

Diseño de algoritmos.
Técnicas descriptivas.

TEMA 23 (25 SAI)

ABACUS NT

Oposiciones 2021

Índice

- 1. Introducción.**
- 2. Concepto de Algoritmo**
- 3. Diseño de Algoritmos**
 - 3.1. Ciclo de Vida del Software**
- 4. Estructura de un Algoritmo**
 - 4.1. Descripción de datos**
 - 4.1.1. Tipos de datos
 - 4.1.2. Ventajas y desventajas de las estructuras de datos estáticas/dinámicas
 - 4.2. Descripción de acciones**
 - 4.3. Expresiones y Operadores**
 - 4.4. Instrucciones**
- 5. Técnicas descriptivas**
 - 5.1. Pseudocódigo**
 - 5.2. Tablas de decisión**
 - 5.3. Diagrama de flujo**
 - 5.4. Organigrama**
 - 5.5. Ordinograma**
 - 5.6. Diagramas de Nassi-Scheiderman (Diagramas N-S)**
 - 5.7. Diagrama estructurado**
- 6. Ampliación: Eficiencia de los Algoritmos**
 - 6.1. Complejidad computacional. Notación O-Mayúscula.**
- 7. Opcional: Técnicas de Diseño de Algoritmos**
 - 7.1. Divide y Vencerás**
 - 7.2. Diseño Top-Down (Diseño Descendente)**
 - 7.3. Bottom-Up (Diseño Ascendente)**
 - 7.4. Recursividad**
 - 7.5. Backtraking**
 - 7.6. Ramifica y Acota**
 - 7.7. Algoritmos voraces**
 - 7.8. Algoritmos heurísticos**
 - 7.9. Otras técnicas:**
- 8. Conclusión**
 - 8.1. Relación del tema con el sistema educativo actual**
- 9. Bibliografía**

1. Introducción.

La principal razón para que las personas aprendan lenguajes y técnicas de programación es **utilizar la computadora como herramienta para resolver problemas**. La resolución de un problema exige al menos de los siguientes **pasos**:

- definición y análisis del problema.
- diseño del algoritmo.
- transformación del algoritmo en un programa
- ejecución y validación del programa.

El propósito del análisis de un problema es **ayudar al programador a llegar a una cierta comprensión de la naturaleza del mismo**. Una buena definición del problema, junto con una descripción detallada de las especificaciones de entrada/salida, son los requisitos más importantes para llegar a una solución eficaz.

Paralelamente y en todas las fases se irá realizando la documentación del programa.

Se distinguen **dos tipos de documentación**:

- Para **programadores**, incluye información técnica del programa, así como los algoritmos y variables y ficheros.
- Para **usuarios**, instrucciones que incluye la descripción de las tareas que realiza un programa y de las instrucciones a seguir para la instalación y arranque.

Como resultado de su trabajo, el programador proporciona un texto que describe por anticipado el conjunto de las operaciones que un ordenador debe ejecutar para resolver el problema del que tiene encomendado hallar una solución informática.

Buena parte de la dificultad del trabajo reside en que esta descripción debe tener en cuenta todas las posibilidades de que se puedan producir, con el fin de evitar que se produzcan fallos en el conjunto del tratamiento por no haber previsto algún caso particular.

2. Concepto de Algoritmo

La programación consiste en la creación de un conjunto concreto de instrucciones que un ordenador puede ejecutar, denominado programa de ordenador. El programa habitualmente se escribe en un lenguaje de programación, aunque también se puede escribir directamente en lenguaje de máquina o ensamblador lo que conlleva cierta dificultad.

Un procesador de un ordenador únicamente puede interpretar y ejecutar programas escritos en lenguaje o código máquina. Dada la complejidad para desarrollar programas, se han desarrollado lenguajes de alto nivel independientes del procesador que permitan por un lado ser fácilmente comprensibles para las personas que los desarrollan, y por otro lado traducibles a código máquina. Esta progresiva simplificación a la hora de codificar los programas, ha ido centrando el esfuerzo de programadores en la **fase de análisis del problema**.

Un algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema; aunque la popularización del término ha llegado con la era de la informática, la palabra algoritmo proviene de Abu Abdallah ibn Musa al-Warizmi, matemático, astrónomo y geógrafo persa del siglo XVIII-IX que es considerado el padre del álgebra y el inventor del algoritmo.

Sin la existencia de los algoritmos numerosos problemas serían irresolubles y los ordenadores serían sólo calculadoras complejas.

Un algoritmo es un conjunto de pasos, exento de ambigüedades, que lleva a la resolución de un problema dado.

Dado un problema pueden existir distintos algoritmos que lo resuelvan. Siempre buscaremos el algoritmo óptimo en cuanto a dos factores fundamentales:

1. Mínimo tiempo de resolución
2. Menor uso de recursos.

El diseño de algoritmos, por tanto, estudia la búsqueda del algoritmo más eficiente para resolver un problema.

Los algoritmos se pueden representar utilizando diferentes herramientas y se pueden utilizar diversas técnicas para su diseño.

Según la definición de algoritmo dada por Niklaus Wirth (1934) -desarrollador de Pascal y Modula 2, tituló uno de sus famosos libros, Algoritmos+Estructuras de datos=Programas, estableciendo así que un algoritmo se divide en dos partes fundamentales:

- Descripción de acciones a completar
- Declaración de datos con los que trabajarán dichas acciones.

La diferencia entre algoritmo y programa, es que este último es su concreción en un conjunto de instrucciones de un lenguaje determinado.

3. Diseño de Algoritmos

3.1. Ciclo de Vida del Software

La creación de software consta generalmente de una serie de pasos claramente diferenciados:

1. Análisis: se recopila documentación, se estudian los requisitos del usuario final (qué se va a implementar, cuáles son las necesidades del usuario). Se suele empezar con una descripción en lenguaje natural de lo que quiere el usuario. Al final de esta etapa tendremos una descripción clara y precisa de qué producto vamos a construir, qué funcionalidades aportará y qué comportamiento tendrá.

2. Diseño: una vez que sabemos qué debemos hacer, debemos determinar cómo lo haremos. Estudiaremos el problema y lo descompondremos en problemas más pequeños; definiremos la estructura de la solución, identificando los módulos que hay que implementar y sus relaciones; definiremos los algoritmos a emplear y el lenguaje de programación a utilizar, etc.

3. Implementación y codificación: se escribe el programa fuente en el lenguaje de programación establecido y se genera el código ejecutable.

4. Pruebas: se comprueba que el programa no tenga errores, y que se cumplen los criterios de calidad.

5. Mantenimiento: el programador se asegura de que el programa siga funcionando, y lo va adaptando también a nuevos requisitos que puedan ir surgiendo



Muchas veces tratamos de acortar el proceso de programación yendo directamente desde el análisis del problema a la codificación del mismo.

Este atajo puede resultar muy tentador e incluso parecer una buena forma de ahorrar tiempo. Sin embargo, este método necesita realmente más tiempo y esfuerzo.

El desarrollo de una solución general antes de escribir realmente el programa puede ayudar al programador a manejar el problema, tomar el camino correcto y evitar errores innecesarios.

Además, puesto que la mayoría de los programas se usarán una y otra vez, la documentación y mantenimiento de programas son partes importantes de la programación.

4. Estructura de un Algoritmo

Como ya hemos visto, podemos dividir un algoritmo en dos partes:

- **Descripción de datos:** consiste en definir los datos a utilizar, asignarle un nombre y el tipo de dato: carácter, numérico, Lógico, estructurado, ...
- **Descripción de acciones:** pueden ser simples: operaciones aritméticas y/o lógicas, representadas a través de expresiones formadas por operadores, operaciones de control o bien sentencias más complejas, como funciones y procedimientos

Vamos a ver los elementos que podemos utilizar en cada caso:

4.1. Descripción de datos

Un dato puede identificarse de forma directa dando su valor, o bien asignándole un nombre con el que vamos a reconocer ese dato.

Hay que ver también si ese valor va a mantener fijo su valor a lo largo del problema o bien se va a mantener constante durante todo el proceso, podemos por tanto encontrar datos literales y asociados a un identificador.

Literales

Un literal es la expresión de un valor de un tipo de dato. Por ejemplo, son literales 3, 3.14, 'a', 'hola'. En realidad, son valores constantes. Un literal puede ser por ejemplo un dato entero, un real, un carácter o una cadena de caracteres.

Identificadores

Los identificadores son nombres que se asocian a los datos para identificarlos. Pueden ser de dos tipos:

- **Variables:** Una variable representa un conjunto de **posiciones de memoria** que tienen asociado un nombre (**identificador**) y un **valor** (contenido) de un determinado tipo. Para utilizar una variable normalmente hay que declararla primero, asignándole un nombre y un tipo de dato.
- **Constantes:** Una constante es un dato cuyo valor no variará a lo largo del algoritmo/programa. Igual que las variables, hay que declararlas, asociando el valor al nombre de la constante.

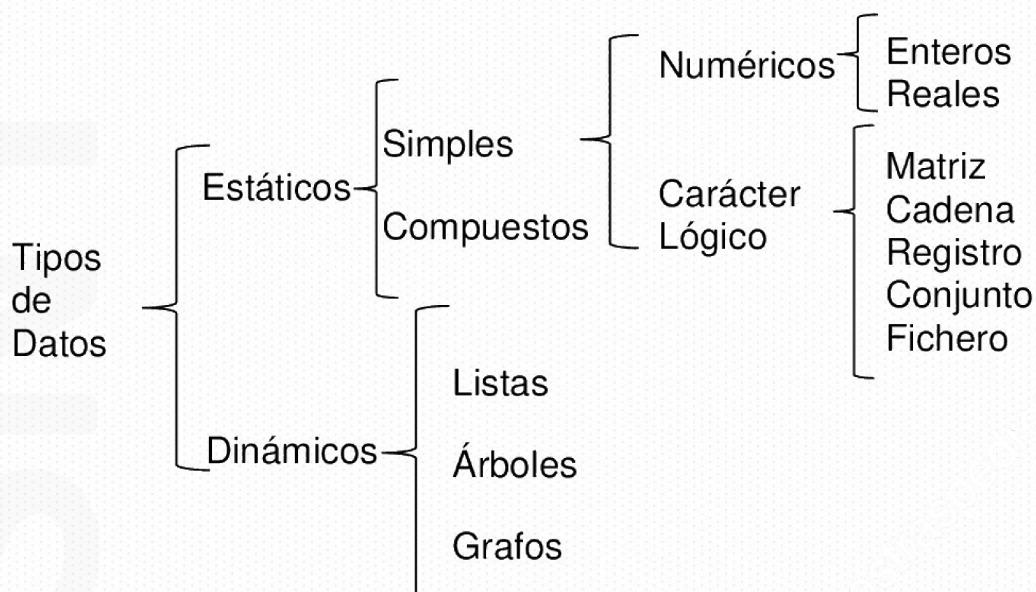
4.1.1. Tipos de datos

Los tipos de datos utilizados en los diferentes lenguajes de programación se pueden clasificar en **tipos simples** y en **tipos compuestos**.

- Los tipos de datos **simples** no están compuestos de **otras estructuras** de datos. Los usados más frecuentemente en casi todos los lenguajes son: **enteros, reales o carácter**. Los tipos lógico y subrango son propios de lenguajes como Pascal. **Los datos simples tienen como característica común que cada variable representa a un elemento.**
- Los tipos de datos compuestos o **estructuras de datos** están construidos **basándose en tipos de datos simples**. El ejemplo más representativo es la **cadena** de caracteres. Los datos estructurados tienen como característica común que **un identificador puede representar a múltiples datos individuales**, pudiendo estos ser referenciados individualmente. Las estructuras de datos se pueden dividir según su modo de almacenamiento en
 - Internas: se almacenan en memoria interna, estas a su vez pueden dividirse en.
 - Estáticas: El tamaño ocupado en la memoria se **define** antes de que el programa se ejecute y no puede modificarse durante la ejecución del programa. Están implementadas en casi todos los lenguajes y son los registros, ficheros y matrices. Dependiendo del tipo de dato pueden ser:
 - Homogéneas: del mismo tipo: arrays o matrices
 - Heterogéneas: registros
 - Dinámicas: No tienen las limitaciones en el tamaño de memoria ocupada propias de las estructuras estáticas. Su tamaño puede cambiar en tiempo de ejecución.
 - Lineales: listas, pilas, colas
 - No lineales: árboles y grafos
 - Externas: Se almacenan en la memoria externa del ordenador: ficheros y bases de datos.

Muy relacionado está también el concepto de **tipo de dato abstracto** que es un **modelo matemático** compuesto por una colección de **operaciones** definidas sobre un conjunto de datos para el modelo (Por ejemplo, el TDA “Cadena”).

El siguiente esquema muestra una clasificación de los tipos de datos más comunes:



4.1.2. Ventajas y desventajas de las estructuras de datos estáticas/dinámicas

Según el autor **J. Glenn Brookshear**, en su libro “**Introducción a la computación**” (2012), una ventaja de las estructuras estáticas es que son más **fáciles** de manejar, ya que las estructuras dinámicas tienen que llevar a cabo varias tareas adicionales, como, por ejemplo, buscar bloques de memoria libres, lo que afecta al rendimiento. Por el contrario, las estructuras dinámicas son más **versátiles**, ya que no siempre se conoce previamente el número de elementos que se van a procesar.

4.2. Descripción de acciones

4.3. Expresiones y Operadores

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Una expresión consta de "operando" y "operadores".

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

Dependiendo del tipo de objetos que manipulan, podremos clasificar las expresiones en: aritméticas, lógicas y de carácter:

Los **operadores** son símbolos que indican la operación a realizar. Podemos tener:

- **Operadores Aritméticos:** Son los que indican la ejecución de una operación aritmética como suma resta, multiplicación. División, etc.
- **Operadores Lógicos:** Son aquellos que indican operaciones relacionadas con el álgebra de Boole. Son operadores lógicos: AND, OR, NOT, NAND, NOR Y NOR.

- **Operadores Relacionales:** Son los que indican la diferencia que hay entre dos variables, constantes, expresiones.; Indicarán igualdad, inferioridad, superioridad, etc.

4.4. Instrucciones

Son expresiones que indican una operación a realizar por el ordenador. La instrucción estará compuesta de un código identificable, y de unos operandos que indican los datos implicados y el lugar donde deben depositarse los resultados. Según la función que desempeñan dentro de un programa, las instrucciones se pueden clasificar en:

- **Instrucciones de declaración:** Son instrucciones que se utilizan para declarar los objetos que se van a usar en el algoritmo.
- **Instrucciones primitivas:** Se denominan así a las instrucciones que se ejecutan de forma inmediata por el procesador. Nos encontramos con tres tipos de Instrucciones primitivas:
 - Entrada: Se utilizan para tomar datos del exterior y en ocasiones, guardarlos en variables.
 - Salida: Se utilizan para presentar; en pantalla o impresora, comentarios, mensajes al usuario, contenido de variables. resultados, etc.
 - Asignación: Este tipo de instrucción se utiliza para asignar valores a las variables.
- **Instrucciones de control:** Se denominan instrucciones de control a las instrucciones que permiten controlar la secuencia de ejecución de otras instrucciones bajo ciertas circunstancias. Distinguimos los siguientes tipos:
 - Selectivas o condicionales: Se ejecuta una acción u otra, según se cumpla o no una determinada conexión, pueden ser:
 - Simples: Este tipo de instrucción se utiliza. cuando en un programa se quiere ejecutar un bloque de instrucciones sólo bajo una determinada condición, en caso de que esta condición no se cumpliese, la ejecución del programa seguiría su curso normal (if .. then ..).
 - Dobles: Se utiliza cuando, se quiere ejecutar un bloque de instrucciones bajo una condición concreta, y otro bloque independiente, si ésta no se cumple. Se llama también **Instrucción alternativa doble** { if . .then .. else).
 - Múltiples: Se ejecutarán unas acciones u otras según el resultado que se obtenga al evaluar una expresión (CASE).
 - Repetitivas: Son estructuras que se repiten un número determinado de veces. Se denomina bucle o ciclo a un proceso que dentro de un algoritmo se repite

un cierto número de veces; normalmente está formado por una o varias acciones que se han de repetir y una condición que determina el número de repeticiones; esta condición necesariamente ha de verse afectada por las acciones del bucle, para no caer en un bucle sin fin. Hay varios tipos:

- Mientras: Este tipo de estructura se caracteriza porque las acciones del cuerpo del bucle se realizan cuando la condición es cierta. Además, se pregunta por la condición al principio del bucle, de donde se deduce que dichas acciones se podrán ejecutar de 0 a N veces.
- Repetir Hasta: las acciones del interior del bucle se ejecutan una vez y continúan repitiéndose hasta que la condición se cumpla. Se interroga por la condición al final del bucle. Se deduce que las acciones se podrán ejecutar de 1 a N veces.
- Para: Se utiliza cuando se conoce, con anterioridad a que empiece a ejecutarse el bucle, el número de veces que se va a iterar. Hace que la instrucción o bloque de instrucciones se ejecute una vez por cada valor dentro del rango Vi a Vf. La variable de control y los valores Vi y Vf, han de ser de tipo ordinal.
- Ruptura secuencial: Son instrucciones que indican que la siguiente instrucción a ejecutar no es la que corresponde según la secuencia física del algoritmo, sino que debe realizar un salto a otro punto del mismo programa. Su uso no es recomendado si se siguen las normas de la programación estructurada. Nos encontramos con los siguientes tipos de saltos:
 - Salto condicional: Revisan si ha ocurrido alguna situación para poder transferir el control del programa a otra sección
 - Salto Incondicional: Permiten cambiar el flujo del programa sin verificar si se cumplió alguna condición.
- Instrucciones compuestas: Representa a un conjunto de instrucciones agrupadas bajo un nombre y que están definidas en otra parte (funciones, procedimientos, etc.). Éstas pueden ser llamadas desde cualquier punto del algoritmo.
- Comentarios: Frases que sirven para aclarar algún aspecto del algoritmo. Se utilizan para facilitar la legibilidad del mismo. Solo son informativas y se pueden incluir en cualquier parte.

5. Técnicas descriptivas

Para representar algoritmos hay diferentes métodos y herramientas que permiten describirlos de forma independiente respecto al lenguaje de programación que finalmente se utilice en su implementación.

Destacamos las siguientes:

- Pseudocódigo
- Tablas de decisión
- Diagrama de flujo
- Diagramas de Nassi-Scheiderman
- Diagrama estructurado

5.1. Pseudocódigo

Un pseudocódigo es una notación mediante la cual podemos describir un algoritmo, utilizando palabras y frases del lenguaje natural, sujetos a unas determinadas reglas. Se puede considerar como un paso intermedio, entre la solución de un problema y su codificación en un lenguaje.

Todo pseudocódigo debe posibilitar la descripción de los siguientes elementos:

- Instrucciones de entrada/salida
- Instrucciones de proceso
- Sentencias de control de flujo
- Acciones compuestas (subprogramas)
- Comentarios

La estructura general de un pseudocódigo es la siguiente:

- Programa: Nombre
- Precondiciones: Tipos y rangos de las variables y estados en los que funcionará.
- Postcondiciones: Resultado de las acciones si se cumplen las precondiciones.
- Entorno: Declaración de estructuras de datos.
- Algoritmo: Secuencia de instrucciones que forman el programa.
- Fin de programa.

La primera parte del pseudocódigo contiene el nombre que el programador asigna al programa. Este nombre debe mantener algún tipo de relación directa o similitud con el funcionamiento del programa.

Así mismo es conveniente que contenga las precondiciones y postcondiciones de funcionamiento del algoritmo, aunque con frecuencia esta parte se omite, es de gran ayuda para documentar el funcionamiento del algoritmo.

La segunda parte es una descripción de los elementos que forman el entorno del propio programa. Incluiríremos la declaración de los objetos del programa, es decir, declaración de variables (numéricas enteras, numéricas reales, alfanuméricas, lógicas) y debe ajustarse a las siguientes normas:

La declaración se puede realizar en una o varias líneas.

Si van en una línea debemos separarlas mediante comas.

La tercera parte indica el secuenciamiento de instrucciones que posteriormente se irán codificando en un lenguaje de programación. Es el algoritmo que resuelve el problema.

5.2. Tablas de decisión

Las tablas de decisión se usan para representar la descripción de situaciones, se representan las distintas alternativas, estados de la naturaleza y las consecuencias, proporcionan una descripción completa, correcta, clara y concisa de una situación que se resuelve por una decisión tomada en un momento específico del tiempo.

Una tabla de decisión es una herramienta que sintetiza procesos en los cuales se dan un conjunto de condiciones y un conjunto de acciones a tomar según el valor que toman las acciones. Puede utilizarse como herramienta en las distintas áreas de modelos de los proyectos: la exposición de los hechos, en el análisis del sistema actual, en el diseño del nuevo sistema y en desarrollo del software.

Esta herramienta ayuda al analista a integrar los datos recopilados por los diversos métodos y a representar de manera más fácil la lógica de un problema cuando ésta es más o menos complicada.

En una tabla de decisión se distinguen cuatro zonas:

- **Identificación de Condiciones:** señala aquellas que son más relevantes. Se detalla una condición por renglón. Se llaman condiciones a situaciones variables que pueden ocurrir
- **Entradas de Condiciones:** indican qué valor, si es que lo hay, se debe asociar para una determinada condición. Se indican valores de las condiciones indicadas en la primera sección, dependiendo del tipo de tabla de decisión (de entrada, limitada o extendida) que se construya para representar el proceso.
- **Identificación de Acciones:** enumera el conjunto de todos los pasos que se deben seguir cuando se presenta cierta condición. Se llaman acciones a los distintos comportamientos que se asumirán en función de los valores que tomen las condiciones. Se escriben en el orden en que deben ser ejecutadas.
- **Entradas de Acciones:** muestran las acciones específicas del conjunto que deben emprender cuando ciertas condiciones o combinaciones de éstas son verdaderas.

CONDICIONES	1	2	3	4
¿Paga contado?	S	S	N	N
¿Compra > \$ 50000?	S	N	S	N
ACCIONES				
Calcular descuento 5% s/importe compra	X	X		
Calcular bonificación 7% s/importe compra	X		X	
Calcular importe neto de la factura,	X	X	X	X

<http://eve-ingsistemas-u.blogspot.com/2012/05/tablas-de-decision-parte-1.html>

5.3. Diagrama de flujo

Un diagrama de flujo es una representación gráfica del flujo de datos u operaciones de un programa. Es una excelente ayuda en el proceso de datos, debido a que nos permite representar el circuito de información desde su entrada como dato, hasta su salida como resultado, esclareciendo la secuencia de operaciones.

En este gráfico existe una secuencia en la información: entra una información de entrada que, a través de un proceso de decisión se transforma en información de salida.

Los diagramas de flujo se utilizan en fases distintas del ciclo de vida del software, siendo distinto el nombre para cada etapa. Así, en la etapa de análisis recibe el nombre de organigrama y en la etapa de programación recibe el nombre de ordinograma.

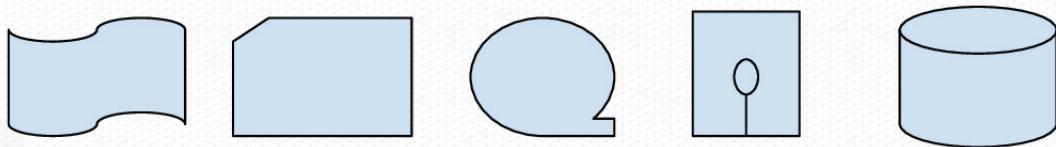
5.4. Organigrama

Los organigramas constituyen un procedimiento común en informática, que consiste en reflejar gráficamente la ubicación de información en los soportes periféricos de entrada y salida, unida por el flujo de datos e informaciones que utiliza un programa de manera clara y sencilla.

En un organigrama se sitúa el nombre del programa en un rectángulo central al que entran desde arriba los datos de entrada y sale hacia abajo la información procesada. A los laterales se representan los soportes de almacenamiento.

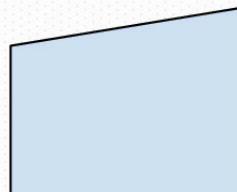
Un organigrama debe dar un visión externa e interna a través de los siguientes elementos:

Soportes de datos de Almacenamiento: Según han evolucionado los soportes se han ido utilizando distintos símbolos que reflejan los distintos medios de almacenamiento:

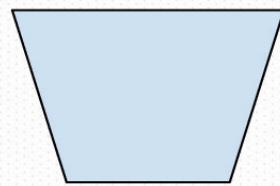


Cinta perforada Tarjeta perforada Cinta Magnética Disquete Disco

Periféricos de Entrada: Que permiten la introducción de datos, tales como el teclado, el ratón o el joystick.



Teclado



Periférico de Entrada

Los Periféricos de Salida: Utilizados para visualizar la información como la pantalla o la impresora:



Impresora



Pantalla

Símbolo	Función
	Proceso u Operación
	Clasificación u ordenación de datos en un fichero
	Fusión de dos ficheros en uno
	Partición de un fichero o extracción de datos
	Mezcla (Intercalación) de uno o más ficheros

	Dirección del flujo de datos
	Línea de teleproceso
	Línea conductora. Une elementos de información

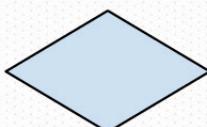
5.5. Ordinograma

Los ordinogramas representan gráficamente paso a paso todas las instrucciones del algoritmo, reflejando la secuencia lógica de las operaciones necesarias para la resolución de un problema. Estas operaciones van encaminadas a su resolución por medio del ordenador.

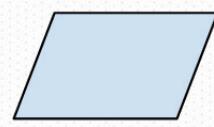
Al igual que en el caso de, los organigramas, para la construcción de un ordinograma se requieren unos determinados símbolos que difieren de los organigramas:



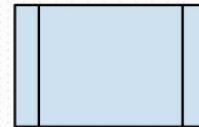
Proceso



Decisión



E/S



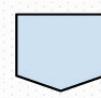
Llamada a procedimiento



Comienzo/Fin



Conector



Conector fuera de la página

El ordinograma detalla paso a paso el proceso que debemos seguir en la etapa de programación, y debe contar con los siguientes elementos:

- Un comienzo que viene señalado con la palabra “Inicio” y que determina el principio del programa.
- Una secuencia de operaciones lo más detallada posible y siguiendo siempre el orden en que se debe ejecutar (De arriba abajo y de izquierda a derecha).
- Un fin de programa marcado por la palabra “Fin”.

Las reglas que hay que seguir para la elaboración de un ordinograma son las siguientes:

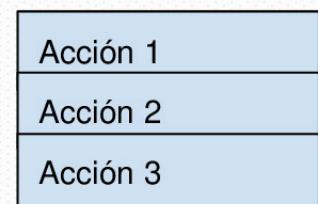
- Todos los símbolos utilizados en el diseño deben estar conectados con líneas de flujos de datos.
- El diseño debe realizarse con la máxima claridad, siempre de arriba abajo y de izquierda a derecha.
- Queda terminantemente prohibido el cruce de líneas de conexión: eso indicaría un mal diseño.
- A un símbolo de proceso pueden llegarle varias líneas de conexión o flujo de datos, pero de él puede salir sólo una línea de conexión.
- A un símbolo de final de proceso o ejecución de programa pueden llegar muchas líneas de conexión, pero de él no puede salir ninguna.

5.6. Diagramas de Nassi-Scheiderman (Diagramas N-S)

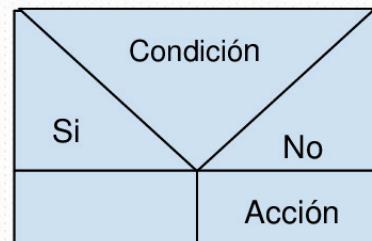
Los diagramas N-S tienen la ventaja de adecuarse mejor a las técnicas de programación estructurada.

Los diagramas N-S no utilizan flechas para indicar el flujo de control.

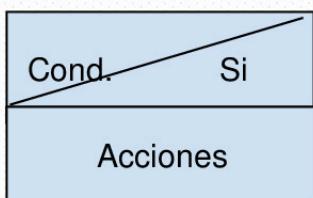
Un diagrama N-S es un dibujo contenido en un rectángulo. Dentro del rectángulo se incluyen una serie de símbolos adyacentes. Los símbolos usados corresponden a las instrucciones de control: secuenciales, condicionales y bucles o ciclos; estos serán:



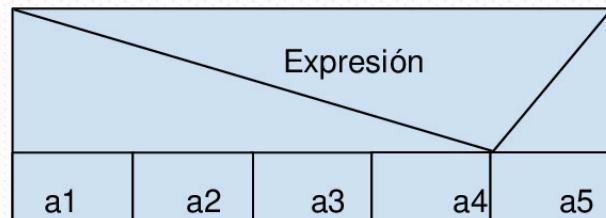
Acciones secuenciales



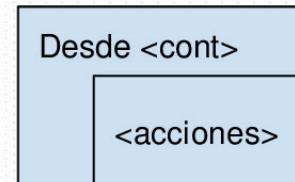
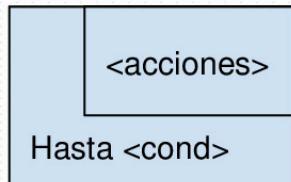
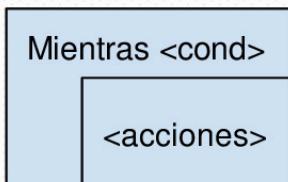
Condicional doble



Condicional simple



Condicional Múltiple



Ciclos

5.7. Diagrama estructurado

Además de los anteriores, hubo diversos autores que desarrollaron sistemas de representación adecuados a la escritura de programas estructurados:

- Método de Warnier
- Método de Jackson
- Método de Bertini
- Método de Tabourier
- Método de Chapin

6. Eficiencia de los Algoritmos

Como ya hemos comentado, para resolver un mismo problema, se pueden definir infinidad de algoritmos.

Por tanto, es necesario, una vez diseñado el primer algoritmo, realizar una evaluación del mismo. Si se decide que éste no es eficiente, será necesario o bien diseñar uno nuevo, o bien optimizar el original.

Optimizar un algoritmo consiste en introducir modificaciones en él, que dependen principalmente dos factores:

- El **tiempo** que se necesita para ejecutarlo. Para comparar 2 algoritmos no es necesario conocer el tiempo real que invertirá la computadora, basta con saber el número de instrucciones de cada tipo necesarias para resolver el problema.
- Los **recursos** que se necesitan para implementar el algoritmo. Concretamente en el caso de que el algoritmo deba ejecutarse en un ordenador, se usan fundamentalmente los siguientes recursos: **memoria** principal para almacenar los datos y las instrucciones, y la memoria externa para almacenar datos auxiliares.

La importancia de estos 2 factores dependerá de la naturaleza del problema y de los medios de que se disponga.

6.1. Complejidad computacional. Notación O-Mayúscula.

En general la mayoría de los problemas tienen un parámetro de entrada que es el número de datos que hay que tratar, esto es N.

La cantidad de recursos del algoritmo es tratada como una función de N, F(N). De esta manera se puede establecer un tiempo de ejecución del algoritmo que suele ser proporcional a una de las siguientes funciones: 1, logN, N, N*logN, n^2 , n^3 , 2 elevado a N, etc.

En general el tiempo de ejecución es proporcional a alguno de los tiempos de ejecución anteriormente propuestos, además de la suma de algunos términos más pequeños. Así, un algoritmo cuyo tiempo de ejecución sea $T=3N^2+6N$ se puede considerar proporcional a $N+N$, esto es, $O(N^2)$.

Conviene no obstante diferenciar entre el peor caso y el esperado. Por ejemplo, el tiempo de ejecución del algoritmo Quick Sort es de $O(N^2)$. Sin embargo, en la práctica este caso no se da casi nunca y la mayoría de los casos son $O(N*logN)$

6.2. Relación del tema con el sistema educativo actual

Este tema puede ser desarrollado en los siguientes módulos formativos (para atribución docente de PES)

- Bachillerato – Tecnologías de la Información y la Comunicación II (PES)
- GS- GS – DAW – Desarrollo Web en Entorno Servidor DAW/DAM – Programación (PES)

Aunque los profesores de SAI no tienen una atribución docente en ningún módulo con aplicación directa del tema, siempre es probable que se diseñe algún pequeño programa en clase, para lo que recurrir a la notación de pseudocódigo o de ordinograma sería un recurso esclarecedor.

7. Bibliografía

- Introduction to Java Programming and Data Structures, Comprehensive Version. Y. Daniel Liang. Pearson, 11^a edición. 2017.
- Java 9. Francisco Javier Moldes Teo. Anaya. 2017.
- Introducción a la computación. J. Glenn Brookshear, Pearson, 11^º edición. 2012
- Fundamentos de programación. Algoritmos, estructuras de datos y objetos. Luis Joyanes Aguilar, McGraw-Hill, 4^º edición. 2008.
- Langsam, Augenstein y Tanembaum: “Estructuras de Datos con C y C++”, Prentice-Hall 1997
- Prieto A., Lloris A. y Torres J.C.: Introducción a la Informática, 4^a ed. McGraw-Hill, 2006

