

Estructuras de programación

Teniendo claro los conceptos de algoritmo y programa resulta fundamental conocer las distintas estructuras de control que se pueden emplear en un programa. Veremos algunas de las técnicas empleadas para representar algoritmos además del uso de un lenguaje de programación estructurado.

Más allá de la estructura secuencial

La composición secuencial de instrucciones permite crear programas muy sencillos en los que se ejecuta un conjunto de instrucciones en un orden fijo. Esta estructura tiene una limitación importante: las instrucciones ejecutadas son siempre las mismas y siempre en el mismo orden, independientemente de los valores que puedan ir tomando las variables de nuestro algoritmo.

Esta limitación nos impide escribir programas que, por ejemplo, traten de resolver problemas que necesiten de la toma de decisiones para ser resueltos; piénsese en algo sencillo como es el caso de averiguar el máximo de dos números enteros, sólo con la composición secuencial de instrucciones que acabamos de ver es imposible resolverlo.

La estructura alternativa

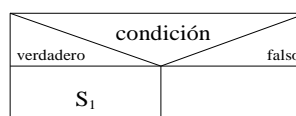
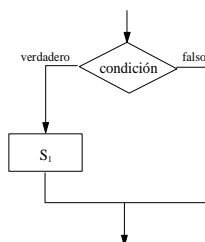
La estructura alternativa nos va a permitir solucionar esta pega dotando de más flexibilidad a nuestro programa ya que le va a permitir seleccionar qué instrucciones se han de ejecutar dependiendo del valor que tengan determinadas variables en un momento dado.

Tenemos *estructura alternativa simple*:

si (condición) entonces

S₁

Donde **condición** representa una expresión booleana y S₁ es un conjunto de instrucciones arbitrarias de nuestro programa que solo se ejecutarán si la condición booleana **condición** se evalúa a *cierto*. Podemos representarla gráficamente del siguiente modo:



La *estructura alternativa*:

si (*condición*) entonces

S₁

si_no

S₂

De manera similar a como hemos visto antes, **condición** representa una expresión booleana, y S₁ y S₂ representan instrucciones arbitrarias de nuestro lenguaje algorítmico. Las primeras solo se ejecutarán si la condición booleana **condición** se evalúa a *cierto*, y las segundas si se evalúa a *falso*.

La podemos representar:

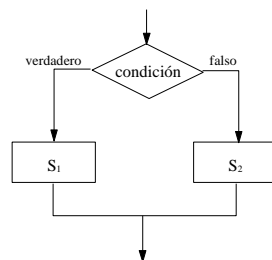


Diagrama de flujo de datos

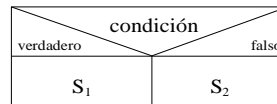


Diagrama de cajas

A medida que se va complicando el programa podemos ir anidando estructuras alternativas o de selección quedando del siguiente modo:

si (*condición*) entonces

S₁

si_no si (*condición*) entonces

S₂

si_no si (*condición*) entonces

S₃

si_no si (*condición*) entonces

S₄

si_no

S₅

en este caso tenemos hasta 5 conjuntos de instrucciones en función de que se cumplan o no determinada condición.

Es aconsejable indentar las estructuras y las instrucciones con el fin de que de un vistazo nos ayude a determinar la estructura de nuestro programa.

Podríamos representar la estructura alternativa múltiple así:

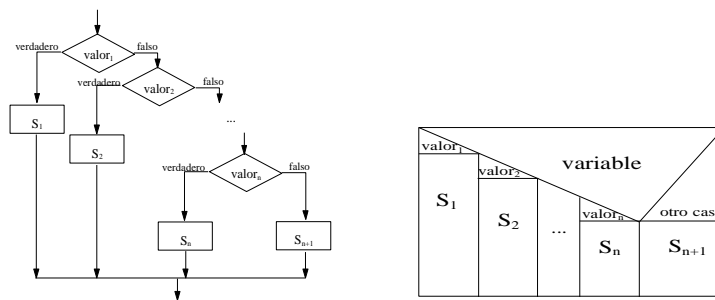


Diagrama de flujo de datos

Diagrama de cajas

Los operadores que podremos emplear en una expresión booleana con el fin de obtener los valores *cierto* y *falso* serán los siguientes:

Lógicos: && || !

Relacionales: < <= == != >= >

La estructura iterativa

Con las estructuras vistas hasta ahora somos incapaces de expresar todas aquellas acciones que requieran una repetición de las mismas durante una cantidad de tiempo determinada. Es, por tanto, necesario que dotemos a nuestro programa de otra estructura que le permita la repetición de las acciones que queramos.

Pensemos en lo que ocurre cuando, para comprar una entrada en el cine, tras llegar a la taquilla, lo lógico es que esperemos en la cola hasta que nos llegue el turno, más o menos algo así:

```

algoritmo Comprar una entrada
    llegar a la taquilla
    mientras haya gente delante
        esperar a que llegue nuestro turno
    comprar entrada
fin_algoritmo.
  
```

Si observamos el algoritmo anterior nos daremos cuenta que aparece la palabra **mientras**, la cual proporciona la idea de repetición *-si hay cola, debemos esperar-* que es lo que hacemos realmente cuando llegamos a la taquilla y tenemos gente delante.

A este tipo de construcciones algorítmicas se las denomina **bucles**, a la condición de este tipo de bucles se le llama **condición de continuación**, y a las instrucciones que se ejecutan de manera repetida **cuerpo** del bucle.

Los diagramas de flujo de datos y diagramas de cajas asociados a esta sentencia de repetición serían:

Repetición con condición inicial

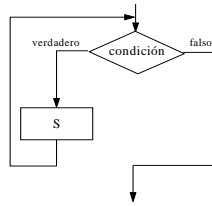


Diagrama de flujo de datos

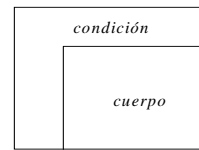


Diagrama de cajas

Repetición con condición final

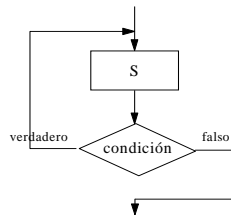


Diagrama de flujo de datos

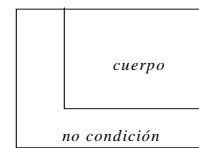


Diagrama de cajas

Repetición con contador

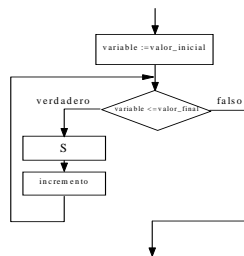


Diagrama de flujo de datos

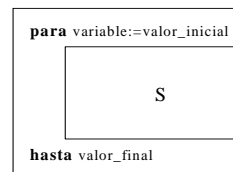


Diagrama de cajas

Técnicas de representación de algoritmos.

Hemos visto maneras de *plasmear* un algoritmo sobre el *papel*.

Por razones obvias, el lenguaje natural es el modo más sencillo y rápido de describir un algoritmo ya que todo el mundo lo conoce y lo entiende, ahora bien, está lleno de imprecisiones y ambigüedades, y un algoritmo se caracteriza porque debe proporcionar una descripción precisa de todos los pasos que lo componen. Básicamente por este motivo es por lo que se emplean otra serie de técnicas para representar un algoritmo.

Estas otras técnicas se caracterizan por que:

- Permiten representar un algoritmo de forma sencilla.
- La representación de un algoritmo está libre de ambigüedades.

Hecha esta introducción, pasamos a ver algunas de estas técnicas.

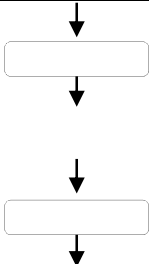
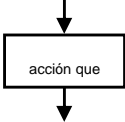
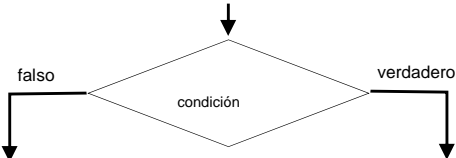
Lenguaje algorítmico

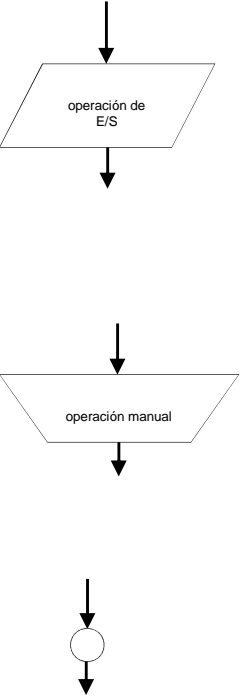
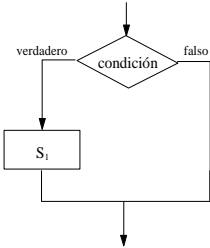
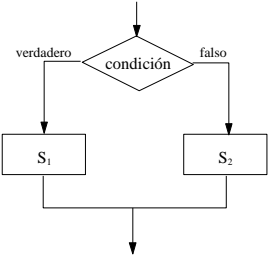
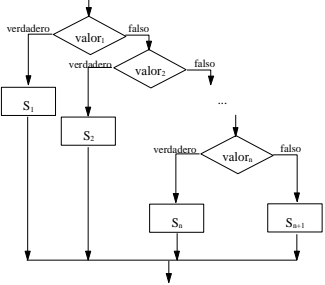
El lenguaje algorítmico (también conocido como pseudocódigo) consiste en una variante del lenguaje natural en la que se han eliminado las posibles ambigüedades mediante el uso de un determinado vocabulario -o conjunto de palabras reservadas- y unas reglas sintácticas de construcción de sentencias. Las *palabras reservadas* son una serie de palabras que el diseñador del lenguaje ha decidido que tienen un significado especial para el *interprete* o *compilador* del mismo, y que por tanto su uso se restringe al ámbito que nos delimite la sintaxis del lenguaje.

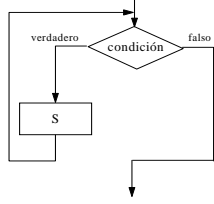
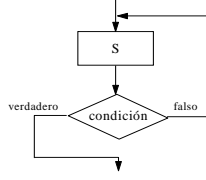
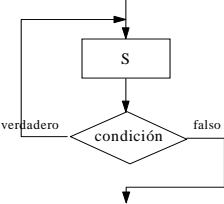
Diagramas de flujo

Los diagramas de flujo de datos -*dfd*- realizan una representación gráfica de los algoritmos. Como su nombre indica, en ella se resalta, principalmente, la lógica o el flujo de control del programa, es decir, lo que se hace en cada momento.

Para realizar esta representación gráfica emplea una serie de símbolos o iconos estándar (existen plantillas con estos iconos para facilitar su uso). estos iconos y su significado los podemos ver a continuación.

	Iconos de comienzo y fin del algoritmo
	Icono de proceso. Describe una acción que realiza
	Icono de decisión. dependiendo del valor de la condición, el flujo del programa sigue por un lado u otro
	Iconos de entrada/salida,

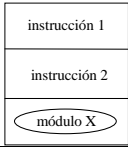
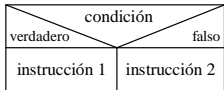
 <pre>graph TD; In1[] --> EIS[/operación de E/S/]; EIS --> Out1[]; In2[] --> OM[/operación manual/]; OM --> Out2[]; In3[] --> C(()); C --> Out3[];</pre>	<p>operación manual</p> <p>y de conexión con otros dfd's del algoritmo.</p>
 <pre>graph TD; In[] --> Cond{condición}; Cond -- verdadero --> S1[S1]; Cond -- falso --> Join(()); S1 --> Join; Join --> Out[];</pre>	<p>Sentencia si</p>
 <pre>graph TD; In[] --> Cond{condición}; Cond -- verdadero --> S1[S1]; Cond -- falso --> S2[S2]; S1 --> Join(()); S2 --> Join; Join --> Out[];</pre>	<p>Sentencia si...sino</p>
 <pre>graph TD; In[] --> V1{valor1}; V1 -- verdadero --> S1[S1]; V1 -- falso --> V2{valor2}; V2 -- verdadero --> S2[S2]; V2 -- falso --> Dots[...]; Dots --> Vn{valor_n}; Vn -- verdadero --> Sn[Sn]; Vn -- falso --> Sn1[S_{n+1}]; S1 --> Join(()); S2 --> Join; Sn --> Join; Sn1 --> Join; Join --> Out[];</pre>	<p>Sentencia caso de</p>

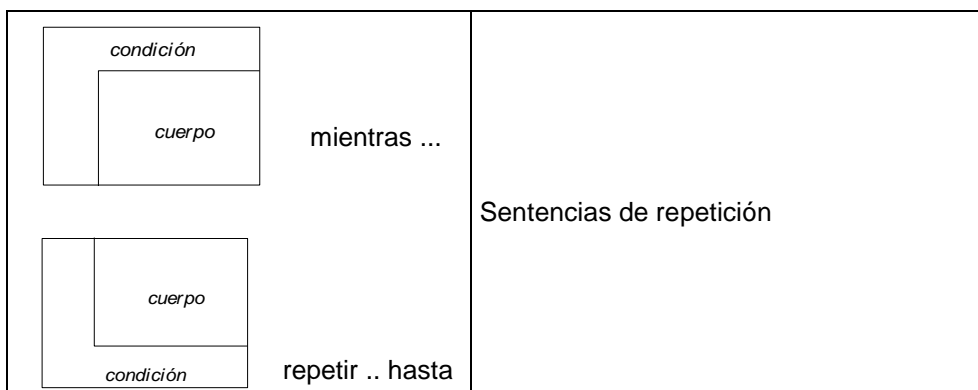
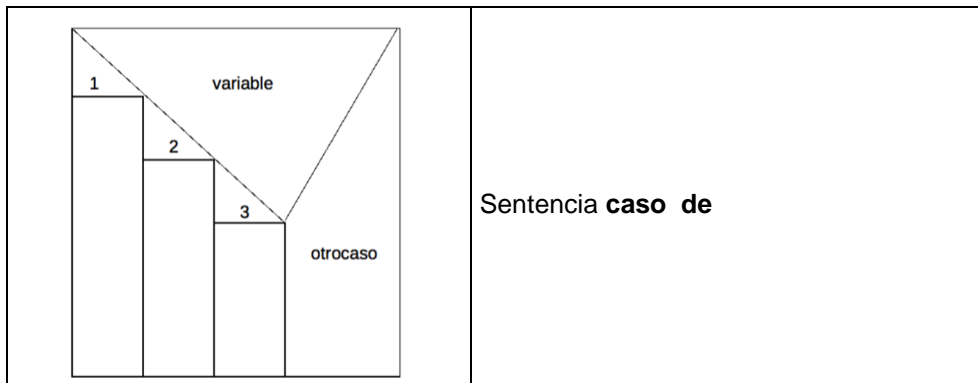
	Sentencia mientras
	Sentencia repetir .. hasta
	Sentencia repetir .. mientras

Diagramas de cajas

Se trata de otro método para representar algoritmos de manera gráfica, también conocido como *diagramas de Chapin*. Al principio es un poco más costoso de aplicar que el método que acabamos de ver, ya que requiere una mayor comprensión del programa por parte del programador, pero finalmente obtenemos una representación de los algoritmos mejor que con los diagramas de flujo. Por el tipo de diagramas que emplea favorece la escritura de algoritmos modulares.

El icono básico empleado por esta metodología es un rectángulo, dentro del cual se van colocando, según sean necesarios, los iconos que representan las sentencias de secuencia de acciones, selección y repetición. Los iconos empleados en esta metodología son los que podemos ver en la tabla siguiente.

	Secuencia de acciones
	Sentencia de selección



Los diagramas de cajas nos permiten comprender mejor la estructura de los programas, así como el enlace entre los distintos módulos de que conste nuestro programa.

Carece de símbolos conectores *-flechas-*, y como el número máximo de símbolos en una página está limitado alrededor de los 20, esto da lugar a algoritmos más compactos. Si fuese necesario representar un algoritmo que excediese una página podemos aislar una parte del mismo que por sí sola pueda constituir un **subalgoritmo**, y representarla aparte, dándole un nombre.