Guía práctica de estudio 03: Algoritmos

Elaborado por:
M.C. Edgar E. García Cano
Ing. Jorge A. Solano Gálvez

Revisado por:
Ing. Laura Sandoval Montaño

Guía práctica de estudio 03: Algoritmos

Objetivo:

A partir del análisis del problema (conjuntos de entrada y salida), elaborar algoritmos que permitan resolver el problema planteado.

Actividades:

- Analizar problemas.
- Crear algoritmos para resolver problemas.

Introducción

Una vez realizado el análisis, es decir, ya que se entendió qué es lo que está solicitando el usuario y ya identificado el conjunto de entrada y el conjunto de salida, se puede proceder al diseño de la solución, esto es, a la generación del algoritmo.

Un problema matemático es computable si éste puede ser resuelto, en principio, por un dispositivo computacional.

La teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que pueden ser resueltos con un algoritmo.

Un algoritmo es un conjunto de reglas, expresadas en un lenguaje específico, para realizar alguna tarea en general, es decir, un conjunto de pasos, procedimientos o acciones que permiten alcanzar un resultado o resolver un problema. Estas reglas o pasos pueden ser aplicados un número ilimitado de veces sobre una situación particular.

Un algoritmo es la parte más importante y durable de las ciencias de la computación debido a que éste puede ser creado de manera independiente tanto del lenguaje como de las características físicas del equipo que lo va a ejecutar.

Las principales características con las que debe cumplir un algoritmo son:

- Un algoritmo debe ser **preciso**, es decir, llegar a la solución en el menor tiempo posible y sin ambigüedades.

- También debe ser **determinista**, es decir, a partir de un conjunto de datos idénticos de entrada, debe arrojar siempre los mismos resultados a la salida.

- El que un proceso sea computable implica que, en algún momento, el proceso va a llegar a su fin. Un algoritmo debe ser **finito**, por tanto, en algún momento debe terminar, lo que al mismo tiempo implica que el dominio del problema debe estar acotado.

Por tanto, un buen algoritmo debe ser correcto (cumplir con el objetivo) y eficiente (realizarlo en el menor tiempo posible), además de ser entendible para cualquier persona.

Dentro del ciclo de vida del software, la creación de un algoritmo se encuentra en la etapa de diseño de la solución del problema:

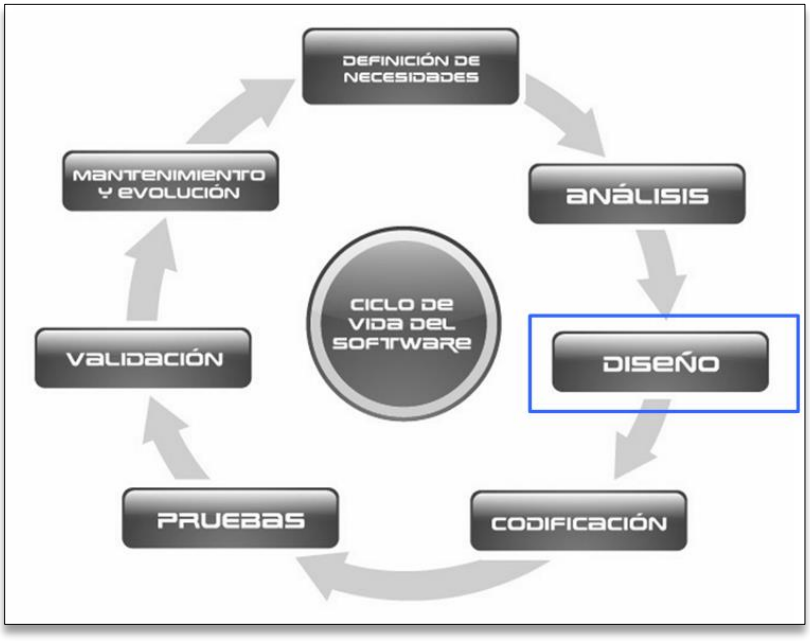
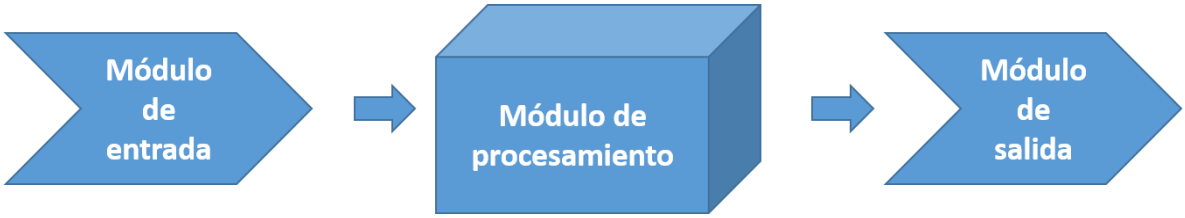


Figura 1: Ciclo de vida del software, resaltando la etapa de *diseño*, la cual corresponde al diseño de algoritmos.

Un algoritmo consta de 3 módulos básicos: módulo de entrada, módulo de procesamiento y módulo de salida.



El **módulo de entrada** representa los datos que requieren para resolver el problema. Estos datos se pueden solicitar al usuario, leer de un archivo, consultar de una base de datos, etc.

El **módulo de procesamiento** de datos representa las operaciones necesarias para obtener un resultado a partir de los datos de entrada.

El **módulo de salida** permite mostrar los resultados obtenidos a partir del módulo de procesamiento de datos. Los resultados pueden mostrarse en diversos sitios: en la pantalla, en un archivo, en una base de datos, etc.

Para saber si un algoritmo es adecuado dos preguntas se vuelven fundamentales:
1. ¿Es correcto?
2. ¿Es eficiente?

Para poder afirmar que un algoritmo es correcto (cumple con el objetivo del problema) es necesario aplicarlo con varios elementos del conjunto de entrada (datos de entrada), realizando una prueba de escritorio para verificar que, para todos los datos del conjunto de entrada, se obtiene la salida esperada (datos del conjunto de salida).

Se puede considerar que un algoritmo es eficiente cuando el tiempo de ejecución es el menor posible para cualquier dato del conjunto de entrada.

Por lo anterior, el siguiente se podría definir como un algoritmo para crear un algoritmo:

```
1. Analizar el problema a resolver (obtener los conjuntos de
   entrada y salida de datos).
2. Idear un algorimto que solvente el problema planteado.
3. Verificar que el algoritmo planteado sea correcto, es decir,
   que resuelva el problema.
4. Analizar la eficiencia del algoritmo.
5. Si el algoritmo planteado no es correcto o si el algoritmo
   planteado no es eficiente, se regresa al punto 2.
6. Si el  algoritmo planteado es correcto y es eficiente, se
   termina el proceso.
```

Fuente: udacity.com, Intro to Algorithms,
https://www.udacity.com/course/viewer#!/c-cs215/l-48747095/m-48691609

Ejemplo 1

PROBLEMA: Determinar si un número dado es positivo o negativo.
RESTRICCIONES: El número no puede ser cero.

DATOS DE ENTRADA: Número real.
DATOS DE SALIDA: La validación de si el número es positivo
DOMINIO: Todos los número reales.

SOLUCIÓN:

```
1. Solicitar un número real.
2. Si el número ingresado es cero, se regresa al punto 1.
3. Si el número ingresado es diferente de cero, se validan las siguientes
   condiciones:
    3.1 Si el número ingresado es mayor a 0 se puede afirmar que el número
        es positivo.
    3.2 Si el número ingresado es menor a 0 se puede afirmar que el número
        es negativo.
```

Prueba de escritorio

El diseño de la solución de un problema implica la creación del algoritmo y la validación del mismo. La validación se suele realizar mediante una *prueba de escritorio.*

Una prueba de escritorio es una matriz formada por los valores que van adquiriendo cada una de las variables del programa en cada iteración. Una iteración es el número de veces que se ejecuta un código y permite ver los valores que van adquiriendo las variables en cada repetición.

Para el ejemplo en cuestión la prueba de escritorio quedaría de la siguiente manera:

Iteración	X	Salida
1	5	El número 5 es positivo

Iteración	X	Salida
1	-29	El número 29 es negativo

Iteración	X	Salida
1	0	-
2	0	-
3	0	-
4	100	El número 100 es positivo

Ejemplo 2

PROBLEMA: Obtener el mayor de dos números dados.
RESTRICCIONES: Los números de entrada deben ser diferentes.

DATOS DE ENTRADA: Número real.
DATOS DE SALIDA: La impresión del número más grande.
DOMINIO: Todos los número reales.

SOLUCIÓN:

```
1. Solicitar un primer número real.
2. Solicitar un segundo número real.
3. Si el segundo número real es igual al primer número real, se regresa al
   punto 2.
4. Si el segundo número real es diferente al primer número real, se validan
   las siguientes condiciones:
    4.1 Si se cumple con la condición de que el primer número es mayor al
        segundo número, entonces se puede afirmar que el primer número es el
        mayor de los números.
    4.2 Si se cumple con la condición de que el segundo número es mayor al
        primer número, entonces se puede afirmar que el segundo número es el
        mayor de los números.
```

Prueba de escritorio:

Iteración	X	Y	Salida
1	5	6	El número 6 es el mayor

Iteración	X	Y	Salida
1	-99	-222.2	El número -99 es el mayor

Iteración	X	Y	Salida
1	15	15	-
2	15	15	-
3	15	15	-
4	15	10	El número 15 es el mayor

Ejemplo 3

PROBLEMA: Obtener el factorial de un número dado. El factorial de un número está dado por el producto de ese número por cada uno de los números anteriores hasta llegar a 1. El factorial de 0 (0!) es 1.
RESTRICCIONES: El número de entrada debe ser entero y no puede ser negativo.

DATOS DE ENTRADA: Número entero.
DATOS DE SALIDA: La impresión del factorial del número.
DOMINIO: Todos los números naturales positivos.

SOLUCIÓN:

1. Solicitar un número entero.
2. Si el número entero es menor a cero regresar al punto 1.
3. Si el número entero es mayor a cero se crea una variable entera *contador* que inicie en 2 y una variable entera *factorial* que inicie en uno.
4. Si la variable contador es menor o igual al número entero se realiza lo siguiente:
   4.1 Se multiplica el valor de la variable *contador* con el valor de la variable *factorial*. El resultado se almacena en la variable *factorial*.
   4.2 Se incrementa en uno el valor de la variable *contador*.
5. Si la variable contador no es menor o igual al número entero se muestra el resultado almacenado en la variable *factorial*.

Prueba de escritorio:

Iteración	X	fact	i	Salida
1	0	1	2	El factorial de 0 es: 1

Iteración	X	fact	i	Salida
1	-2	1	2	-
2	-67	1	2	-
3	5	1	2	-
4	5	2	3	-
5	5	6	4	-
6	5	24	5	-
7	5	120	6	El factorial de 5 es: 120

Iteración	X	fact	i	Salida
1	7	1	2	-
2	7	2	3	-
3	7	6	4	-
4	7	24	5	-
5	7	120	6	-
6	7	720	7	-
7	7	5040	8	El factorial de 7 es: 5040

Referencias

- Carlos Guadalupe (2013). Aseguramiento de la calidad del software (SQA). [Figura 1]. Consulta: Junio de 2015. Disponible en: https://www.mindmeister.com/es/273953719/aseguramiento-de-la-calidad-del-software-sqa

- Michael Littman. (2012). Intro to Algorithms: Social Network Analysis. Consulta: Junio de 2015, de Udacity. Disponible en: https://www.udacity.com/course/viewer#!/c-cs215/l-48747095/m-48691609