

de escritorio aun no han podido sustituir por completo a las grandes computadoras, las cuales también han evolucionado. En la actualidad, existe una gran variedad de computadoras orientadas hacia tareas específicas.

La revolución de las microcomputadoras no sólo ha incrementado el número de estos dispositivos en las oficinas, sino que ha abierto el campo a nuevos hábitats de computación. Un oficial de policía puede utilizar una computadora para registrar notas y pistas sobre un crimen. Otro buen ejemplo lo constituye David Solove, que utiliza una computadora portátil, una cámara digital y un escáner para generar un diario *online* para su familia y amigos acerca de la vida en un circo.

Las computadoras en la actualidad: una breve disección

En la actualidad, la gente trabaja con *mainframes*, supercomputadoras, estaciones de trabajo, portátiles, computadoras de bolsillo, computadoras incrustadas y, por supuesto, con PC. Aunque todas ellas están basadas en la misma tecnología, todas estas máquinas tienen sustanciales diferencias.

Mainframes y supercomputadoras

Antes de la revolución de las microcomputadoras, la mayoría de la información que se procesaba era realizada por **mainframes**, máquinas del tamaño de una gran sala. En la actualidad, las empresas importantes como bancos y líneas aéreas, siguen usando este tipo de computadoras para trabajos que impliquen grandes tareas de computación. Pero los *mainframes* actuales son más pequeños y baratos que sus predecesores; un *mainframe* típico actual puede tener el tamaño de un frigorífico y un precio aproximado a 1 millón de dólares. Estas computadoras de carácter industrial suelen ser invisibles para el público en general porque deben estar encerradas en salas climatizadas. Pero el hecho de no poder verlas no significa que no puedan utilizarse. Cuando usted efectúa una reserva para un vuelo o deposita dinero en su cuenta bancaria, un *mainframe* está involucrado en la operación. La agencia de viajes y el cajero del banco comunican con un *mainframe* a través de un **terminal**: una combinación de teclado y pantalla con algo de potencia operativa que transfiere la información desde y hacia la computadora principal. Dicha computadora puede encontrarse en otra sala o, incluso, en otro país.

Un *mainframe* puede comunicarse con distintos usuarios de forma simultánea mediante una técnica conocida como **compartición de tiempo** (tiempo compartido). Este sistema, por ejemplo, permite que las agencias de viaje de un país efectúen reservas usando la misma computadora y la misma información de vuelos a la vez. La compartición de tiempo también permite que los usuarios con diversas necesidades de computación utilicen equipos de procesamiento que tienen un elevado coste. Por ejemplo, muchos investigadores precisan de más capacidad de procesamiento matemático del que pueden obtener de sus PC, por lo que deben echar mano de un *mainframe* más potente. Una máquina con compartición de tiempo puede atender simultáneamente las necesidades de científicos e ingenieros de diferentes departamentos trabajando en distintos proyectos.

A pesar de ello, existen muchos procesos que no tienen suficiente con la capacidad de procesamiento de los *mainframe*; el tradicional «*big iron*» simplemente no tiene suficiente velocidad para los intensos trabajos de cálculo necesarios en las predicciones meteorológicas, el diseño de redes telefónicas, la simulación de accidentes de tráfico, las prospecciones petrolíferas, la animación por computadora y los diagnósticos médicos. Los usuarios con este tipo de necesidades precisan de computadoras más potentes y rápidas, las **supercomputadoras**. Por ejemplo, la supercomputadora Blue Mountain del Laboratorio Nacional Los Álamos del Departamento de Energía de los Estados Unidos puede llevar a cabo 1,6 billones de operaciones por segundo. La máquina se emplea para simular ensayos nucleares y realizar cálculos intensivos para otros proyectos de investigación.

Servidores, estaciones de trabajo y PC

Para aplicaciones que dan servicio a múltiples usuarios, se emplea una computadora de gama alta llamada **servidor**, una máquina diseñada para ofrecer el software y otro tipo de recursos al resto de computadoras conectadas a una red. Aunque casi cualquier máquina puede usarse como servidor, algunas de ellas están especialmente diseñadas para este propósito (las redes y los servidores se tratan más adelante en este mismo capítulo y en otros de este mismo libro).

La **estación de trabajo** (una computadora de sobremesa de gama alta con gran potencia de procesamiento) se emplea para aplicaciones interactivas como el análisis de datos científicos a gran escala. Las estaciones de trabajo son ampliamente utilizadas por científicos, ingenieros, analistas financieros, diseñadores y animadores cuyos trabajos implican unas grandes necesidades operativas. Desde luego, como la inmensa mayoría de términos informáticos, **estación de trabajo** significa cosas diferentes para cada tipo de persona. Algunos incluyen en este término a todas las computadoras de escritorio y terminales. Los que reservan el término para las máquinas de escritorio más potentes admiten que la línea que separa las estaciones de trabajo de los PC es muy difusa. A medida que las estaciones de trabajo se abaratán y los PC se hacen cada vez más potentes, esta línea se está convirtiendo más en una distinción de marketing que en una de carácter técnico.

La mayoría de usuarios de computadoras no necesitan de la potencia de una estación de trabajo científica para su trabajo diario. Cualquier PC moderno tiene total capacidad para trabajar con procesadores de texto, realizar operaciones contables, jugar, disfrutar con música digital y vídeo, etc. Una computadora personal, tal y como su propio nombre indica, está casi siempre dedicado a un único usuario.

Un comentario acerca de la terminología: hay veces en las que el término **computadora personal** y **PC** generan confusión desde que, en 1981, IBM llamó a su computadora de escritorio IBM Personal Computer. Ésta es la razón por la que, ocasionalmente, ambos términos se utilizan sólo para describir las computadoras IBM o las máquinas compatibles con el hardware IBM («la oficina tiene una red de Mac y PC»). Pero en otro contexto, PC puede describir cualquier computadora monousuario de propósito general («cada estudiante necesita un PC para conectar con Internet»). A lo largo de este libro, cuando hagamos referencia al término PC, estaremos hablando de cualquier PC, no sólo de los fabricados por IBM o los que sean compatibles con sus productos.

Las computadoras personales actuales tienen una gran variedad de formas. El iMac de Apple incluye la CPU, el monitor y los dispositivos de almacenamiento en



Figura 1.3. Las computadoras personales actuales se presentan en variedad de formas. La torre Dell Dimension de la imagen es un diseño más tradicional, con el monitor separado de la CPU y las unidades de almacenamiento. Hay otros diseños que se presentan en un formato «todo en uno», más acordes con las corrientes de diseño actuales.

un diseño «todo-en-uno». Las torres Dell Dimension tienen un diseño más tradicional, con el monitor separado de la CPU, un diseño más cercano a lo que la mayoría estamos acostumbrados a ver.

Computadoras portátiles

Hace dos décadas, los términos **computadora personal** y **computadora de escritorio** fueron intercambiados porque, virtualmente, todos los PC eran computadoras de escritorio. Sin embargo, en la actualidad, uno de los segmentos de mercado de mayor crecimiento tiene que ver con las máquinas que no se encuentran atadas al escritorio: los **portátiles**.

Desde luego, **portátil** es un término relativo. Los primeros «portátiles» eran maletas de unos 9 kilos de peso con teclados plegables y pequeñas televisiones que actuaban a modo de monitores. En la actualidad, estas computadoras de «equipaje» han sido sustituidas por los **notebook** (a veces llamadas también **computadoras laptop**) de pantalla plana y alimentados por baterías que son tan ligeros que pueden estar colocados en su regazo mientras se trabaja con ellos o llevarse en un maletín mientras están cerrados.

Los portátiles actuales pesan entre 1,5 y 4,5 kilos, dependiendo de la máquina, y muchos de ellos funcionan como PC de escritorio; estos pesados aunque potentes dispositivos reciben el nombre de **sustitutos de escritorio**. Los portátiles extra-ligeros y con mucha capacidad de transporte reciben a veces el nombre de **subportátiles**. Para mantener un peso y tamaño bajos, los fabricantes suelen dejar fuera algunos componentes que forman parte habitual de cualquier máquina de escritorio. Por ejemplo, algunos portátiles no disponen de unidades ópticas para la lectura o grabación de un CD-



Figura 1.4. Estos dos portátiles sólo son una pequeñísima muestra de los disponibles en la actualidad. El IBM ThinkPad T40 puede convertirse de un portátil a una computadora de sobremesa usando una *docking station* (en la imagen se ve en la parte posterior de la computadora) o replicador de puertos. Por su parte, el dispositivo Hewlett-Packard iPAQ Pocket PC utiliza una versión de Microsoft Windows diseñada para computadoras de bolsillo e incluye versiones reducidas de las populares aplicaciones de Office.

ROM o un DVD. Algunos de ellos disponen de bahías de expansión en los cuales se pueden insertar estos dispositivos (uno cada vez). Otros tienen varios puertos que permiten la conexión de los mismos mediante cables. Unos cuantos modelos están equipados con ***docking stations*** o **duplicadores de puertos**, los cuales permiten a un usuario conectar el portátil a un monitor, teclado, ratón y disco duro externos. A menudo, muchas personas que tienen una gran movilidad en su trabajo utilizan *docking stations* para convertir sus portátiles en PC de escritorio totalmente operativos cuando regresan a su oficina. Incluso sin este elemento, un portátil puede conectarse fácilmente a periféricos y redes cuando están dentro del entorno fijo.

Las **computadoras de bolsillo**, que con frecuencia son lo bastante pequeñas como para alojarse en un bolsillo, cumplen las necesidades de aquellos usuarios que valoran más la movilidad que un teclado y una pantalla convencionales. Las *docking cradles* para computadoras de bolsillo permiten compartir información con portátiles y PC convencionales. Este tipo de máquinas suelen recibir el nombre de **PDA (Asistente digital personal, Personal Digital Assistants)** o **computadoras de bolsillo**.

A pesar del tamaño, la mayoría de computadoras portátiles son máquinas de propósito general construidas a base de microprocesadores similares a los incluidos en los modelos de escritorio. Pero la portabilidad tiene un precio: los portátiles suelen ser más caros en comparación con una máquina de escritorio de características similares. También resultan más difíciles de actualizar cuando aparece nuevo hardware.

Computadoras incrustadas y de carácter específico

No todas las computadoras son máquinas de propósito general. Algunas están especializadas y llevan a cabo tareas concretas, como controlar la temperatura y la humedad en los edificios de oficinas de última generación o controlar el ritmo cardiaco mien-

tras entrena. Las **computadoras incrustadas** mejoran todo tipo de productos para el consumidor: relojes de pulsera, juguetes, videojuegos, equipos de música, DVRs (Grabadores de vídeo digital, *Digital Video Recorder*) e, incluso, hornos. De hecho, ¡más del 90 por ciento de los microprocesadores que se fabrican en el mundo se encuentran ocultos dentro de electrodomésticos comunes y aparatos electrónicos! Gracias a las computadoras incrustadas, es probable que cualquiera de los nuevos coches tengan más potencia de computación que los PC que se venden.

Muchas de las computadoras de carácter específico son, en el fondo, similares a las de propósito general. Pero a diferencia de sus «primos de escritorio», estas máquinas disponen de programas grabados directamente en la placa y no pueden ser alterados. Este tipo de programa se conoce como ***firmware***: un híbrido de hardware y software.

Conexiones de computadoras: la revolución de Internet

Hemos visto cómo los adelantos en distintas tecnologías han producido nuevos tipos de computadoras. Cada uno de estos avances tecnológicos tuvo un impacto en nuestra sociedad y permitió que la gente encontrara nuevas formas de utilizar dichas computadoras. La mayoría de los historiadores dejaron de contar las generaciones cuando la microcomputadora se hizo algo corriente; ahora mismo, resulta complicado pensar que pueda haber otro avance que tenga el impacto del microprocesador. Pero mientras el mundo intentaba recuperarse de esta revolución, otra nueva estaba empezando a forjarse: la revolución de la red. Hoy en día podemos decir que el principio de los años 90 fue el comienzo de la era de la **computación interpersonal**.

Todas las personas están atrapadas en una **red ineludible de mutualidad**, atadas a una sola prenda de destino. Cualquier cosa que afecta directamente a **una**, afecta a **todas** indirectamente...

—Martin Luther King, Jr.

El surgimiento de las redes

La invención del tiempo compartido en los años 60 permitió que múltiples usuarios se conectaran a un único *mainframe* central mediante terminales individuales. Cuando las computadoras personales comenzaron a sustituir a estos últimos, muchos usuarios se dieron cuenta que tenían toda la potencia de computación que necesitaban en sus escritorios. A pesar de ello, también encontraron que enlazar algunas de estas computadoras en una **LAN (Red de área local, Local Area Network)**, o **red** para abreviar, ofrecía muchas ventajas. Cuando las máquinas se agrupaban, podían compartir recursos como dispositivos de almacenamiento, impresoras e, incluso, capacidad de procesamiento. Mediante una red, una única impresora de alta velocidad podía dar servicio a toda una oficina. Como premio añadido, la gente podía usar las computadoras para enviar y recibir mensajes electrónicos a través de las redes.

Las ventajas de la comunicación electrónica y la compartición de recursos se vio multiplicada cuando las redes más pequeñas se unieron en otras de mayor tamaño. La aparición de la tecnología de telecomunicación permitió que las **WAN (Red de área amplia, Wide Area Network)** no respetaran ni continentes ni océanos. Una computadora remota podía conectar con una red a través de las líneas telefónicas estándar usando un **módem** (un dispositivo electrónico que podía convertir los datos de la computadora en señales compatibles con el sistema telefónico). Los bancos, las agencias gubernamentales y otras instituciones separadas geográficamente comenzaron a cons-

La única frase con la que no he estado de acuerdo es, «**¿Por qué siempre lo hemos hecho de este modo?**». Yo siempre le digo a los jóvenes, «**Adelante, hazlo. Siempre podrás disculparte más adelante.**».

—Grace Murray Hopper

GRACE MURRAY HOPPER NAVEGA POR EL SOFTWARE

La Asombrosa Grace, la gran dama del software, tuvo poco de lo que disculparse cuando murió en 1992 a la edad de 85 años. Más que cualquier otra mujer, Grace Murray Hopper ayudó a diseñar el curso de la industria informática desde sus primeros días.

Hopper obtuvo un doctorado de Yale en 1928 y dio clases de matemáticas durante 10 años en Vassar, antes de unirse a la reserva naval de los Estados Unidos en 1943. La armada la asignó al *Bureau of Ordnance Computation* en Harvard, donde trabajó con el Mark I de Howard Aiken, la primera computadora digital a gran escala. Escribió programas y manuales de operaciones para el Mark I, Mark II y Mark III.

Aiken preguntaba con frecuencia a su compañera, «¿Estás haciendo cálculos?». Cuando ella no «estaba haciendo cálculos», Hopper replicaba que estaba «poniendo a punto» la computadora. En la actualidad, este proceso de «puesta a punto» (conocido por todo programador por su nombre en inglés, *debugging* [depuración]) es el que realiza todo programador para localizar y eliminar los errores de sus programas. Los científicos e ingenieros se han referido a los defectos mecánicos como *bugs* (bichos) durante décadas; Thomas Edison hizo mención en 1978 a estos *bugs* en sus invenciones. Pero cuando Hopper los utilizó por primera vez, se estaba refiriendo a un bicho real: una polilla de unos cinco centímetros que había capturado y que había provocado que la todopoderosa Mark II se detuviese! El cadáver de esta polilla se encuentra en un libro de registros guardado en un museo naval en Virginia.

Hopper se dio cuenta enseguida que las empresas podían hacer un buen uso de las computadoras. Tras la Segunda Guerra Mundial, dejó Harvard para trabajar en la UNIVAC I (la primera computadora comercial de propósito general) y en otras computadoras comerciales. Representó un papel fundamental en el desarrollo del primer compilador (un tipo de traductor al lenguaje de las computadoras) y del COBOL, el primer lenguaje informático diseñado para el desarrollo de aplicaciones empresariales.

A lo largo de su carrera, Hopper permaneció unida a la Armada. Cuando se retiró con el grado de contralmirante a la edad de 79 años, la lista de sus logros ocupaba ocho páginas, a un solo espacio, de su historia en el ejército.

Pero el mayor logro de Hopper fue, probablemente, el resultado de su infatigable cruzada contra el «Siempre lo hemos hecho de esta forma». En los primeros días de la informática, trabajó para persuadir a los hombres de negocios a emplear la nueva tecnología. En sus últimos años, hizo campaña para intentar que el Pentágono y la industria dejaran los *mainframes* y se dirigieran hacia redes de pequeñas computadoras. Su intensa campaña le hizo ganarse fama de polémica y revolucionaria. Esto no molestó a la Asombrosa Grace, cuya máxima favorita era «un barco en el puerto está a salvo, pero esto no es para lo que está construido».

El software informático actual es tan sofisticado que es casi inaccesible para un enorme grupo de usuarios. Al igual que una superproducción puede hacernos olvidar que estamos viendo una película, el mejor software nos permite realizar nuestro trabajo sin preocuparnos del flujo de instrucciones que viajan por el interior de la computadora. Pero ya esté escribiendo en un papel, resolviendo un problema de cálculo, volando en una nave espacial simulada o explorando los rincones y recovecos de Internet, su entorno imaginario permanece en una increíblemente compleja subestructura software. El proceso de creación mediante software es una de las actividades intelectualmente más desafiante realizadas por la gente.

En este capítulo veremos el proceso de convertir las ideas en programas informáticos útiles. Empezaremos echando un vistazo al diseño de sistemas y al ciclo de desarrollo de un programa típico. Examinaremos los lenguajes de programación y el modo en que los programadores los utilizan para crear software. Además, examinaremos cómo los usuarios de las computadoras de benefician de los lenguajes de programación que generan aplicaciones, sistemas operativos y utilidades. También nos enfrentaremos con los problemas implicados en la producción de software fiable y consideraremos las implicaciones que supone depender de un sistema inestable. Durante el proceso de exploración del software, veremos la forma en la que el trabajo de programadores, analistas, ingenieros de software e informáticos afecta a sus vidas y a nuestro trabajo.

¿De qué modo programa la gente?

Muchos usuarios de computadoras dependen de aplicaciones programadas profesionalmente (hojas de cálculo, programas de edición de imágenes, navegadores web, etc.) como herramientas para la resolución de sus problemas. Pero, en ciertas ocasiones, es necesario o deseable escribir un programa en lugar de utilizar uno escrito por otra persona. Como actividad humana, la programación de computadoras es algo relativamente nuevo. Pero la **programación** es una forma especializada del antiguo modelo de resolución de problemas.

Esta actividad suele implicar cuatro pasos:

- **Entender el problema.** Definir el problema con claridad es, con frecuencia, el paso más importante, y casi siempre más descuidado, que debe darse para la resolución de ese problema.
- **Idear un plan para la resolución del problema.** ¿Qué recursos, personas, información, computadoras, software y datos tenemos disponibles? ¿Cómo deben ponerse estos recursos en funcionamiento para resolver el problema?
- **Llevar a cabo el plan.** Esta fase suele estar solapada con el paso 2, ya que muchos esquemas de resolución de problemas suelen desarrollarse sobre la marcha.
- **Evaluar la solución.** ¿Se ha resuelto correctamente el problema? ¿Es una solución válida para otros problemas?

El proceso de programación también suele estar dividido en cuatro fases, aunque casi siempre están solapadas unas con otras:

- Definición del problema.
- Creación, depuración y verificación del algoritmo.

Es el único trabajo en el que puedo pensar como un **ingeniero y un artista**. Es un elemento increíble, riguroso y técnico que me **encanta** porque tienes que **pensar de un modo muy preciso**. Por otro lado, tiene un lado ampliamente creativo donde las **fronteras de la imaginación son sus únicos límites**.

—Andy Hertzfeld,
co-diseñador del Macintosh

- Escritura del programa
- Verificación y depuración del programa.

Casi todos los problemas de programación suelen ser tan complejos que no pueden resolverse de una sola vez. Para transformar un problema en un programa, el programador suele crear una lista de problemas más pequeños, cada uno de los cuales puede subdividirse a su vez en subproblemas que también pueden subdividirse.

Este proceso, llamado **refinamiento por pasos**, es similar al que realiza un escritor para esbozar las líneas maestras de lo que será su obra. Los programadores suelen referirse a este tipo de proceso como un **diseño de arriba a abajo**, ya que el problema se aborda desde arriba, con las ideas principales, y se va desarrollando hacia abajo con los detalles concretos.

El resultado es un **algoritmo**, un conjunto de instrucciones paso a paso que, una vez completadas, resuelven el problema original (recuerde la receta de la tortilla de patatas del Capítulo 4). Los programadores suelen escribir algoritmos en un formato llamado **pseudocódigo**, un cruce entre lenguaje informático y lenguaje real. Cuando todos los detalles del algoritmo están en su sitio, el programador puede traducir todas las pseudoinstrucciones en un lenguaje informático.

De la idea al algoritmo

Vamos a desarrollar un sencillo algoritmo para ilustrar todo el proceso. Emperezaremos con el planteamiento del problema:

Un profesor de un colegio necesita un programa para jugar a «adivinar el número» que ayude a que sus estudiantes desarrollen su lógica y les permita practicar la aritmética. En este juego, la computadora escoge un número comprendido entre 1 y 100 y ofrece al jugador varias oportunidades para adivinarlo. Tras cada intento fallido, la máquina responde diciendo si el número introducido es mayor o menor que el que debe adivinarse.

En resumen, el problema es escribir un programa que pueda:

jugar a "adivinar el número"

Refinamiento por pasos

Lo primero de todo es dividir el problema en tres partes: comienzo, mitad y finalización. Cada una de estas tres partes es en sí misma un pequeño problema de programación por resolver.

```
inicio del juego
repetir el proceso hasta que se acierte el número o se alcance el número
máximo de intentos
fin del juego
```

Estos tres pasos son el esqueleto desnudo del algoritmo. Una vez completado, estas tres partes serán ejecutadas en secuencia. El siguiente refinamiento debe completar algunos detalles de cada parte:

Las Ciencias de la computación o Informática es la disciplina que trata de establecer una base científica para temas tales como el diseño asistido por computadora, la programación de computadoras, el procesamiento de la información, las soluciones algorítmicas de problemas y el propio proceso algorítmico. Proporciona los fundamentos para las aplicaciones informáticas actuales, así como la base para la infraestructura de computación del futuro.

Este libro proporciona una introducción exhaustiva a esta ciencia. Investigaremos un amplio rango de temas, incluyendo la mayoría de los que componen el currículum de los estudios universitarios típicos de Ciencias de la computación. Queremos abarcar el ámbito completo de este campo, así como la dinámica del mismo. Por ello, además de los propios temas, trataremos de analizar su desarrollo histórico, el estado actual de las investigaciones y las perspectivas de futuro. Nuestro objetivo es conseguir una comprensión de las Ciencias de la computación desde el punto de vista funcional; una comprensión que permita al lector continuar con estudios más especializados en esta área, y que también permita a aquellos que trabajan en otros campos sobrevivir en una sociedad cada vez más tecnificada.

0.1 El papel de los algoritmos

Comenzaremos con el concepto más fundamental de las Ciencias de la computación: el concepto de algoritmo. Informalmente, un **algoritmo** es un conjunto de pasos que define cómo hay que realizar una tarea. (Seremos más precisos a este respecto en el Capítulo 5.) Por ejemplo, existen algoritmos para cocinar (recetas), para encontrar el camino en una ciudad desconocida (direcciones), para hacer funcionar una lavadora (instrucciones que normalmente pueden encontrarse en el manual), para tocar música (expresadas mediante partituras) y para realizar trucos de magia (Figura 0.1).

Para que una máquina como una computadora pueda llevar a cabo una tarea, es preciso diseñar y representar un algoritmo de realización de dicha tarea y en una forma que sea compatible con la máquina. A la representación de un algoritmo se la denomina **programa**. Por comodidad de los seres humanos, los programas informáticos suelen imprimirse en papel o visualizarse en las pantallas de las computadoras. Sin embargo, para comodidad de las máquinas, los programas se codifican de una manera compatible con la tecnología a partir de la cual esté construida la máquina. El proceso de desarrollo de un programa, de codificarlo en un formato compatible con la máquina y de introducirlo en una máquina se denomina **programación**. Los programas y los algoritmos que representan se denominan colectivamente **software**, por contraste con la propia máquina que se conoce con el nombre de **hardware**.

El estudio de los algoritmos comenzó siendo un tema del campo de las matemáticas. De hecho, la búsqueda de algoritmos fue una actividad de gran importancia para los matemáticos mucho antes del desarrollo de las computadoras actuales. El objetivo era determinar un único conjunto de instrucciones que describiera cómo resolver todos los problemas de un tipo concreto. Uno de los ejemplos mejor conocidos de estas investigaciones pioneras es el algoritmo de división para el cálculo del cociente de dos números de varios dígitos. Otro ejemplo es el algoritmo de Euclides descubierto por este matemático de la anti-

Figura 0.1 Algoritmo para un truco de magia.

Efecto: el mago coloca boca abajo sobre una mesa algunas cartas extraídas de un mazo de cartas normal y las baraja suficientemente, distribuyéndolas después sobre la mesa. Después, a medida que el público solicita cartas rojas o negras, el mago da la vuelta a cartas del color solicitado.

El truco:

- Paso 1. De un mazo de cartas normal, seleccione diez cartas rojas y diez negras. Coloque estas cartas boca arriba sobre la mesa en dos montones, de acuerdo con su color.
- Paso 2. Anuncie que ha seleccionado algunas cartas rojas y algunas negras.
- Paso 3. Tome las cartas rojas. Con el pretexto de alinear el mazo formado por esas cartas, manténgalas boca abajo en su mano izquierda y, con el pulgar y el índice de su mano derecha tire de cada extremo del mazo, de modo que cada carta quede ligeramente curvada *hacia atrás*. Luego, coloque el mazo de cartas rojas boca abajo sobre la mesa al mismo tiempo que dice: "En este mazo tenemos las cartas rojas".
- Paso 4. Tome las cartas negras. De forma similar a como ha hecho en el Paso 3, proporcione a estas cartas una ligera curvatura *hacia adelante*. Luego devuelva estas cartas a la mesa colocando el mazo boca abajo al mismo tiempo que dice: "Y en este otro mazo están las cartas negras".
- Paso 5. Inmediatamente después de colocar las cartas negras sobre la mesa, utilice ambas manos para mezclar las cartas negras y rojas (que todavía están boca abajo) y distribuirlas sobre la mesa. Explique que está mezclando convenientemente las cartas.
- Paso 6. Mientras que haya cartas boca abajo sobre la mesa, ejecute repetidamente los siguientes pasos:
 - 6.1. Pida a alguien del público que solicite una carta roja o negra.
 - 6.2. Si el color solicitado es rojo y hay alguna carta boca abajo con apariencia cóncava, dé la vuelta a esa carta al mismo tiempo que dice "He aquí una carta roja".
 - 6.3. Si el color solicitado es negro y hay alguna carta boca abajo con apariencia convexa, dé la vuelta a dicha carta al mismo tiempo que dice "He aquí una carta negra".
 - 6.4. En caso contrario, anuncie que ya no hay más cartas del color solicitado y dé la vuelta a las cartas restantes para demostrar su afirmación.

gua Grecia que permite determinar el máximo común divisor de dos números enteros positivos (Figura 0.2).

Una vez que se ha encontrado un algoritmo para llevar a cabo una determinada tarea, la realización de esta ya no requiere comprender los principios en los que el algoritmo está basado. En lugar de ello, la realización de la tarea se reduce al proceso de seguir simplemente las instrucciones proporcionadas. (Podemos emplear el algoritmo de división para calcular un cociente o el algoritmo de Euclides para hallar el máximo común divisor sin necesidad de entender por qué funciona el algoritmo.) En cierto sentido, la inteligencia requerida para resolver ese problema está codificada dentro del algoritmo.

Es esta capacidad de capturar y transmitir inteligencia (o al menos un comportamiento inteligente) por medio de algoritmos lo que nos permite construir máquinas que lleven a cabo tareas de utilidad. En consecuencia, el nivel de

Figura 0.2 El algoritmo de Euclides para calcular el máximo común divisor de dos números enteros positivos.

Descripción: este algoritmo presupone que la entrada está formada por dos enteros positivos y calcula el máximo común divisor de dichos valores.

Procedimiento:

Paso 1. Asigne a M y N el valor del mayor y el menor de los dos números proporcionados como entrada, respectivamente.

Paso 2. Divida M entre N y llame R al resto.

Paso 3. Si R es distinto de 0, entonces asigne a M el valor de N, asigne a N el valor de R y vuelva al Paso 2. En caso contrario, el máximo común divisor será el valor asignado actualmente a N.

inteligencia mostrado por las máquinas está limitado por la inteligencia que podamos transmitir mediante algoritmos. Solo podemos construir una máquina para llevar a cabo una tarea si existe un algoritmo que permita realizar esa tarea. A su vez, si no existe ningún algoritmo para resolver un problema, entonces la solución de ese problema cae fuera de la capacidad de las máquinas disponibles.

La identificación de las limitaciones de las capacidades algorítmicas terminó convirtiéndose en uno de los temas de las matemáticas en la década de 1930 con la publicación del teorema de incompletitud de Kurt Gödel. Este teorema afirma, en esencia, que en cualquier teoría matemática que abarque nuestro sistema aritmético tradicional, hay enunciados cuya verdad o falsedad no puede establecerse por medios algorítmicos. Por decirlo en pocas palabras, cualquier estudio completo de nuestro sistema aritmético, cae fuera de las capacidades de las actividades algorítmicas.

El enunciado de este hecho removió los cimientos de las matemáticas y el estudio de las capacidades algorítmicas que se inició a partir de ahí fue el comienzo del campo que hoy día conocemos como Ciencias de la computación. De hecho, es el estudio de los algoritmos lo que forma la base fundamental de estas ciencias.

0.2 La historia de la computación

Las computadoras actuales tienen una genealogía muy extensa. Uno de los primeros dispositivos de computación fue el ábaco. La historia nos dice que sus raíces se hunden, muy probablemente, en la antigua China y fue utilizado por las antiguas civilizaciones griega y romana. Dicha máquina es muy simple, estando compuesta por una serie de cuentas ensartadas en unas varillas que a su vez se montan sobre un marco rectangular (Figura 0.3). Al mover las cuentas hacia adelante y hacia atrás en las varillas, sus posiciones representan los valores almacenados. Es gracias a las posiciones de las cuentas que esta "computadora" representa y almacena los datos. Para el control de la ejecución de un algoritmo, esta máquina depende del operador humano. Por tanto, el ábaco es, por sí solo, un sistema de almacenamiento de datos; se precisa la intervención de una persona para poder disponer de una máquina de computación completa.

- a. 0000111100001111 b. 00110011000000010000000
 c. 0000101010100000
9. Cite una ventaja de representar las imágenes mediante estructuras geométricas, en lugar de mediante mapas de bits. ¿Qué ventaja tienen las técnicas de mapa de bits frente a las basadas en estructuras geométricas?
10. Suponga que codificamos una grabación estéreo de una hora de música utilizando una tasa de muestreo de 44.100 muestras por segundo como se explica en el texto. Compare el tamaño de la versión codificada con la capacidad de almacenamiento de un CD.

1.5 El sistema binario

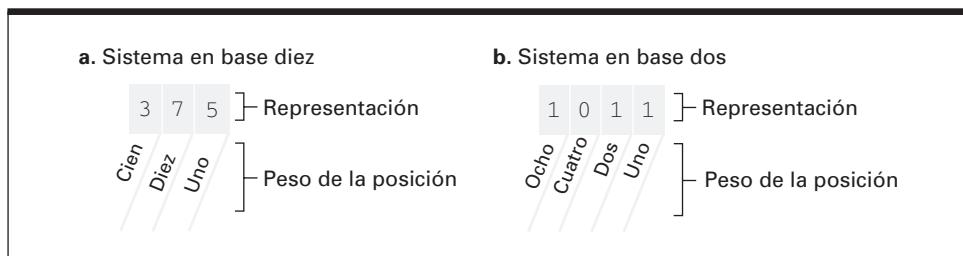
En la Sección 1.4 hemos visto que la notación binaria es una forma de representar valores numéricos utilizando solo los dígitos 0 y 1, en lugar de los diez dígitos 0 a 9 que se emplean en el sistema más común en notación en base diez. Vamos a echar ahora un vistazo más detallado a la notación binaria.

Notación binaria

Recuerde que en el sistema en base diez, cada posición de una representación numérica está asociada con un determinado peso. En la representación 375, el 5 se encuentra en la posición asociada con el peso uno, el 7 está en la posición asociada con el peso diez y el 3 está en la posición asociada con el peso cien (Figura 1.15a). Cada uno de esos pesos es diez veces el peso de la posición situada a su derecha. El valor representado por la expresión completa se obtiene multiplicando el valor de cada dígito por el peso asociado con la posición de este dígito y luego sumando esos productos. Para ilustrar el proceso, el patrón 375 representa $(3 \times \text{cien}) + (7 \times \text{diez}) + (5 \times \text{uno})$, lo que en notación más técnica sería $(3 \times 10^2) + (7 \times 10^1) + (5 \times 10^0)$.

La posición de cada dígito en notación binaria también está asociada con un peso, pero el peso de cada posición es igual al doble del peso asociado con la posición de su derecha. Para ser más precisos, el dígito más a la derecha en una representación binaria está asociado con uno (2^0), la siguiente posición a la izquierda está asociada con dos (2^1), la siguiente está asociada con

Figura 1.15 Sistema en base diez y sistema binario.



cuatro (2^2), la siguiente con ocho (2^3), y así sucesivamente. Por ejemplo, en la representación binaria 1011, el 1 de más a la derecha está en la posición de peso uno, el 1 situado a continuación está en la posición de peso dos, el 0 se encuentra en la posición de peso cuatro y el 1 más a la izquierda está en la posición de peso ocho (Figura 1.15b).

Para obtener el valor correspondiente a una representación binaria, seguimos el mismo procedimiento que en base diez: multiplicamos el valor de cada dígito por el peso asociado con su posición y sumamos los resultados. Por ejemplo, el valor representado por 100101 es 37, como se muestra en la Figura 1.16. Observe que, como la notación binaria solo utiliza los dígitos 0 y 1, este proceso de multiplicación y suma se reduce simplemente a sumar los pesos de las posiciones que están ocupadas por 1s. Por tanto, el patrón binario 1011 representa el valor once, porque los 1s se encuentran en las posiciones de pesos uno, dos y ocho.

En la Sección 1.4 hemos aprendido a contar en notación binaria, lo que nos ha permitido codificar enteros de pequeño tamaño. Para calcular las representaciones binarias de valores grandes es preferible emplear la técnica descrita por el algoritmo de la Figura 1.17. Apliquemos este algoritmo al valor trece (Figura 1.18). Dividimos primero trece entre dos, obteniendo como cociente seis y como resto uno. Dado que el cociente no es cero, el Paso 2 nos dice que hay que dividir el cociente (seis) entre dos, obteniendo como nuevo cociente tres y un resto igual a cero. El nuevo cociente sigue sin ser cero, por lo que lo dividimos entre dos, obteniendo como cociente uno y como resto uno. Una vez más, dividimos el último cociente (uno) entre dos, obteniendo esta vez un cociente de cero y un

Figura 1.16 Decodificación de la representación binaria 100101.

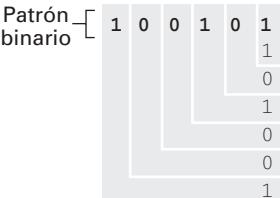
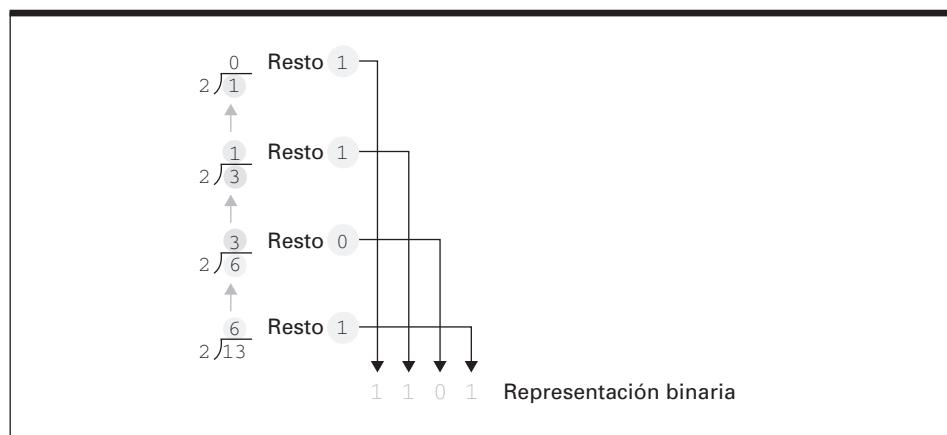
Patrón binario 	1 0 0 1 0 1 1 x uno = 1 0 x dos = 0 1 x cuatro = 4 0 x ocho = 0 0 x dieciséis = 0 1 x treinta y dos = 32	37 Total
Valor del bit Peso de la posición		

Figura 1.17 Algoritmo para determinar la representación binaria de un entero positivo.

- Paso 1. Dividir el valor entre dos y anotar el resto.
- Paso 2. Mientras que el cociente obtenido sea distinto de cero, continuar dividiendo entre dos el último cociente y anotar el resto.
- Paso 3. Ahora que ha obtenido un cociente igual a cero, la representación binaria del valor original está compuesta por los restos obtenidos, ordenados de derecha a izquierda según el orden en que fueron anotados.

Figura 1.18 Aplicación del algoritmo de la Figura 1.17 para obtener la representación binaria de trece.



resto igual a uno. Puesto que ya tenemos un cociente igual a cero, continuamos con el Paso 3, donde vemos que la representación binaria del valor original (trece) es 1101, obtenida a partir de la lista de restos.

Suma binaria

Para comprender el proceso de sumar dos enteros representados en binario, recordemos primero el proceso de suma de valores representados en notación tradicional en base diez. Considere, por ejemplo, el siguiente problema:

$$\begin{array}{r} 58 \\ + 27 \\ \hline \end{array}$$

Comenzamos sumando el 8 y el 7 de la columna situada más a la derecha para obtener la suma 15. Escribimos un 5 debajo de dicha columna y nos llevamos el 1 a la columna siguiente, obteniendo

$$\begin{array}{r} 1 \\ 58 \\ + 27 \\ \hline 5 \end{array}$$

Ahora sumamos el 5 y el 2 de la columna siguiente junto con el 1 que hemos acarreado obteniéndose como suma 8, que será el valor que escribamos debajo de la columna. El resultado es el siguiente:

$$\begin{array}{r} 58 \\ + 27 \\ \hline 85 \end{array}$$

En resumen, el procedimiento consiste en ir avanzando de derecha a izquierda mientras vamos sumando los dígitos de cada columna escribiendo el dígito menos significativo de dicha suma bajo la columna en cuestión y acarreando el dígito más significativo de la suma (si es que hay uno) a la columna siguiente.



Capítulo 1

Resolución de problemas



Objetivos

La resolución de problemas, utilizando como herramienta una computadora, requiere contar con la capacidad de expresión suficiente como para indicar a la máquina lo que debe llevarse a cabo.

Se comenzará resolviendo situaciones del mundo real tratando de utilizar determinados elementos que caracterizan a una secuencia de órdenes que una computadora puede comprender.

El tema central de este capítulo es la definición del concepto de algoritmo y los elementos que lo componen.



Temas a tratar

- ✓ Introducción.
- ✓ Etapas en la resolución de problemas con computadora.
- ✓ Algoritmo.
- ✓ Pre y Postcondiciones de un algoritmo.
- ✓ Elementos que componen un algoritmo: Secuencia de Acciones, Selección, Repetición e Iteración.
- ✓ Importancia de la indentación en las estructuras de control.
- ✓ Conclusiones.
- ✓ Ejercitación.

1.1 Introducción

La Informática es la ciencia que estudia el análisis y resolución de problemas utilizando computadoras.

La palabra ciencia se relaciona con una metodología fundamentada y racional para el estudio y resolución de los problemas. En este sentido la Informática se vincula especialmente con la Matemática.

Si se busca en el diccionario una definición en la palabra *problema* podrá hallarse alguna de las siguientes:

- Cuestión o proposición dudosa, que se trata de aclarar o resolver.
- Enunciado encaminado a averiguar el modo de obtener un resultado cuando se conocen ciertos datos.

La resolución de problemas mediante una computadora consiste en dar una adecuada formulación de pasos precisos a seguir.

Si se piensa en la forma en que una persona indica a otra como resolver un problema, se verá que habitualmente se utiliza un lenguaje común y corriente para realizar la explicación, quizás entremezclado con algunas palabras técnicas. Esto es un riesgo muy grande. Los que tienen cierta experiencia al respecto saben que es difícil transmitir el mensaje y por desgracia, con mucha frecuencia se malinterpretan las instrucciones y por lo tanto se ejecuta incorrectamente la solución obteniéndose errores.

Cuando de una computadora se trata, no pueden utilizarse indicaciones ambiguas. Ante cada orden resulta fundamental tener una única interpretación de lo que hay que realizar. Una máquina no posee la capacidad de decisión del ser humano para resolver situaciones no previstas. Si al dar una orden a la computadora se produce una situación no contemplada, será necesario abortar esa tarea y recomenzar todo el procedimiento nuevamente.

Además, para poder indicar a la computadora las órdenes que debe realizar es necesario previamente entender exactamente lo que se quiere hacer. Es fundamental conocer con qué información se cuenta y qué tipo de transformación se quiere hacer sobre ella.

A continuación se analizarán en forma general las distintas etapas que deben seguirse para poder llegar a resolver un problema utilizando una computadora como herramienta.

1.2 Etapas en la resolución de problemas con computadora

La resolución de problemas utilizando como herramienta una computadora no se resume únicamente en la escritura de un programa, sino que se trata de una tarea más compleja. El proceso abarca todos los aspectos que van desde interpretar las necesidades del usuario hasta verificar que la respuesta brindada es correcta. Las etapas son las siguientes:

Análisis del problema

En esta primera etapa, se analiza el problema en su contexto del mundo real. Deben obtenerse los requerimientos del usuario. El resultado de este análisis es un modelo preciso del ambiente del problema y del objetivo a resolver. Dos componentes importantes de este modelo son los datos a utilizar y las transformaciones de los mismos que llevan al objetivo.

Diseño de una solución

La resolución de un problema suele ser una tarea muy compleja para ser analizada como un todo. Una técnica de diseño en la resolución de problemas consiste en la identificación de las partes (subproblemas) que componen el problema y la manera en que se relacionan. Cada uno de estos subproblemas debe tener un objetivo específico, es decir, debe resolver una parte del problema original. La integración de las soluciones de los subproblemas es lo que permitirá obtener la solución buscada.

Especificación de algoritmos

La solución de cada subproblema debe ser especificada a través de un algoritmo. Esta etapa busca obtener la secuencia de pasos a seguir para resolver el problema. La elección del algoritmo adecuado es fundamental para garantizar la eficiencia de la solución.

Escritura de programas

Un algoritmo es una especificación simbólica que debe convertirse en un programa real sobre un lenguaje de programación concreto. A su vez, un programa escrito en un lenguaje de programación determinado (ej: Pascal, Ada, etc) es traducido automáticamente al lenguaje de máquina de la computadora que lo va a ejecutar. Esta traducción, denominada compilación, permite detectar y corregir los errores sintácticos que se cometan en la escritura del programa.

Verificación

Una vez que se tiene un programa escrito en un lenguaje de programación se debe verificar que su ejecución produce el resultado deseado, utilizando datos representativos del problema real. Sería deseable poder afirmar que el programa cumple con los objetivos para los cuales fue creado, más allá de los datos particulares de una ejecución. Sin embargo, en los casos reales es muy difícil realizar una verificación exhaustiva de todas las posibles condiciones de ejecución de un sistema de software. La facilidad de verificación y la depuración de errores de funcionamiento del programa conducen a una mejor calidad del sistema y es un objetivo central de la Ingeniería de Software.

En cada una de las etapas vistas se pueden detectar errores lo cual lleva a revisar aspectos de la solución analizados previamente.

Dada la sencillez de los problemas a resolver en este curso, la primera etapa correspondiente al análisis del problema, sólo se verá reflejada en la interpretación del



enunciado a resolver. Sin embargo, a lo largo de la carrera se presentarán diferentes asignaturas que permitirán familiarizar al alumno con las técnicas necesarias para hacer frente a problemas de gran envergadura.

Con respecto a la segunda etapa, se pospondrá el análisis de este tema hasta el capítulo 5, ya que se comenzará a trabajar con problemas simples que no necesitan ser descompuestos en otros más elementales.

Por lo tanto, a continuación se trabajará sobre el concepto de algoritmo como forma de especificación de soluciones concretas para la resolución de problemas con computadora.

1.3 Algoritmo

La palabra algoritmo deriva del nombre de un matemático árabe del siglo IX, llamado Al-Khuwarizmi, quien estaba interesado en resolver ciertos problemas de aritmética y describió varios métodos para resolverlos. Estos métodos fueron presentados como una lista de instrucciones específicas (como una receta de cocina) y su nombre es utilizado para referirse a dichos métodos.

Un algoritmo es, en forma intuitiva, una receta, un conjunto de instrucciones o de especificaciones sobre un proceso para hacer algo. Ese algo generalmente es la solución de un problema de algún tipo. Se espera que un algoritmo tenga varias propiedades. La primera es que un algoritmo no debe ser ambiguo, o sea, que si se trabaja dentro de cierto marco o contexto, cada instrucción del algoritmo debe significar sólo una cosa.

Se presentan a continuación algunos ejemplos:

Ejemplo 1.1:

Problema : Indique la manera de salar una masa.

Algoritmo 1: Ponerle algo de sal a la masa

Algoritmo 2: Agregarle una cucharadita de sal a la masa.

El algoritmo 1 presenta una solución ambigua al problema planteado.

El algoritmo 2 presenta una solución adecuada al problema.

Ejemplo 1.2:

Problema: Determinar si el número 7317 es primo.

Algoritmo 1: Divida el 7317 entre sus anteriores buscando aquellos que lo dividan exactamente.

Algoritmo 2: Divida el número 7317 entre cada uno de los números 1, 2, 3, 4, ..., 7315, 7316. Si una de las divisiones es exacta, la respuesta es no. Si no es así, la respuesta es sí.

El algoritmo 1 no especifica claramente cuáles son los valores a lo que se refiere, por lo que resulta ambiguo.

El algoritmo 2 presenta una solución no ambigua para este problema. Existen otros algoritmos mucho más eficaces para dicho problema, pero esta es una de las soluciones correctas.



Ejemplo 1.3:

Problema: Determinar la suma de todos los números enteros.

En este caso no se puede determinar un algoritmo para resolver este problema. Un algoritmo debe alcanzar la solución en un tiempo finito, situación que no se cumplirá en el ejemplo ya que los números enteros son infinitos.

Además de no ser ambiguo, un algoritmo debe detenerse. Se supone también que cuando se detiene, debe informar de alguna manera, su resultado. Es bastante factible escribir un conjunto de instrucciones que no incluyan una terminación y por lo tanto dicho conjunto de instrucciones no conformarían un algoritmo.

Ejemplo 1.4:

Problema: Volcar un montículo de arena en una zanja.

Algoritmo: Tome una pala. Mientras haya arena en el montículo cargue la pala con arena y vuélquela en la zanja. Dejar la pala.

Este algoritmo es muy simple y no ambiguo. Se está seguro que en algún momento parará, aunque no se sabe cuántas paladas se requerirán.

Resumiendo, un algoritmo puede definirse como una secuencia ordenada de pasos elementales, exenta de ambigüedades, que lleva a la solución de un problema dado en un tiempo finito.

Para comprender totalmente la definición anterior falta clarificar que se entiende por “paso elemental”.

Ejemplo 1.5:

Escriba un algoritmo que permita preparar una tortilla de papas de tres huevos.

El enunciado anterior basta para que un cocinero experto lo resuelva sin mayor nivel de detalle, pero si este no es el caso, se deben describir los pasos necesarios para realizar la preparación. Esta descripción puede ser:

*Mezclar papas cocidas, huevos y una pizca de sal en un recipiente
Freír*

Esto podría resolver el problema, si el procesador o ejecutor del mismo no fuera una persona que da sus primeros pasos en tareas culinarias, ya que el nivel de detalle del algoritmo presupone muchas cosas.

Si este problema debe resolverlo una persona que no sabe cocinar, se debe detallar, cada uno de los pasos mencionados, pues estos no son lo bastante simples para un principiante.

De esta forma, el primer paso puede descomponerse en:

*Pelar las papas
Cortarlas en cuadraditos
Cocinar las papas
Batir los huevos en un recipiente
Aregar las papas al recipiente y echar una pizca de sal al mismo*

El segundo paso (freír) puede descomponerse en los siguientes tres:

Calentar el aceite en la sartén

Verter el contenido del recipiente en la sartén

Dorar la tortilla de ambos lados

Nótese además que si la tortilla va a ser realizada por un niño, algunas tareas (por ejemplo batir los huevos) pueden necesitar una mejor especificación.

El ejemplo anterior sólo pretende mostrar que la lista de pasos elementales que compongan nuestro algoritmo depende de quién sea el encargado de ejecutarlo.

Si en particular, el problema va a ser resuelto utilizando una computadora, el conjunto de pasos elementales conocidos es muy reducido, lo que implica un alto grado de detalle para los algoritmos.

Se considera entonces como un paso elemental aquel que no puede volver a ser dividido en otros más simples. De ahora en adelante se utilizará la palabra instrucción como sinónimo de paso elemental.

Un aspecto importante a discutir es el detalle que debe llevar el algoritmo. Esto no debe confundirse con el concepto anterior de paso elemental. En ocasiones, no se trata de descomponer una orden en acciones más simples sino que se busca analizar cuáles son las órdenes relevantes para el problema. Esto resulta difícil de cuantificar cuando las soluciones son expresadas en lenguaje natural. Analice el siguiente ejemplo:

Ejemplo 1.6:

Desarrolle un algoritmo que describa la manera en que Ud. se levanta todas las mañanas para ir al trabajo.

Salir de la cama

Quitarse el pijama

Ducharse

Vestirse

Desayunarse

Arrancar el auto para ir al trabajo

Nótese que se ha llegado a la solución del problema en seis pasos, y no se resaltan aspectos como: colocarse un calzado después de salir de la cama, o abrir la llave de la ducha antes de ducharse. Estos aspectos han sido descartados, pues no tienen mayor trascendencia. En otras palabras se sobreentienden o se suponen. A nadie se le ocurriría ir a trabajar descalzo.

En cambio existen aspectos que no pueden obviarse o suponerse porque el algoritmo perdería lógica. El tercer paso, “vestirse”, no puede ser omitido. Puede discutirse si requiere un mayor nivel de detalle o no, pero no puede ser eliminado del algoritmo.

Un buen desarrollador de algoritmos deberá reconocer esos aspectos importantes y tratar de simplificar su especificación de manera de seguir resolviendo el problema con la menor cantidad de órdenes posibles.

1.4 Pre y Postcondiciones de un algoritmo

Precondición es la información que se conoce como verdadera antes de comenzar el algoritmo.

En el ejemplo 1.1:

Problema: Indique la manera de salar una masa.

Algoritmo: Agregarle una cucharadita de sal a la masa.

Se supone que se dispone de todos los elementos para llevar a cabo esta tarea. Por lo tanto, como precondición puede afirmarse que se cuenta con la cucharita, la sal y la masa.

Postcondición es la información que se conoce como verdadera al concluir el algoritmo si se cumple adecuadamente el requerimiento pedido.

En el ejemplo 1.2:

Problema: Determinar si el número 7317 es primo.

Algoritmo: Divida el número 7317 entre cada uno de los números 1, 2, 3, 4, ..., 7315, 7316. Si una de las divisiones es exacta, la respuesta es no. Si no es así, la respuesta es sí.

La postcondición es que se ha podido determinar si el número 7317 es primo o no.

En el ejemplo 1.4:

Problema: Volcar un montículo de arena en una zanja.

Algoritmo: Tome una pala. Mientras haya arena en el montículo cargue la pala con arena y vuélquela en la zanja. Dejar la pala.



- ¿Cuáles serían las precondiciones y las postcondiciones del algoritmo?

La precondición es que se cuenta con la pala, la arena y está ubicado cerca de la zanja que debe llenar.

La postcondición es que el montículo quedó vacío al terminar el algoritmo.

1.5 Elementos que componen un algoritmo

1.5.1 Secuencia de Acciones

Una secuencia de acciones está formada por una serie de instrucciones que se ejecutan una a continuación de la otra.

Esto se muestra gráficamente en la figura 1.1

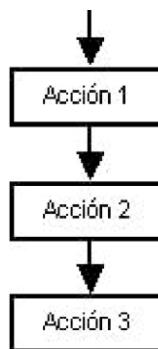


Figura 1.1: Secuencia

Ejemplo 1.7: Escriba un algoritmo que permita cambiar una lámpara quemada.

Colocar la escalera debajo de la lámpara quemada

Tomar una lámpara nueva de la misma potencia que la anterior

Subir por la escalera con la nueva lámpara hasta alcanzar la lámpara a sustituir

Desenroscar la lámpara quemada

Enroscar la nueva lámpara hasta que quede apretada la nueva lámpara

Bajar de la escalera con lámpara quemada

Tirar la lámpara a la basura

Ejemplo 1.8: Escriba un algoritmo que permita a un robot subir 8 escalones

Levantar Pie Izquierdo

Subir un escalón

Levantar Pie Derecho

Subir un escalón

Levantar Pie Izquierdo

Subir un escalón

Levantar Pie Derecho

Subir un escalón

Levantar Pie Izquierdo

Subir un escalón

Levantar Pie Derecho

Subir un escalón

Levantar Pie Izquierdo

Subir escalón

Levantar Pie Derecho

Subir un escalón

Se denomina flujo de control de un algoritmo al orden en el cual deben ejecutarse los pasos individuales.

Hasta ahora se ha trabajado con flujo de control secuencial, es decir, la ejecución uno a uno de los pasos, desde el primero hasta el último.

Las estructuras de control son construcciones algorítmicas que alteran directamente el flujo de control secuencial del algoritmo.

Con ellas es posible seleccionar un determinado sentido de acción entre un par de alternativas específicas o repetir automáticamente un grupo de instrucciones.

A continuación se presentan las estructuras de control necesarias para la resolución de problemas más complejos.

1.5.2 Selección

La escritura de soluciones a través de una secuencia de órdenes requiere conocer a priori las diferentes alternativas que se presentarán en la resolución del problema. Lamentablemente, es imposible contar con esta información antes de comenzar la ejecución de la secuencia de acciones.

Por ejemplo, que ocurriría si en el ejemplo 1.7 al querer sacar la lámpara quemada, el portalámparas se rompe. Esto implica que el resto de las acciones no podrán llevarse a cabo por lo que el algoritmo deberá ser interrumpido. Si se desea que esto no ocurra, el algoritmo deberá contemplar esta situación. Nótese que el estado del portalámparas es desconocido al iniciar el proceso y sólo es detectado al intentar sacar la lámpara quemada. Por lo que usar solamente la secuencia planteada es insuficiente para expresar esta solución.

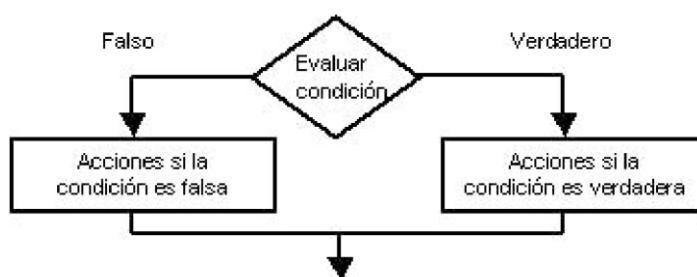


Figura 1.2: Estructura Si-Entonces-Sino

A través de la selección se incorpora, a la especificación del algoritmo, la capacidad de decisión. De esta forma será posible seleccionar una de dos alternativas de acción posibles durante la ejecución del algoritmo.

Por lo tanto, el algoritmo debe considerar las dos alternativas, es decir, qué hacer en cada uno de los casos. La selección se notará de la siguiente forma:

si (condición)
acción o acciones a realizar si la condición es verdadera (1)

sino
acción acciones a realizar si la condición es falsa (2)

donde “condición” es una expresión que al ser evaluada puede tomar solamente uno de dos valores posibles: verdadero o falso.

El esquema anterior representa que en caso de que la condición a evaluar resulte verdadera se ejecutarán las acciones de (1) y NO se ejecutarán las de (2). En caso

contrario, es decir, si la condición resulta ser falsa, solo se ejecutarán las acciones de (2).

En la figura 1.2 se grafica la selección utilizando un rombo para representar la decisión y un rectángulo para representar un bloque de acciones secuenciales.

Analice el siguiente ejemplo:

Ejemplo 1.9: Su amigo le ha pedido que le compre \$1 de caramelos en el kiosco. De ser posible, prefiere que sean de menta pero si no hay, le da igual que sean de cualquier otro tipo. Escriba un algoritmo que represente esta situación.

Ir al kiosco

si(hay caramelos de menta)

Llevar caramelos de menta

(1)

sino

(2)

Llevar de cualquier otro tipo

Pagar 1 peso

Los aspectos más importantes son:

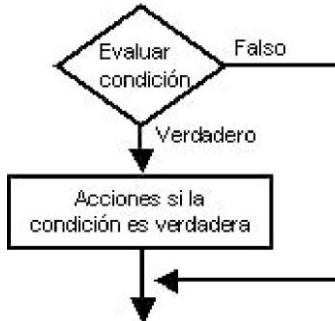


Figura 1.3: Estructura Si-Entonces

- No es posible saber si en el kiosco hay o no hay caramelos de menta ANTES de llegar al kiosco por lo que no puede utilizarse únicamente una secuencia de acciones para resolver este problema.
- La condición “hay caramelos de menta” sólo admite dos respuestas posibles: hay o no hay; es decir, verdadero o falso respectivamente.
- Si se ejecuta la instrucción marcada con (1), NO se ejecutará la acción (2) y viceversa.
- Independientemente del tipo de caramelos que haya comprado, siempre se pagará \$1. Esta acción es independiente del tipo de caramelos que haya llevado.

En algunos casos puede no haber una acción específica a realizar si la condición es falsa. En ese caso se utilizará la siguiente notación:

si (condición)

acción o acciones a realizar en caso de que la condición sea verdadera.

Esto se muestra gráficamente en la figura 1.3.

Ejemplos 1.10: Su amigo se ha puesto un poco más exigente y ahora le ha pedido que le compre \$1 de caramelos de menta en el kiosco. Si no consigue caramelos de menta, no debe comprar nada.

Escriba un algoritmo que represente esta situación.

*Ir al kiosco
si (hay caramelos de menta)
Pedir caramelos de menta por valor de \$1
Pagar \$1*

Con este último algoritmo, a diferencia del ejemplo 1.8, si la condición “hay caramelos de menta” resulta ser falsa, no se realizará ninguna acción.



Haciendo clic en el siguiente link podés acceder a una animación sobre la estructura Selección: [Animación Selección](#)

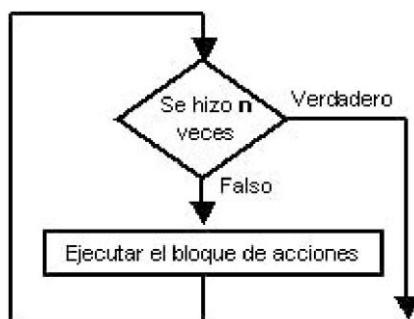


Figura 1.4: Estructura repetitiva

1.5.3 Repetición

Un componente esencial de los algoritmos es la repetición. La computadora, a diferencia de los humanos, posee una alta velocidad de procesamiento. A través de ella, es posible ejecutar, de manera repetitiva, algunos pasos elementales de un algoritmo. Esto puede considerarse una extensión natural de la secuencia.

La repetición es la estructura de control que permite al algoritmo ejecutar un conjunto de instrucciones un número de veces fijo y conocido de antemano.

La notación a utilizar es la siguiente y se muestra en la figura 1.4:

*repetir N
Acción o acciones a realizar N veces.*

Se analizan a continuación algunos algoritmos que presentan repeticiones:

Ejemplo 1.11: Escriba un algoritmo que permita poner 4 litros de agua en un balde utilizando un vaso de 50 cc.

Al plantear una solución posible, se observa que hay dos pasos básicos: llenar el vaso con agua y vaciarlo en el balde. Para completar los cuatro litros es necesario repetir estas dos operaciones ochenta veces. Suponga que se dispone de un vaso, un balde y una canilla para cargar el vaso con agua.

Tomar el vaso y el balde

repetir 80

Llenar el vaso de agua.

Vaciar el vaso en el balde.

Dejar el vaso y el balde.

Nótese que, la instrucción “Dejar el vaso y el balde” no pertenece a la repetición. Esto queda indicado por la sangría o indentación utilizada para cada instrucción. Por lo tanto, se repetirán 80 veces las instrucciones de “Llenar el vaso de agua” y “Vaciar el vaso en el balde”.



Haciendo clic en el siguiente link podés acceder a una animación sobre la estructura Repetición: [Animación Repetición](#)

El ejemplo 1.8, que inicialmente se presentó como un ejemplo de secuencia, puede escribirse utilizando una repetición de la siguiente forma:

Ejemplo 1.12: Escriba un algoritmo que permita a un robot subir 8 escalones.

repetir 4

LevantaPieIzquierdo

Subir un escalón.

LevantaPieDerecho

Subir un escalón

Este algoritmo realiza exactamente las mismas acciones que el algoritmo del ejemplo 1.8. Las ventajas de utilizar la repetición en lugar de la secuencia son: la reducción de la longitud del código y la facilidad de lectura.

Ejemplo 1.13: Juan y su amigo quieren correr una carrera dando la vuelta a la manzana. Considerando que Juan vive en una esquina, escriba el algoritmo correspondiente.

repetir 4

Correr una cuadra

Doblar a la derecha

1.5.4 Iteración

Existen situaciones en las que se desconoce el número de veces que debe repetirse un conjunto de acciones. Por ejemplo, si se quiere llenar una zanja con arena utilizando una pala, será difícil indicar exactamente cuántas paladas de arena serán necesarias para realizar esta tarea. Sin embargo, se trata claramente de un proceso iterativo que consiste en cargar la pala y vaciarla en la zanja.

Por lo tanto, dentro de una iteración, además de una serie de pasos elementales que se repiten; es necesario contar con un mecanismo que lo detenga.

La iteración es una estructura de control que permite al algoritmo ejecutar en forma repetitiva un conjunto de acciones utilizando una condición para indicar su finalización.

El esquema iterativo es de la forma:

mientras (condición)

Acción o acciones a realizar en caso de que la condición sea verdadera.

Las acciones contenidas en la iteración serán ejecutadas mientras la condición sea verdadera. Es importante notar que, la primera vez, antes de ejecutar alguna de las acciones de la iteración, lo primero que se realiza es la evaluación de la condición. Sólo luego de comprobar que es verdadera se procede a ejecutar el conjunto de acciones pertenecientes al mientras.

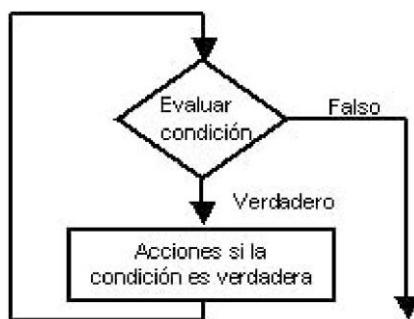


Figura 1.5: Estructura iterativa

Si inicialmente la condición resultara falsa, el contenido del mientras no se ejecutará ni siquiera una sola vez. Este funcionamiento se muestra gráficamente en la figura 1.5.

Es importante que las acciones realizadas en el interior de la iteración modifiquen el valor de verdad de la condición a fin de garantizar que la iteración terminará en algún momento.

Analicemos el siguiente ejemplo:

Ejemplo 1.14: Escriba un algoritmo que permita volcar un montículo de arena en una zanja utilizando una pala.

Tomar la pala.

Ubicarse frente a la zanja.

mientras (no esté vacío el montículo de arena)

cargar la pala con arena

volcar la arena en la zanja

Dejar la pala.



Haciendo clic en el siguiente link podés acceder a una animación sobre la estructura *Iteración*: [Animación Iteración](#)

La iteración indica que, mientras no se vacíe el montículo, se seguirá incorporando arena en la zanja. Cuando el montículo esté vacío, la condición será falsa y la iteración terminará. Es importante destacar, que si el montículo inicialmente estaba vacío, ninguna palada de arena será tomada del montículo ni incorporada a la zanja. Es decir, la condición se verifica ANTES de comenzar la iteración.

En este punto es apropiado hacerse la siguiente pregunta. ¿Qué sentido tiene introducir el concepto de iteración? Con toda seguridad, para los ejemplos antes mencionados no es necesario dicho concepto para establecer clara, simple o comprensiblemente las instrucciones del algoritmo.

Existe una razón bastante obvia para justificar esta estructura de control: es una realidad el hecho de que las computadoras requieren instrucciones detalladas y no ambiguas acerca de lo que deben hacer. Se debe, por lo tanto, dividir los algoritmos en pasos simples, de modo que las computadoras puedan efectuar sus cálculos. Si se quiere que algo sea realizado 80 veces, se le debe indicar que lo repita 80 veces. El empleo de las instrucciones de repetición, en este caso, permite hacer esto sin tener que escribir 80 líneas de instrucciones.

Por otro lado, el concepto de iteración es necesario para una mejor legibilidad o facilidad de lectura de los procesos algorítmicos. La iteración es un proceso fundamental en los algoritmos, y se debe ser capaz de pensar en términos de ciclos de iteración para poder construir los algoritmos.

1.6 Importancia de la indentación en las estructuras de control

Las instrucciones que pertenecen a una estructura de control deben tener una sangría mayor que la utilizada para escribir el comienzo de la estructura. De esta forma, podrá identificarse donde comienza y termina el conjunto de instrucciones involucradas en dicha estructura. A esta sangría se la denomina indentación.

Este concepto se aplica a las tres estructuras de control vistas previamente: selección, repetición e iteración.

El siguiente ejemplo muestra el uso de la indentación en la selección:

Ejemplo 1.15: Suponga que se planea una salida con amigos. La salida depende del clima: si llueve vos y tus amigos irán al cine a ver la película elegida, por el contrario si no llueve irán de pesca. Luego de realizar el paseo se juntarán a comentar la experiencia vivida. Escriba el algoritmo que resuelva esta situación.

Juntarse en una casa con el grupo de amigos

Mirar el estado del tiempo.

si (llueve) (1)

elegir película

ir al cine

sino

preparar el equipo de pesca

ir a la laguna a pescar

Volver a la casa a comentar sobre el paseo (2)

Como puede apreciarse, las acciones que deben ser realizadas cuando la condición es verdadera se encuentran desplazadas un poco más a la derecha que el resto de la estructura. Algo similar ocurre con las acciones a realizar cuando la condición es falsa. De esta forma puede diferenciarse lo que pertenece a la selección del resto de las instrucciones.

En el ejemplo anterior, la instrucción “Volver a la casa a comentar sobre el paseo” se realiza siempre sin importar si llovió o no. Esto se debe a que no pertenece a la selección. Esto queda de manifiesto al darle a las instrucciones (1) y (2) la misma indentación.

Ejemplo 1.16: Ud. desea ordenar una caja con 54 fotografías viejas de manera que todas queden al derecho; esto es, en la orientación correcta y la imagen boca arriba. Las fotografías ordenadas se irán guardando en el álbum familiar. Escriba el algoritmo que le permita resolver este problema.

Tomar la caja de fotos y un álbum vacío.

repetir 54

Tomar una fotografía.

si (la foto está boca abajo)

dar vuelta la foto

si (la foto no está en la orientación correcta)

girar la foto para que quede en la orientación correcta

guardar la fotografía en el álbum

guardar el álbum

Según la indentación utilizada, la repetición contiene a la acción de “Tomar una fotografía”, las dos selecciones y la instrucción “guardar la fotografía en el álbum”. Las instrucciones “Tomar la caja de fotos y el álbum” y “Guardar el álbum” no pertenecen a la repetición.

Ejemplo 1.17: Ud. se dispone a tomar una taza de café con leche pero previamente debe endulzarlo utilizando azúcar en sobrecitos. Escriba un algoritmo que resuelva este problema.



Tomar la taza de café con leche.

Probar el café con leche

mientras (no esté lo suficientemente dulce el café)

Tomar un sobre de azúcar.

Vaciar el contenido del sobre en la taza.

Mezclar para que el azúcar se disuelva.

Probar el café con leche

Tomar el café con leche.

Note que en este último ejemplo no se conoce de antemano la cantidad de sobrecitos de azúcar necesarios para endulzar el contenido de la taza. Además, la condición se evalúa antes de agregar el primer sobre. Según la indentación utilizada, la iteración incluye cuatro instrucciones. La acción “Tomar el café con leche” se ejecutará sólo cuando la iteración haya terminado, es decir, cuando la condición sea falsa.

1.7 Conclusiones

El uso de algoritmos permite expresar, de una forma clara, la manera en que un problema debe ser resuelto. Los elementos que lo componen son característicos de la resolución de problemas con computadora.

La ejercitación es la única herramienta para poder comprender y descubrir la verdadera potencialidad de las estructuras de control. Resulta fundamental alcanzar un total entendimiento del funcionamiento de estas estructuras para poder lograr expresar soluciones más complejas que los ejemplos aquí planteados.



Ejercitación

1. Defina qué es un algoritmo y cuáles son sus características principales.
2. ¿Cuáles son los elementos que componen un algoritmo?
3. Esta noche Juan se encuentra haciendo zapping sabiendo que hay un canal de televisión que está transmitiendo la película “30 años de felicidad”. Luego de terminar de ver la película debe apagar el televisor.
Analice las siguientes soluciones:

Solución 1:

Encender el televisor.
Cambiar de canal hasta encontrar la película.
Ver la película.
Apagar el televisor.

Solución 2:

Encender el televisor.
si (está transmitiendo “30 años de felicidad”)
ver la película.
Apagar el televisor.

Solución 3:

Encender el televisor.
repetir 20
 cambiar de canal.
Ver la película “30 años de felicidad”.
Apagar el televisor.

Solución 4:

Encender el televisor.
mientras (no se transmite en el canal actual “30 años de felicidad”)
 cambiar de Canal.
Ver la película.
Apagar el televisor.

- (a) Compare las soluciones 1 y 4.
(b) Explique por qué las soluciones 2 y 3 son incorrectas.
(c) ¿Qué ocurriría con la solución 4 si ningún canal estuviera transmitiendo la película?
4. Ud. desea comprar la revista “Crucigramas” que cada mes tiene reservada en el puesto de revistas que se encuentra en la esquina de su casa, al otro lado de la calle. Verifique que no pasen autos antes de cruzar. Indique, para cada uno de los siguientes algoritmos, si representa la solución a este problema. Justifique su respuesta.

**Algoritmo 1:**

Caminar hasta la esquina.
mientras (no pasen autos)
 Cruzar la calle
 Comprar la revista “Crucigramas”.

Algoritmo 2:

mientras (no llegue a la esquina)
 dar un paso
mientras (pasen autos)
 esperar 1 segundo
 Cruzar la calle.
Llegar al puesto de revistas.
Comprar la revista “Crucigramas”.

Algoritmo 3:

mientras (no llegue a la esquina)
 dar un paso.
mientras (pasen autos)
 esperar 1 segundo
mientras (no llegue a la otra vereda)
 dar un paso.
Llegar al puesto de revistas.
Comprar la revista “Crucigramas”.

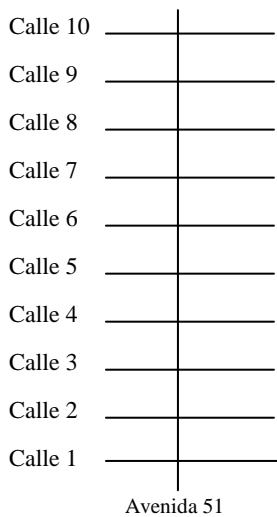
Algoritmo 4:

repetir 10
 dar un paso.
 Cruzar la calle.
 Llegar al puesto de revistas.
 Comprar la revista “Crucigramas”.

5. Utilizando las estructuras de control vistas resolver:

- a) Un algoritmo para que, en caso de ser necesario, permita cambiar el filtro de papel de una cafetera. Considere que está frente a la cafetera y que dispone de un filtro suplente.
 - b) Modifique la solución anterior para que cuando encuentre que el filtro de la cafetera esté limpio, guarde el filtro suplente en el lugar correspondiente.
6. Escriba un algoritmo que le permita preparar un té. Si no dispone de un saquito de té debe preparar un mate cocido. Considere que seguro existe el saquito de mate cocido. Tenga en cuenta que la preparación de las dos infusiones tienen muchos pasos en común.
 7. Escriba un algoritmo que le permita trasladar 70 cajas de 30 kilos cada una, desde la sala A a la Sala B. Considere que sólo llevará una caja a la vez porque el contenido es muy frágil. Para realizar el trabajo debe ponerse un traje especial y quitárselo luego de haber realizado el trabajo.

8. Modifique el algoritmo 7 suponiendo que puede trasladar 60 kilos a la vez.
9. Escriba un algoritmo que le permita guardar fotos en un álbum familiar. El álbum está compuesto por 150 páginas. En cada página entran 10 fotos. El álbum se completa por páginas. Una vez que el álbum está completo, debe guardarse en la biblioteca. Se supone que tiene fotos suficientes para completar el álbum.
10. Modifique el algoritmo anterior si ahora no se conoce la cantidad de fotos que entran en una página. Se cuentan con fotos suficientes para completar el álbum.
11. Modifique el algoritmo del ejer.9) pero suponiendo ahora que no se sabe la cantidad de páginas que tiene el álbum. Se sabe que en cada página entran 10 fotos. Se cuentan con fotos suficientes para completar el álbum.
12. Modifique el algoritmo del ejer.9) pero suponiendo ahora que no se sabe la cantidad de páginas que tiene el álbum ni la cantidad de fotos que entran en cada página. Se cuentan con fotos suficientes para completar el álbum.
13. Suponga que la avenida 51 tiene en sus esquinas faroles y papeleros distribuidos como muestra el dibujo y la tabla. ¿Si una persona ejecutaría el siguiente algoritmo en qué esquina quedaría posicionado?



Suponga:

Esquina (51,1) un papelero.
 Esquina (51,2) una farol y un papelero
 Esquina (51,3) un papelero
 Esquina (51,4) un papelero
 Esquina (51,5) una farol.
 Esquina (51,6) un papelero
 Esquina (51,7) un papelero
 Esquina (51,8) una farol.
 Esquina (51,9) un papelero y una farol
 Esquina (51,10) un papelero

Algoritmo

```
Posicionarse en 51 y 1 mirando hacia 2
caminar una cuadra (1)
mientras (hay un Farol En La Esquina) (2)
    si (Hay Papelero En La Esquina)
        mientras (Hay Papelero En La esquina)
            caminar una cuadra
            caminar una cuadra
        sino
            repetir 2
            caminar una cuadra
```

- ¿Dónde quedaría posicionado si la línea (1) se elimina del algoritmo?
- ¿Dónde quedaría posicionado si en el algoritmo original se cambia la línea (2) por Repetir 2?