

Diseño de algoritmos. Técnicas descriptivas

Índice

1. Introducción.....	2
2. Definición de algoritmo y algoritmia.....	2
3. Técnicas descriptivas.....	2
3.1. Diagramas de flujo.....	2
3.1.1. ANSI.....	2
3.1.2. UML (<i>Unified Modeling Language</i>).....	3
3.1.3. Nassi-Schneiderman.....	4
3.1.4. WML.....	4
3.2. Pseudocódigo.....	4
4. Diseño de algoritmos.....	5
4.1. Divide y vencerás (<i>divide and conquer</i>).....	5
4.2. Algoritmos voraces.....	6
4.3. Programación dinámica.....	6
4.4. Vuelta atrás o <i>backtracking</i>	6
5. Complejidad de los algoritmos.....	6
6. Conclusiones.....	6

1. Introducción

El objetivo principal de los sistemas informáticos es ayudarnos a resolver problemas y realizar operaciones y tareas de manera eficiente y eficazmente. Para solucionar requieren disponer de programas; que son secuencias de instrucciones que obtienen resultados en función de unos valores de entrada. Los programas se codifican en lenguajes de programación que definen secuencia de **instrucciones** y manejan **estructuras de datos**.

La idea general de partida es que hay que pensar antes de codificar en los siguientes aspectos:

- **Estructura de datos** a manipular y
- **Diseño de algoritmo** que solucione el problema planteado de una forma optimizada.

En este tema veremos: **definiciones** de los principales conceptos a tener en cuenta, **técnicas** para **representar** algoritmos (descriptivas) y las **familias** de algoritmos más utilizadas.

2. Definición de algoritmo y algoritmia

Se define **algoritmo** como conjunto ordenado y finito de operaciones no ambiguas que permite hallar una solución a un problema. Características:

- Debe indicar el **orden de realización** de cada paso, y cada uno debe estar descrito de manera precisa.
- Debe **terminar** en un **número finito** de pasos.
- Debe llegar siempre al **mismo resultado para las mismas entradas**.

Puede haber infinitos algoritmos para el mismo problema.

Se define **algoritmia** a la ciencia que estudia la definición, corrección y eficiencia de los algoritmos.

3. Técnicas descriptivas

Las técnicas descriptivas nos permiten definir algoritmos. Son independientes a los lenguajes de programación. Existen diversas técnicas: diagramas de flujo y pseudocódigo.

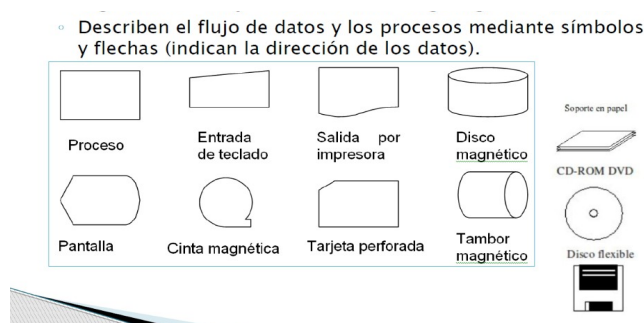
3.1. Diagramas de flujo

Los diagramas de flujo describen el flujo de los datos y de los procesos mediante símbolos y flechas que indican la dirección de los datos. También llamados **organigramas** por ANSI.

3.1.1. ANSI

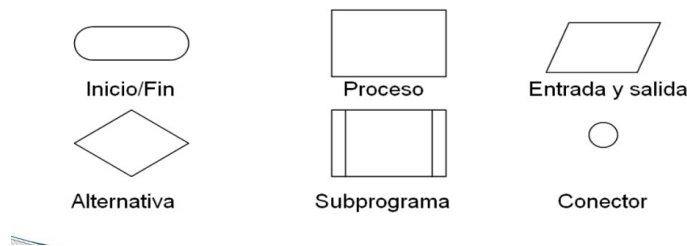
El ANSI (*American National Standard Institute*) define varios tipos con diferentes símbolos:

- Los diagramas de flujo de sistemas u **organigramas**:

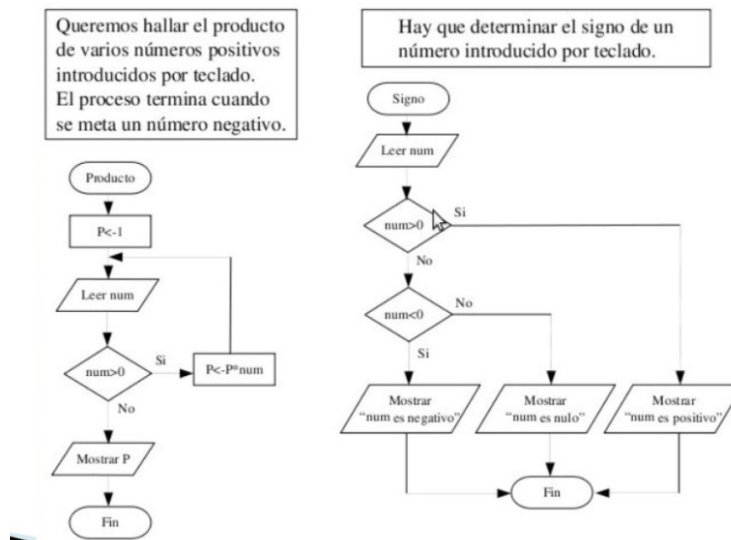


- Los diagramas de flujo de proceso u **ordinogramas** tienen más detalle que los anteriores:

- Más detallados que los anteriores.



Ejemplos:



3.1.2. UML (Unified Modeling Language)

Estándar de descripción de esquemas de software (versión actual 2.5 de junio del 2015). **Diseñado para orientación a objetos** aunque puede aplicarse a otros ámbitos y esquemas.

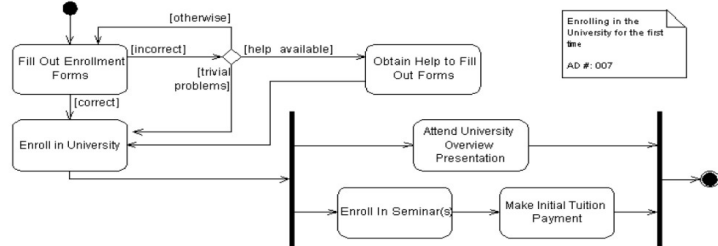
Existen los siguientes diagramas y sus definiciones:

- De casos de uso.** Ayuda a definir lo que existe fuera del sistema (actores) y lo que debe realizar el sistema (casos de uso). El caso de uso capta las interacciones que ocurren entre los productores y consumidores de la información y el sistema en sí.
- De clases.** Especifica las relaciones entre los objetos involucrados.
- De secuencia.** Se usa para mostrar las comunicaciones dinámicas entre objetos durante la ejecución de una tarea.
- De colaboración.** Proporciona otro indicio del orden temporal de las comunicaciones, pero enfatiza las relaciones entre los objetos y clases en lugar del orden temporal.
- De estado.** Método de representación del comportamiento de un sistema que ilustra sus estados y los eventos que ocasionan que el sistema cambie de estado.
- De componentes.** Representa la estructura arquitectónica general.
- De implementación.** Se enfoca en la estructura de un sistema de software y es útil para mostrar la distribución física de un sistema de software entre plataformas de hardware y entornos de ejecución.
- De relaciones de entidad.** Nivel de abstracción alto respecto a las entidades que participan en el proceso.
- De actividad:** que muestra actividades, así como los cambios de una a otra actividad junto con los eventos que ocurren en ciertas partes del sistema.

Técnicas descriptivas. UML

SÍMBOLO	SIGNIFICADO
●	Punto de inicio del proceso
▭	Actividad
◇	Condicional
→	Flujo de secuencia
—	Bifurcación o entrada
●	Punto final del proceso
▭	Swimlanes ("Calles")

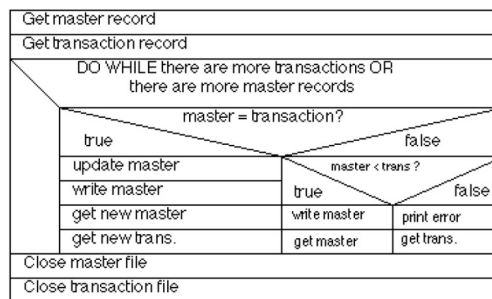
www.agilemodeling.com/style/activityDiagram



3.1.3. Nassi-Schneiderman

Diagrama de cajas apoyado en la programación estructurada. De difícil mantenimiento.

Diagrama de cajas apoyado en la programación estructurada. De difícil mantenimiento.



3.1.4. WML

Para el desarrollo web se están desarrollando así mismo estándares para describir algoritmos y procesos:

- WebML (*Web Modeling Language*) es una notación visual para el diseño de aplicaciones web complejas.
- UWE (*UML-based Web Engineering*) es una extensión de UML para web.

3.2. Pseudocódigo

Algunos algoritmos presentan complejidades que hacen que su representación gráfica no sea adecuada. Recurrimos entonces a un pseudolenguaje independiente de los lenguajes de programación pero que recuerda a ellos, con lo que es más fácil eliminar ambigüedades. No está estandarizado.

Un algoritmo expresado en pseudocódigo debe tener siempre la siguiente estructura:

ALGORITMO <nombre>

```

CONSTANTES
  {Definir las constantes}
VARIABLES
  {Definir las variables}
INICIO
  Acción 1
  Acción n
FIN
  
```

4. Diseño de algoritmos

Los algoritmos **realizan determinadas funciones** para resolver problemas, ya existen algunos bien conocidos y clasificados:

- Algoritmos de **ordenación**: La ordenación es una aplicación fundamental en computación. La ordenación o clasificación es un proceso de organizar un conjunto de datos en algún orden o secuencia específica, tal como creciente o decreciente para datos numéricos o el orden alfabético para datos compuestos por caracteres. Por ejemplo: selección, burbuja, inserción directa o binaria, Shell, *quicksort*, secuencial, etc.
- Algoritmos de búsqueda: **Binaria o dicotómica**: Este algoritmo compara el dato buscado con el elemento central. Según sea menor o mayor se prosigue la búsqueda con el subconjunto anterior o posterior, respectivamente, al elemento central, y así sucesivamente. **Hash**. Aumenta la velocidad de búsqueda, pero que no requiere que los elementos estén ordenados. Consiste en **asignar** a cada elemento un **índice** mediante una transformación del elemento. Esta correspondencia se realiza mediante una función de conversión, llamada función *hash*. Etc.

La algoritmia ha producido familias o técnicas de algoritmos patrón que permiten solucionar problemas genéricos. La idea es encontrar paralelismos entre los problemas a solucionar y estos patrones. Para eso se utilizan distintas técnicas que detallaremos a continuación:

- Divide y vencerás.
- Algoritmos voraces.
- Programación dinámica.
- *Backtracking*.

Existen otras técnicas, que quedan fuera el alcance de este tema:

- Algoritmos paralelos: dividen problemas complejos para ser tratados en distintos procesadores. Por ejemplo: ADN.
- Algoritmos probabilísticos. Usan valores pseudoaleatorios. Por ejemplo: tiempo.
- Algoritmos determinísticos: lineales.
- Algoritmos NO determinísticos: árboles.
- Metaheurística: encuentran soluciones aproximadas.
- Ramificación y acotación: es un *backtracking* optimizado para eliminar soluciones no óptimas.

4.1. Divide y vencerás (*divide and conquer*)

Aplicable si el problema puede dividirse en subproblemas independientes y si sumando las soluciones de estos solucionamos el todo. Pasos:

- 1 Dividir el problema en uno o más instancias más pequeñas (más sencillas).
- 2 Resolver cada una de las nuevas instancias. Si no tienen una solución trivial, usamos recursividad para solucionarlo, es decir, llegar a **un caso base de solución trivial**.
- 3 Combinar, si se necesita, la solución de cada subproblema, para obtener la solución a la instancia general.

```
divideYVenceras (problema, solución)
Si es caso base
    Resolver el problema
    Devolver solución
Si no es caso base
    Dividir el problema (Problema1, ..., Sol1,...)
    divideYVenceras(Problema1, Sol1)
    ...
    divideYVenceras(ProblemaN, SolN)
    CombinarSoluciones (Sol1, ..., SolN)
```

4.2. Algoritmos voraces

Se usan cuando queremos solución óptima: devolver el menor número de monedas, encontrar un camino mínimo, etc.

El algoritmo selecciona candidatos pensando en la **mejor solución momentánea** (pero no verifica si eso es peor a la larga). Cada vez que añadimos un candidato se verifica si es factible la solución con él o si ya hemos llegado a la solución final.

No funciona para todos los casos: funciona si la optimización local lleva a la global.

```
void voraz(conjunto c, conjunto &solucion) {
    // c = conjunto de candidatos
    solucion = ∅;
    Mientras (c != ∅ && !testSolucion(solucion)) {
        x = seleccionar(c);
        c = c \ {x}; //Sacamos al candidato de posibles
        Si (factible(solucion ∪ {x}))
            solucion = solucion ∪ {x};
    }
    Si (!testSolucion(solucion))
        error("No hay solución");
}
```

4.3. Programación dinámica

Elimina recursividad buscando eficacia. Guarda en **tablas temporales** los resultados de las iteraciones parciales. Usa un método *bottom-up*.

4.4. Vuelta atrás o *backtracking*

Los algoritmos voraces no siempre llegan a la mejor solución. *Backtracking* es un algoritmo recursivo y exhaustivo, que prueba todos los caminos posibles hasta encontrar uno válido. Se usa cuando no hay información sobre cuál puede ser la posible solución.

5. Complejidad de los algoritmos

Para medir la eficiencia de un algoritmo se usan dos criterios:

- **Memoria** que consume su procesamiento: depende del número y tipo de las variables utilizadas.
- **Orden de magnitud $O(n)$** del algoritmo: depende de la estrategia seguida para implementar la solución. Para calcularlo vamos a estudiar cómo crece el número de instrucciones que el programa debe ejecutar cuando los datos son muchos. Si lo representamos podemos comparar con familias de funciones que crecen de la misma manera.

6. Conclusiones

El **diseño de algoritmos** surge de la necesidad de tener sistemas de información de calidad en el menor tiempo posible, un buen diseño que recoja los requerimiento de los usuarios y clientes de los sistemas de información ayudan a producir un producto de calidad.

En este tema se han descrito las definiciones de algoritmo y algoritmia así como las principales técnicas descriptivas: diagramas y pseudocódigo. Con el objetivo de agilizar los procesos de construcción han surgido patrones de diseño de algoritmos y en este tema se describen algunos. Para terminar se introduce el tema de complejidad de los algoritmos.