

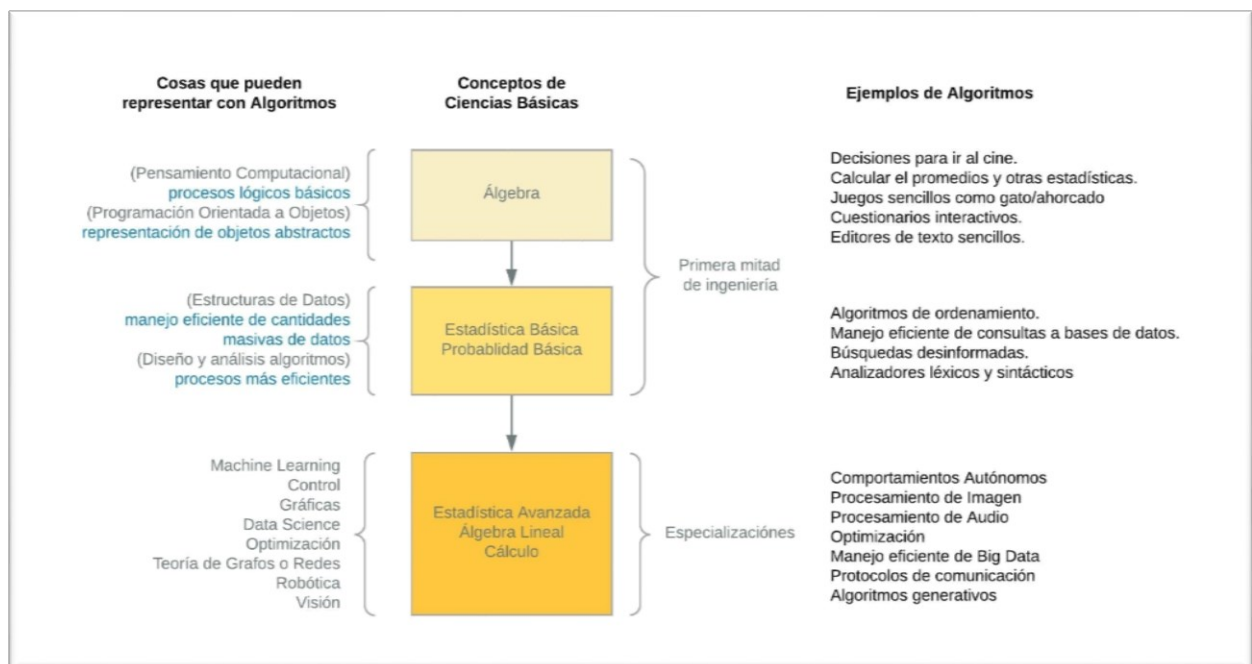
SOBRE LA REPRESENTACIÓN Y LA CREACIÓN DE SOLUCIONES

Un poco de contexto:

Los algoritmos son el repositorio de estrategias que hemos generado en los últimos siglos, para fines recreativos se puede ver https://en.wikipedia.org/wiki/Timeline_of_algorithms.

Los algoritmos representan nuestro conocimiento acumulado sobre cómo resolver problemas de forma automatizada. Saber leer y escribir algoritmos nos abre las puertas a todas las soluciones que hemos creado y documentado hasta la fecha.

Para poder entender algoritmos más complejos necesitaras cada vez bases más amplias. En el siguiente diagrama se relacionan algunas de las áreas de conocimiento que cursarán en su formación como ingenieros con el alcance que tendrán los algoritmos en estas etapas.



Representación ¿Por o cómo empezar?

Para poder generar soluciones descritas mediante algoritmos es necesario poder representar la realidad de una forma más simple, pero sin perder de vista los puntos que son relevantes para resolver el problema. La diferencia más clara entre los programadores más avanzados no es tanto el lenguaje en que programan, sino lo que pueden representar con él. Por eso entrevistas laborales se les pide resolver problemas y después codificarlos.

La representación comienza con el tipo de información que obtenemos del mundo real y como la planteamos en nuestro problema. En la siguiente figura mostramos ejemplos:

Representación de elementos reales en un algoritmo			
Concepto	Ejemplo	Representación	En Código Python
peso	77.5 kg	variable flotante peso	peso = 77.5
cantidad de personas	223	variable entera personas	personas = 223
tonos de rojo desde 0 a 255	91 rojo oscuro	variable entera color_r	color_r = 91
combinación de todos los colores mezclando el rojo verde y azul	135 blanco 135 verdecillo 135 azul	lista/arreglo de 3 elementos rgb	rgb = [135,135,135]
Elementos de ADN citocina, guanini, timina, kitina,	c citocina	variable caracter proteina	proteina = 'c'
secuencia genética de un codón	g guanina c citocina t timina c citocina	lista/arreglo de 4 elementos codon	codon = ['g','c','t','c'] o codon = 'gctc'
nombre de una persona	Benjamín Valdés	cadena string (que es una lista o arreglo)	nombre = 'Benjamín Valdés'

Representación de relaciones

Ya que tenemos nuestros componentes básicos, tenemos que representar también como se relacionan.

En la siguiente figura presentamos ejemplos de varias estrategias donde usamos conceptos básicos de álgebra y aritmética para relacionar los elementos básicos (variables o constantes). Estas estrategias tenlas a la mano porque te servirán en el futuro para integrarlas en tus propios algoritmos:

Representación de relaciones entre elementos		
Tipo de Relación	Ejemplo de Relación	Forma de Calcularla
Diferencia o distancia entre dos elementos numéricos.	diferencia entre perros y gatos	valor absoluto $ perros - gatos $ o $((perros - gatos) * (perros - gatos))$
Relación entre dos variables. Porcentuales	Perros respecto a mascotas	regla de 3 $perros/mascotas*100$
Una variable directamente proporcional a otra	La ganancias obtenidas en pesos, son directamente proporcionales al número de dólares y tipo de cambio.	$Ganacias = Dolares * TipoDeCambio$
Una variable inversamente proporcional a otra	presupuesto asignado a diversión vs alimentación	$diversión/alimentación$ o $alimentación/diversión$
Decremento constante en el tiempo	tiempo restante para que se acabe el examen	$minutos = minutos - 1$
Incremento constante en el tiempo	Contador de goles para un equipo de futbol soccer	$marcador = marcador + 1$
Incremento exponenciales	crecimiento de la población de un lugar donde cada pareja tiene 4 hijos en promedio	$hijos = papas ^ generaciones$
Incremento que va decreciendo con el tiempo	Curva de aprendizaje para una nueva tecnología	$log(incremento)$ o $sqrt(incremento) > 0$

Solucionar Problemas (algoritmos)

Para solucionar cualquier problema, primero debemos:

- 1) Entender que es lo que nos están pidiendo.
- 2) Identificar qué información o recursos nos están brindando.

Esto nos llevará a poder ver la situación de manera más clara. Usualmente a este nivel básico, se nos plantean 2 tipos de situaciones:

- 1) Situación 1: Automatizar un proceso que ya se conoce o que está explícito en los requerimientos. Esto es común cuando ya se cuenta con un algoritmo específico o una fórmula o un proceso clara y no ambiguo sobre cómo solucionar el problema. En esta situación, solo es necesario entonces implementar la solución.

E.g. Crear un algoritmo que calcule el área de un círculo a partir de su radio con la fórmula: $\text{area} = \text{PI} * r^2$

- 2) Situación 2: Automatizar un proceso que no sabemos cómo ocurre, pero del cual tenemos ejemplos de entrada y de salida que esperamos. (De este tipo son los problemas de la programación competitiva)

E.g. Crear un algoritmo que dado 3 números encuentre el más pequeño.

En ambos casos una vez que entendemos el problema generamos ejemplos o casos de prueba. Esto es importante porque nos permite entender dada qué información esperamos qué resultado. Con estos casos probaremos nuestros algoritmos para saber si son correctos o están mal hechos. Siguiendo los ejemplos anteriores tenemos que:

E.g.

	entrada		salida
Circulo_Area(3)	- >	28.27	
Circulo_Area(5)	- >	78.54	
Circulo_Area(-5)	- >	78.54	

E.g.

	entradas		salida
núm más pequeño(10, 23,1)	- >	1	
núm más pequeño(200, -23,9)	- >	-23	
núm más pequeño(0, 0, 0)	- >	0	

Una forma útil de comenzar con el proceso es: **primero resolver el problema a mano** con ejemplos concretos, y escribir cada paso de forma detallada.

Si un paso no se puede pasar a una instrucción de programación, entonces el paso ambiguo y debe descomponerse en pasos más específicos. Los algoritmos para los ejemplos anteriores podrían ser los siguientes:

<pre> circulo_area(PI = 3.1415, r) <- Estado Inicial area = PI*r*r regresa(area) <- Estado Final </pre>	<pre> circulo_area(PI = 3.1415, r) <- Estado Inicial area = r*r area = PI*area regresa(area) <- Estado Final </pre>
<pre> numero_menor(a, b, c) <- Estado Inicial if(a > b > c) res = c esle if (a < b < c) res = a else res = b regresa(res) <- Estado Final </pre>	<pre> numero_menor(a, b, c) <- Estado Inicial if(a > b > c) regresa (c) y termina <- Estado Final esle if (a < b < c) regresa (a) y termina <- Estado Final else regresa (c) y termina <- Estado Final </pre>

Como se puede ver hay varios algoritmos que pueden solucionar el mismo problema. Es importante contemplar que aunque pueden haber varias soluciones, no todas son igual de buenas y es común sacrificar algunos aspectos (velocidad, complejidad, memoria, transparencia, escalabilidad) para mejorar otros. Nosotros como ingenieros tenemos que desarrollar el criterio para saber que conviene más sacrificar en escenarios reales, pero eso ya lo verán cursos posteriores.