 0 o 1 se llama "bit" (la palabra bit es la contracción de *binary digit*) y se implementa en la memoria principal de una computadora por una celda con el nivel de voltaje encendido o apagado. En los discos duros los bits son representados por la dirección de un campo magnético sobre una superficie revestida. En los discos compactos la parte de la superficie que corresponde a un bit puede reflejar (una cresta) o no reflejar la luz (un valle).

[1] Recuperada de Fundación para el Avance de los Estudios Mesoamericanos

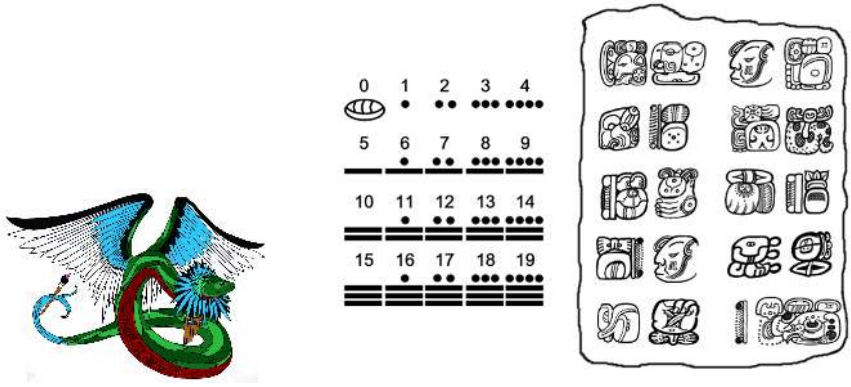


Figura 3.5: -Kukulkán- Los mayas fueron grandes astrónomos, artistas, arquitectos y matemáticos. A la izquierda se muestra su sistema de numeración en base 20 y a la derecha su sistema de escritura basado en glifos. El mensaje contenido en la estela maya (piedra tallada) es el siguiente: "Me llamo Alan. Soy estudiante y jugador de pelota. Mi madre se llama María. Ella trabaja en tejidos y es de Yaxchilán. Mi padre se llama Tomás. Él es agricultor y sabio. Él es de Palenque".

Un bit por sí mismo no puede representar demasiado por lo que usualmente se juntan en grupos para representar números, texto, imágenes, música, videos, video juegos, etc. ¡Es sorprendente que toda la amplia variedad de entidades almacenadas en una computadora sólo son secuencias de ceros y unos llamados bits!

3.5.1. Representando números

Un grupo de bits que comúnmente se usa para representar números en una computadora se denomina "byte", un byte está compuesto por ocho bits y puede representar números del 0 al 255. A fin de mostrar cómo representar los números en las computadoras utilizaremos un grupo de cinco bits.

Observa el grupo de cinco estelas mostradas en la Figura 3.6, cada estela contiene puntos marcados en la cara que se encuentra boca arriba. El conjunto de estelas comienza con la estela que contiene un solo punto y cada estela que se encuentra a la izquierda contienen el doble de puntos de la estela adyacente que se encuentra a su derecha, en este caso, 1, 2, 4, 8 y 16.

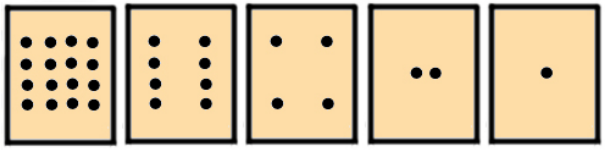


Figura 3.6: La estela que se encuentra a la izquierda contienen el doble de puntos.

Cuándo la estela se encuentra boca arriba representa al 1 y cuando se encuentra volteada el 0, mediante la concha de caracol maya (ver Figura 3.7). Es decir, el 1 significa que los puntos están visibles y el 0 indica que los puntos se encuentran ocultos. Para obtener el número decimal a partir de la distribución de estelas mostrada en la Figura 3.7 únicamente se suman los puntos que se encuentran visibles. Por ejemplo, el número binario 01001 representa al número nueve.

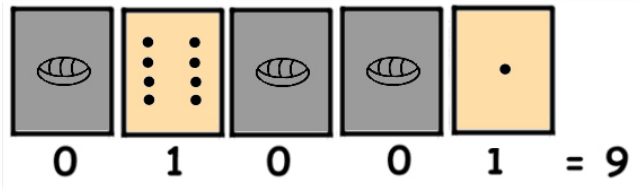


Figura 3.7: El número nueve es 01001 en el sistema binario.

Intenta escribir el número del mes en que naciste con un número binario, por ejemplo, Tukkul nació el 1 de abril de 1987. El número decimal a representar

es cuatro, y se obtiene al colocar boca arriba la tercera estela que contiene cuatro puntos y dejando volteadas todas las demás.

Algunas características interesantes de los números binarios son las siguientes:

- Dada una estela seleccionada, contiene el doble de puntos de la estela que se encuentra inmediatamente a su derecha. Como puedes ver, sólo son necesarias algunas estelas para representar números muy grandes.

- Dada una estela seleccionada que se encuentra visible y todas las estelas que se encentran a su derecha también son visibles, el resultado de la suma de dichas estelas es el valor de la estela que se encuentra inmediatamente a la izquierda de la estela seleccionada menos 1. Por ejemplo, si seleccionas la tercer estela con valor de 4 y lo sumas con los valores de las estelas anteriores, el resultado es $1 + 2 + 4 = 7$, que es el valor de la cuarta estela (con valor 8) menos 1.

- Otra característica interesante de los números binarios es qué sucede cuando se pone un cero en el lado derecho del número. Si estamos trabajando en la base 10 (decimal), cuando se pone un cero en el lado derecho del número, el número se multiplica por 10. Por ejemplo, 9 adquiere el valor de 90. En la misma manera, cuando se agrega un cero del lado derecho de un número binario se multiplica por dos, por ejemplo, $1001 = 9$ adquiere el valor de $10010 = 18$.

3.5.2. Representando texto

El texto en las computadoras se representa mediante la asociación de cada símbolo del alfabeto con números binarios. Por esta razón, se define un tamaño

de representación fijo denominado código bloque. Por ejemplo, el código ASCII (*American Standard Code for Information Interchange*) es de siete bits y ha sido ampliamente utilizado en el mundo. Actualmente, debido a que algunos países usan alfabetos con demasiados símbolos se diseño el código *Unicode*.

Observa una simplificación del código ASCII mostrada en la Figura 3.8 donde se utilizan cinco bits para representar un alfabeto que tiene 29 símbolos. Son necesarias al menos cinco estelas para tener los números suficientes para asociar cada símbolo, es decir, $2^5 = 32$ números (sobran tres números).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	
a	b	c	ch	d	e	f	g	h	i	j	k	l	ll	
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
m	n	ñ	o	p	q	r	s	t	u	v	w	x	y	z

Figura 3.8: Código para representar las letras del alfabeto (simplificación del ASCII) con un tamaño de representación fijo de cinco bits.

Por ejemplo, el mensaje "ayuda" se representa mediante bloques de cinco bits de la siguiente manera: 00001, 11100, 11000, 00101, 00001 (las comas separan cada letra para visualizar con mayor claridad el bloque que representa al símbolo).

Ahora intenta representar tu nombre, por ejemplo, tukkul en binario se escribe de la siguiente manera: 10111, 11000, 01100, 01100, 11000, 01101.

3.5.3. Representando imágenes

Las computadoras representan imágenes utilizando números binarios. Las pantallas de las computadoras se dividen en una cuadricula de pequeños pun-

tos llamados píxeles (pixel es la contracción de *picture element*). Los píxeles se pueden controlar independientemente para que emitan luz o no.

Si consideramos unicamente el caso de pantallas en blanco y negro, entonces el pixel emite luz blanca o no (se ve negro). Una manera simple de representar la imagen de la Figura 3.9 es mediante 342 bits (aproximadamente 42 bytes), 18 bits para cada uno de los 19 renglones. Usando la convención de codificar los renglones de izquierda a derecha y de arriba a abajo, los primeros tres renglones son: 000000000000000000, 000000000000000000, 000000111111000000 (en este ejemplo, el 1 representa un pixel negro, y el 0 uno blanco).

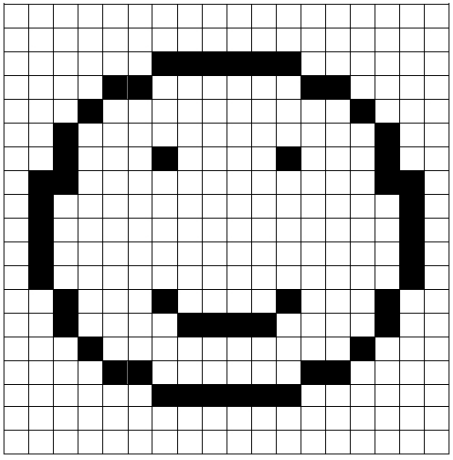


Figura 3.9: Imagen en blanco y negro de una carita feliz.

En pantallas a colores los píxeles emiten luz de distintos colores. Cada color se asocia con un número binario. Por ejemplo, en la Figura 3.10 se muestra la imagen de *Mario Bros* a la izquierda y la representación de dicha imagen en binario a la derecha. La imagen puede representarse utilizando bloques de tres bits donde cada bloque representa uno de los ocho colores (negro, rojo, verde, etc.).

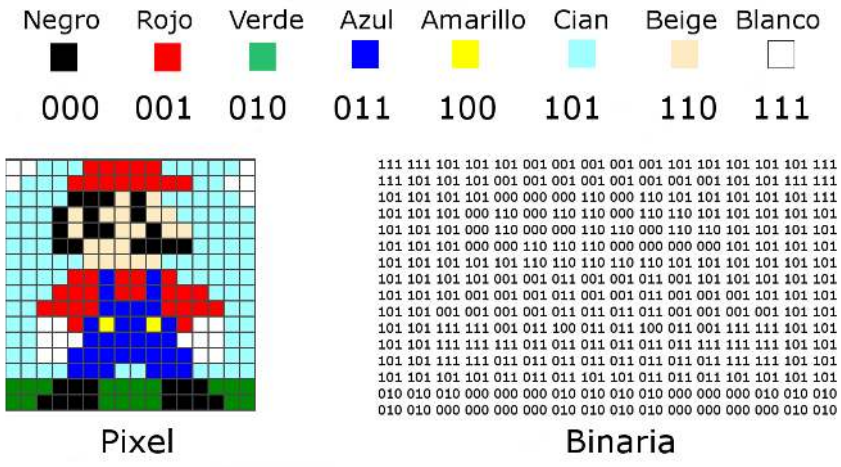


Figura 3.10: La imagen está representada por grupos de tres bits que representan el color correspondiente a la imagen.

En las imágenes de mapa de bits cada píxel se codifica mediante un conjunto de bits de longitud fija. Puede codificarse un píxel con un byte, de manera que cada píxel admite hasta 256 variaciones de color. En las imágenes llamadas de color verdadero, normalmente se usan tres bytes (24 bits) para definir el color de un píxel, en total se puede representar 16, 777, 216 variaciones de color. Cada 8 bits (byte) representa la intensidad de uno de los colores llamados *primarios*: rojo, verde y azul; de forma que la variedad de colores que vemos en una pantalla o televisión de color es producto de diferentes intensidades de dichos colores primarios. Este mismo principio es el que usa nuestra vista para reconocer los diferentes colores en el mundo.

3.6. Compresión

En la actualidad la cantidad de datos que se necesitan almacenar crece constantemente y las computadoras tienen un espacio limitado de almacenamiento. Por esta razón, es fundamental almacenar datos de manera eficiente. La *compresión es el acto de reducir el volumen de los datos tratables para representar una determinada información empleando una menor cantidad de espacio*.

Los datos se comprimen antes de ser almacenados y se descomprimen cuando se necesitan, permitiendo que la computadora pueda almacenar más información y también enviar información más rápido. Aunque la capacidad de almacenamiento de las computadoras ha crecido rápidamente en los últimos años, las necesidades de almacenamiento han crecido aún más rápido, por lo que las técnicas de compresión continúan siendo de suma importancia en los sistemas de cómputo que deben manejar archivos enormes de imágenes, audio y video. En la actualidad, el proceso de compresión y descompresión que las computadoras aplican en los datos es común.

3.6.1. Compresión de texto

Una de las técnicas que las computadoras utilizan para comprimir texto es el principio de apuntadores a secuencias previas del escrito. En la Figura 3.11 se muestra el poema *Vivir sin tus caricias*, escrito por Amado Nervo, donde se aplica la técnica de apuntadores. Dicha técnica consiste en retulizar fragmentos de letras que se repiten. Por ejemplo, las letras “am”, se repiten en las palabras: des(am)paro y (am)oroso.

La técnica técnica de apuntadores se denomina codificación *Ziv-Lempel* (LZ) y en la actualidad se conoce como *ZIP*. Además, se utiliza también en el formato

Vivir sin tus caricias es mucho desamparo;
Vivir sin tus palabras es mucha soledad;
Vivir sin tu amoroso mirar, ingenuo y claro,
es mucha oscuridad…

Amado Nervo

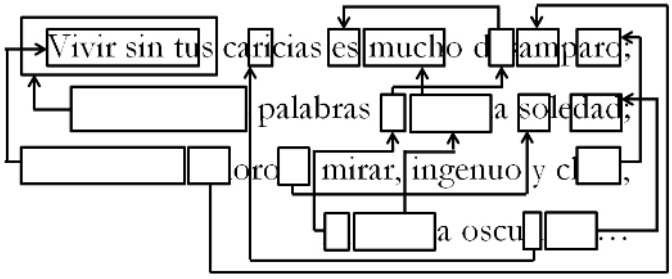


Figura 3.11: Compresión por apuntadores. El poema contiene 124 símbolos (sin contar espacios en blanco). Al utilizar apuntadores solo se necesitan 79 símbolos y 11 apuntadores.

de imágenes *Graphics Interchange Format* (GIF). Otros métodos, como el *Joint Photographic Experts Group* (JPEG) plantean que las letras más utilizadas deben tener códigos más pequeños, la clave Morse se basa en dicha idea.

3.6.2. Compresión de imágenes

Si las imágenes no se comprimen, entonces se toma más tiempo en transmitirlas y se requiere más espacio para almacenarlas. En las secciones anteriores se mostró como representar las imágenes en blanco y negro mediante una secuencia bits. Una imagen en blanco y negro también puede representarse con números porque sólo es necesario almacenar secuencialmente cuantos pixeles

son negros y blancos, es decir, el número consecutivo de pixeles blancos, seguido por negros, luego blancos y así sucesivamente..

Por ejemplo, los 3 primeros renglones de la Figura 3.9 mediante la representación de secuencias de bits tiene una longitud de 54 bits: 000000000000000000, 000000000000000000, 000000111111000000. Ahora, mediante la representación de números de bloques de ceros y unos, es: 18, 18, 6, 6, 6, la misma representación en términos de bits es: 10010, 10010, 00110, 00110, 00110, y tiene una longitud de 25 bits. Mediante la representación de números, los primeros tres renglones se comprimieron aproximadamente el 50 %.

A menudo las imágenes de de documentos (que se obtienen, por ejemplo, al tomar una foto a un documento) tienen grandes bloques de color blanco (los márgenes) o de color negro (una línea horizontal). Para ahorrar espacio de almacenamiento en este tipo de imágenes se suelen utilizar una variedad de técnicas de compresión de datos. La técnica de comprimir la imagen de la carita con números se denomina *Run-Length Encoding* (RLE), y es una manera eficaz de comprimir imágenes.

Las imágenes de color también tienen mucha repetición de bloques de un mismo color. Para ahorrar la cantidad de espacio requerido para almacenar imágenes a color, se utilizan una variedad de técnicas de compresión de datos como el *Joint Photographic Experts Group* (JPEG). Las imágenes se comprimen a menudo a una décima parte o incluso una centésima parte de su tamaño original. Esto permite almacenar muchas más imágenes en un disco y hace posible verlas en menor tiempo en la *Web*.

Los videos en la computadora se representan como una secuencia de imágenes, normalmente 30 imágenes por segundo, y por lo tanto ocupan mucha memoria. Por ello, se han desarrollados formas muy elaboradas para compri-

mir videos que combinan la repetición de bloques dentro de una imagen, con la repetición de bloques entre imágenes consecutivas (tienden a parecerse mucho dos imágenes consecutivas en un video). Las técnicas de compresión más comunes para video se conocen como MPEG (*Moving Pictures Expert Group*), que incluyen diferentes estándares de compresión de video como el MPEG-4.

3.7. Corrección de errores

Cuando los datos se almacenan en un disco o se transmiten de una computadora a otra, generalmente asumimos que los datos no cambian en el proceso. Sin embargo, debido a diferentes factores como la exposición a radiaciones magnéticas o eléctricas, por el calor, por daños físicos o por fallas de interferencia en el medio de transmisión, se presentan errores en los datos.

Imagina que estás depositando 10 000 pesos en tu cuenta bancaria. El cajero escribe la cantidad del depósito y la envía a la computadora central. Al mismo tiempo ocurre una interferencia en la línea y la cantidad de 10 000 se cambia por 100. Esto sí que será un problema para ti. Ahora, considera el caso donde se reciben datos desde una sonda espacial lejana, sería muy ineficiente esperar la retransmisión de los datos si ocurre un error. Por ejemplo, el tiempo para obtener una señal de radio de Júpiter a la Tierra cuando se encuentra en su punto más cercano es aproximadamente de media hora.

La computadora debe de estar dotada de un método de *detección de errores* capaz de reconocer cuando se han corrompido los datos y de un método de *corrección de errores* capaz de reconstruir los datos originales. La idea esencial para hacer frente a errores es expandir los datos (contrario a comprimir) para tener representaciones capaces de garantizar que lo que se recupera, luego de enviar

o almacenar datos, sea lo mismo que se envió o almacenó originalmente, o al menos que se puedan detectar errores.

Observa la rejilla de 5×5 mostrada en la Figura 3.12. Ahora, se añade otra fila y otra columna extras de manera que haya un número par de celdas coloreadas de negro en cada uno de los renglones y columnas. A la celda extra se le llama *celda de paridad*.

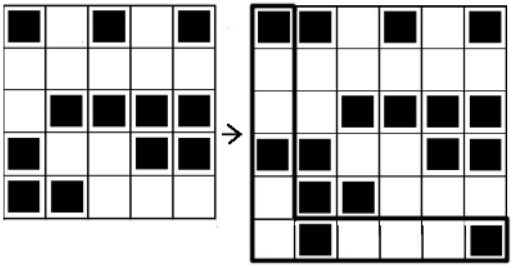


Figura 3.12: Rejilla de 5×5 a la izquierda y la misma rejilla con la adición de las celdas de paridad a la derecha.

Ahora, se colorea de blanco la celda del renglón 3 y la columna 3 (ver Figura 3.13) ¿Qué notas en la fila y en la columna de esa celda? La respuesta es que tienen un número impar de celdas coloreadas. Las celdas de paridad se utilizan para mostrar cuando se ha cometido un error.

Técnicas similares a la mostrada anteriormente se utilizan en las computadoras. Al colocar bits de paridad es posible detectar no sólo si se ha producido un error, sino también en dónde ha ocurrido. El bit incorrecto es cambiado de nuevo y con ello es realizada la corrección del error.

Por supuesto, las computadoras suelen utilizar sistemas de control de errores más complejos que son capaces de detectar y corregir múltiples errores. El disco duro de una computadora asigna una gran cantidad de su espacio para la

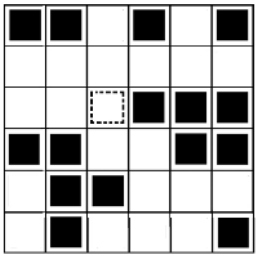


Figura 3.13: Si hay filas y columnas con un número impar de celdas coloreadas de negro, entonces puede deducirse dónde ha ocurrido un error.

corrección de errores, de modo que trabaje de manera confiable incluso si fallan partes del disco. Este tipo de sistemas están estrechamente relacionados con el esquema de paridad.

3.8. Criptografía

Los datos se pueden comprimir, expandir y también esconder. *La criptografía es una técnica de cifrado que altera las representaciones lingüísticas de determinados mensajes a fin que sean incomprensibles a receptores no autorizados*.

El objetivo de la criptografía es que dos entidades diferentes, A y B, logren comunicarse de modo que sólo ellos puedan entender el significado de los mensajes. Se presupone entonces la existencia de una tercera entidad, el enemigo, que está permanentemente al acecho procurando comprender lo que A y B se dicen. Para que el enemigo no tenga éxito, A y B deben enviarse mensaje cifrados.

El proceso criptográfico básico es un *método de cifrado* que recibe como entrada datos comprensibles y un elemento adicional llamado *clave de cifrado*. La salida del método es un *mensaje cifrado*. El mensaje cifrado es entonces transmitido o almacenado en un medio susceptible a ser intervenido por el enemigo. El

receptor del mensaje recupera los datos originales usando el mismo método, pero ahora para descifrar. El *método de descifrado* recibe como entrada el mensaje cifrado y una clave de descifrado (correspondiente a la clave de cifrado) con lo que se obtiene el mensaje original.

3.8.1. Cifrado por sustitución

Uno de los cifrados más antiguos que se conoce es el *cifrado de César*, atribuido a Julio César. En el cifrado de César cada letra del alfabeto se remplaza por la letra que se encuentra tres posiciones adelante. Es decir, *a* se vuele *D*, *b* se vuelve *E*, *c* se vuelve *F*,..., *x* se vuelve *A*, *y* se vuelve *B* y *z* se vuelve *C*. Por ejemplo, la palabra *mundo* se vuelve *PXQGR*. En los ejemplos de esta sección el texto llano se muestra en minúsculas y el texto cifrado en mayúsculas.

A los métodos en el que cada letra o grupo de letras se reemplaza por otra letra o grupo de letras para disfrazarla se les denomina *cifrado por sustitución*. Otra técnica de sustitución más robusta que el cifrado de César es remplazar cada uno de los símbolos del texto llano con alguna otra letra del alfabeto. El texto general de sustitución de símbolo por símbolo del alfabeto se llama *sustitución monoalfabetica*. Por ejemplo,

Alfabeto: a b c d e f g h i j k l m n o p q r s t u v w x y z
Sustitución: Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

La clave es la cadena de sustitución de 26 letras, por ejemplo, el texto llano de la palabra *mundo* se transforma en el texto cifrado *ZXYMG*. La técnica de sustitución monoalfabetica parece razonablemente segura debido a que se tendría que probar entre las 26! $\approx 4 \times 10^{26}$ claves posibles para descifrar la clave. Incluso, si

se utilizará una computadora que obtuviera una solución en 1 nseg tardaría 10^{10} años en probar todas las claves.

Sin embargo, puede descifrarse fácilmente al aprovechar las estadísticas de los lenguajes naturales. Por ejemplo, en el idioma inglés, la letra *e* es la más común, seguida de *t*, *o*, *a*, *n*, *i*, etc. Las combinaciones de dos letras más comunes son *th*, *in*, *er*, *re* y *an*. Las combinaciones de tres letras más comunes son *the*, *ing*, *and* e *ion*.

Mediante un análisis de frecuencias que consiste en primero calcular la frecuencia de las letras que aparecen en el texto cifrado y luego asociar letras de texto llano a ellas. Una gran frecuencia de *X* podría sugerir que las *X* son *e*, por lo que se asignaría tentativamente *X* a *e* y la siguiente más común a la letra *t*. Posteriormente, probaría con combinaciones de tres letras comunes como *tXe*, lo que sugeriría fuertemente que *X* es *h*, y se continuaría en la misma manera tratando de adivinar mediante las palabras comunes y conociendo los patrones probables de las vocales y consonantes. Se intentarían varias combinaciones hasta descifrar el texto.

En la actualidad los mensajes se codifican utilizando las computadoras, donde cada letra se representa mediante un número (utilizando el código ASCII, por ejemplo), así que un mensaje es simplemente una secuencia de números. Por ejemplo, la palabra *adiós* es equivalente al número "6568737982". Convertidos en números, los mensajes se pueden cifrar mediante operaciones aritméticas. La llave en este caso esta definida por cierto número y ciertas operaciones aritméticas. Supongamos que la llave es 2 y la operación es la multiplicación. Podemos entonces cifrar *adiós* multiplicándolo por 2 ($6568737982 \times 2 = 13137475964$). El receptor simplemente divide entre 2 para descifrar el mensaje.

Actualmente la criptografía es muy utilizada en el comercio electrónico para transmitir información delicada, por ejemplo los números de tarjeta de crédito. Pero hay un problema, cómo compartir la llave sin que otras personas la conozcan. Para ello se invento un esquema de criptografía muy ingenioso basado en una llave *pública*. La criptografía de llave pública se basa en dos ideas principales. La primera es la idea de la llave asimétrica: la llave para codificar el mensaje es diferente de la llave para decodificarlo. De esta forma puedes hacer pública la llave para cifrar los mensajes secretos que te vayan a enviar, pero solo tú conoces la llave secreta para descifrarlos. La segunda idea clave es el uso de ciertas funciones matemáticas que son fáciles de aplicar en un sentido pero difíciles en el sentido opuesto, llamadas funciones de un solo sentido. Por ejemplo, el producto de dos números primos (solo divisibles entre uno y el mismo número) es fácil de obtener, pero dado el producto, es muy difícil obtener los factores.

3.9. Aplicando la información

La información es la materia prima fundamental para para tomar decisiones y solucionar problemas en cualquier ámbito de la vida cotidiana porque aumenta el conocimiento. Una persona informada puede tomar decisiones responsables y asertivas.

Actualmente, estamos rodeados de fuentes de información: televisión, radio, periódicos, Internet, etc. Durante el día consumimos información a través de un noticiero, una revista o en la *Web*. La información que absorbemos día con día pasa muchas veces desapercibida ante nosotros y no nos percatamos de la importancia que tiene en nuestra vida personal.

Por ejemplo, cada uno de nosotros aplica la información de diversas maneras, desde la persona que toma un paraguas antes de irse a trabajar, porque vio el estado del tiempo, hasta el inversionista que compra o vende acciones con base en la información de la Casa de Bolsa.

En la actualidad, vivimos la era de la información porque cada día se generan cantidades masivas de información en Internet. No obstante, gran parte de la información contenida en Internet es repetitiva, errónea o falseada. Por está razón, las destrezas contenidas en el pensamiento computacional son vitales para el hombre moderno porque le permiten discernir qué información es confiable y qué información se debe desechar.

Las computadoras seguirán provocando cambios en la sociedad, las empresas y la vida de los individuos. El acceso a la información es de vital importancia en la sociedad porque le permite a cada individuo tener la posibilidad de crear, consultar, utilizar y compartir la información y el conocimiento, de tal manera que las sociedades puedan emplear plenamente sus posibilidades en la promoción de su desarrollo sostenible y en la mejora de su calidad de vida.

3.10. Ejercicios

1. De acuerdo a los siguientes valores, indica el tipo de dato primitivo que se debería utilizar en su representación en un programa de computadora.

Valor	Tipo de dato
Número de peldaños de una escalera	
Altura de un edificio	
Contador igual o mayor que 20	
"Z"	
Calle de la Alegría Num.14	

2. Un hospital desea iniciar el registro de los datos de pacientes que acuden por primera vez al hospital. Investiga que datos le pide normalmente un hospital a un paciente cuando ingresa. Elabora una propuesta de tabla de datos en la que se indique que dato se desea guardar, de que tipo se propone y un ejemplo de valor de ese dato. Se incluyen un par de ejemplos de datos:

Nombre del dato	Tipo de dato	Un valor del dato
Identificador_paciente	Entero	987615
Nombre_paciente	Cadena de caracteres	Tukkul Ch'íich' Kaan
...		

3. Escribe en binario tu fecha de nacimiento en el formato: día, mes, año.

4. Investiga el código ASCII que utiliza 7 bits para representar cada símbolo de texto, y escribe tu nombre en binario utilizando ASCII.

5. Para la imagen en blanco y negro de la Figura 3.9, escribe en binario la imagen completa (los 19 renglones).

6. Representa la imagen del problema anterior mediante la compresión en bloques de ceros y unos por renglón. ¿Cuántos bits requiere la imagen original y cuántos la comprimida?

7. En cierto lugar el clima se comporta estadísticamente de la siguiente manera: de 365 días, 200 lluvioso, 60 nublado, 40 soleado, 20 nevado, 20 tormenta, 10 graniza, 10 viento y 5 llovizna. ¿Si cada día se envía un mensaje con el clima, qué información da para cada tipo de clima?

8. Para el problema anterior, ¿Cuál es el promedio de bits de información que da el mensaje?

9. Codifica el mensaje *pensamiento computacional* mediante el código de sustitución de la Sección 3.8.1.

10. Decodifica el siguiente mensaje secreto “UIUOPYTU OP ITPZITUE OP YADPXAE", que esta cifrado por sustitución. Para esto necesitas saber cuales son las letras más comunes en español y usar tu imaginación.

Capítulo 4

Algoritmos

Un objetivo sin un plan es solamente un deseo.
Antoine de Saint-Exupéry

El vocablo *algoritmo* es de origen árabe y proviene del sobrenombre del matemático Al-Khowarizmi. Cuando escuchan la palabra algoritmo, las personas frecuentemente consideran que es demasiado sofisticada y exclusivamente relacionada con las ciencias. Sin embargo, los algoritmos están ampliamente relacionados con nuestra vida cotidiana y con nuestro quehacer profesional. *Un algoritmo es una serie de pasos ordenados que se siguen para resolver un problema*.

En la vida cotidiana se emplean algoritmos ampliamente, por ejemplo, en las recetas para preparar platillos. Un algoritmo para preparar un pastel de queso (tarta de queso) se presenta a continuación.

1. *Mezclar en una licuadora* 100*gr de galletas María y* 50*gr de mantequilla hasta que quede una pasta de galletas parecida a la arena mojada.*

2. *Aplastar la pasta de galletas en un recipiente hondo hasta que quede una base compacta.*

3. *Colocar el recipiente hondo en un refrigerador por* 15 *minutos.*

4. *Añadir en una cazuela* 500*gr de queso crema,* 500*gr de nata,* 100*gr de azúcar y* 15*gr de gelatina en polvo.*

5. *Mezclar los ingredientes anteriores a fuego lento mientras haya grumos.*

6. *Añadir la mezcla sobre la base de galletas del recipiente hondo.*

7. *Colocar el recipiente hondo en el refrigerador por* 4 *horas.*

8. *Cubrir la superficie de la tarta con* 300*gr de mermelada de fresa.*

El pastel de queso tiene diferentes recetas para prepararse. Por esta razón, una persona que está tratando de hacer un delicioso pastel de queso tiene varias recetas que puede utilizar para lograr su objetivo. Los resultados no serán los mismos porque algunas recetas requieren mayor tiempo de preparación, tienen ingredientes costosos o difíciles de conseguir, tienen demasiada azúcar, u otras características que afectan el producto final.

Al igual que en las recetas, un *problema* en particular frecuentemente puede resolverse de varias maneras. Por esta razón, hay varios algoritmos que pueden solucionarlo con diferentes ventajas y desventajas, algunos *tardan más tiempo en encontrar una solución* u otros *consumen más memoria*.

Las computadoras, al igual que el ser humano, usan varios lenguajes, denominados *lenguajes de programación*, que indican cuales instrucciones deben ejecutarse. Un algoritmo implementado en un lenguaje de programación se denomina *programa*. Todo el software que utilizamos diariamente como los sistemas operativos (Windows, Mac OS, Linux, etc.), las aplicaciones ofimáticas (Word, Power Point, Excel, etc.), los videojuegos (The Legend of Zelda, Street Fighter,

Tomb Raider, etc.), los navegadores (Chrome, Firefox, Explorer, etc.), son algoritmos que se implementan por medio de lenguajes de programación (ver figura 4.1).

Figura 4.1: Los video juegos son ejemplos de algoritmos implementados en computadoras por medio de lenguajes de programación.

Las características subyacentes de un algoritmo no se ven afectadas por las capacidades de un lenguaje de programación o las capacidades de una computadora en particular. Los algoritmos son uno de los tópicos principales del pensamiento computacional porque representan las soluciones a nivel conceptual en el proceso de resolución de problemas.

4.1. Representación de algoritmos

Los algoritmos se representan de varias formas, incluyendo el *lenguaje natural* (utilizado para representar el algoritmo del pastel de queso presentado anteriormente), *diagramas de flujo*, *seudocódigo* y *lenguajes de programación*. Las descripciones en lenguaje natural pueden ser ambiguas y extensas, mientras que los diagramas de flujo y el seudocódigo evitan las ambigüedades del lenguaje na-

tural porque son representaciones más estructuradas de los algoritmos. Además, son independientes de los lenguajes de programación. Por estas razones, nos enfocaremos en los diagramas de flujo y el seudocódigo.

4.1.1. Diagramas de flujo

Los diagramas de flujo son la representación visual de cada paso del algoritmo por medio de símbolos que representan las operaciones ejecutadas sobre los datos. Hay símbolos aceptados como estándar, a partir de las propuestas de organizaciones como: *American National Standards Institute* (ANSI) y la *International Organization for Standarization* (ISO).

Los diagramas de flujo se usan para introducir a los estudiantes en el desarrollo de algoritmos por su facilidad de lectura. Sin embargo, abarcan demasiado espacio y su construcción es laboriosa, por esta razón, los diagramas de flujo se usan principalmente para representar algoritmos pequeños. En la figura 4.2 se muestran los símbolos básicos de los diagramas de flujo. En las siguientes secciones se explicará su significado y cómo utilizarlos.

4.1.2. Seudocódigo

El seudocódigo es una descripción informal de alto nivel de un algoritmo que utiliza las convenciones de un lenguaje de programación real. El seudocódigo está diseñado para que el algoritmo sea leído por un humano. Por esta razón, el seudocódigo se complementa, donde sea conveniente, con descripciones detalladas en lenguaje natural, o con notación matemática. Además, omite detalles que no son esenciales para su comprensión, por ejemplo, el tipo de variables y la implementación de algunas funciones (sub-algoritmos, que son componentes de los algoritmos que resuelven sub-problemas específicos).

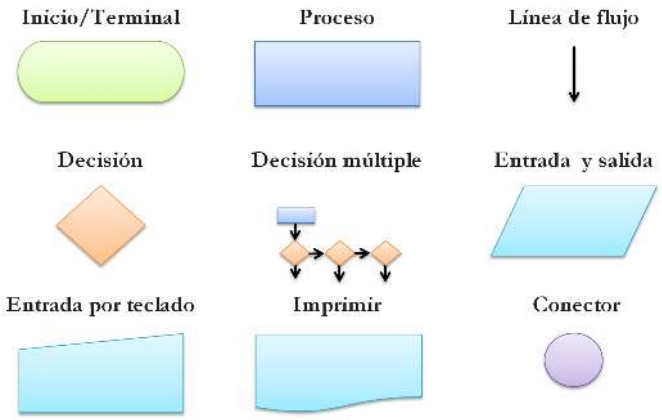


Figura 4.2: Símbolos básicos de los diagramas de flujo.

En el seudocódigo no hay una sintaxis estándar y por lo general, no obedece a las reglas de sintaxis de ningún lenguaje de programación. Dependiendo del diseñador, el seudocódigo puede diferir en su estilo; puede acercarse a una descripción en lenguaje natural (en prosa), o en el otro extremo, aproximarse a una imitación casi exacta de un lenguaje de programación real.

El seudocódigo tiene las siguientes ventajas sobre los diagramas de flujo:

- El espacio utilizado en la descripción del problema es menor.
- Las operaciones complejas se representan de forma sencilla.
- El seudocódigo es sencillo de trasladar a un lenguaje de programación.
- Las reglas de sangrías permiten observar claramente los niveles en la estructura del algoritmo.

En las secciones siguientes los algoritmos se explican principalmente utilizando seudocódigo.

4.2. Variables

Una variable es un espacio reservado en la memoria que, como su nombre indica, puede cambiar de contenido a lo largo de la ejecución de un programa. En los algoritmos, cada diseñador aplica algún tipo de convención. Por ejemplo, la instrucción para asignar el *valor de la variable x a la variable y* puede representarse como: $x = y$, $x \leftarrow y$ o $x := y$.

Las operaciones usualmente se representan de manera matemática de la siguiente manera:

$volumen = \pi r^2 h$

$hipotenusa = \sqrt{a^2 + b^2}$

$resultado = \text{sen}(a)$

En los lenguajes de programación las variables suelen asociarse a determinados tipos de datos. Hay variables de tipo *entero* o *flotante* que almacenan números enteros o reales. Hay variables *booleanas* que solamente almacenan el valor de 1 ó 0. Además, hay variables del tipo *carácter* que almacenan valores alfanuméricos. En los algoritmos algunas variables sirven como auxiliares en la ejecución de instrucciones, como *contadores* y *acumuladores*.

4.2.1. Contadores

Los contadores son variables que cuentan el número de eventos ejecutados dentro de un algoritmo. Un contador aumenta o disminuye su contenido en un valor constante. El contador inicia generalmente en 0 ó 1. Por ejemplo, en el video juego Mario Bros™ se utiliza un contador para almacenar las vidas, el tiempo o la cantidad de monedas.

Las operaciones básicas que se llevan a cabo con una variable contador son:

1. *Inicialización*: *contador* = *valor inicial*. Esta operación asigna el valor inicial de la variable contador. Por ejemplo: *vidas* = 1.

2. *Incremento o decremento*. Para el incremento: *contador* = *contador* + *constante*, y para el decremento: *contador* = *contador* − *constante*. Dichas operaciones se realizan cada vez que ocurre el evento que se desea contar. Usualmente el incremento o decremento corresponde a la constante 1. Por ejemplo: *vidas* = *vidas* + 1. Desde luego, pueden utilizarse otras constantes. Por ejemplo: *total* = *total* + 5.

Imagina que Tukkul es el personaje principal de un videojuego, cuando Tukkul obtiene un elote aumenta su número de vidas en uno (similar a los champiñones de Mario). La expresión utilizada sería *vidas* = *vidas* + 1. Si al inicio el contador *vidas* = 1 y Tukkul obtiene tres elotes, el contador *vidas* adquirirá los valores que se indican en la tabla 4.1.

Tabla 4.1: Contador de la expresión *vidas* = *vidas* + 1

vidas =	*vidas*	*constante*
2	1	1
3	2	1
4	3	1

4.2.2. Acumuladores

Los acumuladores incrementan o decrementan su contenido en un valor variable. La acumulación inicia generalmente en 0 o 1.

La operación básica que se debe realizar con ellos es:

1. *Inicialización*: *acumulador = valor inicial*. Esta operación asigna el valor inicial de la variable acumulador y depende del tipo de operación que se realizará. Por ejemplo: *energía* = 100

2. *Incremento o decremento*. Para el incremento: *acumulador = acumulador+ variable*, y para el decremento: *acumulador = acumulador − variable*. Dichas operaciones se realiza cada vez que ocurre el evento que se desea acumular. Por ejemplo: *vitalidad = vitalidad − impacto*.

Nuevamente, imagina que Tukkul es el personaje principal de un videojuego, si Tukkul recibe impactos de enemigos, entonces su energía disminuye en una cantidad proporcional al nivel de fuerza del enemigo. La expresión utilizada sería *vitalidad = vitalidad − impacto*. Si al inicio el acumulador *vitalidad* = 100 y Tukkul recibe tres impactos de diferentes enemigos, el acumulador *vitalidad* adquirirá los valores que se indican en la tabla 4.2.

Tabla 4.2: Acumulador de la expresión *vitalidad = vitalidad − impacto*. Donde la *vitalidad* en la columna central es antes del impacto y en la columna izquierda después del impacto.

vitalidad	*vitalidad*	*impacto*
75	100	25
25	75	50
24	25	1

4.3. Operadores de asignación e igualdad

A diferencia de la noción usual que se enseña en los cursos de matemáticas con respecto al operador de igualdad, en los algoritmos y lenguajes de programación, el operador de igualdad tiene dos significados diferentes: asignar un valor o verificar una igualdad. Por ejemplo: en el lenguaje de programación *C* se utiliza el símbolo = para la asignación y el símbolo == para verificar la igualdad entre dos valores. En el caso de los algoritmos, usualmente el contexto en el que se encuentran las instrucciones permite determinar cual operación se debe aplicar.

En una operación de asignación usualmente se asigna el valor que resulta de los cálculos ejecutados en la parte derecha a la variable que está en la parte izquierda. Por ejemplo: si $x = 0$ y se ejecuta la siguiente operación: $x = x + 1$, entonces el valor adquirido al ejecutar la operación $x = 0 + 1$ es $x = 1$.

La operación de verificación de igualdad usualmente se aplica cuando se está evaluando una condición. Por ejemplo: si $x = 0$ y $z = 0$, entonces al ejecutar la operación $x = z$, el valor de x permanece en 0.

4.4. Estructuras de control

Un algoritmo estructurado se construye a partir de tres *estructuras de control*: la estructura de *secuencia*, la estructura de *decisión* y la estructura de *repetición*. Así mismo, en la *programación estructurada* los programas son creados a partir únicamente de dichas estructuras de control. En las siguientes secciones se explica a detalle dichas estructuras.

4.4.1. Estructura de secuencia

En un algoritmo, *la estructura de secuencia establece que las instrucciones escritas en seudocódigo se ejecutan en orden secuencial de arriba a abajo*. Considera el siguiente bloque de instrucciones:

1: $instrucciones_1$
2: $instrucciones_2$
3: ...
4: $instrucciones_n$

Las $instrucciones_1$ se ejecutan primero (renglón número 1), posteriormente se ejecutan las $instrucciones_2$, y de manera sucesiva se ejecutan las demás instrucciones hasta ejecutarse las $instrucciones_n$.

El bloque de instrucciones de la estructura secuencia se representa por el diagrama de flujo de la figura 4.3. *El símbolo proceso (rectángulo) representa la ejecución de una o más instrucciones que manipulan los datos.*

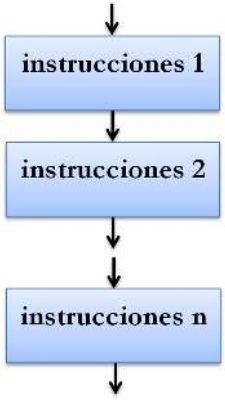


Figura 4.3: Diagrama de flujo que muestra el funcionamiento de la estructura de secuencia.

Por ejemplo, considera el siguiente algoritmo que realiza Paat para levantarse de la cama y asistir al laboratorio.

1: *levantarse*
2: *quitarse la pijama*
3: *tomar un baño*
4: *vestirse*
5: *desayunar*
6: *tomar el transporte hacia el laboratorio*

Paat realiza el conjunto de instrucciones de manera ordenada con la finalidad de presentarse en el laboratorio en tiempo y forma. Ahora, supón que las mismas instrucciones del algoritmo se ejecutan en un orden ligeramente diferente.

1: *levantarse*
2: *quitarse la pijama*
3: *vestirse*
4: *tomar un baño*
5: *desayunar*
6: *tomar el transporte hacia el laboratorio*

En este caso, Paat llegará mojada al trabajo. *El orden en que se ejecutan las instrucciones determina el resultado de un algoritmo*.

4.4.2. Estructura de decisión

En un algoritmo, *la estructura de decisión permite decidir cuales instrucciones se deben ejecutar dependiendo de una condición*. En la vida real, cada día las personas constantemente tienen que decidir cuales actividades realizar. Por ejemplo, salir a correr o quedarse en la cama; escoger entre estudiar o jugar; o decidir si comer a las 2:00 PM o a las 3:00 PM.

En esta sección se presentan tres variantes de la estructura de decisión: *decisión simple*, *decisión disyuntiva* y *decisión múltiple*.

Decisión simple

La estructura de decisión simple decide cuales instrucciones deben ejecutarse dependiendo del cumplimiento de una condición. La condición es una expresión booleana, es decir, el valor de la condición es verdadero o falso (1 ó 0). Considera el siguiente bloque de instrucciones.

```
1: si condición entonces
2:     instrucciones1
3: fin si
```

Si la *condición* es `verdadera` (especificada en el renglón 1), entonces se ejecutan las *instrucciones*$_1$ que están dentro del cuerpo de la estructura. Si la condición es `falsa` las *instrucciones*$_1$ no se ejecutan. La estructura de decisión simple se muestra en la figura 4.4 con un diagrama de flujo. *El rombo representa la evaluación de la condición para determinar cuales instrucciones deben seguirse.*

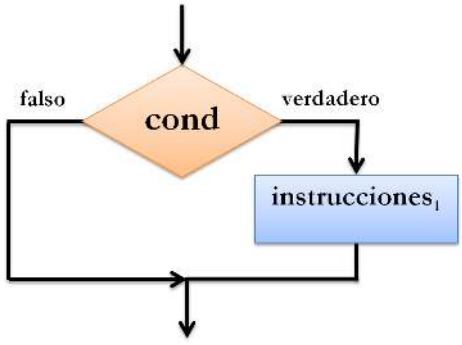


Figura 4.4: Diagrama de flujo que muestra el funcionamiento de la estructura de decisión simple.

La estructura de decisión simple se utiliza cuando:

- El problema requiere ejecutar o evitar determinadas acciones dependiendo de una condición.

Por ejemplo: supón que Tukkul ha decidido asistir a un curso de natación solo si el horario de las sesiones es posterior a las 7:00 hrs. El bloque de instrucciones mediante la estructura de *decisión simple* se presenta a continuación.

```
1: si horario_sesión > 7 entonces
2:     Asistir al curso de natación
3: fin si
```

donde el horario se almacena en la variable *horario_sesión*.

Decisión disyuntiva

La estructura de decisión disyuntiva decide cuales instrucciones deben ejecutarse entre dos posibles opciones dependiendo del cumplimiento de una condición. Al igual que en la estructura de decisión simple, la condición es una expresión booleana. Considera el siguiente bloque de instrucciones.

```
1: si condición entonces
2:     instrucciones_1
3: si no
4:     instrucciones_2
5: fin si
```

Por un lado, si la *condición* es `verdadera` (especificada en el renglón 1), entonces se ejecutan las $instrucciones_1$ que están dentro del primer bloque de la estructura. Por otro lado, si la condición es `falsa`, entonces se ejecutan las $instrucciones_2$ que están dentro del segundo bloque de la estructura. El bloque anterior se muestra en el diagrama de flujo 4.5.

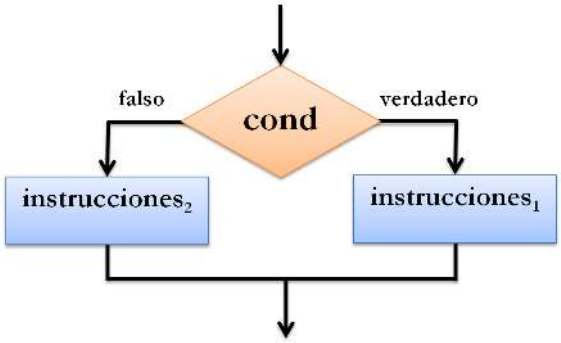


Figura 4.5: Diagrama de flujo que muestra el funcionamiento de la estructura de decisión disyuntiva.

La estructura de decisión disyuntiva se utiliza cuando:

- El problema requiere elegir cuales instrucciones ejecutar entre dos opciones dependiendo de una condición.

Por ejemplo, supón que un profesor califica un examen de historia. Si la calificación de un estudiante es mayor o igual que 6, entonces el estudiante aprueba. En caso contrario, el estudiante reprobará. Al profesor no le gusta que sus alumnos reprueben, pero no hay otra opción. El bloque de instrucciones mediante la estructura de *decisión disyuntiva* se presenta a continuación.

```
1: si calificación ≥ 6.0 entonces
2:     aprobar estudiante
3: si no
4:     reprobar estudiante
5: fin si
```

donde la calificación se almacena en la variable *calificación*.

Decisión múltiple

La estructura de decisión múltiple selecciona entre varias opciones cuales instrucciones se ejecutan dependiendo del valor de una expresión. Opcionalmente puede agregarse un caso por omisión. Es decir, cuando no se cumple ninguna de las condiciones. Considera el siguiente bloque de instrucciones.

```
1: seleccionar expresión
2: caso valor_1
3:     instrucciones_1
4: caso valor_2
5:     instrucciones_2
6: caso valor_n
7:     instrucciones_n
8: caso por omisión
9:     instrucciones_o
10: fin seleccionar
```

Si el valor de la *expresión* es igual al valor de alguno de los casos, entonces se ejecutan las instrucciones correspondientes. Cuando se encuentra la primera coincidencia, se ejecutan las instrucciones y se sale de la estructura *seleccionar*. Si el valor de la expresión no coincide con ninguno de los casos, entonces se ejecutan las instrucciones por omisión $instrucciones_o$. El diagrama de flujo correspondiente se muestra en la figura 4.6.

La estructura de decisión múltiple se utiliza cuando:

- El problema requiere elegir cual conjunto de instrucciones ejecutar entre múltiples opciones dependiendo del valor de una expresión.

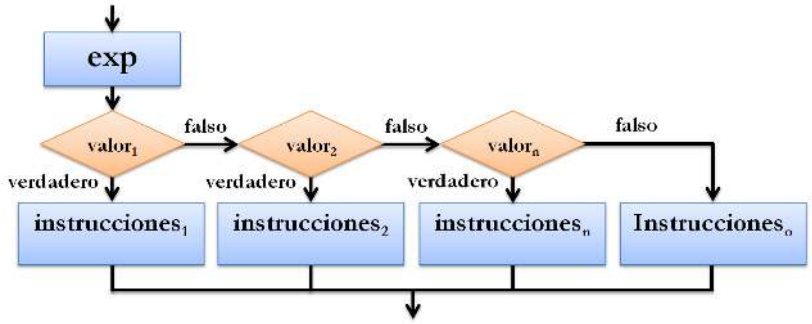


Figura 4.6: Diagrama de flujo que muestra el funcionamiento de la estructura de decisión múltiple.

Imagina que Tukkul desea obsequiarle un regalo a Paat. Tukkul tiene tres opciones que le gustarían: unas orquídeas, un libro de Feymman o el disco *Surfing with the Alien*.

```
1: seleccionar dinero
2: caso 1500
3:     Comprar Orquídeas
4: caso 500
5:     Comprar El placer de descubrir de Feymman
6: caso 200
7:     Comprar Surfing with the Alien
8: caso por omisión
9:     Escribir Poema
10: fin seleccionar
```

Tukkul seleccionará el regalo dependiendo del dinero que tenga. Por ejemplo, si tiene solo 200 pesos, entonces comprará el disco *Surfing with the Alien*. Si tiene 50 pesos, entonces escribirá un poema.

4.4.3. Estructura de repetición

En un algoritmo, *la estructura de repetición permite ejecutar varias veces determinadas instrucciones dependiendo de una condición*. En la vida real, cada día las personas constantemente tienen que repetir actividades. Por ejemplo, actividades como caminar, escribir o hablar, implican repetir acciones.

En esta sección se presentan tres variantes de la estructura de repetición: *repetir mientras*, *repetir hasta* y *repetir para*.

Repetir mientras

La estructura repetir mientras permite ejecutar varias veces determinadas instrucciones mientras su condición permanezca verdadera. Considera el siguiente bloque de instrucciones.

```
1: mientras condición hacer
2:     instrucciones1
3: fin mientras
```

Si la *condición* es `verdadera` (especificada en el renglón 1), entonces se ejecutan las *instrucciones*$_1$ que están dentro del cuerpo de la estructura. Posteriormente, comienza un nuevo *ciclo* (repetición periódica). Es decir, vuelve a evaluarse la *condición*, si es `verdadera` se ejecutan nuevamente las *instrucciones*$_1$. Finalmente, si la *condición* es `falsa`, entonces terminan las iteraciones (repeticiones). El bloque anterior se muestra por medio del diagrama de flujo en la figura 4.7. La estructura *repetir mientras* se utiliza cuando:

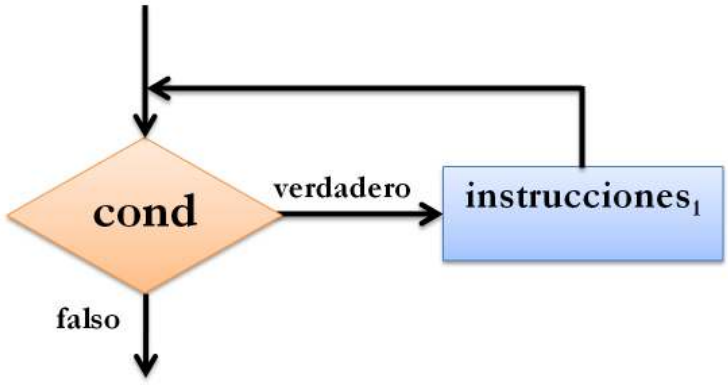


Figura 4.7: Diagrama de flujo que muestra el funcionamiento de la estructura *repetir mientras*.

- El problema a resolver requiere que la condición sea evaluada primero.
- El número de ciclos que deben ejecutarse para resolver el problema es desconocido.

Por ejemplo, considera la receta del pastel de queso presentada al inicio. La instrucción 5 dice: *Mezclar los ingredientes anteriores a fuego lento mientras haya grumos*. La instrucción implica que el cocinero debe mover repetidamente la cuchara para mezclar los ingredientes. El cocinero no sabe cuántas veces tendrá que mover la cuchara, por lo tanto, el proceso se repetirá mientras todavía haya grumos. El bloque de instrucciones de la estructura de *repetir mientras* se presenta a continuación:

```
1: mientras hay_grumos = verdadero hacer
2:     mover la cuchara para mezclar los ingredientes
3: fin mientras
```

donde el estado de la mezcla de los ingredientes se almacena en la variable *hay_grumos*.

Repetir hasta

La estructura repetir hasta permite repetir determinadas instrucciones hasta que la condición sea verdadera. A diferencia de la estructura *repetir mientras*, en la estructura *repetir hasta* primero se ejecuta el bloque de instrucciones y después se evalúa la condición. Considera el siguiente bloque de instrucciones.

```
1: repetir
2:     instrucciones1
3: hasta que condición
```

Las *instrucciones*$_1$ dentro del cuerpo de la estructura se ejecutan primero, por lo tanto, dichas instrucciones se ejecutan por lo menos una vez. Posteriormente la *condición* se evalúa (especificada en el renglón 3). Si la *condición* es `verdadera`, entonces nuevamente se ejecutan las *instrucciones*$_1$ dentro del cuerpo de la estructura. El proceso anterior se repite hasta que la *condición* es `verdadera` y por lo tanto la repetición termina. El bloque anterior se muestra por medio del diagrama de flujo en la figura 4.8.

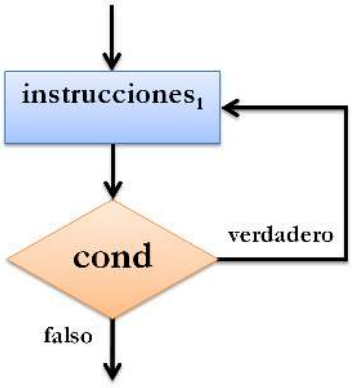


Figura 4.8: Diagrama de flujo que muestra el funcionamiento de la estructura *repetir hasta*.

La estructura *repetir hasta* se utiliza cuando:

- El problema a resolver requiere ejecutar determinadas instrucciones primero y posteriormente la condición.
- El número de ciclos que deben ejecutarse para resolver el problema es desconocido.

Por ejemplo, supón que un jugador debe lanzar dos dados hasta obtener como suma un valor mayor que 10. Probablemente en el primer intento lo consiga o hasta después de *n* intentos. No obstante, tiene que lanzar los dados por lo menos una vez para evaluar la condición. El bloque de instrucciones de la estructura de *repetir hasta* se presenta a continuación.

```
1: repetir
2:     suma = valor de los dados lanzados
3: hasta que suma ≥ 10
```

Repetir para

La estructura repetir para permite repetir instrucciones un determinado número de veces. Se utiliza cuando se conoce el número de repeticiones que se deben realizar. El bloque de instrucciones de la estructura *repetir para* se presenta a continuación.

```
1: para inicialización; condición; incremento hacer
2:     instrucciones_1
3: fin para
```

La estructura *repetir para* utiliza un *contador* para realizar el control del número de ciclos que se ejecutaran. La estructura *para* (renglón 1) se divide en tres

partes: *inicialización*, *condición* e *incremento*. La inicialización y el incremento corresponden a las operaciones de un contador (explicadas en la sección 4.2). Si la *condición* es `verdadera`, entonces se ejecutan las *instrucciones*$_1$ que están dentro del cuerpo de la estructura. Posteriormente el proceso anterior se repite, se evalúa la *condición*, si la *condición*$_1$ es `verdadera` se ejecutan las *instrucciones*$_1$ dentro del cuerpo de la estructura. Finalmente, cuando la *condición* es `falsa` la repetición termina. El bloque anterior se muestra por medio del diagrama de flujo en la figura 4.9.

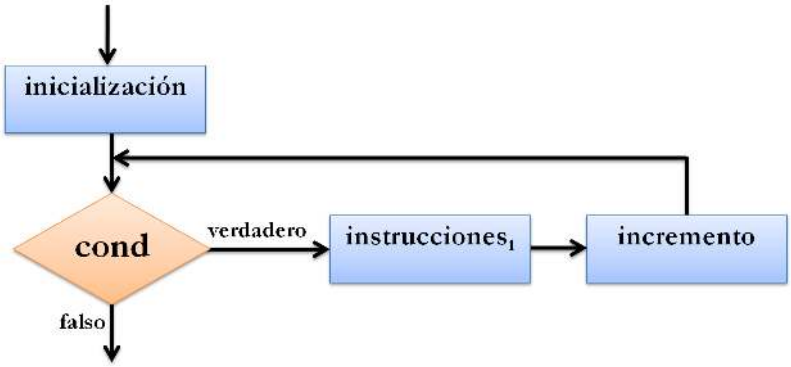


Figura 4.9: Diagrama de flujo que muestra el funcionamiento de la estructura *repetir para*.

La estructura *repetir para* se utiliza cuando:

- Se conoce el número de ciclos que deben ejecutarse.
- Se desea acceder consecutivamente a los elementos de una estructura de datos.

Por ejemplo, supón que Paat compra los productos que necesita a partir de una lista de compras. El número de veces que se repetirá la acción de comprar un producto se conoce con anticipación porque está determinado por el número

de elementos en la lista. El bloque de instrucciones del ejemplo se presenta a continuación.

1: **para** $i = 1; i \leq$ *número elementos*; $i = i + 1$ **hacer**
2: *comprar producto*$_i$
3: **fin para**

4.5. Apilamiento y anidamiento

El apilamiento se refiere a colocar una estructura de control después de otra. En la misma manera que la secuencia, sin embargo, específicamente con estructuras de control. El siguiente ejemplo muestra la estructura de control *decisión simple* y *repetir mientras* apiladas.

1: **si** *condicion*$_1$ **entonces**
2: *instrucciones*$_1$
3: **fin si**
4: **mientras** *condición*$_2$ **hacer**
5: *instrucciones*$_2$
6: **fin mientras**

El anidamiento se refiere a colocar una estructura de control dentro de otra. El siguiente ejemplo muestra la estructura de control *repetir mientras* anidada dentro de la estructura *decisión simple*.

1: **si** *condicion*$_1$ **entonces**
2: **mientras** *condición*$_2$ **hacer**
3: *instrucciones*$_1$
4: **fin mientras**
5: **fin si**

Imagina que tienes un videojuego donde Tukkul es el protagonista. Si Tukkul obtiene una estrella, entonces lo vuelve invencible por un tiempo y todo enemigo que toca muere. El siguiente seudocódigo muestra una estructura de control *decisión simple* anidada dentro de una estructura *repetir mientras* y que se ejecutaría mientras el tiempo no termine.

```
1: mientras Tiempo en estado estrella no termine hacer
2:     si Tocas enemigo entonces
3:         Enemigo muere
4:     fin si
5: fin mientras
```

Cualquier estructura de control puede ser sustituida por otra estructura de control. Además, las estructuras de control apiladas y anidadas forman algoritmos de mayor complejidad. En la misma manera que al apilar y anidar bloques de plástico nos permite crear estructuras más complejas (ver figura 4.10).

Figura 4.10: Los bloques de plástico interconectables son apilados y anidados para crear diversas estructuras.

4.6. Niveles de abstracción

Los niveles de abstracción están determinados por el grado de detalle con el que se especifica un problema [1]. Una técnica denominada enfoque descendente (*top-down*) es una técnica esencial para el diseño de algoritmos estructurados. En el enfoque descendente se formula una representación del algoritmo en un alto nivel de abstracción (sin especificar detalles) de lo que el algoritmo debe realizar. Posteriormente, la representación del algoritmo se especifica en un nivel de abstracción inferior al dividirse en tareas más detalladas. Cada tarea del algoritmo se describe con más detalle hasta que la especificación del algoritmo es lo suficientemente concreta.

Considera el siguiente problema: *Desarrolla un programa que calcule el promedio de horas de programación que dedican al día un número arbitrario de estudiantes*.

El algoritmo para el promedio de horas de programación se desarrollará mediante el enfoque descendente. Como se mencionó anteriormente, comenzamos con una representación en seudocódigo en un alto nivel de abstracción:

1: *Determinar el promedio de las horas de programación*

La descripción anterior es una instrucción individual que contiene la función general del programa. Sin embargo, contiene muy pocos detalles para escribir un programa.

[1] Jeannette Wing resalta la importancia de la abstracción en el pensamiento computacional, haciendo hincapié en la necesidad de pensar en múltiples *niveles de abstracción*

4.6.1. Nivel de abstracción 1

A partir de una descripción general, usualmente muchos algoritmos se pueden dividir de manera lógica en tres fases:

1. *Fase de inicialización* que inicializa las variables del algoritmo.
2. *Fase de procesamiento* que introduce los valores de los datos y ajusta las variables del algoritmo de manera adecuada.
3. *Fase de terminación* que calcula y despliega los resultado finales.

La observación anterior a menudo es todo lo que necesita para realizar el primer refinamiento. Comencemos con el proceso de refinamiento dividiendo la instrucción general en las tres fases anteriormente mencionadas y las listamos en el orden que deben ejecutarse. El resultado del primer refinamiento es el siguiente:

1: *Inicializar las variables*
2: *Introducir las horas de programación, sumarlas y contarlas*
3: *Calcular y desplegar el promedio de horas*

El refinamiento anterior utiliza solo la estructura de secuencia, es decir, que las instrucciones deben ejecutarse en orden una después de la otra. El proceso de refinamiento continua dividiendo las instrucciones en tareas más pequeñas. Para continuar con el siguiente nivel de abstracción se identifican las variables específicas.

4.6.2. Nivel de abstracción 2

En este ejemplo, necesitamos una variable para recibir el valor de cantidad de horas conforme el usuario la introduce que denominamos *horas*, una variable

para almacenar la suma de las horas denominada *total*, una variable que cuenta el número de horas denominada *n* y una variable para almacenar el promedio calculado denominada *promedio*. Las instrucciones en un nivel de abstracción inferior para la primera instrucción *Inicializar las variables* son las siguientes:

```
1: total = 0;
2: n = 0;
```

Observa que solo las variables *total* y *n* deben inicializarse antes de utilizarse. Las variables *horas* y *promedio* no necesitan inicializarse.

Las instrucciones en un nivel de abstracción inferior para la instrucción *Introducir las horas de programación, sumarlas y contarlas* requiere una estructura de repetición que introduzca cada hora. No sabemos por adelantado cuantas horas se van a procesar, por esta razón, utilizamos una estructura de control del tipo *repetir mientras*. Las instrucciones en un nivel de abstracción inferior para dicha instrucción son las siguientes:

```
1: introducir(horas)
2: mientras horas ≠ −1 hacer
3:     total = total + horas                    //Acumulador
4:     introducir(horas)
5:     n = n + 1                                //Contador
6: fin mientras
```

El usuario ingresa las horas una por una y después de introducir la ultima hora, el usuario introduce un valor no valido (−1). El algoritmo evalúa el valor de *horas* después introducir cada valor y termina el ciclo cuando el usuario introduce el valor −1 desde el teclado.

Las instrucciones en un nivel de abstracción inferior para la instrucción *Calcular y desplegar el promedio de horas* son las siguientes:

```
1: si n ≠ 0 entonces
2:     promedio = total/n;
3: si no
4:     No hay horas;
5: fin si
```

Observa que en una división el valor del denominador podría ser cero y en consecuencia ocurría un error. Por esta razón, debemos hacernos cargo de este problema de manera apropiada dentro del algoritmo (por ejemplo, se puede desplegar un mensaje de error), en lugar de permitir que ocurra un error.

El algoritmo está terminado cuando esté especificado con suficientes detalles para codificarlo en un lenguaje de programación. El algoritmo 1 resuelve un problema general de promedios. Este algoritmo se desarrolló después de sólo dos niveles de refinamiento. Para algoritmos más complejos se requerirán más niveles de refinamiento y subdividirlos en más tareas.

4.7. Aplicando los Algoritmos

Para el pensamiento computacional los algoritmos se consideran una práctica clave para poder formular problemas y aplicar soluciones de manera metódica y ordenada. Ahora que has adquirido los conceptos básicos sobre algoritmos, piensa en la forma que se encuentran presentes en tu vida cotidiana y en tu quehacer profesional. Se creativo, analiza el entorno que te rodea, las personas, sus actividades o la naturaleza, e identifica donde se aplica el concepto de secuencia, el concepto de decisión o el concepto de repetición.

Algoritmo 1 Calcular promedio de horas

```
1:  total = 0
2:  n = 0;
3:  introducir(horas)
4:  mientras horas ≠ −1 hacer
5:      total = total + horas                    //Acumulador
6:      introducir(horas)
7:      n = n + 1;                               //Contador
8:  fin mientras
9:  si n ≠ 0 entonces
10:     promedio = total/n
11: si no
12:     No hay horas
13: fin si
```

Promueve tu pensamiento lógico y algorítmico desde las actividades que tienes que realizar para prepararte y asistir a la escuela o al trabajo, hasta para resolver un problema matemático complejo, o en la toma de decisiones que realizas en tu lugar de trabajo. Con el tiempo, tendrás la habilidad para resolver problemas complejos a partir de ideas estructuradas.

4.8. Ejercicios

1. Diseña un algoritmo en seudocódigo para ayudar a Paat a encontrar un lugar para vivir cuando vaya a la universidad. A Paat le gustaría vivir cerca de la universidad, en un lugar con mucha luz y que le permitan tener mascotas. Pero también le gustaría que el costo de la renta no fuera muy elevado.

2. Tukkul está planeando que hacer el próximo fin de semana, y le gustaría conocer el clima para elegir la mejor opción y comprar lo necesario. En caso de un fin de semana soleado, a Tukkul le gustaría ir a una zona arqueológica y necesitará un sombrero. Pero si se trata de un fin de semana lluvioso, entonces le gustaría ir a la biblioteca a leer un buen libro y necesitará un paraguas. Diseña un algoritmo en seudocódigo para ayudar a Tukkul a planear el fin de semana.

3. Paat cursó 5 materias este semestre y le gustaría saber si puede solicitar una beca. Para ello, necesitaría un promedio por lo menos de 8.5. Diseña un algoritmo en seudocódigo para determinar si Paat puede solicitar una beca. Utiliza la estructura de control “repetir mientras”.

4. Un profesor desea felicitar a sus mejores estudiantes y motivar a los otros a aplicarse más en las materias. Los mejores estudiantes son aquellos que tienen un promedio igual o mayor que 9. El grupo tiene 20 estudiantes. Diseña un algoritmo en seudocódigo para ayudarle al profesor a cumplir con su objetivo. Utiliza la estructura de control "repetir para".

5. El auto de un amigo de Tukkul y Paat tiene una llanta ponchada, y quieren cambiarla, pero no saben cómo. Diseña un algoritmo en seudocódigo para ayudarlos a cambiar la llanta ponchada.

6. Tukkul ha conseguido un trabajo para el verano y está tratando de calcular cuánto dinero puede ganar cada semana. Su salario será de 250 pesos por hora si trabaja 40 horas o menos. Sin embargo, por cada hora adicional a las 40 recibirá un pago de 300 pesos. Diseña un algoritmo en seudocódigo que le permita a Tukkul calcular su ingreso semanal cuando conoce el número de horas trabajadas.

7. A Paat le han pedido en clases compartir sus conocimientos de la lengua maya con sus compañeros. Para motivar esta tarea, ha decidido escribir un traductor automático. Diseña un algoritmo en seudocódigo para traducir inicialmente los números del 1 al 4 (uno = juntúule', dos = ka'ap'éel, tres = óoxp'éel, cuatro = kantúulo'on). Por el momento, si el algoritmo recibe un número fuera de este rango, emitirá un mensaje indicando que la traducción aún no está disponible. Utilizar la estructura "decisión múltiple".

8. Tukkul y Paat han decidido iniciar una cuenta de ahorros en la que cada quien aportará 50 pesos mensuales. Para motivarlos, el banco les ha ofrecido un rendimiento de 1 % mensual. Diseña un algoritmo en seudocódigo

para ayudarles a calcular cuánto dinero tendrán al final de un plazo dado en meses, suponiendo que cumplen con su plan de depositar la misma cantidad cada mes, y además reinvierten los intereses obtenidos. Utilizar la estructura "repetir para".

9. Paat ha formado un club de amigos, de quienes registra su nombre y edad. Cada vez que registra un nuevo integrante, desea saber quién es el mayor y el más joven de sus amigos. Diseñar un algoritmo en pseudocódigo que realice esta tarea. Utilizar la estructura "repetir hasta".

10. Tukkul ha organizado una caja de ahorro para sus amigos, quienes pueden realizar depósitos y retiros. Diseñar un algoritmo en eudocódigo que reciba estos movimientos y permita un retiro únicamente si la cantidad no excede el monto depositado disponible. Al finalizar , el algoritmo reporta el monto disponible en caja. Utilizar la estructura "repetir hasta".

Capítulo 5

Funciones y recursión

La función del hombre en la Tierra es vivir, no existir.

Javier Reverte

Una estrategia para afrontar la complejidad es que los problemas se dividan en problemas menores y se resuelvan por separado. Dicha estrategia se denomina *divide y vencerás*. Los algoritmos usualmente son divididos a partir de piezas o componentes denominados *funciones* o *procedimientos*. *Las funciones o procedimientos son subalgoritmos del algoritmo original.*

Hay diferentes motivos para dividir un algoritmo en funciones o procedimientos:

- Las funciones o procedimientos evitan la repetición de instrucciones al reutilizarlas mediante una llamada a la función o procedimiento, por lo que el algoritmo se vuelve más compacto.
- Las funciones y procedimientos son algoritmos más fáciles de diseñar, corregir, mantener y modificar.

- Las funciones o procedimientos *ocultan los detalles* de su especificación, de manera que para utilizarlas sólo es necesario conocer su interfaz.

El ocultamiento de los detalles se refiere a esconder la especificación de cómo es ejecutada una función. Imagina una función como una caja negra de la cual nos interesa sólo su forma de interactuar con el medio que le rodea, entendiendo qué es lo que hace, pero sin dar importancia a cómo lo hace.

El poder de las funciones es que los usuarios pueden suponer que realizan su tarea sin necesidad de entender los detalles internos. ¿Sabes cómo realiza sus cálculos la función sin *x* incluida en tu calculadora? No tienes que saberlo, únicamente asumes que funciona.

Piensa cuánto más difícil sería manejar un coche si se tuviera que entender, antes de poder utilizarlo, cómo funciona la dirección hidráulica. Por lo regular, el único interés de un usuario es manejarlo y no cómo funciona internamente.

Los ingenieros podrían haber agregado en cada generación de nuevas tecnologías una gran cantidad de botones e interruptores, pero sabiamente han mantenido la misma *barrera de abstracción*, es decir, las mismas entradas y salidas: pedal derecho acelera y pedal izquierdo frena. Dicha barrera de abstracción ha permanecido a través de varias generaciones tecnológicas, porque es una interfaz sencilla que oculta detalles innecesarios.

5.1. Definición de funciones y procedimientos

En computación, una *función es un subalgoritmo que realiza una tarea específica y que devuelve un resultado*. Al igual que una función matemática, una función *recibe uno o varios valores de entrada y regresa un valor de salida*. Así mismo, un procedimiento *recibe uno o varios valores de entrada*, sin embargo,

no genera un valor de salida. No obstante, un procedimiento podría devolver resultados a través de sus parámetros de entrada.

En una función y un procedimiento es necesario especificar claramente cuáles son sus entradas. Usualmente los parámetros de entrada se especifican entre paréntesis al principio de la declaración de la función o procedimiento.

Una función especifica el resultado que devuelve. La estructura de una función en seudocódigo se muestra a continuación:

```
1: función NOMBREFUNCIÓN(parámetros)                    //Entrada
2:     instrucciones
3:     devolver resultado                                //Salida
4: fin función
```

Por ejemplo, el seudocódigo de una función que permite calcular la hipotenusa se presenta a continuación.

```
1: función CALCULARHIPOTENUSA(a, b)                      //Entrada
2:     c = √(a² + b²)
3:     devolver c                                        //Salida
4: fin función
```

Un procedimiento no especifica un resultado a devolver. La estructura de un procedimiento en seudocódigo se muestra a continuación:

```
1: procedimiento NOMBREPROCEDIMIENTO(parámetros)         //Entrada
2:     instrucciones
3: fin procedimiento
```

Por ejemplo, el seudocódigo de un procedimiento que imprime un mensaje en la pantalla se presenta a continuación:

```
1: procedimiento imprimirPantalla(opción)          //Entrada
2:     si opción = 1 entonces
3:         imprimir : Hola Mundo
4:     si no
5:         imprimir : Adiós Mundo
6:     fin si
7: fin procedimiento
```

Al diseñar una función o un procedimiento, si no puedes asignar un nombre conciso que exprese lo que la función realiza, entonces es posible que tu función intente realizar demasiadas y diversas tareas. Por lo general, es mejor dividir dicha función en varias funciones. La función original puede llamar a funciones más pequeñas para realizar la tarea completa. Cuando diseñes funciones considera lo siguiente:

- Diseña funciones pequeñas y concisas para promover su reutilización. Si tu función requiere muchos parámetros, entonces podría estar realizando demasiadas tareas. Por esta razón, divide tu función en varias funciones para realizar tareas separadas. Esto hace a los algoritmos más fáciles de corregir y modificar.

- Elige nombres de función y nombres de parámetros con significado para que los algoritmos sean claros y ayuden a evitar el uso excesivo de comentarios (notas explicativas dentro del código de un programa).

5.1.1. Llamadas a función

Una función se invoca para realizar la tarea para la que fue diseñada mediante una *llamada a función*. La llamada a función especifica el nombre de la

función y usualmente proporciona parámetros de entrada requeridos para que la *función invocada* realice su tarea. El algoritmo principal o una *función invocadora* llama a la función invocada para realizar la subtarea.

Por ejemplo, el algoritmo 2 realiza el cálculo del área de dos círculos (ver figura 5.1).

Algoritmo 2 principal

```
1: radio = 7
2: área = ÁreaCírculo(radio)          //La entrada es un radio = 7
3: radio = 14
4: área = ÁreaCírculo(radio)          //La entrada es un radio = 14
```

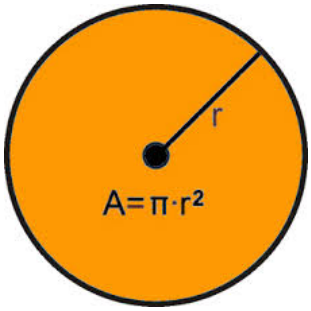


Figura 5.1: Área del circulo.

El algoritmo 5.1 realiza dos llamadas a función en la línea 2 y 4. La función `ÁREACÍRCULO` calcula el área total de un círculo con *radio* variable. La primer llamada a función `ÁREACÍRCULO` pasa el parámetro con un *radio* = 7 y el valor retornado se almacena en *área*. Así mismo, la segunda llamada a función de `ÁREACÍRCULO` pasa el parámetro con una *radio* = 14 y la salida se almacena nuevamente en *área*.

El área del círculo es calculada por la función `ÁREACÍRCULO`. La fórmula para calcular el área de un círculo es $área = \pi r^2$. La función `ÁREACÍRCULO` se muestra en el algoritmo 3.

Algoritmo 3 Área del circulo

```
1: función ÁREACÍRCULO(radio)                    //Entrada
2:     área = π * CUADRADO(radio)
3:     devolver área                             //Salida
4: fin función
```

La función `ÁREACÍRCULO` ejecuta una llamada a la función `CUADRADO` para calcular el cuadrado del radio. La *función invocadora* es `ÁREACÍRCULO` y la *función invocada* es `CUADRADO`. La función `CUADRADO` se muestra en el algoritmo 4.

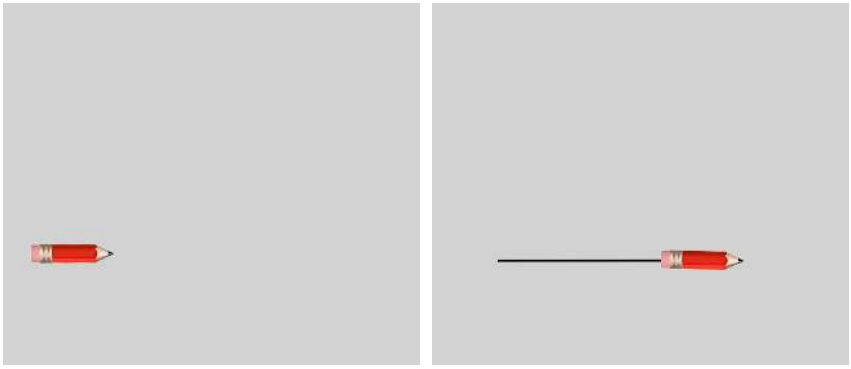
Algoritmo 4 Cuadrado

```
1: función CUADRADO(x)                           //Entrada
2:     y = x * x
3:     devolver y                                //Salida
4: fin función
```

5.1.2. Funciones para dibujar

Imagina que estás diseñando un programa de dibujo basado en procedimientos. Tu programa cuenta con dos procedimientos, *mover* y *girar*, para trazar líneas dentro de un lienzo gris.

El procedimiento *mover* traslada el objeto lápiz en la dirección de su punta *n* pixeles. Por ejemplo, para mover al objeto lápiz 100 pixeles se aplica el siguiente procedimiento: `mover`(100). El resultado se muestra en la figura 5.2.



(a) Estado inicial del lápiz (b) Dibuja el trazo recorrido.

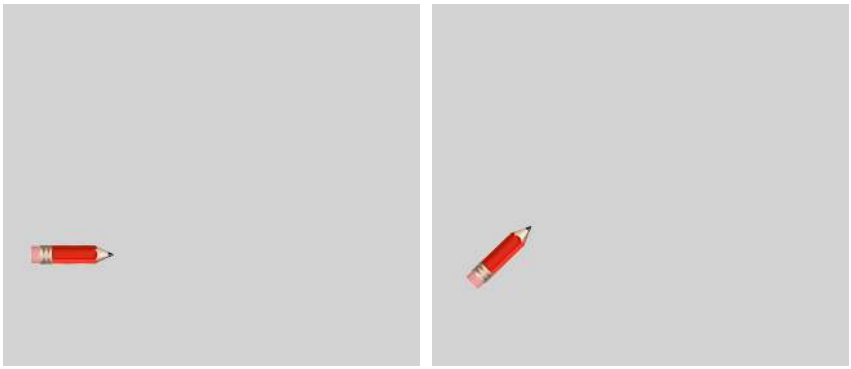
Figura 5.2: El procedimiento mover cambia de posición el lápiz y dibuja el trazo recorrido.

El procedimiento *girar* rota el objeto lápiz α (alfa) grados. Por ejemplo, para rotar el objeto lápiz 45 grados se aplica el siguiente procedimiento: `girar`(45). El resultado se muestra en la figura 5.3.

Al aplicar los procedimientos *girar* y *mover* en secuencia es posible dibujar en diferentes direcciones. Por ejemplo, para rotar el objeto lápiz 45 grados y después moverlo 100 pixeles se aplican los siguientes procedimientos:

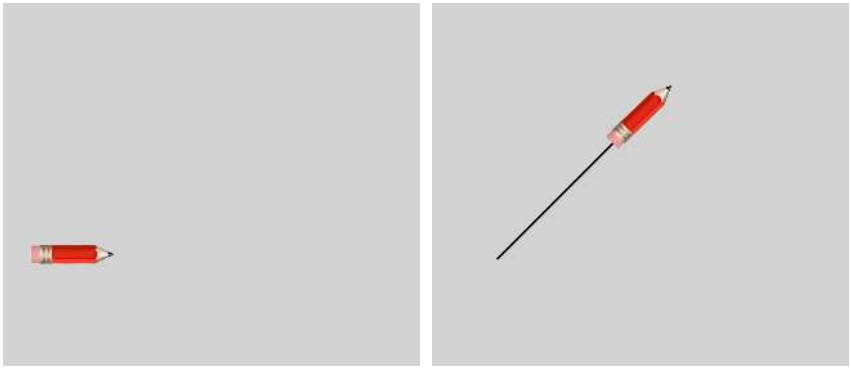
1: *girar*(45)
2: *mover*(100)

El resultado se muestra en la figura 5.4.



(a) Estado inicial del lápiz (b) Rota el lápiz 45 grados

Figura 5.3: El procedimiento girar rota el objeto lápiz para dibujar en otras direcciones.



(a) Estado inicial del lápiz (b) Dibuja el camino recorrido.

Figura 5.4: Cuando se aplican el procedimiento girar y mover es posible dibujar en distintas direcciones.

5.2. Recursión

La recursión es la especificación de un proceso basado en su propia definición. La solución de un problema resuelto mediante la recursión depende de las soluciones de pequeñas instancias del mismo problema. En las siguientes secciones veremos dos ejemplos de algoritmos recursivos: dibujar una concha de nautilus y resolver una sumatoria.

5.2.1. Concha de nautilus

Para ilustrar la recursión, en esta sección abordaremos un algoritmo *recursivo* que dibujará una concha de nautilus, como la mostrada en la figura 5.5. La clave para entender la recursión es notar que el dibujo de la concha de nautilus contiene varias versiones de sí misma de tamaño menor. Es decir, la concha de nautilus está compuesta por cuadrados que disminuyen su tamaño y rotan a partir de su esquina inferior izquierda en el sentido contrario de las manecillas del reloj.

El objetivo de esta sección es dibujar una concha de nautilus mediante llamadas recursivas. Vamos a crear un procedimiento `CONCHA`, que invocará los procedimientos *mover* y *girar* de la siguiente manera:

1. Invocar a *mover* para dibujar la base del cuadrado de izquierda a derecha e invocar a *girar* para rotar la dirección del objeto lápiz a 90°.

2. Invocar a *mover* para dibujar el lado derecho del cuadrado de abajo hacia arriba e invocar a *girar* para rotar la dirección del objeto lápiz a 90°.

3. Invocar a *mover* para dibujar el lado superior del cuadrado de derecha a izquierda e invocar a *girar* para rotar la dirección del objeto lápiz a 90°.

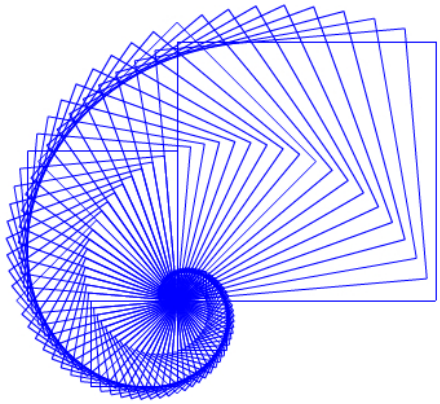


Figura 5.5: La concha de nautilus es un objeto geométrico cuya estructura se repite varias veces a diferentes escalas lo que se conoce como un fractal.

4. Invocar a *mover* para dibujar el lado izquierdo del cuadrado de arriba hacia abajo.

5. Invocar a *girar* para rotar la dirección del objeto lápiz a 95° de manera que el siguiente cuadrado sea dibujado ligeramente inclinado.

6. Invocará al procedimiento ¡`CONCHA`! para dibujar otro cuadrado de tamaño menor, y así sucesivamente hasta llegar al cuadrado que tenga un tamaño de lado menor que 10.

Quizá te preguntaras ¿cómo utilizar el procedimiento `CONCHA` dentro del procedimiento `CONCHA`?, ¡pues todavía no ha sido definido! Esa es la idea esencial detrás del concepto de recursión. Para ilustrar la idea paso por paso, definiremos el procedimiento `CONCHA1`, que solo dibuja un cuadrado (ver figura 5.6).

El procedimiento `CONCHA1` mostrado en el algoritmo 5 invoca el procedimiento *mover* cuatro veces para dibujar los lados del cuadrado y el procedimiento *girar* tres veces para poner el lápiz en la dirección adecuada. El parámetro de

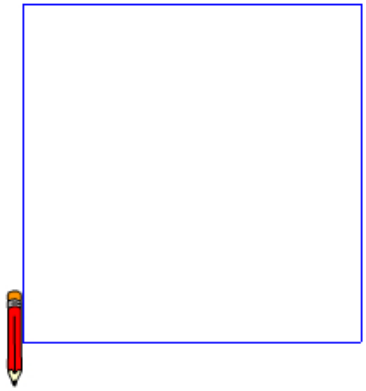


Figura 5.6: Concha 1.

entrada *tamaño* especifica el ancho de los cuatro lados. Hasta aquí todo parece muy simple, sin embargo, pronto se pondrá interesante.

Ahora, definimos el procedimiento `CONCHA2`, que dibuja dos cuadrados (ver figura 5.7).

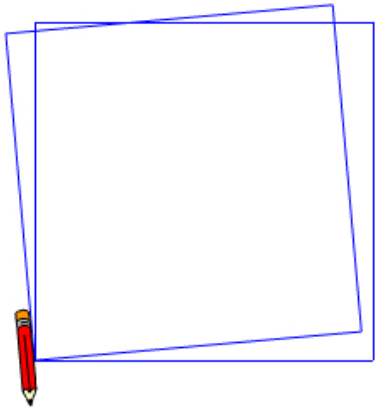


Figura 5.7: Concha 2.

Algoritmo 5 Concha 1

```
1: procedimiento CONCHA1(tamaño)                          //Entrada
2:     mover(tamaño)          //Lado base (el lápiz apunta a 0° al inicio)
3:     girar(90)                                  //Apuntar hacia arriba
4:     mover(tamaño)                                      //Lado derecho
5:     girar(90)                                  //Apuntar hacia la izquierda
6:     mover(tamaño)                                      //Lado superior
7:     girar(90)                                  //Apuntar hacia abajo
8:     mover(tamaño)                                      //Lado izquierdo
9: fin procedimiento
```

El procedimiento `CONCHA2` mostrado en el algoritmo 6 invoca el procedimiento *mover* cuatro veces para dibujar los lados del primer cuadrado y el procedimiento *girar* tres veces para apuntar el lápiz en la dirección adecuada. Posteriormente, invoca el procedimiento *girar* en 95° para apuntar el lápiz ligeramente hacia arriba (5° por arriba del lado base del primer cuadrado). Finalmente, para dibujar el segundo cuadrado el algoritmo invoca nuevamente a los procedimientos *mover* cuatro veces reduciendo el tamaño de la entrada en 97 % y *girar* tres veces.

Nota que la parte donde se dibuja el segundo cuadrado del procedimiento `CONCHA2` es repetitiva. Por esta razón, podemos usar `CONCHA1` para simplificar el procedimiento `CONCHA2`. Tú puedes comparar el procedimiento `CONCHA2A` del algoritmo 7 con el procedimiento `CONCHA2` del algoritmo 6 para convencerte que hacen lo mismo.

El procedimiento dentro de `CONCHA2A` (del algoritmo 7) es el procedimiento `CONCHA1`, no el procedimiento `CONCHA2A`. No es inusual que un procedi-

Algoritmo 6 Concha 2

```
1:  procedimiento CONCHA2(tamaño)                                    //Entrada
2:      mover(tamaño)                                                //Lado base
3:      girar(90)                                                    //Apuntar hacia arriba
4:      mover(tamaño)                                                //Lado derecho
5:      girar(90)                                                    //Apuntar hacia la izquierda
6:      mover(tamaño)                                                //Lado superior
7:      girar(90)                                                    //Apuntar hacia abajo
8:      mover(tamaño)                                                //Lado izquierdo
9:      girar(95)                  //Apuntar cinco grados más para inclinar el lápiz
10:     mover(0.97 * tamaño)                  //Lado base del segundo cuadrado
11:     girar(90)
12:     mover(0.97 * tamaño)                  //Lado derecho del segundo cuadrado
13:     girar(90)
14:     mover(0.97 * tamaño)                  //Lado superior del segundo cuadrado
15:     girar(90)
16:     mover(0.97 * tamaño)                  //Lado izquierdo del segundo cuadrado
17: fin procedimiento
```

Algoritmo 7 *Concha2a* que usa *Concha*1

```
1:  procedimiento CONCHA2A(tamaño)                          //Entrada
2:      mover(tamaño)                                       //Lado base
3:      girar(90)                                           //Apuntar hacia arriba
4:      mover(tamaño)                                       //Lado derecho
5:      girar(90)                                           //Apuntar hacia la izquierda
6:      mover(tamaño)                                       //Lado superior
7:      girar(90)                                           //Apuntar hacia abajo
8:      mover(tamaño)                                       //Lado izquierdo
9:      girar(95)          //Apuntar cinco grados más para inclinar el lápiz
10:     CONCHA1(0.97 * tamaño)                              //Dibuja el segundo cuadrado
11: fin procedimiento
```

miento invoque a otro. Hasta aquí, no se presenta la situación misteriosa de que `CONCHA` invoque a `CONCHA`.

Para dibujar tres cuadrados puedes crear el procedimiento `CONCHA3` que invoque el procedimiento `CONCHA2A`. Después, si deseas cuatro cuadros, definirías el procedimiento `CONCHA4` que invoque el procedimiento `CONCHA3`. Dicha estrategia es demasiado tediosa, porque cada vez que se requiere un cuadro más es necesario definir un nuevo procedimiento.

La recursión nos permite simplificar el problema de dibujar conchas nautilus con n cuadros. La idea esencial es que podemos escribir una función o procedimiento que se invoque a sí mismo dentro de la propia función o procedimiento.

Anteriormente, el procedimiento `CONCHA3` invoca a `CONCHA2A`; el procedimiento `CONCHA2A` invoca a `CONCHA1`. Ahora, la idea es que el procedimiento

`CONCHA` invocará a `CONCHA`, pero con un caso más sencillo, es decir, reduciendo el número de cuadros en 1. No obstante, ¿cuándo se detendrían las invocaciones? La respuesta es que las invocaciones continuarían de manera indefinida. Por esta razón, siempre es necesario incluir un *caso base* para la estrategia recursiva en el que no es necesario hacer una llamada recursiva. *El caso base es el paso donde el proceso recursivo se detiene*.

El caso base de nuestro ejemplo se presenta cuando el valor de *tamaño* es menor que 10, donde solo se debe dibujar un cuadrado como el que se realizó en el procedimiento `CONCHA1`. El procedimiento recursivo final se presenta en el algoritmo 8.

El lector es invitado a consultar el algoritmo implementado en *scratch* en el enlace: `scratch.mit.edu/projects/213924232/`.

Paat te explica la estructura de cualquier procedimiento recursivo de la siguiente manera:

1. En el *caso base* se hace el trabajo sencillo y no es necesario invocar un procedimiento recursivo (líneas 3-9 del algoritmo 8).

2. Si no es el caso base, entonces pensamos: ¡Ay! Está bien difícil, solo haré una pequeña parte del problema (líneas 11-18 del algoritmo 8); el trabajo restante que lo haga alguien más (línea 19 del algoritmo 8).

5.2.2. Sumatoria

En esta sección analizaremos una función recursiva sencilla, definida como `SUMATORIA`(n), con la que realizaremos el seguimiento de cada invocación para entender qué sucede. La función recibe como argumento de entrada un número entero $n \geq 0$ y retorna la suma: $n + (n-1) + (n-2) + ... + 0$. Por ejemplo, si el

Algoritmo 8 Concha nautilus general

```
1:  procedimiento CONCHA(tamaño)                    //Entrada
2:      si tamaño < 10 entonces
3:          mover(tamaño)
4:          girar(90)
5:          mover(tamaño)
6:          girar(90)
7:          mover(tamaño)
8:          girar(90)
9:          mover(tamaño)
10:     si no
11:         mover(tamaño)
12:         girar(90)
13:         mover(tamaño)
14:         girar(90)
15:         mover(tamaño)
16:         girar(90)
17:         mover(tamaño)
18:         girar(95)
19:         CONCHA(0.97 * tamaño)
20:     fin si
21: fin procedimiento
```

procedimiento es invocado con el parámetro 3, entonces `SUMATORIA(3)` retorna 6, que es la suma de 3 + 2 + 1 + 0.

El primer paso para diseñar la función recursiva es pensar en el caso base, es decir, el caso más sencillo donde no es necesario delegar el trabajo a otra invocación recursiva. El caso base se presenta cuando $n = 0$, donde únicamente la función tiene que retornar 0. Ahora tenemos que averiguar qué hacer para números mayores que 0. Supongamos que alguien nos da un número que no sea 0, por ejemplo, el número 712. ¿Qué haríamos? Siguiendo el consejo de Paat:

1. "¡Ay! Está bien difícil, solo haré una pequeña parte del problema". Únicamente sumamos el número $n = 712$ con la sumatoria de $n - 1$.

2. "El trabajo restante que lo haga alguien más". Para realizar la sumatoria restante de $n - 1$ invocamos a la función `SUMATORIA(711)`, es decir, con $712 - 1$ como entrada.

La función recursiva `SUMATORIA` se muestra en el algoritmo 9.

Algoritmo 9 Sumatoria

```
1: función SUMATORIA(n)                          //Entrada
2:     si n = 0 entonces                         //Caso base
3:         devolver 0                            //Salida
4:     si no
5:         devolver n + SUMATORIA(n − 1)         //Salida
6:     fin si
7: fin función
```

¿Qué pasa cuando llamamos a la función con el argumento 3? El diagrama 5.8 muestra lo que sucede. Cada vez que se realiza una invocación se crea una

copia de la función para que realice lo mismo, pero con un número menor. Las flechas están numeradas para mostrar la secuencia de eventos.

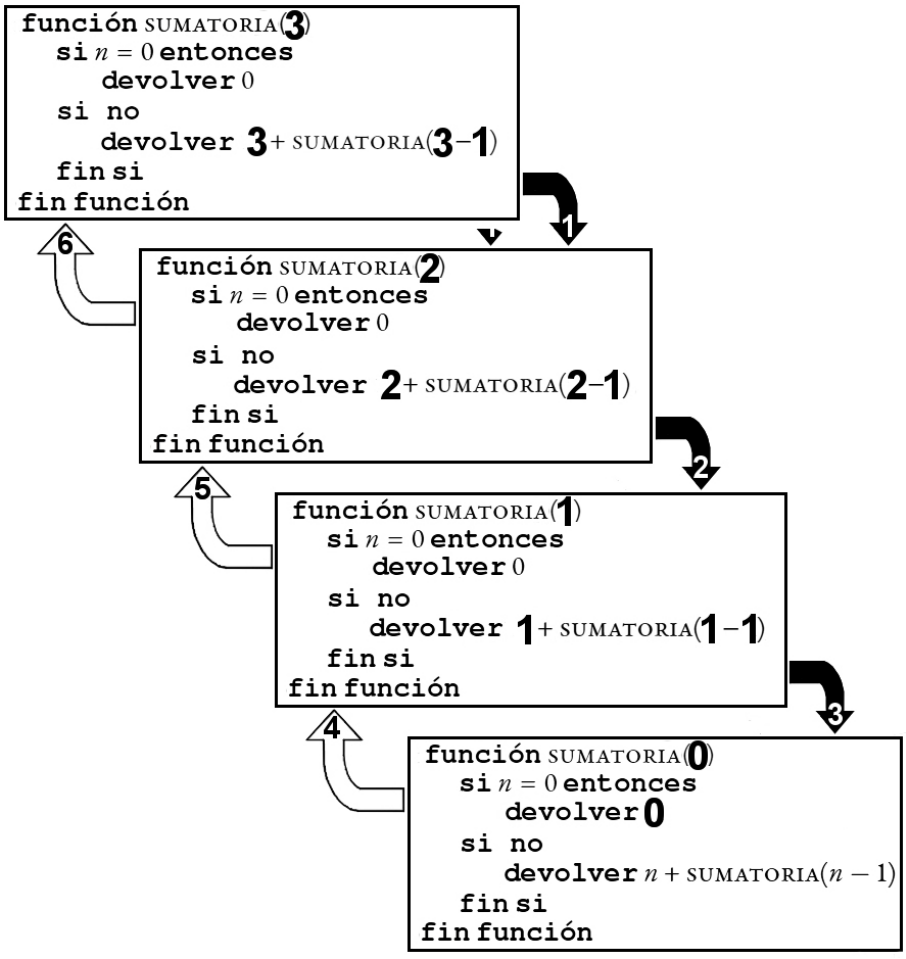


Figura 5.8: Copias recursivas. Los encabezados de los procedimientos muestran el valor del parámetro *n*. Las flechas negras muestran la "delegación" en otra copia del procedimiento (es decir, una invocación recursiva). Las flechas blancas indican la devolución del control al procedimiento que invocó la copia recursiva.

Como observaste en el ejemplo anterior, las soluciones recursivas implican una sobrecarga en memoria porque para cada invocación recursiva se crea una copia de la función. Con un programa no recursivo se puede eliminar dicha sobrecarga. Sin embargo, en ocasiones la vía más lógica y natural de resolver un problema es mediante la recursión.

5.3. Aplicando funciones y recursión

Las funciones son una herramienta poderosa porque permiten ocultar los detalles de su implementación. En el ejemplo de la función que calcula el área del círculo, el usuario de la función puede suponer que realiza la tarea que describe sin preocuparse por conocer cómo la realiza. Por lo tanto, el poder de las funciones es que los usuarios pueden suponer que ejecutan una determinada tarea sin la necesidad de entender los detalles del algoritmo.

El abstraer los algoritmos en funciones permite a los diseñadores la posibilidad de modificar fácilmente los algoritmos utilizados. Si se empieza con una versión muy ineficiente de determinada función, se puede actualizar fácilmente el algoritmo implementado sin cambiar nada que esté fuera de dicha función. La entrada y la salida se mantendrán igual, sólo los pasos para llegar al resultado van a cambiar y no afectarían el algoritmo principal.

Un conductor sólo necesita conocer la interfaz de su vehículo para poder manejarlo; es decir, qué hacen el volante, la palanca de velocidades y los pedales. Piensa en cuánto más difícil sería manejar un coche si tuvieras que entender, antes de poder usarlo, cómo funciona la dirección hidráulica, el motor, etc.

Por ejemplo, considera a un jefe de una empresa. El jefe le pide a un trabajador que realice una tarea (llamada a función) y que informe de los resultados cuando la tarea esté hecha. El jefe no sabe cómo el trabajador la realiza. Incluso, el trabajador podría llamar a otros trabajadores para colaborar en la ejecución de la tarea encomendada y el jefe no se enteraría de esto.

5.4. Ejercicios

1. Modifica la función `ÁREACÍRCULO` para que tome como entrada el diámetro en lugar del radio. Además, modifica el Algoritmo 3 para que imprima en la pantalla las áreas de dos círculos de 14 y 28 cm de diámetro.

2. Escribe una función que reciba cuatro números y regrese el rango, que es la diferencia entre el número más grande y el más pequeño. Por ejemplo, si los números dados son 5, 1, 3 y 9, se obtiene $mayor = 9$ y $menor = 1$, quedando $rango = mayor - menor = 8$.

3. Escribe una función que calcule el volumen de un cono. Las entradas de la función deben ser el radio y la altura. Escribe también un algoritmo que llame a tu función para calcular el volumen de dos conos, el primero de $radio = 3$ cm y $altura = 5$ cm, y el segundo con el doble de estos valores. Recuerda que el volumen de un cono está dado por la fórmula $volumen = (\pi * radio^2 * altura)/3$.

4. Escribe una función para jugar a adivinar un número. La entrada de esta función es el número de intentos que tiene un jugador. La computadora, mediante la función `ALEATORIO`(10), selecciona un número entre 1 y 10. El jugador adivina un número y lo introduce mediante la función

`LEER`(número). La computadora, mediante la función `IMPRIMIR`(mensaje), muestra el mensaje "ganaste" si el jugador adivinó el número o el mensaje "perdiste" en caso contrario.

5. Escribe una función que dibuje un rectángulo de cualquier dimensión. Las entradas de esta función, nombrada `DIBUJARECTÁNGULO`, son la base y la altura. Recuerda que para dibujar cuentas con las funciones mover() y girar().

6. Utiliza la función `DIBUJARECTÁNGULO` para elaborar un dibujo plano de una bandera con asta. El asta debe ser un rectángulo de 1cm de base y 25cm de altura. A la derecha del asta, en la parte superior, coloca la bandera que es un rectángulo de 20cm de base y 5cm de altura.

7. Utiliza la función `DIBUJARECTÁNGULO` para elaborar un dibujo plano de un pastel de tres capas colocadas simétricamente. Considera que todas las capas tienen la misma altura de 5cm. La base de la primera capa, de abajo hacia arriba, es de 20cm; la siguiente de 10cm y la última de 5cm.

8. El algoritmo 8 construye una concha nautilus dibujando cuadros cuyo tamaño se reduce en un 97 % en cada llamada recursiva. El tamaño del cuadro es el único parámetro de entrada del procedimiento recursivo. Modifica este procedimiento para que también reciba de entrada el porcentaje de reducción.

9. El máximo común divisor (mcd) de dos números a y b, no negativos y diferentes de cero, es el entero más grande que los divide sin dejar residuo. Por ejemplo, mcd(48, 60) = 12. El cómputo del mcd(a, b) se puede realizar recursivamente como sigue: si b es 0, entonces la respuesta es a; en otro

caso, la respuesta es mcd(b, a mód b). Recuerda que el operador mód obtiene el residuo de la división, por ejemplo 5 mód $2 = 1$. Escribe en formato de algoritmo la función recursiva mcd(a, b).

10. Construye una tabla para mostrar la ejecución por pasos del cálculo recursivo de mcd(48, 60). Por ejemplo, para mcd(1265, 440) = 55 se obtiene la siguiente tabla.

Nivel de llamada	a	b
1	1265	440
2	440	385
3	385	55
4	55	0

Capítulo 6

Análisis de eficiencia de algoritmos

Estar contigo o no estar contigo es la medida de mi tiempo.
Jorge Luis Borges

Si un problema tiene que resolverse, entonces posiblemente haya varios algoritmos capaces de solucionarlo. Desde luego, debes aplicar el mejor algoritmo. Por lo tanto surge la siguiente pregunta: ¿cuál es el criterio que determina la superioridad de un algoritmo? Por ejemplo, un cocinero seleccionará la mejor receta para preparar un pastel considerando los siguientes criterios: el tiempo de preparación, el costo de los ingredientes o la cantidad de azúcar.

Ahora considera el problema de obtener la suma de la serie de números de 1 a n. Una forma natural para resolver el problema es sumar los números uno a uno. Así, el resultado de la serie 1, 2, 3, 4, 5, 6, 7 es $1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$. Una alternativa diferente es aplicar el algoritmo inventado por Friedrich Gauss. Consiidera el siguiente cuento:

Paat inventó una nave que puede realizar viajes cósmicos en el tiempo mediante el *motor de Alcubierre*; su sueño era conocer a las grandes mentes de la historia de la humanidad. Por esta

razón, el primer viaje que decidió hacer fue al año 1789, cuando Friedrich Gauss, el "príncipe de las matemáticas", era un niño de 12 años.

En esa época, Paat asistió a la clase de matemáticas del profesor Büttner, que estaba molesto por el mal comportamiento de su grupo de primaria. Así que, como castigo, el profesor encomendó a sus alumnos resolver el problema de obtener la suma de la serie de números de 1 a 100. Supuso que les tomaría un buen rato terminarlo. Sin embargo, a los pocos segundos de haber planteado el problema, un niño se levantó y colocó el resultado sobre el escritorio del maestro. Ese niño era Gauss.

Cuando Paat preguntó a Gauss cómo obtuvo el resultado, Gauss le explicó que se dio cuenta que la suma $1 + 2 + 3 + ... + 100$ es equivalente a la suma:

$$\begin{array}{rccccc} & 1 & 2 & 3 & \cdots & 50 \\ + & 100 & 99 & 98 & \cdots & 51 \\ \hline & 101 & +101 & +101 & \cdots & +101 \end{array}$$

Hay 50 pares que suman 101, por tanto, Gauss concluyó que con la operación $50 \times 101 = 5050$ se puede obtener rápidamente la respuesta correcta.

En general, si deseas resolver el problema de obtener la suma de todos los números de 1 a n, debido a que hay $\frac{n}{2}$ pares de la suma $(n+1)$, la fórmula resultante es $\frac{n(n+1)}{2}$.

Imagina que Paat se enfrenta a Gauss en una competencia para saber quién puede resolver en menor tiempo el problema de obtener la suma de los números de 1 a n, teniendo en cuenta que Gauss utiliza su fórmula y Paat realiza la suma uno por uno. ¿En qué casos Paat ganaría, empataría o perdería con los siguientes valores de n?

- ¿Qué pasaría si $n = 2$?
- ¿Qué pasaría si $n = 5$?
- ¿Qué pasaría si $n = 100$?

Probablemente concluiste que Paat ganaría con $n = 2$, Paat empataría con $n = 5$ y Gauss ganaría con $n = 100$. Además, quizá notaste que a mayor cantidad de datos, Gauss sería definitivamente el vencedor. *El tiempo que demora un algoritmo en calcular la solución de un problema, especialmente considerando grandes conjuntos de datos, es un criterio que determina su superioridad*. En el caso del problema de obtener la suma de 1 a n, el método de Gauss es superior a sumar cada número uno a uno.

La principal razón de construir computadoras es que procesan cálculos de una manera más rápida y precisa que el ser humano. Además, dicha superioridad es notable cuando se procesan grandes cantidades de datos. Las computadoras son rápidas, pero *no son lo suficientemente rápidas*. Por esta razón, los científicos e ingenieros de la computación realizan continuamente esfuerzos por descubrir algoritmos que se ejecuten de manera rápida. Imagina qué tan diferente sería un buscador de páginas web si demorara en entregar los resultados en el orden de minutos u horas.

6.1. Medición de la eficiencia

Al igual que en las recetas, los algoritmos tienen diferentes ventajas y desventajas relacionadas, principalmente, con el *tiempo de ejecución* y el *espacio en memoria*.

- *El tiempo de ejecución es la cantidad de tiempo que tarda un algoritmo en encontrar soluciones.*
- *El espacio en memoria es la cantidad de almacenamiento requerido por un algoritmo.*

Normalmente, el tiempo de ejecución es considerado el criterio más importante para determinar la eficiencia de un algoritmo cuando es analizado. *La eficiencia se mide por el tiempo de ejecución que tarda un algoritmo en completar la tarea*. Un algoritmo es superior a otro si es más eficiente.

Para un algoritmo, una *instancia* es cada uno de los ejemplares que pertenecen a su *dominio de definición*. El número de instancias pertenecientes al dominio de definición de un algoritmo puede ser finito o infinito. Por ejemplo, considera el problema de multiplicar dos enteros positivos. Una instancia de dicho problema es la multiplicación de los enteros positivos 274 y 382, y su dominio de definición es todo el conjunto de los enteros positivos. Nota que dicho conjunto tiene una cantidad infinita de instancias. Sin embargo, si la multiplicación se restringe solo a los números dígitos (0, 1, 2, 3, 4, 5, 6, 7, 8 y 9) el dominio de definición es finito.

Un algoritmo es correcto si funciona perfectamente para todas las instancias que el algoritmo declara resolver. En el caso de la multiplicación: para todo el conjunto de enteros positivos. *El tamaño de una instancia es indicado por un número entero que mide de manera significativa la magnitud de sus componentes*. En la multiplicación, el tamaño de la instancia estará en términos del número de dígitos que contengan los números pertenecientes al ejemplar. El tamaño de la instancia 274 y 382 es 6.

Si en un problema únicamente se tiene que resolver un limitado número de instancias y de pequeño tamaño, entonces es suficiente con seleccionar el algoritmo que sea más sencillo de implementar, sin preocuparse por sus propiedades teóricas. Sin embargo, si el problema exige resolver grandes conjuntos de instancias, el análisis de qué algoritmo es el más adecuado debe hacerse de manera cuidadosa.

6.2. Tiempo de ejecución

El tiempo de ejecución está relacionado con el tiempo que le toma a un algoritmo encontrar la solución de un determinado problema, y es el criterio que normalmente decide qué algoritmo es el mejor. Para seleccionar un algoritmo se usan dos enfoques: el *enfoque empírico* y el *enfoque teórico*.

El enfoque empírico consiste en implementar cada algoritmo en un lenguaje de programación y realizar pruebas con numerosas instancias del problema: el algoritmo que obtenga los mejores resultados será el seleccionado. La desventaja del enfoque empírico es que hay una gran diversidad de computadoras con capacidades de procesamiento distintas que toman diferentes cantidades de tiempo al realizar las mismas operaciones. En consecuencia, hace imposible referirse a una medida de tiempo general, por ejemplo, en segundos.

El enfoque teórico determina matemáticamente la cantidad de recursos requeridos considerando el tiempo de ejecución como la cantidad de operaciones elementales que realiza un algoritmo en función del tamaño de la instancia de entrada. Se considera una *operación elemental* aquella que toma una cantidad de tiempo constante, como lo son la suma, la multiplicación, la división, etc. El tamaño de la instancia es una propiedad del algoritmo independiente de la computadora en la que se ejecuta. En el problema de sumar la serie de números de 1 a n, el tamaño de la instancia es n, es decir, la cantidad de números que se deben sumar. En el caso de una multiplicación el tamaño de la entrada está en función del número de dígitos que contienen los operandos.

A partir del análisis de las operaciones elementales que ejecuta un algoritmo se estima su eficiencia mediante su aproximación por los *órdenes* de complejidad, que revelan la cantidad de recursos de cómputo requeridos por el algoritmo. El algoritmo que tenga órdenes menores será seleccionado. Para designar el orden

de un algoritmo se utiliza la notación asintótica O, que indica el comportamiento de los algoritmos en el límite. Es decir, para valores suficientemente grandes en el tamaño de sus instancias. Los órdenes que frecuentemente se presentan en el análisis de los algoritmos son: *constante*, *lineal*, *polinómico* y *logarítmico*, denotado como $O(1)$, $O(n)$, $O(n^c)$, $O(log\ n)$, respectivamente.

6.3. Tiempo constante

Los algoritmos de tiempo constante responden en un tiempo constante independientemente del tamaño de la instancia de entrada y se denotan como $O(1)$. El algoritmo de Gauss, que resuelve el problema de sumar los números de 1 a n, es un algoritmo de *tiempo constante* porque no tiene que sumar todos los números que están en la serie 1 a n. El algoritmo sólo necesita la cantidad de números en el conjunto. Independientemente del valor de n, ya sea 10 o 10, 000, 000 números, la fórmula $\frac{n(n+1)}{2}$ siempre ejecuta 3 operaciones elementales: una suma, una multiplicación y una división.

Como ya se mencionó, normalmente cualquier operación aritmética elemental, como la suma, la resta, la multiplicación, la división y la exponenciación (en algunos casos), es considerada una operación de tiempo constante.

Los algoritmos de tiempo constante son los más codiciados por los científicos de la computación. Por desgracia, la mayoría de los algoritmos no son de tiempo constante.

6.4. Tiempo lineal

Los algoritmos de tiempo lineal responden en un tiempo lineal con respecto al tamaño de la instancia de entrada y se denotan como $O(n)$. El algoritmo de la

sumatoria de la serie 1 a n, donde los números se suman de uno a uno (sin usar la estrategia de Gauss), es un algoritmo de tiempo lineal. Si la lista de números es mayor, entonces la solución tardará más tiempo en calcularse porque deben sumarse más números. Sin embargo, estará linealmente acotada en función de la entrada.

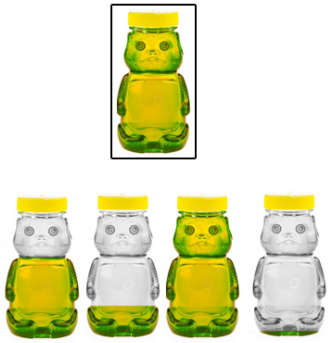
Los algoritmos de *tiempo lineal* también son muy codiciados en ciencias de la computación. Para muchos problemas, son los algoritmos más rápidos que se pueden encontrar.

6.4.1. Búsqueda Lineal

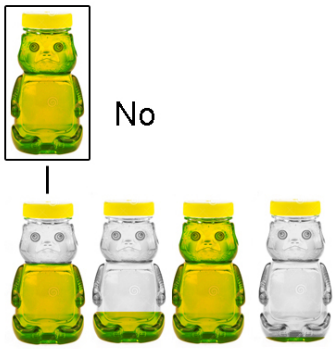
El algoritmo de búsqueda lineal consiste en comparar cada uno de los elementos de la lista hasta encontrar el elemento buscado o llegar al final de la lista. La búsqueda lineal aplica un enfoque de "fuerza bruta" porque es la manera más simple e intuitiva de buscar un elemento en una lista de elementos. Nota que si hay más elementos en la entrada, entonces será mayor el tiempo necesario para encontrar un determinado elemento.

Por ejemplo, el objetivo es encontrar el envase lleno entre el grupo de cuatro envases denotado como A, que se muestra en la figura 6.1.

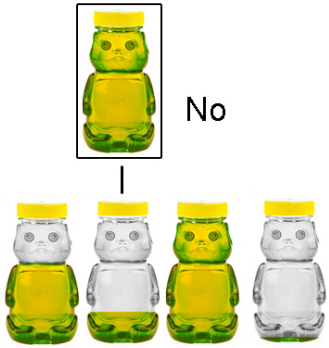
Las computadoras procesan información muy rápido. Por esta razón, se podría pensar que la búsqueda lineal sería suficiente para encontrar de manera práctica elementos en una lista. Sin embargo, el algoritmo de búsqueda lineal es muy lento incluso para las computadoras. Por ejemplo, si un supermercado tiene 10, 000 productos y un cajero escanea el código de barras de un producto, entonces la computadora debe buscar en una lista de 10, 000 códigos para encontrar el nombre y precio del producto. Si la computadora únicamente tardara una milésima de segundo en revisar cada código, entonces se necesitarían



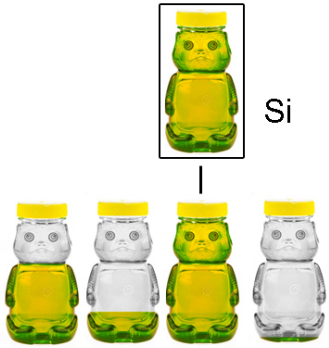
(a) El envase que se desea buscar en el grupo *A* se encuentra en la parte superior.



(b) El envase es comparado con el envase de la posición 1. El resultado es *falso* porque el envase de la posición 1 no es el envase a buscar.



(c) El envase es comparado con el envase de la posición 2. El resultado es *falso* porque el envase de la posición 2 no es el envase a buscar.



(d) El envase clave es comparado con el envase de la posición 3. El resultado es *verdadero* porque el envase de la posición 3 es el envase a buscar.

Figura 6.1: Algoritmo de *Búsqueda Lineal*.

10 segundos para recorrer toda la lista. Imagina el tiempo que el cajero tardaría en cobrar las compras de tu familia.

6.5. Tiempo polinómico

Los algoritmos de tiempo polinómico responden en un tiempo polinómico con respecto al tamaño de la instancia de entrada y se denotan como $O(n^c)$. Cuando c es 2 se denomina *tiempo cuadrático*, cuando n es 3 se denomina *tiempo cúbico* y, en general, tiempo polinómico. Intuitivamente podríamos decir que el tiempo polinómico es el último de los tiempos aceptables en la práctica. Los algoritmos con tiempos más grandes, como lo son tiempo exponencial $O(c^n)$, tiempo factorial $O(n!)$ y tiempo combinatorio $O(n^n)$, son demasiado ineficientes, sobre todo cuando n se incrementa.

En la siguiente sección examinaremos un algoritmo de ordenamiento en tiempo cuadrático denominado *ordenamiento por inserción*.

6.5.1. Ordenamiento por inserción

Los diccionarios, los índices de los libros y los directorios telefónicos se encuentran ordenados alfabéticamente. Si la información no estuviera ordenada, entonces nuestras labores cotidianas serían mucho más difíciles de realizar. ¡Qué difícil sería localizar un número telefónico si los nombres no se encontraran ordenados! Por esta razón, se han creado diversos algoritmos de ordenamiento para organizar la información.

Imagina que tienes diez cartas numeradas del 1 al 9 en orden aleatorio (al azar): $\{3, 5, 1, 9, 4, 2, 7, 8, 6\}$. Ahora, el objetivo es organizar las cartas en orden ascendente: $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Un método intuitivo para ordenar las cartas

es formar dos grupos: el grupo de las cartas desordenadas y el grupo de las cartas ordenadas. Al comienzo, únicamente tendrías un grupo con todas las cartas desordenadas. La idea es colocar un elemento a la vez en el grupo de las cartas ordenas hasta que contenga todos los números ordenados (ver tabla 6.1).

Tabla 6.1: Ordenamiento de cartas.

Grupo desordenado	Grupo ordenado	Comparaciones
$\{3, 5, 1, 9, 4, 2, 7, 8, 6\}$	$\{\}$	-
$\{5, 1, 9, 4, 2, 7, 8, 6\}$	$\{3\}$	0 (3 con ninguno)
$\{1, 9, 4, 2, 7, 8, 6\}$	$\{3, 5\}$	1 (5 con 3)
$\{9, 4, 2, 7, 8, 6\}$	$\{1, 3, 5\}$	2 (1 con 5 y 3)
$\{4, 2, 7, 8, 6\}$	$\{1, 3, 5, 9\}$	1 (9 con 5)
$\{2, 7, 8, 6\}$	$\{1, 3, 5, 9\}$	3 (4 con 9, 5 y 3)
$\{7, 8, 6\}$	$\{1, 3, 4, 5, 9\}$	5 (2 con 9, 5, 4, 3 y 1)
$\{8, 6\}$	$\{1, 2, 3, 4, 5, 9\}$	2 (7 con 9 y 5)
$\{6\}$	$\{1, 2, 3, 4, 5, 7, 9\}$	2 (8 con 9 y 7)
$\{\}$	$\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$	4 (6 con 9, 8, 7 y 5)

El número de comparaciones que se realizaron es $0+1+2+1+3+5+2+2+4 = 20$. En el *peor de los casos*, es decir, si las cartas estuvieran en orden inverso $\{9, 8, 7, 6, 5, 4, 3, 2, 1\}$, entonces el número de comparaciones es $1+2+3+4+5+6+7+8$. Es decir, podemos utilizar la fórmula de Gauss, $\frac{n(n+1)}{2} = \frac{9(9+1)}{2} = 36$.

En el peor de los casos y para un número de elementos n, son necesarias $\frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$ comparaciones en total. El término mayor es de orden cuadrático, por lo tanto el algoritmo de inserción es de tiempo cuadrático y se denota como $O(n^2)$.

El algoritmo de ordenamiento *inserción* es un algoritmo ineficiente, pero útil como introducción a la enseñanza de algoritmos de ordenamiento debido a la simplicidad de su implementación. Considera un grupo de envases A, como el que se muestra en la figura 6.2.

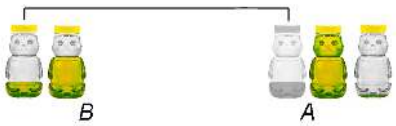
Figura 6.2: Grupo A inicialmente desordenado.

A continuación se describe cómo funciona el algoritmo por inserción:

- Extrae el primer envase del grupo A y forma un nuevo grupo B con dicho envase.
- Extrae el siguiente envase del grupo A e insértalo en la posición correcta del grupo B.
- Repite el proceso hasta que ya no queden más envases en el grupo A.

Ver el ejemplo mostrado en la figura 6.3.

(a) El primer envase del grupo *A* es colocado en la posición 1 del grupo *B*. Un grupo con un solo envase ya se encuentra insertado en su posición correcta.



(b) El siguiente envase del grupo *A* es insertado en la posición número 1, la posición correcta, del grupo *B*.

(c) El siguiente envase del grupo *A* es insertado en la posición número 3 del grupo *B*.

(d) El siguiente envase del grupo *A* es insertado en la posición número 1 del grupo *B*. El procedimiento termina porque ya no quedan más envases en el grupo *A*.

Figura 6.3: Ordenamiento por inserción.

6.6. Tiempo logarítmico

Los algoritmos de tiempo logarítmico responden en un tiempo logarítmico con respecto al tamaño de la instancia de entrada y se denotan como $O(log\ n)$. El tiempo logarítmico crece muy lentamente. Los tiempos logarítmicos son los inversos de los tiempos exponenciales, que crecen muy rápidamente, de modo que si $log_2 n = x$, entonces $n = 2^x$. Ejemplo: como $log_2 128 = 7$, sabemos que $2^7 = 128$.

En la siguiente sección analizaremos otro algoritmo de búsqueda denominado *búsqueda binaria*, que está en tiempo logarítmico. El requisito de dicho algoritmo es que la lista de números esté previamente ordenada.

6.6.1. Búsqueda binaria

El algoritmo de búsqueda binaria consiste en comparar el elemento que se desea encontrar con el elemento central de la lista. Si los elementos coinciden, entonces el algoritmo termina porque encontró el elemento deseado. En caso contrario, el algoritmo continúa su búsqueda en la mitad de la lista donde puede encontrarse el elemento deseado. Por ejemplo, si buscas la página de un libro, entonces puedes abrirlo a la mitad (más o menos) y decidir de qué lado está la página que buscas. En la mitad seleccionada repites el procedimiento, es decir, la partes por la mitad y decides en qué mitad está la página que buscas. Con seguridad, puedes ignorar la otra mitad.

El tiempo de ejecución no es constante. Si la lista es mayor, entonces más elementos tienen que verificarse. El tiempo de ejecución tampoco es lineal, el algoritmo termina sin tener que verificar todos los elementos de la lista. El tiempo de ejecución del algoritmo de búsqueda binaria debe estar en algún punto intermedio.

Contemos el número de divisiones que se tienen que realizar para distintos tamaños de páginas de un libro. Si el libro sólo tiene cuatro páginas, entonces tenemos que realizar como máximo dos divisiones antes de encontrar el número que estamos buscando, o para decidir que el número de la página no está en el libro. En un libro de tamaño ocho, después de la primera división, sólo se necesita buscar en la mitad del libro de tamaño cuatro. Por esta razón, tenemos que realizar como máximo tres divisiones antes de terminar, como acabamos de

indicar, incluyendo la primera. Así mismo, para un libro con 16 páginas, después de la primera división se buscaría en sólo ocho páginas; es decir, sólo se tienen que hacer como máximo cuatro divisiones (ver tabla 6.2).

Tabla 6.2: Número de divisiones en la búsqueda binaria.

Tamaño de la lista	4	8	16	32
Número de divisiones	2	3	4	5

Observa que si el tamaño de la entrada se duplica, entonces el número de divisiones (y por lo tanto el tiempo de ejecución del algoritmo) se incrementa en uno. Además, observa que la fila superior está creciendo mucho más rápido que la fila inferior. Este comportamiento es similar al de la función logarítmica, donde cada incremento multiplicativo en la entrada se convierte en un aumento aditivo en la salida. Tal crecimiento se denomina logarítmico. Por esta razón, el algoritmo de búsqueda binaria es de *tiempo logarítmico*.

Los algoritmos de tiempo logarítmico también son muy codiciados debido a que sus tiempos de ejecución se encuentran cerca del tiempo constante. Si se aumenta el tamaño de las entradas por un factor grande, el tiempo de ejecución se incrementará en una cantidad pequeña, lo que los hace muy valiosos.

Retomando el ejemplo de la sección 6.4.1 del supermercado, la búsqueda lineal encontrará el producto deseado en una lista de 10, 000 productos en un tiempo máximo de 10 segundos. Ahora, por medio de la búsqueda binaria el producto deseado puede encontrarse en la lista de 10, 000 productos sólo con catorce pruebas, que equivale a dos centésimas de segundo.

6.6.2. QuickSort

El algoritmo de ordenamiento *quicksort* es más rápido que el ordenamiento por inserción, en particular con listas muy largas. Es uno de los mejores algoritmos de ordenamiento que se conocen. Considera el siguiente grupo de envases A mostrado en la figura 6.4.

Figura 6.4: Grupo A inicialmente desordenado.

A continuación se describe cómo funciona *quicksort*:

- Selecciona el envase que se encuentra en el extremo derecho, dicho envase se denomina pivote.
- Compara el pivote con cada uno de los envases restantes y coloca aquellos que son más livianos a la izquierda y los envases más pesados a la derecha, formando el grupo I y el D, respectivamente. Idealmente, el pivote quedará cerca del centro, pero es posible que termines con muchos más envases en un lado que en el otro.
- Aplica el mismo procedimiento en el grupo I y D. Repite el procedimiento en cada nuevo grupo I y D que vayas formando hasta que ninguno de los grupos tenga más de un envase. Cuando todos los grupos hayan sido divididos en grupos de un solo envase, entonces los envases estarán ordenados del más liviano al más pesado.

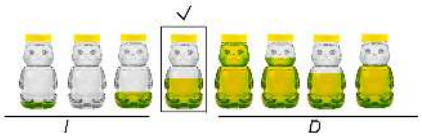
Ahora, observa el ejemplo mostrado en las figuras 6.5 y 6.6.

(a) El grupo *A* se encuentra inicialmente desordenado.

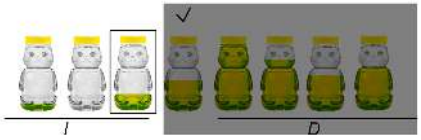
(b) El último envase del grupo *A* se selecciona como pivote.

(c) El primer envase se mantiene en el lado izquierdo del pivote porque es menor.

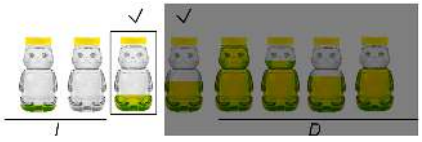
(d) El segundo envase se desplaza al lado derecho del pivote porque es mayor.



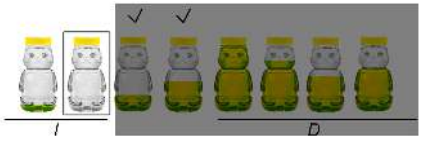
(e) Al comparar todos los envases, los grupos *I* y *D* tienen los envases menores y mayores al pivote, respectivamente. El pivote está en su posición y ya no interviene.



(f) El *quicksort* se vuelve a aplicar en el grupo *I*. El último envase del grupo *I* se selecciona como pivote. Por el momento, el grupo *D* ya no interviene.

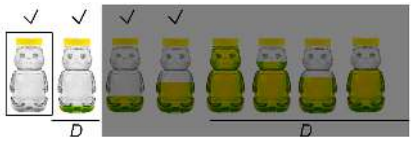


(g) Solo se formó el grupo *I* porque solo hubo envases menores. El pivote está en su posición correcta y ya no interviene.

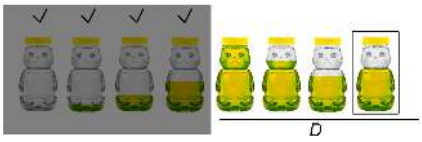


(h) El *quicksort* se vuelve a aplicar en el grupo *I*. El último envase del grupo *I* se selecciona como pivote.

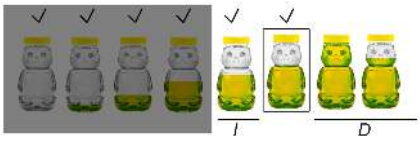
Figura 6.5: Algoritmo de ordenamiento *quicksort*.



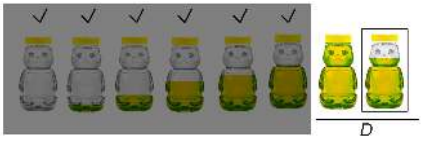
(a) Después de comparar el envase restante se formó el nuevo grupo *D*. No se aplica el *quicksort* porque D solo tiene un envase y el pivote, los dos elementos están en su posición correcta.



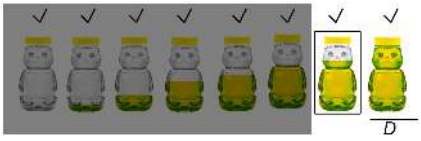
(b) El *quicksort* se vuelve a aplicar en el grupo *D*. El último envase del grupo *D* se selecciona como pivote. Los envases del lado izquierdo estan ordenados y no se toman en cuenta.



(c) Después de comparar los envases restantes se formó el grupo *I* y el *D*. El pivote y el envase del grupo *I* estan en su posición correcta.



(d) El *quicksort* se vuelve a aplicar en el grupo *D*. El último envase del grupo *D* se selecciona como pivote.



(e) Después de comparar el envase restante se formó el nuevo grupo *D*. No se aplica el *quicksort* porque D solo tiene un envase y el pivote, los dos elementos están en su posición correcta.

(f) El *quicksort* finaliza porque no hay grupos a ordenar. El grupo *A* está ordenado en su totalidad.

Figura 6.6: Algoritmo de ordenamiento *quicksort* (continuación).

6.7. Aplicando el análisis de algoritmos

Como habrás notado, hay una gran variedad de maneras para resolver un problema. En una situación particular, tú puedes desarrollar la habilidad para decidir el mejor algoritmo que solucione un determinado problema de acuerdo al análisis de las ventajas y desventajas que ofrezca. Por ejemplo, si el problema implica el manejo de pocos datos, quizá puedas aplicar algoritmos que sean fáciles de implementar. Sin embargo, si es necesario procesar grandes conjuntos de datos, entonces lo más conveniente sería implementar algoritmos eficientes.

Las computadoras almacenan grandes cantidades de información y necesitan poder encontrarla rápidamente. Uno de los problemas de búsqueda más desafiantes se encuentran en los motores de búsqueda en Internet, que deben revisar miles de millones de páginas web en fracciones de segundo.

Las computadoras ejecutan de manera repetida algoritmos de ordenamiento sobre la información, como cuando sus archivos se encuentran ordenados por nombre. Por tal motivo, los científicos de la computación han dedicado gran esfuerzo en descubrir algoritmos de ordenamiento más eficientes. Algunos de los algoritmos más lentos, como el ordenamiento por inserción, pueden ser útiles en situaciones especiales. Sin embargo, usualmente se utilizan algoritmos más rápidos, como el *quicksort*.

Con respecto a las actividades que realizas a diario, sé innovador y busca alternativas para realizar tus tareas de manera más eficiente a fin de minimizar el tiempo que te toma realizarlas.

6.8. Complejidad

Como hemos visto en las secciones anteriores, el tiempo de ejecución de un algoritmo se puede determinar en función del tamaño de la instancia de entrada, y puede ser constante, logarítmico, lineal o polinomial. Todos estos algoritmos se pueden resolver eficientemente en una computadora. Sin embargo, existen otros algoritmos cuyo tiempo de ejecución aumenta en forma exponencial en función del tamaño de la instancia de entrada. En este caso, el tiempo aumenta muy rápidamente. Por ejemplo, si el tiempo aumenta en el orden de N^2, donde N es el tamaño de la instancia de entrada, para $N = 10$ el tiempo ya es del orden de 1, 000, y para $N = 20$, del orden de 1, 000, 000, y así crece muy rápidamente; de forma que si N es muy grande tomaría mucho tiempo incluso en una computadora muy poderosa.

Hay problemas para los cuales no se han encontrado algoritmos eficientes (polinomiales), que se conocen como problemas *NP*, por lo que no pueden ser resueltos en forma general por una computadora. En estos casos se restringen a casos de tamaño de la instancia de entrada no muy grande, o se buscan algoritmos que encuentren una solución aproximada en un tiempo razonable.

6.9. Ejercicios

1. Encontrar el mínimo. Un algoritmo muy popular se basa en encontrar el elemento mínimo (o más pequeño) de todos los elementos de una lista.

   (a) Si la lista estuviera ordenada, ¿qué algoritmo se te ocurre para encontrarlo?

(b) En el caso de que la lista esté desordenada, ¿qué algoritmo se te ocurre para encontrarlo?

(c) Determina el orden del algoritmo que propones para los incisos anteriores.

2. Encontrar el elemento central. Imagina que tienes una lista de elementos y quieres encontrar el elemento central de esa lista (eso se llama mediana).

   (a) Diseña un algoritmo para el caso en que la lista esté desordenada.

   (b) Diseña un algoritmo para el caso en que la lista esté ordenada.

   (c) Determina el orden del algoritmo que propones para los incisos anteriores.

3. Un algoritmo de ordenamiento muy intuitivo consiste en encontrar el elemento más pequeño de todos y ponerlo en primer lugar, hacer lo mismo para el resto de los elementos (es decir, con los que sobran, buscas el elemento mas pequeño, o sea el que debería ir en la 2a posición, etc). Este algoritmo se conoce como *Ordenamiento por Selección*. Determina el orden de este algoritmo.

4. Hay un algoritmo de ordenamiento que seguramente lo haz aplicado, divides tu lista de elementos en listas más pequeñas, y te encargas de cada una por separado. Por ejemplo: imagina que tienes documentos de muchas personas (algunos cientos) y debes ordenarlos en forma alfabética. Una forma de hacer esta división es, agrupas los nombres que empiecen con alguna letra entre la A y la D, el segundo grupo sería con nombres cuya primer letra esté entre la E y H, etc. Este algoritmo se conoce como

Ordenamiento por Cubetas. Una vez que tengas los grupos, ¿qué algoritmo de ordenamiento aplicarías?

5. Algoritmo de Ordenamiento por Mezcla. Este algoritmo de ordenamiento consiste en dividir la lista de elementos en dos sublistas (escogidos aleatoriamente). Te encargas de ordenar cada sublista de manera independiente. Una vez que tengas ambas sublistas ordenadas entonces las mezclas, de la siguiente manera.

   (a) Comparas el menor elemento de ambas sublistas y ves cual es el menor, ese será el menor de todos los elementos de la lista completa. ¿Por qué esta aseveración es cierta?

   (b) Una vez que pasaste el inciso anterior, repites el proceso con ambas sublistas. De esta manera tendrás toda la lista ordenada. ¿Por qué este algoritmo dejará ordenada a toda la lista?

Capítulo 7

Simulación

No se trata sólo de prever el futuro, sino de hacerlo posible.
Antoine de Saint-Exupéry

Jeanette Wing sostiene que el pensamiento computacional complementa y combina el pensamiento matemático y la ingeniería, porque se basa en las matemáticas como sus fundamentos y recurre a la ingeniería por la interacción de los sistemas con el mundo real. Estos sistemas están limitados por la física del dispositivo subyacente, pero por medio de la computadora se pueden construir *simulaciones* o *mundos virtuales* sin las restricciones de la realidad física.

La simulación computacional es la conjunción de algoritmos matemáticos que modelan el comportamiento dinámico de sistemas físicos y herramientas computacionales que permiten reproducir y visualizar esta dinámica.

La simulación computacional es una herramienta científica que, al inicio, fue utilizada en la meteorología y en la física nuclear después de la Segunda Gue-

rra Mundial. Posteriormente, su uso se extendió a la astrofísica, física de partículas, ciencia de los materiales, ingeniería, mecánica de fluidos, climatología, biología evolutiva, ecología, economía, teoría de decisiones, medicina, sociología, epidemiología, diseño de fármacos, etc. Inclusive, algunas disciplinas crecieron significativamente a partir del uso de esta herramienta, como la teoría del caos y sistemas complejos.

En el libro *Los sueños de la razón* del físico Heinz Pagels se describe el potencial de la computadora como instrumento científico. También, considera a la computadora como el instrumento de la próxima generación de científicos y la base de "la segunda parte de la Revolución Científica". Heinz Pagels argumenta que, históricamente, los grandes avances de la ciencia han estado asociados con la invención de un instrumento que permita una nueva forma de aproximarse a la verdad de la naturaleza. Por ejemplo: el telescopio, que reveló la estructura a gran escala del universo, o el microscopio, que reveló los componentes imperceptibles del mundo material. Análogamente, consideraba que los instrumentos anteriores describían lo que era muy grande o muy pequeño para ser percibido a simple vista, y que la computadora nos proporciona los medios para simular lo que es demasiado complejo para ser entendido únicamente con la mente. Heinz Pagels estuvo en lo cierto. Actualmente, la simulación por computadora ha sido designada como el tercer pilar de la ciencia, junto con la teoría y la experimentación clásica.

Para Heinz Pagels el uso de la computadora como un nuevo instrumento científico no era simplemente instalar una computadora en un laboratorio para ayudar a realizar cálculos, sino crear un nuevo laboratorio dentro de la computadora basado en simulaciones. En el "mundo interno" de la computadora la imaginación del programador adquiere una "vida propia" porque, al contrario

de una novela literaria o una película, un programa por computadora se desarrolla de acuerdo a su propia lógica interna. El programador, en múltiples ocasiones, no podrá pronosticar lo que hará el programa no porque no tenga idea de las órdenes que escribió, sino porque el resultado de la interacción entre dichas órdenes es complejo y la única alternativa es ejecutar el programa para visualizar los resultados (ver Figura 7.1). Por dicha razón, el concepto de simulación es una de las principales ideas del pensamiento computacional en el proceso de solución de problemas.

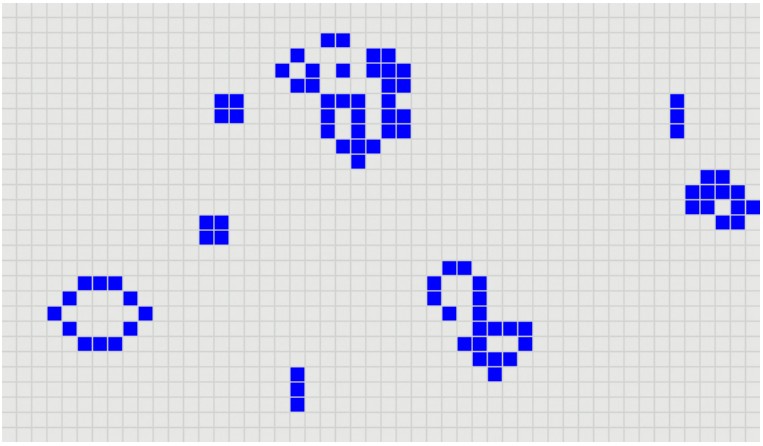


Figura 7.1: *El juego de la vida* es un autómata celular diseñado por el matemático británico John Horton Conway en 1970. Desde su publicación, ha atraído mucho interés debido a la gran variabilidad de la evolución de los patrones. Es interesante para los científicos, matemáticos, economistas y otros observar cómo patrones complejos pueden surgir de reglas muy sencillas.

Además, para determinados problemas no es posible un método de experimentación clásica. Por ejemplo: investigar el fenómeno de una estrella cuando llega al final de su vida y explota como una supernova. Un método experimental no es posible porque no puedes explotar una estrella en un laboratorio. Otra alternativa es que ocurra el evento en la realidad para ser estudiado. Sin embargo,

las supernovas son poco frecuentes y, afortunadamente, ¡ocurren a distancias astronómicas! Los métodos teóricos también tienen una utilidad limitada porque, aunque los procesos físicos básicos de la estrella se entienden, la integración de los diversos procesos en un análisis matemático se vuelve complejo. En este punto es donde la computadora es el instrumento clave, pues es posible implementar un modelo de una supernova y ejecutar la simulación en la computadora. Si el modelo de la estrella explota en forma similar a una estrella, entonces se tiene cierta confianza de que el modelo es correcto.

7.1. Simulación computacional y modelado

Un modelo es una representación abstracta (matemática, declarativa, visual, etc.) de fenómenos, sistemas o procesos. El humano ha podido plantear leyes o modelos que representan la esencia de los fenómenos a fin de comprenderlos matemáticamente o mediante simulaciones por computadora. La simulación computacional nos permite simular modelos y crear animaciones. Dichas animaciones se basan en la geometría analítica, álgebra matricial, trigonometría y las operaciones que comúnmente conoces de suma, resta, multiplicación y división.

En las siguientes secciones analizaremos dos modelos de la naturaleza: *cuerpos en caída libre* y *movimiento colectivo de cardúmenes*, y los simularemos en el lenguaje de programación *scratch*.

7.1.1. Cuerpos en caída libre

La caída de un cuerpo debido a la atracción gravitacional de la Tierra es un ejemplo conocido de movimiento con aceleración constante. Los experimentos muestran que, si puede omitirse el efecto del aire, todos los objetos caen con la misma aceleración hacia abajo sea cual fuere su tamaño o peso. El modelo idealizado de movimiento con aceleración constante se denomina *cuerpos en caída libre*. La ecuación (representación matemática) de los *cuerpos en caída libre* es:

$$y = y_0 + v_{0y}t + \frac{1}{2}a_y t^2, \tag{7.1}$$

donde se asume que el objeto cae libremente sobre el eje vertical y. Las variables y_0 y v_{0y} son la posición y la velocidad inicial. La aceleración a_y esta dada por el valor de la gravedad cerca de la superficie terrestre y t es el tiempo transcurrido en segundos.

Ahora, imagina que Tukkul deja caer una piedra en el cenote Ik Kil, la piedra parte del reposo y cae libremente. A fin de establecer una referencia, Tukkul toma como origen el punto de partida O y la dirección hacia arriba como positiva (ver figura 7.2). Por esta razón, la aceleración es hacia abajo, en la dirección negativa, así que $a_y = -9.8 m/s^2$ que es el valor aproximado de la gravedad cerca de la superficie terrestre. Si Tukkul deseara conocer la posición de la piedra después de 1s, 2s y 3s, entonces ¿qué puede hacer?

Una estrategia es utilizar el modelo de caída libre para pronosticar la posición de la piedra. Inicialmente, y_0 y v_{0y} son ambas cero porque la piedra parte del origen y está en reposo. Como y_0 y v_0 son ambas cero la ecuación resultante es:

$$y = \frac{1}{2}a_y t^2. \tag{7.2}$$

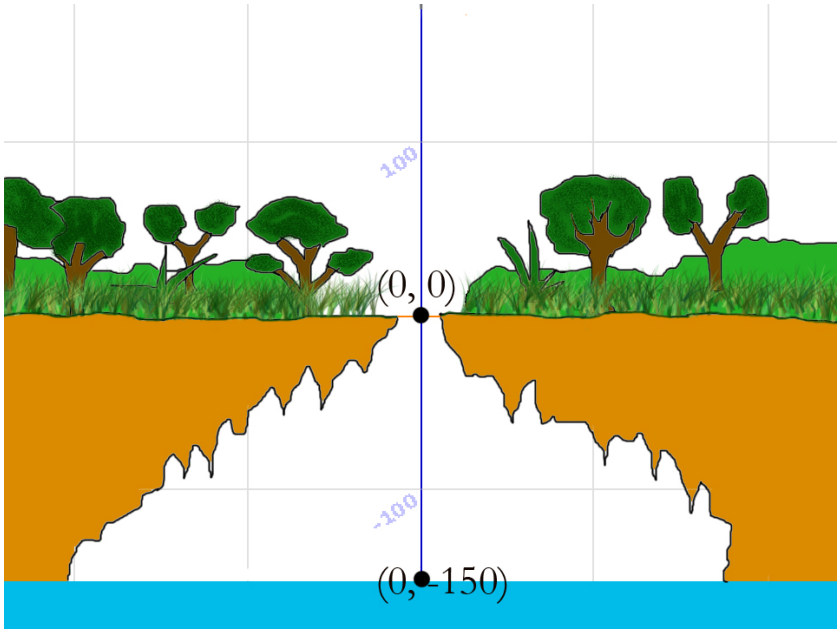


Figura 7.2: Representación esquemática del problema de caída libre.

Cuando el valor de $t = 1$, entonces $y = \frac{-9.8m/s^2}{2} 1s^2 = -4.9$m. Por lo tanto, la piedra se encuentra a 4.9m debajo del origen después de que Tukkul la dejó caer. La posición en los segundos 2s y 3s se obtiene de la misma manera. En $t = 2$s, $y = -19.6$m y en $t = 3$s, $y = -44.1$m.

El algoritmo 10 simula por un minuto la caída libre de un cuerpo. Con el algoritmo 10 es posible desarrollar una simulación por computadora de los *cuerpos en caída libre*. El lector es invitado a consultar la simulación en: `https://scratch.mit.edu/projects/98683363/`.

Algoritmo 10 Cuerpos en caída libre

1: $a_y = -9.8$;
2: $y_0 = 0$;
3: $v_{0y} = 0$;
4: $t = 0$;
5: **mientras** $t \leq 60$ **hacer** //Un minuto
6: $y = y_0 + v_{0y}t + \frac{1}{2}a_y t^2$
7: $t = t + 1$; //Contador
8: **fin mientras**

7.1.2. Movimiento colectivo de cardúmenes

Los cardúmenes de peces son un fenómeno colectivo intrigante en la naturaleza, ocasionado por cientos o miles de peces moviéndose como una unidad coordinada (ver figura 7.3 [1]). La conformación de cardúmenes reduce la probabilidad de que los peces sean detectados por un depredador.

Un modelo clásico que simula cardúmenes de peces, parvadas de aves o rebaños se denomina *boids*, porque cada pez o ave en el modelo se denomina genéricamente como *boid* (contracción de *bird-oid object*). En el modelo de *boids*, cada *boid* aplica de manera autónoma tres *fuerzas de dirección* que controlan su movimiento: *cohesión*, *separación* y *alineación*.

Cada *boid i* tiene asignado un vector de posición $\overrightarrow{p_i}$ y un vector de velocidad $\overrightarrow{v_i}$ que describen su movimiento en el espacio. Además, cada *boid* tiene una vista local de su entorno nombrada *área de percepción* que está relacionada a una de las fuerzas de dirección mencionadas anteriormente. El *área de percepción* está

[1] Recuperada de `https://en.wikipedia.org/wiki/Shoaling_and_schooling`

Figura 7.3: La figura muestra un cardumen de peces moviéndose coordinadamente en su medio ambiente.

determinada por un radio r donde sólo los vecinos que están dentro del área de percepción son seleccionados (ver Figura 7.4).

En la fuerza de *cohesión* cada *boid* se mueve hacia el centro percibido de un grupo de vecinos, de modo que se acerque a sus vecinos y minimice la exposición al exterior de la formación. La fuerza de cohesión del *boid i* está descrita como el vector $\overrightarrow{c_i}$ y se obtiene en dos pasos. Primero se calcula la posición promedio de todos los *boids j* percibidos, y después se calcula la diferencia de la posición promedio con respecto a la posición del *boid i*. La fórmula de cohesión $\overrightarrow{c_i}$ se expresa en la ecuación 7.3.

$$\overrightarrow{c_i} = \left(\frac{1}{m_c} \sum_{j=1}^{m_c} \overrightarrow{p_j} \right) - \overrightarrow{p_i}, \tag{7.3}$$

donde m_c es el número de *boids* percibidos por el *boid i* en su radio de cohesión.

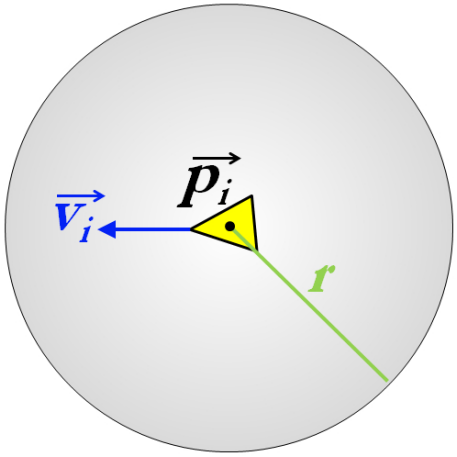
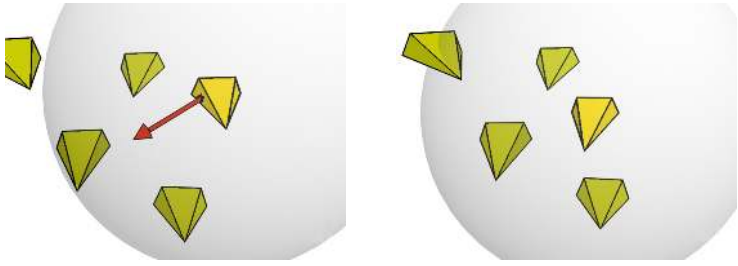


Figura 7.4: La figura muestra la posición que representa las coordenadas donde está ubicado el cuerpo del *boid*. La velocidad determina el desplazamiento del *boid*. El *área de percepción* está determinada por un radio.

El resultado de la aplicación de esta fuerza de cohesión es que el *boid i* se dirigirá hacia el centro de sus vecinos (ver figura 7.5).



(a) El vector de fuerza de cohesión resultante hace al *boid* dirigirse al centro percibido.

(b) Se muestra la posición del *boid* después de la aplicación de la fuerza de cohesión.

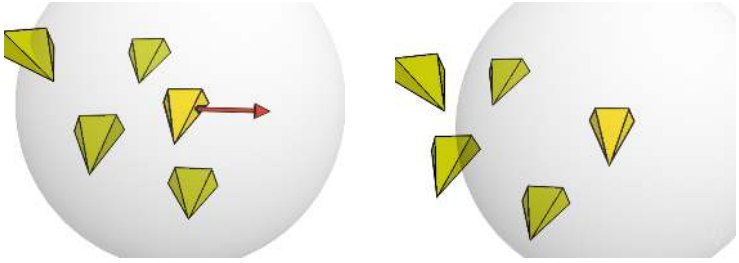
Figura 7.5: Fuerza de Cohesión.

En la fuerza de *separación* cada *boid* se mueve evitando estar muy cerca de sus vecinos, de modo que se reduzcan las colisiones en la formación. La fuerza de separación del *boid i* está descrita como el vector $\overrightarrow{s_i}$ y se obtiene de la suma de las diferencias de los *boids j* percibidos. Posteriormente, la suma de las diferencias es multiplicada por una unidad negativa para que el vector resultante tenga una fuerza repulsiva. La fórmula de separación $\overrightarrow{s_i}$ se expresa en 7.4.

$$\overrightarrow{s_i} = -\sum_{j=1}^{m_s} \left(\overrightarrow{p_j} - \overrightarrow{p_i}\right), \tag{7.4}$$

donde m_s es el número de *boids* percibidos por el *boid i* en su radio de separación.

El resultado de la aplicación de la fuerza de separación es que el *boid i* se dirigirá hacia el sentido contrario de los *boids* que se están muy cerca, como se ilustra en la figura 7.6.



(a) El vector de fuerza de separación hace al *boid* alejarse de sus vecinos más cercanos.

(b) Se muestra la posición del *boid* después de la aplicación de la fuerza de separación.

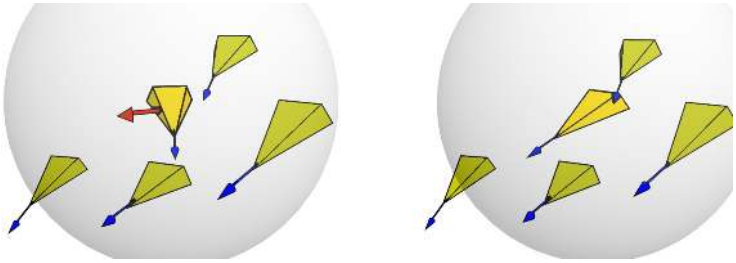
Figura 7.6: Fuerza de Separación.

En la fuerza de *alineación* cada *boid* se mueve en la dirección y con la rapidez promedio de sus vecinos, de modo que se oriente con respecto a la formación. La

fuerza de alineación del *boid i* está descrita como el vector $\overrightarrow{a_i}$ y se obtiene en dos pasos. Primero se calcula la velocidad promedio de todos los *boids j* percibidos. Posteriormente, se calcula la diferencia de la velocidad promedio con respecto a la velocidad del *boid i*. La fórmula de alineación se expresa en 7.5.

$$\overrightarrow{a_i} = \left(\frac{1}{m_a} \sum_{j=1}^{m_a} \overrightarrow{v_j} \right) - v_i, \tag{7.5}$$

donde m_a es el número de *boids* percibidos por el *boid i* en su radio de alineación. El resultado de la aplicación de la fuerza de alineación es que el *boid i* se desplazará a la velocidad promedio de la formación (ver figura 7.7).



(a) El vector de de fuerza de alineación hace al *boid* orientarse en la dirección de sus vecinos.

(b) Se muestra la posición del *boid* después de la aplicación de la fuerza de alineación.

Figura 7.7: Fuerza de alineación.

La formación de un cardumen o parvada es el resultado de la suma de las fuerzas de dirección. Si cada *boid i* aplica las fuerzas de dirección: cohesión $\overrightarrow{c_i}$, separación $\overrightarrow{s_i}$ y alineación $\overrightarrow{a_i}$, entonces emerge la formación. Los *boids* se man-

tienen unidos y se mueven coordinadamente. La fórmula para la formación f_i se expresa en (7.6).

$$\overrightarrow{f_i} = \overrightarrow{c_i} + \overrightarrow{s_i} + \overrightarrow{a_i} \tag{7.6}$$

El lector es invitado a consultar la simulación de los *boids* en 2D en el enlace: `https://scratch.mit.edu/projects/73676630/`

7.2. Simuladores en física

Los simuladores en el área de la física se han utilizado ampliamente como herramientas educativas para mejorar los procesos de aprendizaje y la resolución de problemas. Actualmente, existen una gran cantidad de simuladores en línea que pueden enriquecer, reforzar y ampliar el concepto de pensamiento computacional aplicado en la física. Es decir, los programas de computadora que simulan fenómenos físicos son una excelente herramienta educativa para mejorar los procesos de aprendizaje y la resolución de problemas.

A continuación describiremos algunos de los más populares.

7.2.1. PhET

Physics Education (PhET) es un proyecto de uso libre de la Universidad de Colorado, creado en 2002 (ver figura 7.8). El objetivo de PhET es mejorar la manera en que la ciencia y la tecnología es aprendida y enseñada. PhET comenzó con temas sobre física, posteriormente se expandió a otras disciplinas. Actualmente, tienen 125 simuladores relacionados a física, química, biología y matemáticas.

Figura 7.8: Physics Education (`https://phet.colorado.edu/es/simulations`)

7.2.2. Educaplus

Educaplus es un proyecto de uso libre del profesor Jesús Peñas, creado en 1998 (ver figura 7.9). El objetivo de Educaplus es compartir diversos simuladores para apoyar la práctica docente principalmente en la comunidad hispanoablante. Los simuladores abarcan temas de matemáticas, física, química, biología, ciencias de la tierra, educación artística y juegos.

7.2.3. EduMedia

EduMedia es un proyecto financiado por la Asociación de Tecnología Integrada (ver figura 7.10). El objetivo de EduMedia es ofrecer una gran cantidad de recursos y animaciones que sirvan como apoyo didáctico en matemáticas y ciencias. EduMedia hace más amena la interacción entre el alumno y la tecnología a través de recursos didácticos donde los alumnos se divierten aprendiendo. Incluye un sistema adaptativo para los diferentes tipos de aprendizaje: auditivo, visual y kinestésico. Existen aproximadamente 800 recursos interactivos dispo-

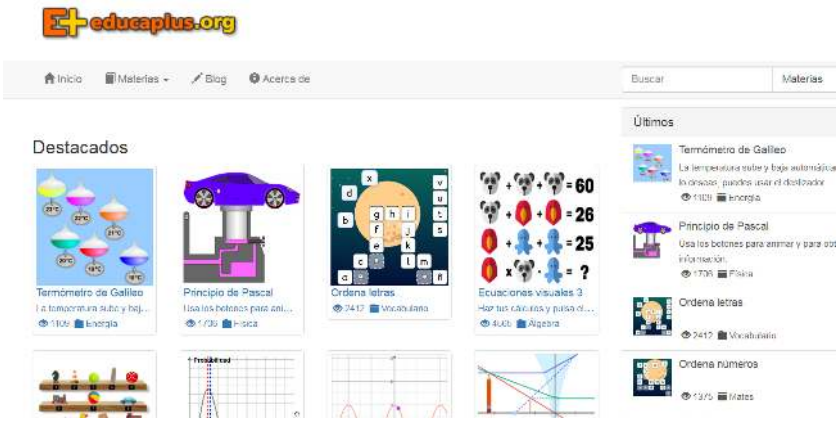


Figura 7.9: Educaplus (`www.educaplus.org/`)

nibles; sin embargo, para acceder a todos los recursos es necesario adquirir una licencia.

Figura 7.10: EduMedia (`http://www.edumedia-sciences.com/es/`)

7.2.4. GeoGebra

Geogebra es un proyecto de uso libre creado por Markus Hohenwarter en 2001 (ver figura 7.11). Actualmente, es mantenido por la Universidad de Linz. El objetivo de Geogebra es proporcionar un software matemático interactivo para la educación en escuelas y universidades. Geogebra es ampliamente conocido por sus capacidades para resolver ecuaciones, graficar funciones, analizar datos y explorar gráficas en 3D. Además, Geogebra tiene una división de simuladores de temas variados como física, matemáticas y química, entre otros.

Figura 7.11: EduMedia (`https://www.geogebra.org/?lang=es`)

7.3. Videojuegos y películas

Actualmente, los videojuegos y las películas animadas aplican un subcampo de la computación denominado *gráficas por computadora*. Las gráficas por computadora se centran en los fundamentos matemáticos y computacionales para generar y procesar imágenes realistas, no sólo en cuestiones puramente es-

téticas. Es decir, los video juegos y las películas que nos han impresionado con animaciones muy elaboradas aplican las matemáticas de manera ordinaria.

Una versión simplificada del proceso necesario para la creación de gráficos por computadora en los videojuegos y las películas es la siguiente:

1. Modelado. Es el proceso de construcción de un objeto a partir de elementos geométricos: polígonos, aristas y vértices (ver figura 7.12).

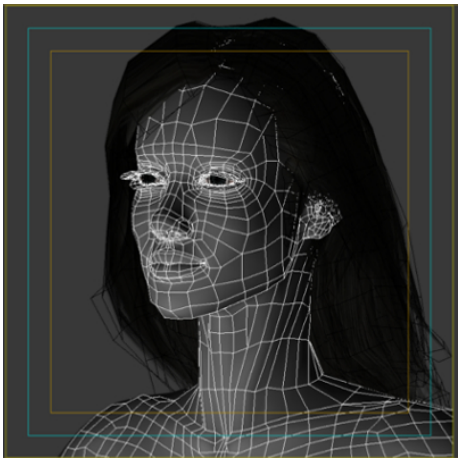
2. Animación. Es el proceso de dotar de movimientos a los objetos para percibir la sensación de vivacidad.

3. Iluminación. Es el proceso de simular la propagación de la luz en la escena para visualizar diversas propiedades de los objetos (ver figura 7.12).

4. Renderizado (*render* en inglés). Es el proceso de generar una imagen a partir de una escena 3D (ver figura 7.12).

Si deseas saber más acerca de las matemáticas utilizadas en las películas, entonces consulta los siguientes videos:

- Math and movies (`www.youtube.com/watch?v=mX0NB9IyYpU`)
- The magic ingrediente (`www.youtube.com/watch?v=Z1R1z9ipFnM`)
- Computer simulation (`www.youtube.com/watch?v=YeYW8TIWLG8`)



(a) Modelado.

(b) Iluminación y render.

Figura 7.12: Ejemplo de gráficas por computadora: modelado, iluminación y render.

7.4. Aplicando la simulación

La simulación computacional involucra implementar modelos a fin de realizar experimentos y probar hipótesis. Además, debido a que el programador tiene la libertad de ordenar en la simulación cualquier cosa que imagine, la simulación computacional también puede utilizarse para fines artísticos. Por ejemplo: el modelo de *boids* ha tenido aplicaciones en videojuegos y en las películas para proporcionar representaciones realistas de grupos moviéndose coordinadamente. En particular, la película *Batman Returns* utilizó el modelo de *boids* para simular bandadas de murciélagos y los ejércitos de pingüinos que caminan en las calles de Ciudad Gótica.

Las simulaciones por computadora se han utilizado ampliamente para la enseñanza de diferentes disciplinas. Por ejemplo: programas diseñados para simulación matemática (Geogebra y Pari/GP), simulación de sistemas dinámicos (PhET, JMCAD y Forcepad) o procesamiento de señales (Scilab, Octave y Matlab). Este tipo de simulaciones se encuentran también en otras áreas, como la química (Jmol y E-TECH), astronomía (Celestia) y ciencias en general (E-TECH). Diversas plataformas de simulación han sido reconocidas por *WISE Awards* al ser los proyectos de innovación más exitosos ante los retos educativos a nivel mundial. Además, existen simulaciones en otras áreas que son de gran importancia para la sociedad, como, la simulación en medicina del *Human Anatomy Atlas*, que visualiza órganos y reconstruye imágenes. Algunas empresas que venden este tipo de aplicaciones son *Medical Simulator* y *Virtamed*.

A nivel mundial existen varios centros de investigación y empresas que se especializan en el desarrollo de herramientas, algoritmos y nuevas propuestas para la simulación en diferentes áreas. Por ejemplo, el *Center for Computational Simulation*, el *Join Institute for Computational Science* y el *Simudyne*.

7.5. Ejercicios

1. Ingresa al simulador *Aterrizaje Lunar*: `https://phet.colorado.edu/es/simulation/legacy/lunar-lander`. Explora la relación entre los vectores de fuerza de propulsión de la nave y la gravedad de la luna para aterrizar la nave de manera segura. Si la nave estuviera a 200m de altura y no tuviera combustible, ¿en qué tiempo caería la nave sobre la superficie lunar?

2. Ingresa al simulador *Laboratorio de densidad*: `http://www.educaplus.org/game/laboratorio-de-densidad`. Mide la masa de todos los objetos y prueba los objetos en líquidos con distintas densidades. Apunta los valores correspondientes de cada objeto y la densidad del liquido en la que ya no se hunden. Si aumentas la densidad del liquido, ¿qué deduces?

3. Ingresa al simulador *Flotabilidad*: `https://phet.colorado.edu/sims/density-and-buoyancy/buoyancy_es.html`. Explora cómo funciona la flotabilidad modificando las propiedades de los bloques y del fluido. ¿Cuáles son los materiales que flotan? Los que tienen mayor o menor densidad que el líquido. ¿Cómo sería la densidad de una piedra comparada con la densidad del agua de un estanque? Mayor o menor que el agua. ¿Qué material usarías para construir un barco capaz de mantenerse a flote?

4. La ley de Pascal, enunciada por el físico y matemático francés Blaise Pascal (1623-1662), se puede resumir como: La presión ejercida en un fluido incompresible y contenido en un recipiente de paredes indeformables se transmite con igual intensidad por todos los puntos del fluido. Ingresa al simulador *Principio de Pascal*: `http://www.educaplus.org/game/principio-de-pascal`. Relaciona la descripción de la ley con el ejemplo de la prensa hidráulica que levanta el auto.

5. Ingresa al simulador *Cambios y formas de energías*: `https://phet.colorado.edu/es/simulation/legacy/energy-forms-and-changes`. Los materiales (hierro, ladrillo y el agua) al enfriarse o calentarse generan un

aumento o disminución de la energía. Observa cómo se transfiere la energía entre objetos.

6. El esquema de la figura 7.2 representa el movimiento vertical de una piedra que Tukkul deja caer. El movimiento de la piedra está determinado por la ecuación $y = \frac{1}{2}(-9.8)t^2$. Imagina que una canoa comienza a moverse desde el punto $(-50, -150)$ en un tiempo $t_0 = 0$. Es decir, la velocidad inicial de la canoa es cero y su posición en $x = -50$.

   (a) El movimiento de la canoa está determinado por la ecuación $x = \frac{1}{2}a_x t^2$. ¿Cuál es la aceleración a_x que la canoa necesita para que la piedra caiga sobre ella?

   (b) Implementa la simulación del movimiento de la piedra y de la canoa en scratch.

7. Diseña en scratch un simulador del tiro parabólico. Imagina que es un cañón el que está disparando. Coloca un objeto a cierta distancia e introduce distintos ángulos y velocidades iniciales hasta impactar en el objeto.

8. Diseña en scratch un simulador de algún fenómeno o sistema de tu interés. Por ejemplo, un fenómeno físico como el movimiento de un satélite en órbita alrededor de la Tierra o un sistema biológico como los caminos creados por las hormigas hacia lugares con mayor cantidad de comida.

Capítulo 8

Experiencia de aprendizaje

En el documento *Computational Thinking Teacher Resouces* se presentan nueve *Computational Thinking Learning Experiences* (Experiencias de Aprendizaje del Pensamiento Computacional). Las Experiencias de Aprendizaje del Pensamiento Computacional (EAPC) son una muestra de prototipos de actividades de enseñanza en distintos niveles de educación y asignaturas. Las EAPC están destinadas a ilustrar actividades del PC en un formato amigable y resaltando los conceptos clave y el vocabulario del pensamiento computacional. Las EAPC pueden diseñarse de tal manera que las actividades requieran o no de la utilización de una computadora.

Las EAPC presentadas en dicho documento son una guía para que cada docente diseñe sus propias experiencias de aprendizaje utilizando el PC en sus planes y lecciones de estudio. En este capítulo se presentan dos experiencias de aprendizaje sin computadora denominadas: *Escribe tu nombre en glifos Mayas* y *Tejiendo el huipil de Ixchel*.

8.1. Escribe tu nombre en glifos Mayas

Habilidades del pensamiento computacional:

- Organizar datos de manera lógica y analizarlos.
- Representar datos mediante abstracciones.
- Automatizar soluciones mediante pensamiento algorítmico.

Actitudes del Pensamiento Computacional:

- Confianza en el manejo de la complejidad.
- Tolerancia a la ambigüedad.
- Habilidad para comunicarse y trabajar con otros para alcanzar una meta o solución común.

Vocabulario:

- Analizar datos: Darle sentido a los datos, hallar o establecer patrones y sacar conclusiones.

- Representar datos: Representar y organizar los datos en gráficas, cuadros, palabras o imágenes apropiadas.
- Descomponer problemas: Dividir una tarea en partes pequeñas y manejables.
- Algoritmos y procedimientos: Serie de pasos ordenados que se siguen para resolver un problema o lograr un objetivo.

Metas:

- Desarrollar ejercicios relacionados con descomposición de problemas, abstracción y representación de la información usando diferentes tipos de códigos.
- Reconocer la importancia de codificar y decodificar la información para diferentes propósitos.
- Despertar el interés en conocer los glifos Mayas u otros tipos de códices.

Asignaturas relacionadas:

- Computación, historia, lenguas y arte.

Evidencia:

- Los estudiantes dividieron el problema complejo en problemas más simples.
- Los estudiantes organizaron los datos de manera clara y lógica logrando una representación ingeniosa y artística para ellos.
- La información fue codificada y decodificada correctamente.
- Los estudiantes identificaron los conceptos de abstracción y descomposición.

Material

- Lápiz y goma, un plumón de color negro y una hoja de foami (el color de tu preferencia). En caso de no contar con el plumón y la hoja foami, puede sustituirse por una pluma y una hoja de papel bond. ¡Lo que nunca debe faltar es el entusiasmo!

Actividad

La civilización Maya destacó por sus avances en matemáticas, astronomía, arquitectura y arte. Los mayas inventaron el concepto del cero y un sistema de escritura que refleja el ingenio, la inteligencia y el avance de la civilización Maya. La escritura maya está compuesta de signos y símbolos llamados glifos similares a los iconos con los que comúnmente asocias un archivo o un programa en la computadora. En esta actividad aprenderás a escribir tu nombre con glifos mayas. Observa la oración mostrada en la Figura 8.1 para darte una idea de como escribían los mayas [1].

Por muchos años el significado de los glifos mayas fue un misterio. Afortunadamente, en los últimos 30 años varios glifos han sido descifrados. Los glifos son una ventana al pasado que revela un poco de su cultura. En está experiencia de aprendizaje aprenderás cómo descifrar un mensaje escrito con glifos mayas como el mostrado en el siguiente cuento.

[1] Recuperada de Fundación para el Avance de los Estudios Mesoamericanos

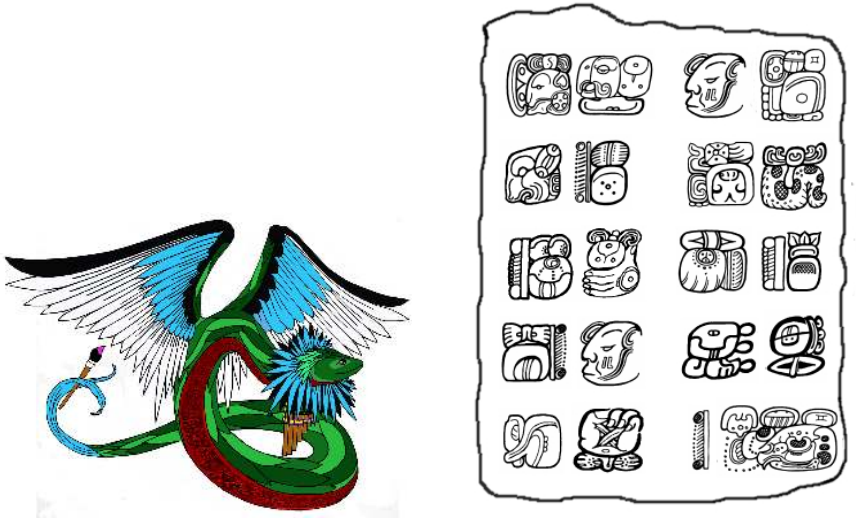


Figura 8.1: El mensaje contenido en la estela maya (piedra tallada) es el siguiente: "Me llamo Alan. Soy estudiante y jugador de pelota. Mi madre se llama María. Ella trabaja en tejidos y es de Yaxchilán. Mi padre se llama Tomás. Él es agricultor y sabio. Él es de Palenque".

Tukkul es un arqueólogo porque anhela encontrar la respuesta de cuál es el propósito de la vida en las culturas antiguas. Ha estudiado en detalle a los sumerios, los egipcios, los griegos y las culturas mesoamericanas. Él es un arqueólogo muy famoso porque descubrió, en el templo de las inscripciones, un acceso bloqueado por escombros hacia una cámara secreta. Después de tres años de excavaciones para poder acceder, Tukkul ingresa en la cámara y encuentra una estela maya como la que se aprecia en la Figura 8.2.

Estaba anocheciendo por lo que Tukkul decide regresar a su campamento. En su alcoba, mientras se arropaba con su bata de piel de ardilla, cae en un profundo sueño. En el sueño, Tukkul se encontraba en una exploración nocturna en la selva Lacandona y en un descuido resbala y cae dentro de un cenote. La caída lo deja aturdido, se levanta con dificultad y percibe un olor fétido, se da la vuelta y a la luz de la luna que entra por el cenote, queda horrorizado al ver un cuerpo en descomposición que comienza a levantarse. El rostro purulento de aquel ser se burlaba de él. Tukkul comprende que está en Xibalbá (el inframundo) y que el ser aterrador es el

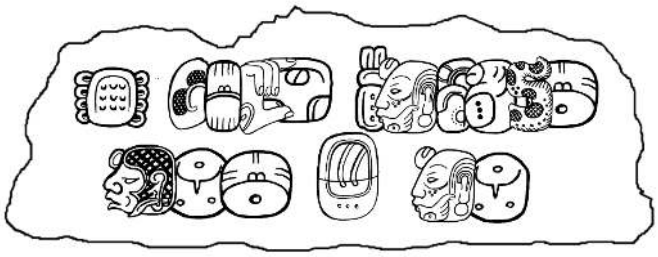


Figura 8.2: Estela maya dentro del templo de las inscripciones.

Dios de la muerte Ah Puch. Tukkul comienza a correr, las piernas le tiemblan, Ah Puch está en su acecho. En el ajetreo Tukkul tropieza y tiene como nunca antes la sensación de la proximidad de su muerte.

En ese instante, Tukkul recuerda varios pasajes de su vida. «La vida no me ha arrebatado nada, ahora comprendo que nada me pertenece», pensó Tukkul, y dejo de tener miedo, en lo más profundo de su espíritu tiene un sentimiento de gratitud y bienaventuranza por todo lo que ha vivido. Estando en el suelo Tukkul pronuncia: «bendigo todos los días la salida del sol», y besa la tierra. Al instante, aparece su amigo Kukulkán y escupe un grano de maíz en la tierra. El maíz crece hasta volverse una enorme milpa que traspasa el subsuelo y llega a la superficie. Tukkul aprovecha la oportunidad para escapar de Ah Puch y trepa por la milpa. En el momento que sale y escapa, despierta.

Al atardecer, Tukkul está reflexionando en las escalinatas del templo de Palenque: «¿Por qué tendría que tener un propósito la vida? ¿Qué no sería más bello que no tenga un propósito y que cada uno sea libre de darle el suyo? ¿Quizá el propósito de la vida es evolucionar y alcanzar la liberación espiritual?». Estando inmerso en sus cavilaciones, Paat llega corriendo alegremente y le dice: «mirá al cielo y veras a los angelitos». Tukkul mira al cielo y transcurre alrededor de cinco segundos. «¿Los ves?» inquirió Paat. «No veo nada», pronunció Tukkul. «No seas menso, los angelitos son aquellas luciérnagas que van destellando por el cielo», dijo Paat.

De pronto, Tukkul tiene una revelación, «No busques, simplemente párate y mira», recordó Tukkul el mensaje de la estela maya. Él había orientado sus esfuerzos en la búsqueda de una sabiduría oculta y milagrosa, creía que cuando obtuviera el conocimiento arcano se iluminaría de improviso. Sin embargo, ahora comprende que la vida en sí misma, con sus alegrías y sus

sufrimientos, es el propósito y lo que construya con ella. Ahora, el propósito de su vida ya no es un enigma para Tukkul, tiene un innegable sentido de entrega a la simplicidad de la existencia, a la bondad y al amor, y él es lo suficientemente fuerte para implantar ese sentido en ella.

Yuri Knorozov descubrió que la escritura maya es un sistema mixto que combina signos fonéticos (*silabogramas*) y signos para palabras completas (*logogramas*). En el caso de los signos fonéticos cada glifo representa una combinación *consonante-vocal*. Es decir, una sílaba. Los mayas usaban glifos para las vocales *a*, *e*, *i*, *o* y *u*. Además, tenían símbolos para la mayoría de las consonantes acompañadas por una vocal. Por ejemplo: *ma*, *me*, *mi*, *mo* y *mu* (ver Figura 8.3).

Como únicamente los mayas inventaron glifos para las sílabas que terminaban en vocal, Knorozov supuso correctamente que una palabra maya formada por una combinación *consonante-vocal-consonate* era escrita con dos glifos y la vocal del segundo glifo no se pronunciaba.

Al escribir un sonido como la sílaba *tab*, usaban las sílabas *ta* + *ba* y no pronunciaban la última *a*. Utilicemos la siguiente convencion: *ta* — *b*(*a*), donde los paréntesis indican que la última *a* es muda. Por lo general, en escritura maya la última vocal y la vocal anterior son idénticas lo que se denomina *la regla de la armonía*. Los glifos que significan palabras completas y no son formadas con sílabas son denominados *logogramas*. Por ejemplo, los logogramas que representan la palabra blanco (SAK), cielo (CHAN) y jaguar (B'ALAM) se muestran en la Figura 8.3.

Los epigrafistas habían estado buscando un solo símbolo para cada sonido silábico, hasta que David Stuart descubrió que un sonido podía tener varias representaciones diferentes. Por lo general, los mayas tenían más de una manera de escribir una sílaba, lo cual es diferente en la mayoría de las lenguas modernas. Nosotros siempre escribimos el sonido *ma* como *m* + *a*, sin embargo, los mayas tenían muchos modos de escribir el sonido *ma* (ver Figura 8.3).

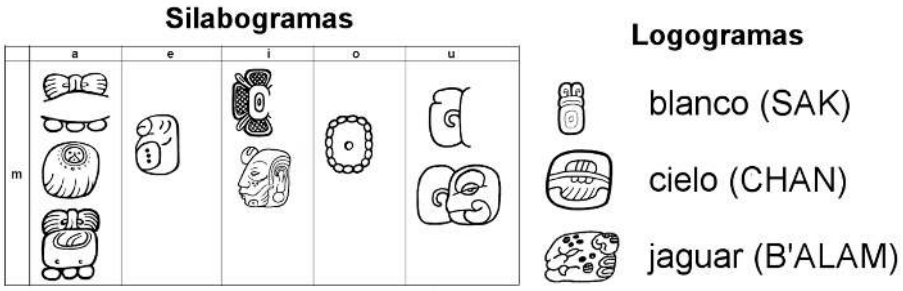


Figura 8.3: Silabogramas con *m* y algunos logogramas.

Algoritmo

En esta sección mostraremos el algoritmo que te permitirá escribir tu nombre con glifos mayas. Usamos como ejemplo el nombre: María. Realiza los siguientes pasos en una hoja en blanco.

1. Dividir el nombre en sílabas mayas. Si es necesario, usar la regla de la armonía y eliminar los acentos.

   $$María = ma - ri - a$$

   Si tu nombre es *Alan*, entonces la división silábica es $A - lan$. La sílaba *lan* termina en consonante. Por esta razón, debes usar la regla de la armonía para que todas las sílabas terminen en vocal. Al agregar una *a* a la sílaba *lan*, el resultado es $A - la - n(a)$.

2. Sustituir las consonantes: *c, d, f, g, h, q, r, v* o *z*, por sus equivalentes del sistema maya de acuerdo a la tabla 8.1. Por ejemplo, los mayas no usaban el sonido *r*, sustituir *l* por *r*.

   $$María = ma - li - a$$

Tabla 8.1: Sonidos equivalentes

Sustituciones de Consonantes			
Consonante	Sonido	Sustitución	Ejemplo
C	Suave	S + vocal	Celia = se - li a
C	Fuerte	K + vocal	Catarina = ka - ta - li - na
D	-	T + vocal	David = ta - bi - d(i)
F	-	P + vocal	Fabiana = pa - bi - a - na
G	-	K + vocal	Gustavo = ku - su - ta - bo
H	-	Eliminar H	Helia = e - li - a
Q	-	K + vocal	Quiroa = ki - lo - a
R	-	L + vocal	Rebeca = le - be - ka
V	-	B + vocal	Vicente = bi - se - n(e) - te
Z	-	Tz + vocal	Zoila = tzo - i - la

3. Seleccionar la sílaba principal. Idealmente, es la sílaba acentuada en la pronunciación de tu nombre.

   La sílaba principal del nombre *María* es *rí* (li).

   ma — li — a

   Si la palabra tiene dos sílabas, entonces usa la sílaba que tiene el acento como sílaba principal. Por ejemplo, la sílaba principal en *Ana* es la primera *A*. Si tu nombre tiene tres o cinco sílabas, la sílaba principal es la sílaba de en medio (como en *María*). Si tu nombre tiene cuatro o seis sílabas, entonces debes escoger una de las sílabas que esté en medio de tu nombre como sílaba principal. Por ejemplo, la sílaba principal en *Guadalupe* es *lu*.

4. Sustituir cada sílaba del nombre por el correspondiente *silabograma* (ver Figura 8.5). Recuerda que en algunos casos, una sílaba puede representarse de varias maneras. La opción que elijas depende de la que más te guste. En el nombre *ma* – *li* – *a*, una posibilidad se muestran en la Figura 8.4.

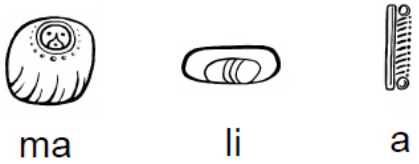


Figura 8.4: Sílabas de María en glifos mayas.

Si el cuadrado de las sílabas que necesitas no tiene glifos, significa que el glifo maya de esa sílaba no se ha descubierto. ¿Qué debes hacer? Imagina que necesitas la sílaba *wu*. En los silabarios no hay ningún glifo para *wu*, pero sí hay el glifo *wa*. En estos casos, usa la *consonante* + *a* y la vocal que necesitas. Es decir, la sustitución de *wu* es *wa* + *u*.

5. Organizar los silabogramas de acuerdo a las normas que los mayas usaban para mostrar tu nombre de manera entendible y artística. El orden común de las partes en los bloques de glifos es de izquierda a derecha y de arriba hacia abajo. La Figura 8.6 muestra la disposición de los bloques en las que normalmente son colocados los glifos mayas.

   Si es posible el glifo principal debe ser uno de los glifos grandes y cuadrados. Normalmente, la primera sílaba del nombre se ubica el espacio prefijo. Las otras sílabas previas a la sílaba principal se ubican en el espacio del superíndice (si necesitas más espacio, también en el espacio de prefijo). Las sílabas posteriores a la sílaba principal se ubican en los espacio del sufijo y del subíndice. A excepción de la sílaba principal, debes tratar de usar

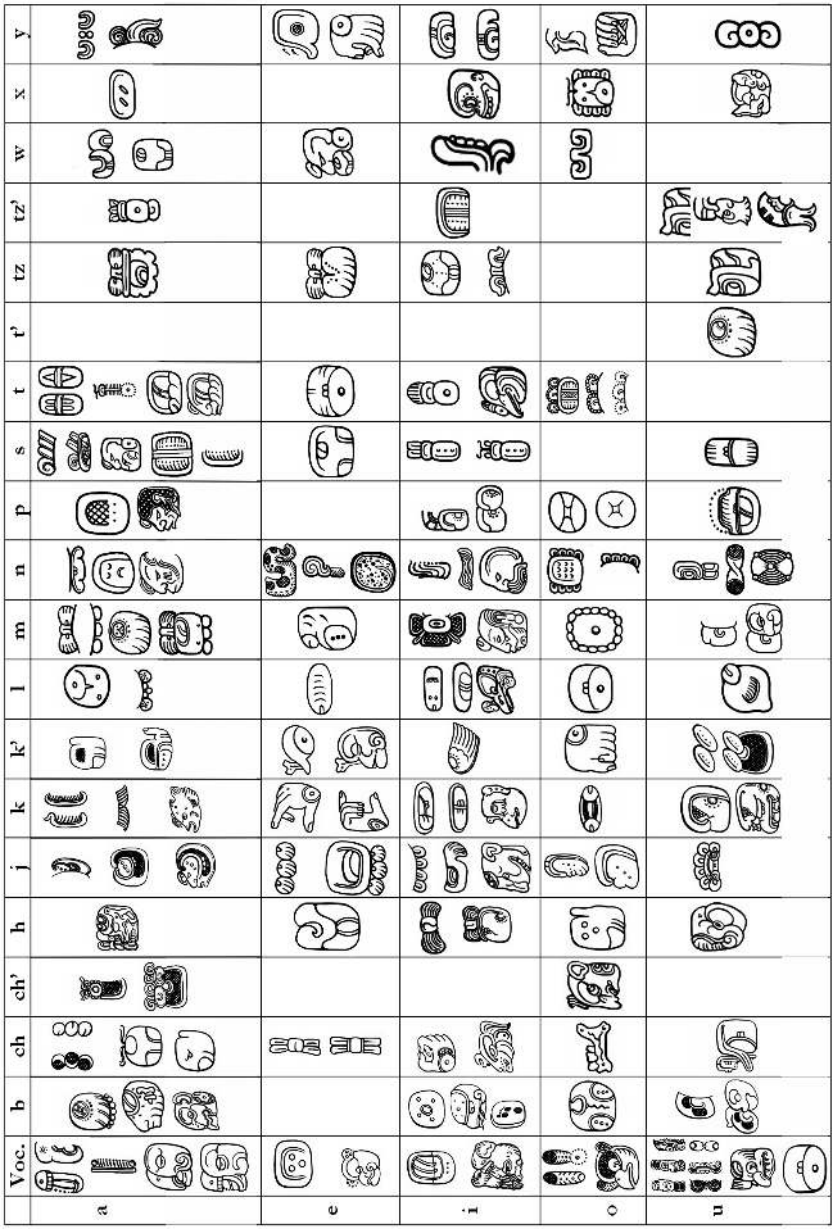


Figura 8.5: Silabograma Maya.

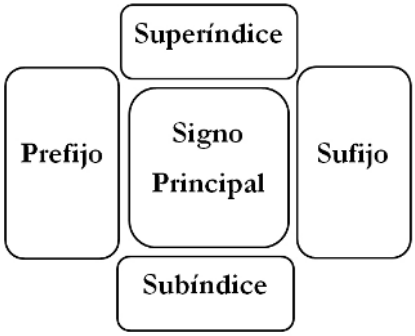


Figura 8.6: Disposición en la que normalmente son colocados los glifos mayas.

los glifos menores para las demás sílabas. Si es posible, las sílabas deberán agruparse cerca del glifo principal y tocarlo. Rota los glifos para hacer que encajen en los espacios alrededor del glifo principal. Estira los glifos para hacer que el grupo sea agradable. Al final, lo deseable es un bloque que parezca un cuadrado, como una piedrecilla con esquinas redondeadas.

Un posible glifo final del nombre *María* es el mostrado en la Figura 8.7.

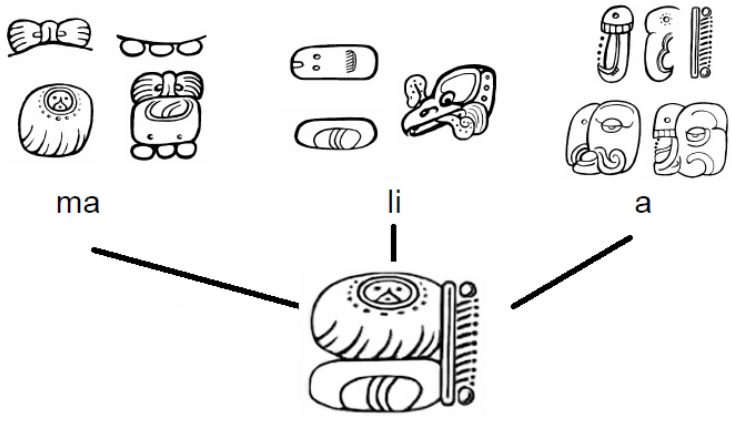


Figura 8.7: El nombre María en glifos mayas.

Si tu nombre tiene cinco sílabas, la primera sílaba se ubica en la posición del prefijo y la segunda sílaba en la posición del superíndice. La sílaba central se ubica en el lugar de la sílaba principal, es decir, en el espacio del glifo principal. La penúltima sílaba se ubica en la posición de sufijo y la última sílaba se ubica en la posición de subíndice. Si tu nombre tiene más de cinco sílabas o quieres experimentar con distintos bloques, entonces usa los diseños de la Figura 8.8 [2].

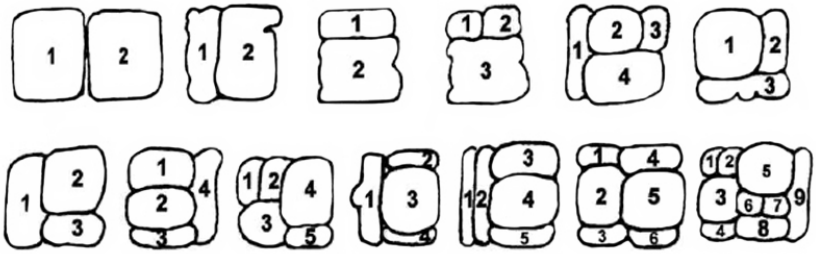


Figura 8.8: Diseños de glifos mayas.

6. Dibujar tu nombre en la hoja fomi (ver Figura 8.9).

Los escribas mayas eran mujeres y hombres muy creativos e inteligentes. A los mayas les gustaba divertirse cuando escribían, preferían crear nuevas y originales formas de expresarse, por esta razón, les encantaban las sustituciones. Dos mujeres que se llaman María pueden deletrear su nombres con diferentes glifos mayas. Si te interesa profundizar en la escritura maya puedes consultar la página: `www.famsi.org/spanish/mayawriting/`.

Finalmente, Tukkul pudo descifrar el mensaje de la estala maya de la Figura 8.2 porque conoce de memoria el silabario maya y pudo sustituir las sílabas de acuerdo a cada sibalograma como se muestra en la Figura 8.10.

[2] Recuperada de Fundación para el Avance de los Estudios Mesoamericanos

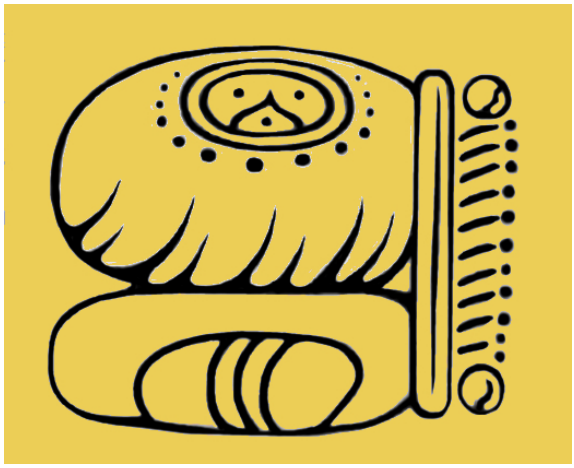


Figura 8.9: Resultado final del nombre María en glifos mayas.

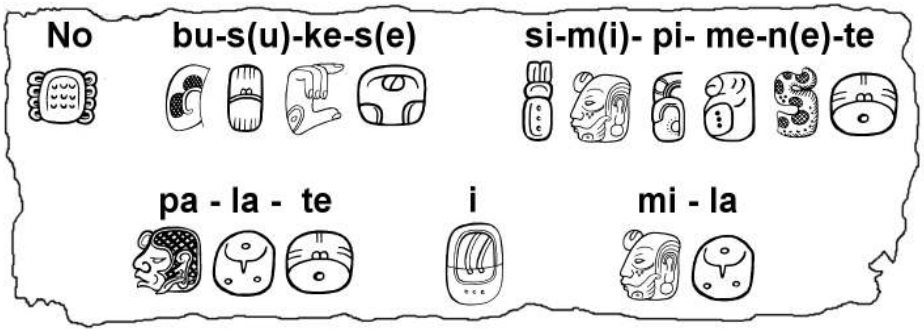


Figura 8.10: Mensaje descifrado.

La especificación del algoritmo anterior en un nivel de abstracción menor puede implementarse en un lenguaje de programación. El algoritmo 11 muestra el paso dos que sustituye las sílabas consonantes.

Algoritmo 11 Sustituir letras

```
1: mientras Revisión de sílabas no terminada? hacer
2:     si La sílaba contiene las letras C, D, F, G, H, Q, R, V o Z? entonces
3:         seleccionar letra
4:           caso C suave
5:             Sustituir C por S
6:           caso C fuerte
7:             Sustituir C por K
8:           caso D
9:             Sustituir D por T
10:          caso F
11:            Sustituir F por P
12:          caso G
13:            Sustituir G por K
14:          caso H
15:            Eliminar la H;
16:          caso Q
17:            Sustituir Q por Q
18:          caso R
19:            Sustituir R por L
20:          caso V
21:            Sustituir V por B
22:          caso Z
23:            Sustituir Z por Tz
24:        fin conmutador
25:    fin si
26: fin mientras
```

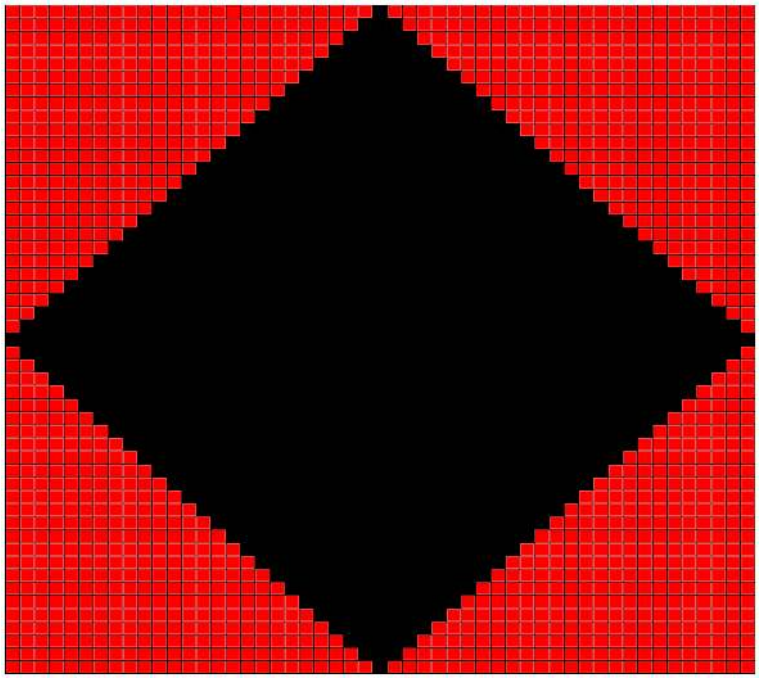
Programa de glifos Mayas

Imagina que quieres escribir un programa que descifre los glifos mayas escritos en ruinas y que también escriba los glifos a partir de un dictado en Español. Piensa cómo podrías hacerlo y qué se necesita. Un sistema de visión computacional es necesario para identificar en dónde hay glifos. Uno de los problemas principales en la identificación es el desgaste de los glifos.

Normalmente, se recurre a arqueólogos y linguistas especializados para obtener los glifos. Supongamos que puedes aislar exitosamente todos los glifos, ¿cómo podrías interpretarlos? Como hay diferentes formas de escribir e interpretar los glifos, el sistema tendría que proporcionar las interpretaciones más probables en Maya. De manera similar, a los sistemas que interpretan el lenguaje hablado que generan varias alternativas de partes obtenidas de un audio y proporcionan la interpretación más probable. Posteriormente, se ejecuta una traducción al Español.

En la escritura se requiere conocer los sonidos del lenguaje Español y hacer la traducción de las palabras a glífos. En esta actividad vimos como hacerlo. Además, hay que definir qué símbolos usar, cuál es el sonido dominante, cómo distribuir el resto de las sílabas, etc. Hay algunos sistemas que pueden ayudarte. Por ejemplo, ver: `https://www.paleoaliens.com/event/mayan_glyphs/`.

8.2. Tejiendo el huipil de Ixchel



Habilidades del pensamiento computacional:

- Automatizar soluciones mediante pensamiento algorítmico.
- Identificar, analizar e implementar posibles soluciones con el objetivo de lograr la combinación más eficiente y efectiva de pasos y recursos.
- Formular problemas de una manera que nos permita usar una computadora y otras herramientas para ayudar a resolverlos.

Actitudes del Pensamiento Computacional:

- Confianza en el manejo de la complejidad.

- Persistencia en trabajar con problemas difíciles.

Vocabulario:

- Descomponer problemas: Dividir una tarea en partes pequeñas y manejables.
- Algoritmos y procedimientos: Serie de pasos ordenados que se siguen para resolver un problema o lograr un objetivo.
- Representación de datos: Representar y organizar apropiadamente datos en gráficos, tablas, palabras o imágenes.
- Una libreta cuadriculada, un color rojo y negro.

Asignaturas relacionadas:

- Computación y arte.

Evidencia:

- Los estudiantes dividieron el problema complejo en problemas más simples.
- Los estudiantes identificaron el concepto de algoritmo.
- Los estudiantes desarrollaron algoritmos en seudocódigo.

Actividad

El arte textil mexicano es considerado parte fundamental del patrimonio cultural del país y es reconocido por la UNESCO como patrimonio cultural de la humanidad. La riqueza y variedad del arte textil mexicano se encuentra a lo largo de todo el país: Oaxaca (Mazateco, Chinanteco, Mixes y Zapoteco), Yucatán

(Maya), Michoacán (Purépechas y Otomí), San Luis Potosi (Teneek y Huastecos), Veracruz (Totonaca), Estado de México (Mazahua), Guerrero (Amuzgos), Chiapas (Tsotsil, Tzeltal y Zoque), Chihuaha (Rarámuri-Tarahumara) y Nayarit (Huichol-Wixárikas).

La gran diversidad y colorido del arte textil se puede apreciar en diversos museos de México, como el Museo de Arte Popular (`http://www.map.cdmx.gob.mx/index.php`), el Centro de Textiles del Mundo Maya (`http://sic.gob.mx/ficha.php?table=museo&table_id=1758`) en donde hay más de 2, 500 piezas, el Museo Textil de Oaxaca (`http://www.mexicoescultura.com/recinto/56393/museo-textil-de-oaxaca.html`).

El vestido es una prenda que distingue y da rasgo de identidad a las diferentes comunidades en México. La Figura 8.11 muestra sólo algunos ejemplos de estos bordados[3], pero existen muchos más.

Figura 8.11: Huipil.

[3]Recuperada de Centro de Textiles del Mundo Maya

El siguiente cuento nos introduce en la fascinante tarea de elaborar un huipil.

En la paradisíaca Cozumel, Tukkul hizo muchas actividades. En el día, buceó en los arrecifes de coral y disfrutó de los bailes y ritmos del carnaval. En la noche, contempló fijamente la luna llena por unos minutos y justo antes de caer en los brazos del sueño, alucinó un vórtice de vibrantes colores que comenzó a tomar la forma de un huipil. El huipil estaba siendo tejido por una mujer que tenia un conejo a su lado. Tukkul entendió que esa mujer era la diosa Ixchel (diosa de la Luna y del tejido). Maravillado, Tukkul tuvo la curiosidad de saber cuál es el proceso de creación del huipil. Ixchel lo intuyo y le explicó que el proceso para crear un huipil involucra el cultivo de algodón, la limpieza del algodón, la fabricación del hilo, el teñido de los hilos, la elaboración de las madejas. Una vez creados los telares, la artesana monta los hilos en el telar y tensa los hilos entre un árbol y su cintura para comenzar el proceso de bordado. Ixchel le dijo a Tukkul que una pieza artesanal de gran colorido y belleza puede llevar meses en su elaboración.

El proceso de bordado generalmente sigue un algoritmo para cada detalle que se quiere incluir en el textil. Por ejemplo, suponiendo que ya tenemos dos hilos teñidos, podríamos pensar en tener una tela con puntos alternados de dos colores (rojo y negro). Para hacer una línea alternada de colores podríamos tener el siguiente pseudo-código:

```
1: para j = 1; j == M; j = j + 1 hacer        //M es la mitad del ancho
2:     pintar en color rojo
3:     pintar en color negro
4: fin para
```

El resultado lo puedes apreciar en la Figura 8.12. ¿Porqué el ciclo va de uno hasta la mitad del ancho de la prenda?

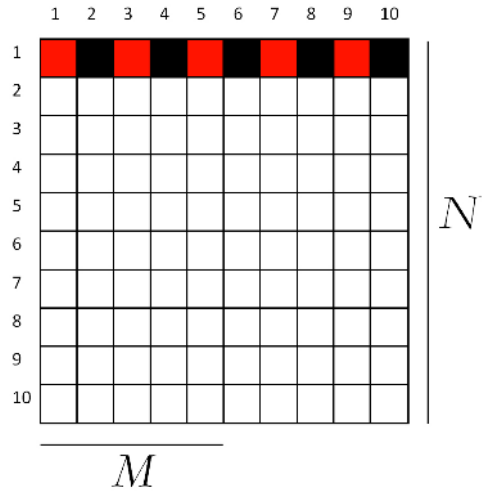


Figura 8.12: Tejido 1.

Si queremos producir una tela completa, no solo una línea, entonces tenemos que repetir el ciclo anterior, tantas veces como queramos que sea el alto de la prenda.

```
1: para i = 1; i == N; i = i + 1 hacer          //N es alto de la prenda
2:     para j = 1; j == M; j = j + 1 hacer      //M es la mitad del ancho
3:         pintar en color rojo
4:         pintar en color negro
5:     fin para
6:     Cambia de línea
7: fin para
```

¿Qué produce el código anterior? El algoritmo anterior produce una tela de líneas verticales rojas y negras (ver Figura 8.13).

¿Qué tendrías que hacer para tejer una cuadrícula del tipo ajedrez con con los colores rojo y negro? Una posibilidad es hacer una línea horizontal rojo-

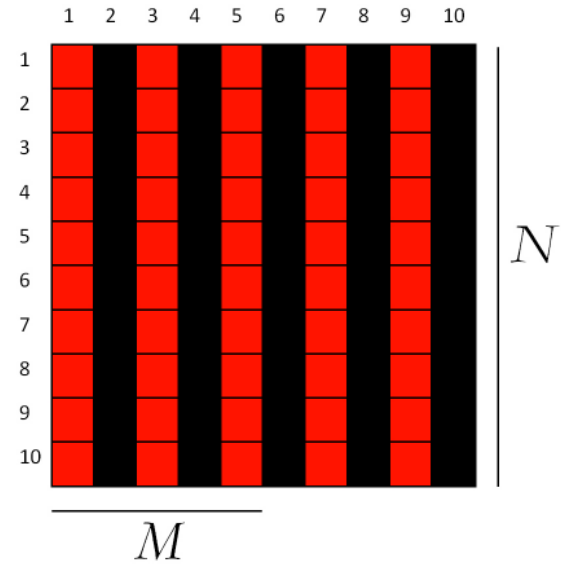


Figura 8.13: Tejido 2.

negro seguida de una línea negro-rojo. Esto lo podemos hacer repitiendo el código para hacer una línea, pero intercambiando los colores.

```
1:  para i = 1; i == N/2; i = i + 1 hacer          //N es alto de la prenda
2:      para j = 1; j == M; j = j + 1 hacer        //M es la mitad del ancho
3:          pintar en color rojo
4:          pintar en color negro
5:      fin para
6:      Cambia de línea
7:      para j = 1; j == M; j = j + 1 hacer        //M es la mitad del ancho
8:          pintar en color negro
9:          pintar en color rojo
10:     fin para
11:     Cambia de línea
```

12: **fin para**

El resultado lo puedes apreciar en la Figura 8.14. ¿Porqué el ciclo externo sólo va de uno hasta la mitad de la altura de la prenda?

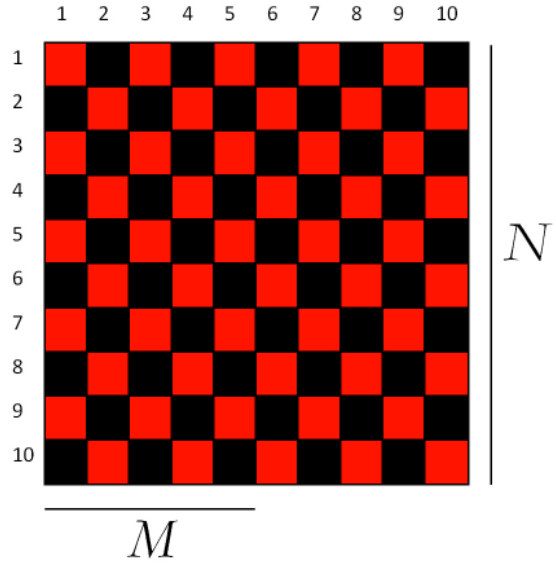


Figura 8.14: Tejido 3.

¿Qué tendrías que hacer para tejer un rombo negro dentro de un cuadro rojo? Tienes que pensar cómo insertar gradualmente más puntos negros mientras disminuyes los rojos de cada lado, hasta llegar a la mitad y después, cómo disminuir los negros gradualmente, aumentando proporcionalmente los rojos. Por ejemplo, para una tela de 51 puntos (ancho) por 51 puntos (alto), hay un solo punto negro en la primera línea, después hay 3 puntos negros, y así sucesivamente hasta llegar a 51 puntos negros. Posteriormente, el número de puntos negros disminuye hasta llegar a 1 punto negro. Tu algoritmo tendría que producir la secuencia de la tabla 8.2. El patrón resultantes se muestra en la Figura 8.15.

Tabla 8.2: Secuencia

Rojo	Negro	Rojo
25	1	25
24	3	24
23	5	23
...		
1	49	1
0	51	0
1	49	1
...		
25	1	25

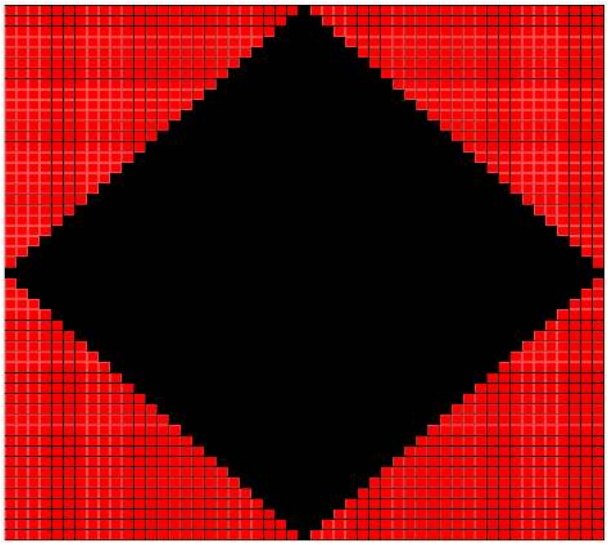


Figura 8.15: Tejido 4.

¿Cómo podrías hacer un algoritmo que dibuje el rombo?

Está actividad consiste en que escribas un algoritmo que dibuje el patrón del rombo. Una vez que hayas escrito tu programa, en una hoja cuadriculada pinta cada cuadro de acuerdo con las instrucciones de algoritmo para corroborar que funciona correctamente.

El algoritmo 12 es una posible solución que dibuja el patrón del rombo.

Algoritmo 12 Dibujar rombo

```
1:  X = 1                                          //Número de negros, inicialmente 1
2:  para Y = 1; Y == N/2; Y = Y + 1 hacer          //Largo de la tela - primera parte de la tela
3:      para i = 1; i == (N + 1)/2 − X; i = i + 1 hacer   //Primera parte de rojos
4:          color = rojo
5:      fin para
6:      para j = 1; j == X; j = j + 1 hacer        //Parte de negros
7:          color = negro
8:      fin para
9:      para i = 1; i == (N + 1)/2 − X; i = i + 1 hacer   //Lo mismo para la segunda parte de rojos
10:         color = rojo
11:     fin para
12:     X = X + 2                                  //Incrementa los negros
13:     Cambia de línea
14: fin para
```

15: $X = X - 2$ //Decrementa los negros y repetimos lo mismo para la segunda parte de la tela, pero ahora disminuyendo los negros
16: **para** $Y = 1;\ Y == N/2; Y = Y + 1$ **hacer** //Largo de la tela - segunda parte de la tela
17: **para** $i = 1; i == (N+1)/2 - X; i = i + 1$ **hacer** //Primera parte de rojos
18: *color = rojo*
19: **fin para**
20: **para** $j = 1; j == X; j = j + 1$ **hacer** //Parte de negros
21: *color = negro*
22: **fin para**
23: **para** $i = 1; i == (N+1)/2 - X; i = i + 1$ **hacer** //Lo mismo para la segunda parte de rojos
24: *color = rojo*
25: **fin para**
26: $X = X - 2$ //Decrementa los negros
27: *Cambia de línea*
28: **fin para**

El algoritmo describe cómo hacer un solo rombo de un color. Imagínate lo elaborado del trabajo y el tiempo que se requiere para hacer tejidos como los presentados en la Figura 8.11. Esto sin tomar en cuenta todo el proceso anterior requerido para conseguir madejas de colores.

Bibliografía

[1] Bell, T., Witten, I. H. y Fellows, M. (2015). Computer Science Unplugged, Estados Unidos.

[2] Brassard, G. y Bratley P. (1997). Fundamentos de Algoritmia. Madrid, España: Prentice Hall.

[3] Bribiesca, E., Galaviz, J. y Rajsbaum, S. (2010). Computación. En Enciclopedia de Conocimientos Fundamentales (509-785). Ciudad de México, México: UNAM-Siglo XXI.

[4] Cormen, T. H., Leiserson C. E., Rivest R. L. y Stein C. (2009). Introduction to Algorithms (2nda edición). Estados Unidos: MIT Press y McGraw-Hill.

[5] CSTA y ISTE. (2011). Computational Thinking Leadership Toolkit. Computer Science Teachers Association y la International Society for Technology in Education.

[6] CSTA y ISTE. (2011). Computational Thinking, Teacher Resources. Computer Science Teachers Association y la International Society for Technology in Education.

[7] Deitel, H. M. y Deitel P. J. (2003). Cómo Programar en C/C++, Pearson Educación.

[8] Garcia, D. (2012). The Beauty and Joy of Computing. University of California, Berkeley. Sitio web: https://bjc.berkeley.edu/

[9] Hewitt, P. G. (2007). Física Conceptual, México: Pearson Educación.

[10] Kramer, J. (2007). Is abstraction the key to computing? Communications of the ACM, 50, 36-42.

[11] Pagels, H. R. (1991). Los sueños de la razón: el ordenador y los nuevos horizontes de las ciencias de la complejidad. España: Gedisa.

[12] Paul, R. y Elder, L.(2003). La Mini-guía para el Pensamiento Crítico Conceptos y Herramientas. Fundación para el Pensamiento Crítico.

[13] Pitts, M. y Matson, L. (2008). Escribir con Glifos Mayas: Nombres, lugares y oraciones simples, una introducción no técnica a los glifos mayas. The Aid and Education Project, Inc.

[14] Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. SIGGRAPH '87: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, 21, 25-34.

[15] Starir, R. M. y Reynolds, G. W. (2000). Principios de Sistemas de Información. International Thomson Editores.

[16] Tanenbaum, A. S. (2003). Redes de Computadoras. Pearson Educación.

[17] Wing, J. M. (2008). Computational Thinking and Thinking about Computing. Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 366, 3717-3725.

[18] Wing, J. M. (2006). Computational Thinking. Communications of the ACM, 49, 33-35.

[19] Young, H. D. y Roger A. F. (2009), Física Universitaria, Volumen 1, México: Pearson Educación.

[20] Zapotecatl, J. L., Munoz-Meléndez A. y Gershenson C. (2016). Performance Metrics of Collective Coordinated Motion in Flocks. Proceedings of the Artificial Life Conference 2016, 322–329.

Glosario

Abstracción: Concepto general formado por la extracción de características de ejemplos concretos o el acto de dejar fuera de consideración una o más propiedades de un objeto complejo para atender otras.

Acumulador: Variable numérica entera o real que se utiliza en un algoritmo para almacenar un valor que se incrementa de una manera no constante.

Algoritmo: Serie de pasos ordenados que se siguen para resolver un problema.

Algoritmo de tiempo constante: Responde en un tiempo constante independientemente del tamaño de la instancia de entrada y se denota como $O(1)$.

Algoritmo de tiempo lineal: Responde en un tiempo lineal respecto al tamaño de la instancia de entrada y se denota como $O(n)$.

Algoritmo de tiempo logarítmico: Responde en un tiempo logarítmico con respecto al tamaño de la instancia de entrada y se denota como $O(log\ n)$.

Algoritmo de tiempo polinómico: Responde en un tiempo polinómico respecto al tamaño de la instancia de entrada y se denota como $O(n^c)$, para $c >= 1$.

Animación: Proceso de dotar de movimientos a los objetos para percibir la sensación de vivacidad.

Árbol taxonómico: Estructura que permite agrupar entidades de forma jerárquica según sus caracterísiticas.

Asignación: Instrucción secuencial utilizada en un algoritmo para dar un valor a una variable de cualquier tipo. Dicho valor resulta de evaluar una expresión.

Boids: Modelo clásico que simula cardúmenes de peces, parvadas de aves o rebaños, donde a cada pez o ave en el modelo se denomina genéricamente como boid (contracción de bird-oid object), al cual se aplican de manera autónoma tres fuerzas de dirección que controlan su movimiento: cohesión, separación y alineación.

Cárdumen de peces: Fenómeno colectivo ocasionado por cientos o miles de peces que se mueven como una unidad coordinada.

Compresión: Acto de reducir el volumen de los datos tratables para representar una determinada información empleando una menor cantidad de espacio.

Computación: Conjunto de conocimientos científicos y tecnológicos que estudian los métodos, procesos, y funcionamiento de computadoras que tienen la capacidad de almacenar, procesar y hacer uso de datos.

Contador: Variable numérica entera utilizada en un algoritmo para registrar el número de veces que ocurre un evento dado. Para ello, dicha variable se incrementa o decrementa de manera constante cada vez que ocurre un evento.

Corrección de Errores: Método de computadora capaz de reconocer cuando se han corrompido o dañado los datos y aplicar técnicas para reconstruir los datos originales.

Criptografía: Técnica de cifrado que altera las representaciones lingüísticas de determinados mensajes a fin de que sean incomprensibles a receptores no autorizados.

Datos: Representación de realidades concretas en su estado primario.

Diagrama de flujo: Representación de los pasos de un algoritmo a través de la combinación de elementos gráficos simples para cada uno de los tipos de instrucciones, tales como entrada o salida de datos, estructuras de decisión y proceso de datos.

Divide y vencerás: Estrategia de diseño de algoritmos para afrontar la complejidad de los problemas. Consiste en dividir un problema en problemas menores, resolverlos por separado y combinar las soluciones parciales para obtener la solución del problema original.

Economía del conocimiento: Sistema económico en el que la generación de valor tangible e intangible depende de la cantidad y calidad de información disponible.

Eliminación de los detalles: Proceso de dejar fuera de consideración una o más propiedades de un objeto con la finalidad de enfocarse sólo en algunas propiedades.

Estructura de control: Instrucción que controla el orden en que se ejecuta un algoritmo, de acuerdo con el los valores de variables y/o los requerimientos del algoritmo, por lo que permite modificar el flujo de ejecución de las instrucciones.

Función: Sub-algoritmo que realiza una tarea específica y devuelve un resultado.

Generalización: Proceso de formular conceptos genéricos a través de la extracción de cualidades comunes de ejemplos concretos.

Gráficas por computadora: Se centran en los fundamentos matemáticos y computacionales para generar imágenes y procesar imágenes.

Información: Es un conjunto de datos organizados de tal modo que adquieren un valor adicional más allá del propio.

Instancia: Cada uno de los ejemplares que pertenecen a su dominio de definición.

Iluminación: Proceso de simular la propagación de la luz en la escena para visualizar diversas propiedades de los objetos.

Lenguaje de programación: Lenguaje artificial que se usa para definir una secuencia de instrucciones para que una computadora las procese. Los lenguajes de programación se usan para construir programas que ejecuten los algoritmos en una computadora.

Llamada a función: El algoritmo principal, o una función invocadora, llama a una función invocada para realizar una subtarea. Esta definición se aplica también a los procedimientos.

Modelo: Representación abstracta (matemática, declarativa, visual, etc.) de fenómenos, sistemas o procesos.

Nivel de abstracción: Grado de detalle con el que se especifica una representación. Una representación en un alto nivel de abstracción especifica menos detalles que una representación en un bajo nivel de abstracción.

Orden de un algoritmo: Cantidad de recursos requeridos por un algoritmo para su realización.

Pensamiento computacional: 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). De manera más general, estrategia de solución de problemas basada en dichos procesos de pensamiento.

Pensamiento crítico: Modo de pensar en el cual el pensante mejora la calidad de su pensamiento al apoderarse de las estructuras inherentes del acto de pensar y al someterlas a estándares intelectuales.

Problema: Relación entre un conjunto de instancias, datos o situaciones (entrada) y un conjunto de soluciones (salida).

Procedimiento: Sub-algoritmo que realiza una tarea específica y no devuelve un resultado.

Recursión: Especificación de un proceso basado en su propia definición. La solución de un problema resuelto mediante la recursión depende de las soluciones de pequeñas instancias del mismo problema.

Renderizado: Término empleado en ambientes gráficos para referirse al proceso de interpretación y generación de una imagen, fotorrealista o no, partiendo de un modelo en 2D o 3D.

Seudocódigo: Descripción informal de alto nivel de un algoritmo que utiliza las convenciones de un lenguaje de programación real.

Simulación computacional: Conjunción de algoritmos matemáticos que modelan el comportamiento dinámico de sistemas físicos y herramientas computacionales que permiten reproducir y visualizar esta dinámica.

Simuladores en física: Programas de computadora que simulan fenómenos físicos y se utilizan como una herramienta educativa para mejorar los procesos de aprendizaje y la resolución de problemas.

Teoría de la Información: Shannon formaliza en esta teoría el concepto de información, estableciendo las bases para las comunicaciones, la compresión de datos; la detección y corrección de errores; y la criptografía.

Toma de decisiones: Proceso mediante el cual se realiza una elección entre diferentes opciones o formas posibles de resolver diferentes situaciones en la vi-

da. Básicamente, consiste en elegir una opción entre las disponibles, a efectos de resolver un problema.

Variable: Valor que puede modificarse durante la ejecución de un algoritmo, pero que es identificado mediante el mismo símbolo alfanumérico.

Introducción al pensamiento computacional: conceptos básicos para todos,
se terminó de imprimir en septiembre de 2018 en Agys Alevín S.C.
Retorno de Amores No. 14-102. Col. Del valle.
C. P. 03100, Ciudad de México.
En su composición se utilizó tipo Garamond.
Impreso en papel couché mate de 115 grs.
La edición consta de 120 ejemplares.