CAPÍTULO I

Una breve introducción a algoritmos, estructuras de datos y Python

Para comenzar a abordar los diferentes temas referidos a algoritmos y estructuras de datos y aprovechar las potencialidades que el lenguaje de programación Python presenta para su implementación es imprescindible establecer las bases conceptuales mínimas, necesarias para comprender el resto de los capítulos de esta asignatura.

Algoritmo ¿Qué cosa es?

El nombre algoritmo (del latín *algorithmus*) proviene del matemático persa del siglo IX Abu Abdallah Muḥammad ibn Mūsā al-Khowârizmî, conocido como al-Juarismi. Podemos encontrar muchas definiciones en distintas fuentes de bibliografías dependiendo del enfoque de cada autor:

> Informalmente, un algoritmo es cualquier procedimiento computacional bien definido que toma algún valor, o conjunto de valores, como entrada y produce algún valor, o conjunto de valores, como salida. (Cormen at. al., 2009: 5)

> Un algoritmo es una secuencia de pasos computacionales que transforma la entrada en la salida. (Cormen at. al., 2009: 5)

> Podemos ver un algoritmo como una herramienta para resolver un problema computacional bien especificado. La declaración del problema especifica en términos generales la relación de entrada/salida deseada. El algoritmo describe un procedimiento computacional específico para lograr esa relación de entrada/salida. (Cormen at. al., 2009: 5)

> Un conjunto de reglas para efectuar algún cálculo bien sea a mano o en una máquina. (Brassard y Bratley, 1997: 2)

> Un conjunto de instrucciones sencillas, claramente especificado, que se debe seguir para resolver un problema. (Weiss, 1995: 17)

> Según el Diccionario de la lengua española (DLE) es un “Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema”; o un “Método y notación en las distintas formas del cálculo”.

> Una secuencia de instrucciones que representan un modelo de solución para determinado tipo de problemas. (Joyanes, 2003: 4)

> Un conjunto o grupo de instrucciones que realizadas en orden conducen a obtener la solución de un problema. (Joyanes, 1990: 7)

De forma general, se puede definir un algoritmo como una secuencia de instrucciones o pasos que al ejecutarlos recibe datos de entrada, de manera directa o indirecta y los transforma en una salida. También se lo puede definir con un conjunto de instrucciones ordenadas de manera lógica tal que al ser ejecutadas por una computadora o persona dan solución a un determinado problema para el cual fue diseñado.

Los algoritmos son independientes de los distintos lenguajes de programación en los que se desee implementarlos. Para cada problema se puede escribir un algoritmo, que luego puede codificarse y ejecutarse en diferentes lenguajes de programación. El algoritmo es la infraestructura de la solución a un problema, que luego puede ser implementada en cualquier lenguaje de programación.

Es importante destacar que los algoritmos deben cumplir determinadas características para ser considerados como tales. En primer lugar, un algoritmo debe ser preciso, es decir debe definirse de manera rigurosa, sin dar lugar a ambigüedades en las instrucciones que lo forman. Este también debe ser definido, esto implica que si se ejecuta dos veces el mismo algoritmo con la misma entrada
–salvo sean algoritmos aleatorios– se debe obtener el mismo resultado de salida. Además, tiene queser finito. Esto quiere decir que debe terminar de ejecutarse en algún momento. Pueden tener cero o más elementos de entrada y a su vez debe producir un resultado, los datos de salida serán los resultados de efectuar las instrucciones que lo forman sobre la entrada. Finalmente se concluye queun algoritmo debe ser suficiente para resolver el problema para el cual fue diseñado.

Frente a dos algoritmos que lleven cabo un mismo objetivo, es decir que resuelvan el mismo problema –ocurre bastante a menudo–, siempre será preferible el más corto. Esto no es una tarea trivial, dado que se debe tener en cuenta el análisis del algoritmo para la optimización de los tiempos de ejecución y recursos que consume, como se verá más adelante en profundidad y detalle en el capítulo III.

Además, los algoritmos pueden clasificarse en algoritmos cualitativos y cuantitativos. En el caso de los primeros, refieren a los que en sus pasos o instrucciones que lo forman no están involucradas o intervienen cálculos numéricos. Por ejemplo, las instrucciones para desarrollar una actividad física o encontrar un tesoro, tomar mate, buscar una palabra en el diccionario, etc. Mientras que los segundos son aquellos algoritmos en los que dentro de los pasos o instrucciones involucran cálculos numéricos. Por ejemplo, solución de una ecuación de segundo grado, conteo de elementos que cumplen una determinada condición, cálculo de un promedio, etc.

¿Programa y algoritmo es lo mismo? Es importante destacar que no es lo mismo un programa que un algoritmo. Un programa es una serie de instrucciones ordenadas, codificadas en un lenguaje de programación que expresa uno o varios algoritmos y que puede ser ejecutado en una computadora. Por lo general un programa contempla un conjunto de instrucciones encargadas de controlar el flujo de ejecución del programa –a través de una interfaz que puede ser de consola o gráfica– y hacer las llamadas a los distintos algoritmos que lo forman.

Algoritmia, una manera de estudiar los algoritmos

A su vez, el término *algoritmia* está relacionado de manera directa con los algoritmos, o podríamos decir que deriva del mismo. Existen varias definiciones para este término:

> Ciencia que nos permite evaluar el efecto de diferentes factores externos (como los números implicados, la forma en que se presenta el problema, o la velocidad y capacidad de almacenamiento de nuestro equipo) sobre los algoritmos disponibles, de tal modo que sea posible seleccionar el que mejor se ajuste a nuestras circunstancias particulares. (Brassard y Bratley, 1997: 3)

> El estudio de los algoritmos. (Brassard y Bratley, 1997: 3)

> Según DEL es "Ciencia del cálculo aritmético y algebraico, teoría de los números".

Se puede definir entonces la algoritmia como la ciencia que estudia a través de determinadas técnicas y herramientas cuál es el mejor algoritmo dentro de un conjunto de algoritmos diseñados para resolver un mismo problema, de acuerdo a un determinado criterio, necesidades y características del problema.

¿Para qué necesitamos las estructuras de datos?

Las estructuras de datos son las principales herramientas o recursos que tienen los programadores para el desarrollo de sus algoritmos. Es difícil pensar en un algoritmo o conjunto de estos que no tenga como macroestructura de soporte una estructura de datos. Entre las definiciones más conocidas de estructuras de datos podemos mencionar:

> Una estructura de datos es una forma de almacenar y organizar datos para facilitar el acceso y las modificaciones. Ninguna estructura de datos funciona bien para todos los propósitos, por lo que es importante conocer las fortalezas y limitaciones de cada una de ellas. (Cormen at. al., 2009: 9)

> Una estructura de datos es una colección de datos (normalmente de tipo simple) que se caracterizan por su organización y las operaciones que se definen en ellos. (Quetglás at. al., 1995: 171)

En otras palabras, una estructura de datos es una colección de datos simples o compuestos –como los registros– con un sentido u orden concreto para quien lo está utilizando, sobre la cual se definen un conjunto de operaciones que por lo general describen el comportamiento de la estructura.

La estructura de datos más simple que disponemos es una *variable*. Con esta podemos realizar las siguientes operaciones: *obtener* y *modificar* su valor de manera directa utilizando su identificador. Por su parte, un *registro* define un conjunto de atributos o campos a los que podemos acceder mediante su identificador y luego utilizar el operador punto seguido del nombre del campo para acceder a su valor: "registro.altura", "registro.peso".

Existen numerosos tipos de estructuras de datos, generalmente construidas sobre otras más simples. Por ejemplo, vector, matriz, pila, cola, lista, árboles, montículo, grafos, archivo, etc. Estas son lineales o no lineales –también llamadas ramificadas– las mismas pueden ser estáticas o dinámicas. Por lo general, al trabajar sobre estas estructuras de datos se suelen contemplar algunas actividades mínimas como *insertar*, *eliminar* y *buscar* un dato en la misma. A lo largo de los capítulos de esta asignatura se describen en profundidad varias de las estructuras de datos clásicas.

Pensamiento algorítmico ¿De qué se trata?

Quizás, uno de los aspectos más importantes de esta obra es su intención de crear y fortalecer el pensamiento algorítmico del alumno. Por esta razón, en cada guía de ejercicios se encontrarán algunos problemas sencillos para aplicar los conceptos abordados en el capítulo y luego otros másavanzados e integradores, donde se deberá atacar los problemas con un enfoque algorítmico parapoder obtener una solución. Para comenzar a introducir al alumno en esta mecánica práctica se puede plantear un primer ejercicio: "resuelva el problema de mover el caballo de ajedrez sobre el tecla-do de un teléfono".

Suponga que tiene a su disposición dicha ficha del ajedrez, y que se puede mover en ciertas formas particulares como se observa en la figura 1. Ahora, desea saber cuántos movimientos válidos pueden realizarse partiendo con el caballo desde todos los números del teclado realizando un movimiento desde cada número.

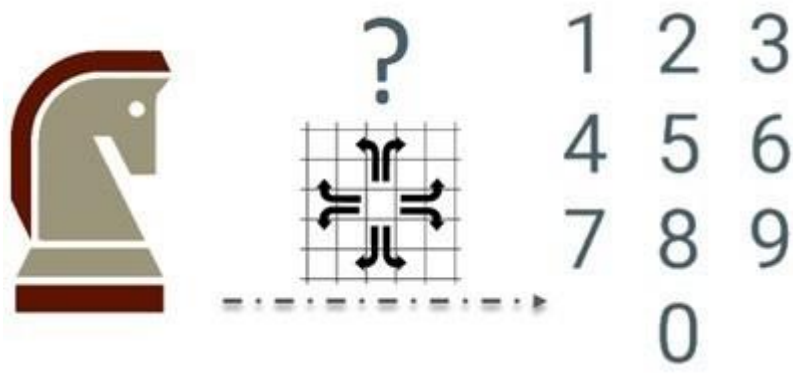

Figura 1. Problema de los movimientos de caballo de ajedrez

Si se parte del 1 se puede ir al 6 y al 8 (dos movimientos); si se sale del 2 se puede llegar al 7 y al 9 (dos movimientos más); iniciando desde el 3 se puede arribar al 4 y al 8 (se suman nuevamente dos); si se arranca desde el 4 las posibilidades son 3, 9 y 0 (ahora se acumulan tres movimientos); pero si la posición inicial es el 5 no se puede mover la ficha a ningún lugar dado que no hay movimientos válidos –sin embargo aún restan varias posibilidades más para seguir probando–; desde el 6 se pueden alcanzar el 1, 7 y 0 (nuevamente se agregan tres más); por su parte desde el 7 se puede mover la ficha hasta el 2 y el 6 (la cantidad se incrementa en dos); si se toma el 8 como inicio se pueden alcanzar el 1 y el 3 (se adicionan dos movimientos); si se posiciona la ficha en el 9 las opciones para moverse son 2 y 4 (nuevamente se tienen dos movimientos); y por último si se sale desde el 0 los movimientos válidos son 4 y 6 (se suman los últimos dos). En total se pueden realizar veinte movimientos válidos con esta ficha.

Ahora, diseñe un algoritmo que permita calcular cuántos posibles movimientos válidos puede realizar la ficha del caballo, recibiendo como entrada la cantidad de movimientos a realizar desde el inicio, partiendo de todos los números. Por ejemplo, como mostramos anteriormente si la cantidad de movimientos es uno, la cantidad de movimientos válidos son veinte. Pero si la cantidad de movimientos son dos y se sale desde el 1 se puede ir hasta el 6 y el 8 (un movimiento), a continuación, a partir del 6 hasta el 1, 7 y 0 (dos movimientos de la ficha), luego se sigue desde el 8 hasta el 1 y 3 (para alcanzar los dos movimientos de la ficha). En resumen, se tienen cinco posibles movimientos válidos partiendo desde el 1 (1-6-1, 1-6-7, 1-6-0, 1-8-1 y 1-8-3) a estos se deben sumar todos los movimientos que resulten de partir de los demás número. En total la cantidad de posibles movimientos válidos para dos movimientos son 46. Una vez desarrollado el algoritmo complete la siguiente tabla.

Cantidad de movimientos	**Posibilidades válidas**
1	20
2	46
3	104
5	
8	
10	
15	
18	
21	
23	
32	

Pasemos a trabajar en otro ejemplo para fortalecer aún más nuestro pensamiento algorítmico, en este caso el problema de las n-reinas, el mismo consiste en ubicar *n* reinas en un tablero de ajedrez de tamaño n x n, sin que las mismas se amenacen. Recuerde que la reina desplaza de manera horizontal, vertical y diagonal como se puede observar en la figura 2, además podemos ver una solución al problema de las 4 reinas. Nótese que una parte importante para resolver un problema es de que manera representar la solución, para este caso particular usamos un vector de n posiciones (columnas) y el valor almacenado representa la fila donde se ubica dicha reina.

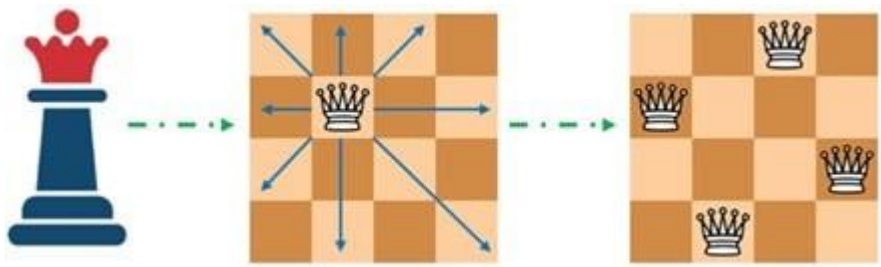

Figura 2. Problema de las n-reinas

Cuando haya entendido el problema y tenga una solución en mente, desarrolle un algoritmo que permita hallar al menos una solución para distintas cantidades de reinas, y luego complete la siguiente tabla.

n-reinas	**Todas las soluciones**
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
15	

Del algoritmo a las estructuras de datos: ¡A implementar!

Para implementar los algoritmos y definir las estructuras de datos se puede utilizar cualquier lenguaje de programación como C, Java, Delphi, C++, Javascript. En esta asignatura se opta por trabajar con el lenguaje Python por su versatilidad, creciente popularidad y demanda en estos últimos años. A su vez, se buscó darle un enfoque ágil sin dejar de lado las metodologías estructuradas de estos temas. Los ejemplos están codificados de la manera clásica utilizando unos pocos comandos particulares del lenguaje, para que puedan ser codificados en cualquier otro len guaje que se prefiera, sin realizar cambios significativos. Los ejemplos pueden ser ejecutados utilizando cualquier versión "3.x" de Python dado que la versión "2.x" esta descontinuada a partir de enero del 2020 (si se trabaja con alguna versión anterior se deberá hacer algunas modificacionespara adaptar el código y que funcione en dicha versión).

¿Qué es Python?

Python es un lenguaje de programación creado por Guido Van Rossum a principios de los años noventa, es un lenguaje que posee una sintaxis clara y estructurada que favorece a generar un código legible. Es administrado por la Python Software Foundation, posee una licencia de código abierto, denominada Python Software Foundation License, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1.

Se trata de un lenguaje interpretado o de script, con tipado dinámico, multiplataforma y orientado a objetos, además cuenta con una gran cantidad de librerías y *frameworks* disponibles. La sintaxis de Python es sencilla y cercana al lenguaje natural. Por estas razones se trata de uno de los mejores

lenguajes para comenzar a programar, –ya que es como programar en pseudocódigo–, y la curva de aprendizaje es bastante alta y rápida.

Es una tecnología muy utilizada en la actualidad, por su simplicidad y potencia permite realizar aplicaciones de una manera rápida, sencilla y óptima para el despliegue en distintas plataformas, además permite utilizar algoritmos de *machine learning* e inteligencia artificial en producción. Existen muchos casos de grandes empresas que utilizan Python en sus aplicaciones con gran éxito, tal es el caso de Google, la NASA, Netflix, Mercado libre, Spotify, Dropbox, Instagram, Pinterest, Globant, Sattelogic y un largo etcétera.

Un lenguaje interpretado o de script es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente una computadora. La ventaja de los lenguajes compilados es que su ejecución es más rápida. Sin embargo, los lenguajes interpretados son más flexibles y más portables.

Python tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es seminterpretado. En Python, como en Java, Delphi y muchos otros lenguajes, el código fuente se traduce la primera vez que se ejecuta a un pseudocódigo máquina intermedio llamado *bytecode*, generando archivos ".pyc" o ".pyo", que son los que se ejecutarán en sucesivas ocasiones.

La característica de tipado dinámico se refiere a que no es necesario declarar el tipo de dato que va a contener una variable, sino que su tipo se determinará en tiempo de ejecución según el tipo del valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo, lo cual le otorga una gran flexibilidad a la hora de programar.

En los lenguajes con esta característica no se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente. Por ejemplo, si tenemos una variable que contiene un texto (variable de tipo cadena o *string*) no podremos tratarla como una variable numérica, sin previamente realizar una conversión.

La programación orientada a objetos es un paradigma en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones e intercambio de mensajes entre los objetos. No obstante, el lenguaje soporta también los paradigmas de programación imperativa y funcional (esteúltimo le permite al lenguaje añadir características avanzadas muy interesantes).

El intérprete de Python está disponible en muchas plataformas de las más comunes (Linux, UNIX, Mac OS, Solaris, DOS, Windows, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin tener que realizar ningún tipo de cambio. Esto nos permite desarrollar programas que sean portables de un sistema operativo a otro.

La filosofía de los creadores de Python entiende que la gran sencillez, flexibilidad y potencia del lenguaje debe ir acompañada de una correcta formación del programador que lo utiliza para poder realizar las actividades de desarrollo con criterio, dado que la inexperiencia de los nuevos programadores puede conducirlos a programar de manera incorrecta (aunque lleguen a un resultado).