TEMA 1: Algoritmos y programas

1.1.-Introducción

La razón principal para utilizar un ordenador es para resolver problemas (en el sentido más general de la palabra), o en otras palabras, procesar información para obtener un resultado a partir de unos datos de entrada.

Los ordenadores resuelven los problemas mediante la utilización de programas escritos por los programadores. Los programas de ordenador no son entonces más que métodos para resolver problemas. Por ello, para escribir un programa, lo primero es que el programador sepa resolver el problema que estamos tratando.

Ejemplo: *Resolución de una ecuación de segundo grado.* $2 \cdot x^2 - 3 \cdot x + 1 = 0$

Datos: 2 / -3 / 1

Procesamiento: $x = \frac{3 \pm \sqrt{3^2 - 4 \cdot 2 \cdot 1}}{2 \cdot 2} = \frac{3 \pm \sqrt{9-8}}{4} = \frac{3 \pm 1}{4}$

Resultados: $\begin{cases} x_1 = 1 \\ x_2 = \frac{1}{2} \end{cases}$

El procesamiento de la información se realizará mediante la utilización de un método para resolver el problema que denominaremos *algoritmo*.

1.2.-Concepto de algoritmo

Definición de la Real Academia:

"*Conjunto ordenado y finito de operaciones que permiten resolver un problema*"

Consideramos, por tanto, un algoritmo tanto una receta de cocina, donde el problema a resolver es por ejemplo realizar un pastel, como un método matemático para multiplicar dos números.

Sin embargo esta definición no es completa. Hay ciertas condiciones que debe cumplir un algoritmo que no se han expresado. A continuación enumeraremos todas las características que ha de cumplir un algoritmo:

- Tiene un número finito de pasos.
- Acaba en un tiempo finito. Si nunca acaba no resolverá el problema.
- Las operaciones están definidas de forma precisa y sin ambigüedad.
- Interacciona con el entorno. Es decir, tiene como mínimo una salida y puede tener entradas.

Con estas características definiremos algoritmo como el conjunto **finito** de pasos y acciones que especifican de forma **clara y concisa (sin ambigüedades)** la **secuencia** de operaciones a realizar para procesar adecuadamente unos datos con un determinado objetivo.

Ejemplo: *Algoritmo de Euclides: MCD de dos números enteros.*

Datos de entrada: dos números enteros **A** y **B**

Datos de salida: el **MCD**

1. Si **B** es mayor que **A** intercambiar los valores.
2. Calcular el resto de dividir **A** por **B** y poner ese valor en **R**
3. Si **R** es igual a **0**, el **MCD** es **B**, y FIN
4. Ponemos en **A** el valor contenido en **B**, y
   ponemos en **B** el valor contenido en **R**
5. Volver a *2*.

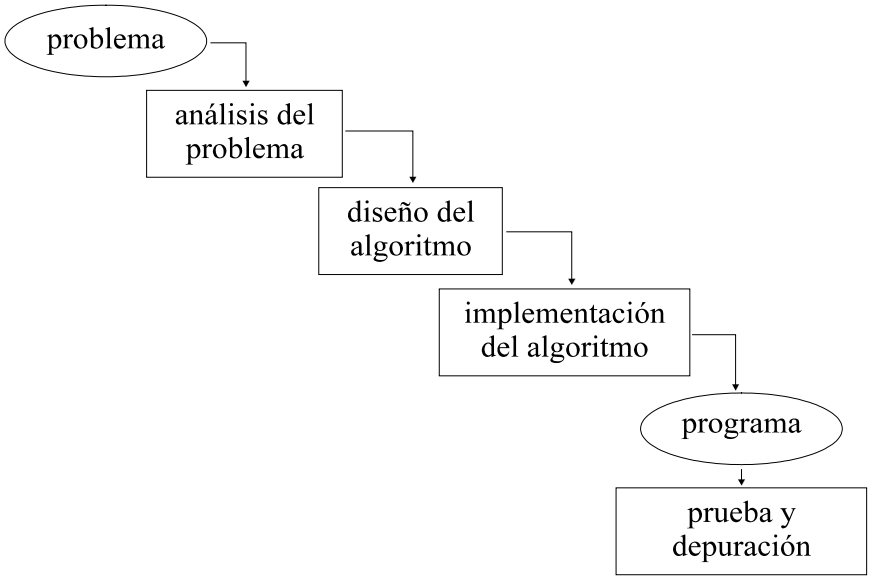
Es importante resaltar que no hace falta entender cómo funciona un algoritmo para utilizarlo. Se puede utilizar el algoritmo anterior sin saber por qué funciona.

La codificación de un algoritmo en un ordenador se denomina *programa*. El programa también se puede considerar un algoritmo, pero en este caso las operaciones son instrucciones del ordenador.

1.3.-Análisis, diseño y programación de algoritmos

¿Cómo se hace para, ante un problema determinado, obtener un programa que lo solucione?

Para realizar esto se sigue un proceso fijo que consiste en los siguientes pasos:



1-Análisis del problema

a.- Acotar y especificar el problema con total precisión (obtener el máximo de información acerca de lo que debemos resolver y las soluciones a determinar.) El problema ha de ser comprendido perfectamente antes de realizar el algoritmo.

b.- Definir los datos iniciales o de partida (que datos necesitamos proporcionar del problema para resolverlo.)

c.- Definir que datos o resultados debe proporcionar el algoritmo.

Ejemplo: *¿Cuantos metros cuadrados tiene mi habitación?*

Definición clara del problema: Calcular área de un rectángulo.

Datos de entrada: Dos lados contiguos de la habitación.

Datos de salida: Área de la habitación.

2-Diseño del algoritmo

No hay un método general para encontrar el algoritmo para un problema. Es una cuestión de experiencia e ingenio.

Sin embargo si que hay métodos sobre cómo proceder para realizar el algoritmo. Uno de los más importantes es el *diseño descendente (TOP-DOWN)*.

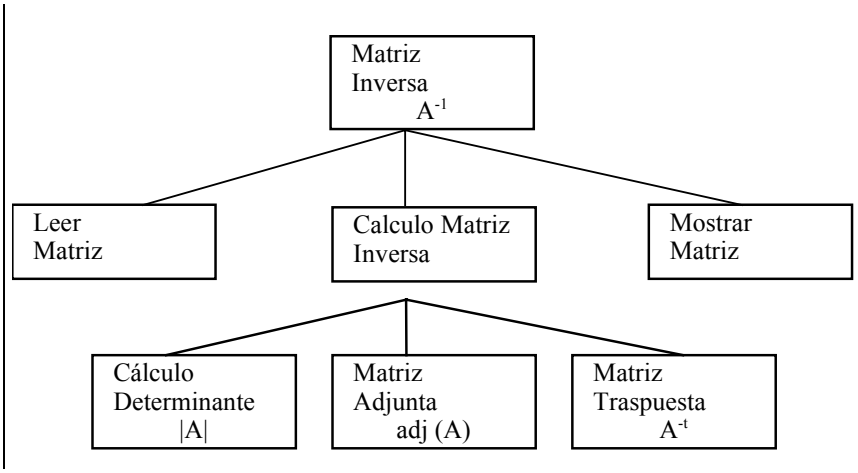
Consiste en descomponer un problema en subproblemas más sencillos. Este proceso se puede repetir sucesivamente sobre cada subproblema.

Es el más adecuado para la programación estructurada y modular (que ya veremos). El programa está dividido en módulos cada uno de los cuales resuelve una parte del problema. Además el programa tiene una estructura clara y es más fácil realizar modificaciones y localizar errores.

Para cada módulo debe de estar perfectamente claro que tarea ha de realizar, así como sus datos de entrada y sus datos de salida.

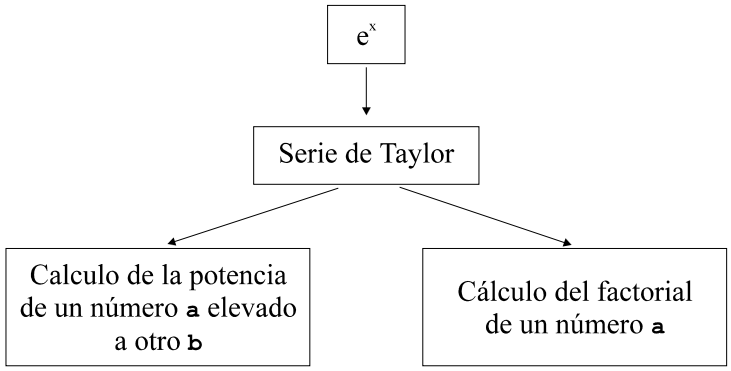
Ejemplo: *Cálculo de la Matriz Inversa*

$$A^{-1} = \frac{1}{|A|}[adj(A)]^t$$



Ejemplo: *Calcular una aproximación de e^x mediante la serie de Taylor.*

$$e^x = \sum_{i=0}^{n} \frac{x^i}{i!}$$



3-Implementación del algoritmo

Consiste en la codificación del algoritmo en un programa. Esta codificación se deberá realizar utilizando un determinado *lenguaje de programación*.

4- Prueba y depuración

Una vez realizado el programa se deberá ejecutar para comprobar que realmente hace lo que se pretendía, es decir, que no existan *errores de codificación* ni *errores en el algoritmo*. Cada módulo se probará por separado y, en caso de que funcionen bien, se probará todo el conjunto.

1.4.-Representación de algoritmos

Existen numerosas formas de representar algoritmos. Los más importantes son el pseudocódigo y los organigramas o diagramas de flujo, aunque actualmente estos últimos se utilizan menos.

El **pseudocódigo** es una manera de escribir algoritmos de forma poco estricta (con una sintaxis relajada) o estructuras de datos poco detalladas, pero intentando acercar las ideas del algoritmos a estructuras y sintaxis parecidas a las de los lenguajes de alto nivel en los que vamos a programar el algoritmo.

Es para ser leído por personas, por tanto no se preocupa en detalles sintácticos.

Ejemplo: *Dado un número, decir si es positivo o negativo.*

```
Algoritmo Positivo_Negativo
Variables
      x: Entero
Inicio
      Leer (x)
      Si (x<0) entonces
            Escribir ('Numero negativo')
      sino
            Escribir ('Numero positivo')
      Fin_si
Fin
```

Ejemplo: *Algoritmo para decir si un número es par (pseudocódigo)*

```
leer N
mientras N > 2 hacer
    N <- N – 2
fmientras
si N = 2 entonces
    escribir "Es par"
sino
    escribir "Es impar"
fsi
```

Ejemplo: *Búsqueda de números primos entre 2 y un cierto valor MAX*

Pseudocódigo: **Método 1**

.1. ***X*** = 2

.2. ***I*** = 2

.3. Hacer (***X***/***I***)

.4. Si ***I*** es menor que ***X*** y la división es entera entonces **X no es primo** y pasar a **.7.**

.5. Si ***I*** es igual que ***X*** entonces **X es primo** y pasar a 7

.6. Incrementar ***I*** y pasar a .3.

.7. Si ***X*** es más pequeño que **MAX** entonces incrementar ***X*** y pasar a **.2.** Sino hemos terminado.

Pseudocódigo: **Método 2:** ***Criba de Eratóstenes***

.1. Poner todos los números entre 2 y **MAX** uno detrás de otro.

.2. Si hay números sin tachar, el primero de ellos es primo

.3. Tachar de la lista todos los múltiplos del primer número

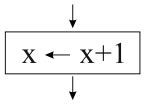
.4. Borrar el primer número

.5. Borrar los tachados y pasar a .2.

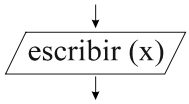
Los **organigramas** o **diagramas de flujo** son dibujos que representan de manera gráfica tanto las tareas como la sucesión de tareas del algoritmo. Las tareas se

representan mediante rectángulos, rombos y romboides y el flujo de tareas mediante flechas que enlazan las diferentes tareas.

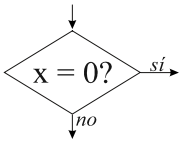
Las instrucciones se representan en rectángulos:



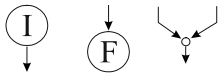
Las entradas y salidas en romboides:



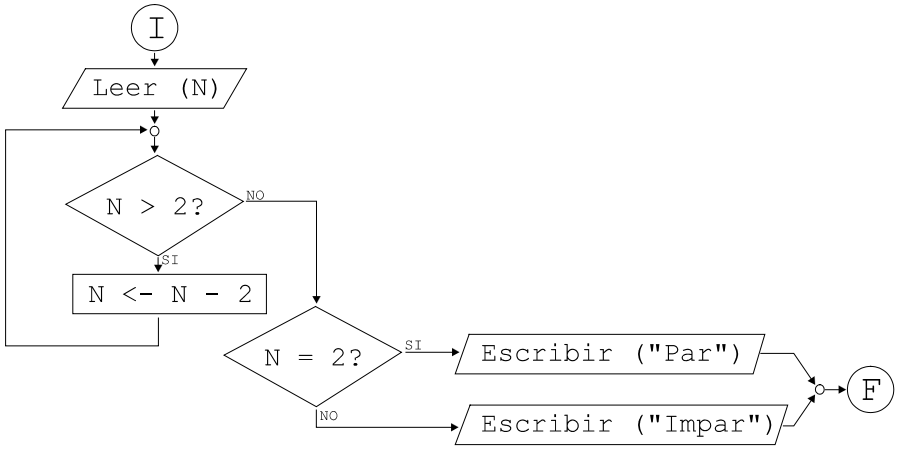
Las condiciones en rombos:



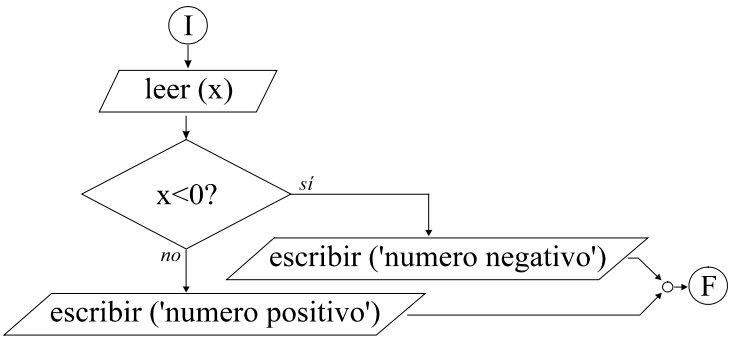
El inicio, el final y los puntos de reunión de flujo en círculos:



Ejemplo: *Algoritmo para decir si un número es par (organigrama)*



Ejemplo: *Dado un número, decir si es positivo o negativo.*



Ejemplo: *Obtener el mayor de dos números enteros introducidos por teclado*