Sistemas y Aplicaciones Informáticas

Tema 25. Diseño de Algoritmos. Técnicas Descriptivas.

1. Ámbito de docencia.

- Sistemas informáticos monousuario y multiusuario (ASI 1).
- Sistemas informáticos multiusuario y en red (DAI 1).
- Sistemas operativos en entornos monousuario y multiusuario (ESI 1).

2. Diseño de algoritmos.

2.1. Algoritmos.

2.1.1. Concepto. Resolución de problemas. Algoritmos y programas.

- Un algoritmo es una secuencia ordenada de operaciones que permite llevar a cabo una tarea determinada. La resolución de problemas es el proceso que consiste en el diseño de un conjunto de algoritmos que permitan resolver un problema partiendo de su descripción.
- El objetivo principal de la Informática es utilizar los sistemas informáticos como herramientas para la resolución de problemas. Para ello es necesario transformar los algoritmos en programas codificados en un lenguaje de programación adecuado para obtener los resultados deseados.
- Por tanto, para llegar a la realización de un programa es necesario el diseño previo del algoritmo.

2.1.2. Características de un algoritmo.

- Debe ser preciso y sin ambigüedades, e indicar el orden de realización de cada paso.
- Debe tener un número finito de pasos y finalizar en algún momento.
- Debe producir siempre los mismos resultados a partir de la misma situación inicial, y poder repetirse el número de veces necesarias en cualquier momento.
- Debe ser independiente tanto del lenguaje de programación en que se expresa como del sistema informático que lo ejecuta.
- Debe carecer de errores y obtener la solución al problema en poco tiempo.

2.1.3. Clasificación de los algoritmos.

2.1.3.1. Según su esquema de funcionamiento.

2.1.3.1.1. Algoritmos divide y vencerás.

- Consisten en descomponer el problema original en varios subproblemas más sencillos, para luego resolver éstos mediante un cálculo sencillo. Por último, se combinan los resultados de cada subproblema para obtener la solución del problema original.
- Si los subproblemas son demasiado grandes, pueden dividirse a su vez en nuevos subproblemas hasta que su solución sea trivial, o hasta que su tamaño sea superior a un cierto umbral.

2.1.3.1.2. Algoritmos voraces.

- Se aplican normalmente a problemas de decisión y optimización. Son procesos repetitivos sencillos que tratan sucesivamente los diferentes elementos del problema.
- Inicialmente, el conjunto de elementos que forman la solución está vacío. En cada paso se intenta añadir el mejor de los elementos restantes a dicha solución parcial. Si este conjunto ampliado satisface las restricciones del problema, el elemento se incorpora definitivamente. Por el contrario, si dicho conjunto no es válido se desecha el elemento.

2.1.3.1.3. Algoritmos de programación dinámica.

- Se aplican a problemas de optimización, y producen varias secuencias de decisiones.

- Si un problema se resuelve tras tomar una determinada secuencia de decisiones, y existen varias opciones posibles para cada una de las decisiones, se evita explorar todas las secuencias posibles por medio de la resolución de subproblemas de tamaño creciente y del almacenamiento en una tabla de sus soluciones óptimas para facilitar la solución de los problemas más grandes.

2.1.3.1.4. Algoritmos de vuelta atrás.

- Se aplican a problemas de búsqueda. Realizan una búsqueda sistemática, probando todo lo posible hasta encontrar la solución o encontrar que no existe solución al problema.
- Para conseguir este propósito, se separa la búsqueda en varias búsquedas parciales o subtareas, que a su vez suelen incluir más subtareas. En el caso de no encontrar una solución en una subtarea se retrocede a la subtarea original y se prueba otra distinta a las probadas anteriormente.

2.1.3.1.5. Algoritmos de ramificación y acotación.

- Son una variante de los algoritmos de vuelta atrás para problemas donde se trata de encontrar el valor máximo o mínimo de alguna función.
- Se hace un recorrido exhaustivo del espacio de búsqueda en el que sólo se descartan aquellos caminos que no son factibles o que no pueden conducir a una solución mejor que la que se tiene en estos momentos. La utilización de una función de estimación correcta es imprescindible para garantizar el buen funcionamiento del algoritmo.

2.1.3.2. Según su tiempo de ejecución.

- **Logarítmicos**. Aquellos que son proporcionales a *log N*, siendo *N* el número de datos que debe tratar el algoritmo. En la práctica se puede considerar como constante.
- **Polinomiales**. Aquellos que son proporcionales a N^k, siendo *N* el número de datos que debe tratar el algoritmo y *k* una constante. Existe en general al menos un algoritmo de este tipo capaz de resolver un problema en un tiempo razonable.
- **Exponenciales**. Aquellos que son proporcionales a k^N, siendo *N* el número de datos que debe tratar el algoritmo y *k* una constante. No suelen ser muy útiles en la práctica por su elevadísimo tiempo de ejecución, salvo un número de datos de entrada muy reducido.

2.2. Especificación y desarrollo de algoritmos.

2.2.1. Especificación. Semántica axiomática.

- La <u>especificación</u> de un algoritmo describe qué debe hacer mediante la relación existente entre la situación inicial previa a la ejecución del algoritmo y los resultados de la misma.
- Un método formal de formular la especificación de un algoritmo es la <u>semántica axiomática</u>:
  - Se basa en expresar los valores de todos los datos que debe manejar el algoritmo mediante fórmulas lógicas denominadas <u>predicados</u>.
  - La especificación de un algoritmo consta de lo siguiente:
    - **Declaración de las variables** que utilizará el algoritmo.
    - **Precondición**. Predicado que indica el estado de los datos antes de su ejecución.
    - **Postcondición**. Predicado que indica el estado de los datos después de su ejecución.
  - Esto significa que si al empezar a ejecutarse el conjunto de instrucciones se cumple la precondición, al acabar necesariamente tiene que cumplirse la postcondición. Si la precondición no se cumple, el algoritmo no puede ejecutarse.

2.2.2. Elementos de un algoritmo.

- **Acción**. Cada uno de los pasos que lleva a cabo el algoritmo, actuando sobre los datos.
- **Proceso**. Es el conjunto de acciones que realiza el algoritmo para obtener los resultados.
- **Datos de entrada**. Son los datos iniciales que posee el algoritmo antes de ejecutarse.
- **Datos internos**. Son los datos no accesibles desde el exterior que condicionan el algoritmo.
- **Datos de salida**. Son los resultados que obtiene finalmente el algoritmo.
- **Recursos**. Deben tenerse en cuenta al implementar el algoritmo en un sistema informático:
  - *Tiempo de ejecución*. Período transcurrido entre el inicio y el fin del algoritmo.
  - *Memoria utilizada*. Cantidad de memoria que necesita el algoritmo para su ejecución.

2.2.3. Entidades algorítmicas.

2.2.3.1. Datos. Tipos de dato.

- Cualquier dato manejado por un algoritmo tiene un tipo asociado que define el conjunto de valores que puede tomar y las operaciones básicas que se pueden realizar sobre dicho conjunto.
- Los tipos de dato pueden ser:
  - *Simples*. Son aquellos que no pueden descomponerse en otros tipos de dato:
    - **Numéricos**. Representan a los números enteros y los reales. Este tipo de dato permite realizar operaciones aritméticas comunes.
    - **Lógicos**. Son aquellos que sólo pueden tener dos valores, verdadero o falso.
    - **Alfanuméricos**. Es una secuencia de caracteres alfanuméricos que es interpretada como un único dato. Es posible representar números como alfanuméricos, pero no es posible hacer operaciones aritméticas con ellos.
  - *Estructurados*. Son aquellos que están construidos a partir de otros tipos de dato.

2.2.3.2. Constantes y variables. Identificadores.

- Una constante es un dato cuyo valor no cambia durante el desarrollo del algoritmo. Una variable es un dato cuyo valor cambia a medida que se suceden las acciones del algoritmo.
- Tanto las variables como las constantes tienen asociado un tipo de dato, y se representan mediante un identificador formado por una secuencia de caracteres alfanuméricos.
- Los identificadores en general deben seguir ciertas reglas:
  - Debe ser único y no puede utilizarse ninguna palabra reservada. Su longitud puede ser de varios caracteres y debe dar una idea del valor que contiene.
  - Debe comenzar con una letra y no pueden contener espacios en blanco. Las letras, los dígitos y ciertos caracteres especiales están permitidos después del primer carácter.

2.2.3.3. Operadores. Expresiones.

- Los operadores son símbolos que relacionan una o más variables y/o constantes, y que indican el tipo de operación que se realiza entre los operandos. Pueden ser:
  - *Aritméticos*. Permiten la realización de operaciones matemáticas. Los más importantes son suma (+), resta (-), multiplicación (*), división (/), módulo (mod) y exponenciación (^).
  - *Relacionales*. Se utilizan para comparar dos valores. Son los siguientes: mayor que (>), menor que (<), mayor o igual que (>=), menor o igual que (<=), distinto (<>) e igual (=).
  - *Lógicos*. Se utilizan para realizar operaciones lógicas. Son los siguientes: and, or y not.

- Las expresiones son combinaciones de constantes, variables 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. Se clasifican en:
  - *Aritméticas*. Tienen las siguientes características:
    - Están formadas por variables y constantes numéricas y por operadores aritméticos.
    - Producen un resultado numérico entero si los valores implicados son enteros, y real si los valores implicados son enteros y/o reales.
    - En una misma expresión los operadores se evalúan en el siguiente orden: exponenciación, multiplicación, división, modulo, suma y resta.
  - *Relacionales*. Tienen las siguientes características:
    - Están formadas por variables y constantes del mismo tipo y por operadores relacionales.
    - Producen un resultado lógico verdadero si la comparación entre los valores es cierta, y falso si la comparación entre los valores no es cierta.
    - En una misma expresión los operadores relacionales tienen el mismo nivel de prioridad. Los operadores relacionales tienen menor prioridad que los aritméticos.
  - *Lógicas*. Tienen las siguientes características:
    - Están formadas por variables y constantes lógicas y por operadores lógicos.
    - Producen un resultado lógico que viene determinado por las tablas de verdad correspondientes a cada uno de los operadores.

A	B	A AND B	A OR B	NOT A
F	F	F	F	V
F	V	F	V	
V	F	F	V	F
V	V	V	V	

    - En una misma expresión los operadores se evalúan en el siguiente orden: not, and y or. Los operadores lógicos tienen menor prioridad que los relacionales.
  - *Alfanuméricas*. Están formadas por variables y constantes alfanuméricas y producen un resultado alfanumérico. Se suele utilizar el operador '+' para la concatenación de caracteres.
- Una expresión puede estar formada por un conjunto de expresiones unidas por diferentes operadores, aunque debe mantenerse la integridad entre los valores devueltos por cada una de las expresiones y el tipo de operador que las une.
- Cuando en una expresión hay más de un operador, hay que tener en cuenta las reglas de orden de prioridad de las operaciones. Para alterar este orden se usan los paréntesis, de forma que toda expresión encerrada entre paréntesis se ejecuta primero. En caso de coincidir la prioridad de dos o más operadores, éstos se ejecutan de izquierda a derecha.

2.2.3.4. Declaración de variables y constantes. Primitivas. Subalgoritmos.

- La declaración de variables y constantes consiste en listar al principio del algoritmo todas las variables y su tipo de dato asociado, y todas las constantes con su respectivo valor. Sirve para advertir y documentar el uso de variables y constantes en el algoritmo.
- Una primitiva es una acción sobre los datos que se lleva a cabo inmediatamente, y puede ser una:

- *Asignación*. Consiste en pasar a una variable un determinado valor, que puede ser una constante o el resultado de una expresión, compatible con el tipo de dato de la variable.
- *Entrada de datos*. Consiste en recibir desde un dispositivo de entrada (por ejemplo el teclado) un determinado valor, y almacenarlo en una variable.
- *Salida de datos*. Consiste en enviar a un dispositivo de salida (un monitor o una impresora) el contenido de una variable, un resultado o un mensaje.

– Un subalgoritmo consiste en agrupar un conjunto de acciones que realiza una determinada tarea e identificarlo por un nombre, sustituyendo dicho conjunto por su nombre dentro del algoritmo.

2.2.3.5. Estructuras algorítmicas.

2.2.3.5.1. Estructuras de control alternativas.

– Se basan en la evaluación de una expresión lógica o relacional para controlar la realización de ciertas acciones y alterar de este modo el orden secuencial del algoritmo.
– Existen tres tipos básicos de estructuras de control alternativas:
  - *Simples*. Si la expresión devuelve un valor lógico verdadero, se lleva a cabo un conjunto de acciones. En caso contrario no se realiza y se continúa con la siguiente acción.
  - *Dobles*. Si la expresión devuelve un valor lógico verdadero, se lleva a cabo un conjunto de acciones. En caso contrario se realiza otro conjunto diferente de acciones.
  - *Múltiples*. Controla la realización de varios conjuntos de acciones mediante el valor final de una expresión, de tal forma que cada conjunto de acciones está ligado a un posible valor de la expresión. Normalmente existe un bloque que especifica las acciones a llevar a cabo en caso de que la expresión tenga algún valor no definido.

2.2.3.5.2. Estructuras de control iterativas.

– Se basan en la evaluación de una expresión lógica o relacional para repetir la realización de ciertas acciones y alterar de este modo el orden secuencial del algoritmo.
– Existen tres tipos básicos de estructuras de control iterativas:
  - *Mientras*. Primero se evalúa la expresión. Si devuelve un valor lógico verdadero, se lleva a cabo un conjunto de acciones y se vuelve a evaluar la expresión, repitiéndose la realización de las acciones hasta que la expresión tenga un valor lógico falso. Si en la primera evaluación el valor de la expresión es falso, el conjunto de acciones no se realiza.
  - *Repetir hasta*. Primero se lleva a cabo un conjunto de acciones y después se evalúa la expresión. Si devuelve un valor lógico falso, se repite la realización de las acciones y la evaluación de la expresión hasta que ésta tenga un valor lógico verdadero. En cualquier caso el conjunto de acciones se realiza al menos una vez.
  - *Para-Hasta*. Repite un conjunto de acciones un número concreto de veces en función del contenido de una variable de control. En primer lugar se le asigna un determinado valor y después se evalúa una expresión en la que participe dicha variable. Si devuelve un valor lógico verdadero, se lleva a cabo un conjunto de acciones, se actualiza el valor de la variable y se vuelve a evaluar la expresión, repitiéndose la realización de las acciones y la actualización de la variable hasta que la expresión tenga un valor lógico falso. Si en la primera evaluación el valor de la expresión es falso, el conjunto de acciones no se realiza.

2.2.4. Fases de desarrollo.

- **Análisis**. En esta fase se determina cuál es exactamente el problema a resolver, qué datos forman la entrada del algoritmo y cuáles deberán obtenerse como salida. Para ello es necesario:
  - Acotar y especificar el problema con total precisión para obtener el máximo de información acerca de lo que se debe resolver y las soluciones a determinar. El problema ha de ser comprendido perfectamente antes de realizar el algoritmo.
  - Definir los datos iniciales que se deben proporcionar al problema para resolverlo.
  - Definir que datos o resultados debe proporcionar el algoritmo.
- **Diseño**. Consiste en elaborar el algoritmo de manera que se ajuste a lo establecido en el análisis. La técnica más utilizada es el diseño descendente, que consiste en una serie de descomposiciones sucesivas del problema inicial, estableciendo una serie de niveles de mayor a menor complejidad que den solución al problema. Los niveles se estructuran jerárquicamente de manera que un nivel y su inmediato inferior se relacionan mediante entradas y salidas de información.
- **Implementación**. Consiste en la codificación de programas a partir de la especificación del algoritmo, y deberá realizarse utilizando un determinado lenguaje de programación.
- **Prueba**. Se comprueba que el algoritmo obtiene la salida esperada para todas las entradas.

3. Técnicas descriptivas.

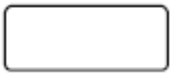
3.1. Descripción de algoritmos.

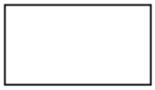
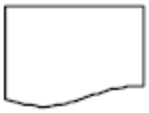
3.1.1. Notaciones algorítmicas. Tipos.

- Una notación algorítmica es un conjunto de convenios adoptados que permite describir las operaciones realizadas y los datos manipulados por el algoritmo, y controlar la realización de las acciones, indicando el modo de organización de estas acciones en el tiempo.
- La notación algorítmica puede ser:
  - *Gráfica*. Representa de forma gráfica las operaciones que realiza un algoritmo.
  - *Textual*. Representa de forma descriptiva las operaciones que realiza un algoritmo.

3.1.2. Diagramas de flujo.

- Es una notación algorítmica gráfica que utiliza un conjunto de símbolos gráficos normalizados y expresa de forma clara los posibles caminos de ejecución de las acciones. Sin embargo, si el algoritmo es complejo este tipo de notación no es adecuada.
- Las normas para la elaboración de diagramas de flujo son las siguientes:
  - El principio y el final del algoritmo sólo deben aparecer una vez cada uno.
  - El flujo del algoritmo se toma de principio a fin y de izquierda a derecha del diagrama.
  - No deben quedar líneas de flujo sin conectar, y se debe evitar el cruce de líneas utilizando los conectores, aunque sólo cuando sea necesario.
  - Deben utilizarse solamente líneas de flujo horizontales y verticales.
- Algunos de los símbolos utilizados son los siguientes:

Símbolo	Descripción
	Indica el inicio y el final del diagrama de flujo.

Símbolo	Descripción
	Indica la entrada y salida de datos.
	Símbolo de proceso que indica la ejecución de una acción.
	Indica la salida de información por impresora.
	Conector dentro de la misma página.
	Conector en otra página.
	Indica la salida de información en la pantalla o monitor.

3.1.3. Pseudocódigo.

- Es una notación algorítmica textual que permite la declaración de los datos manipulados por el algoritmo, y dispone de un conjunto pequeño y completo de palabras reservadas que permite expresar las acciones que se han de realizar.
- Se trata de un lenguaje intermedio entre el lenguaje natural y el lenguaje de programación de alto nivel que impone algunas limitaciones para evitar la ambigüedad. Para mejorar la lectura y explicar el funcionamiento del algoritmo, es posible introducir comentarios precedidos por '//'.
- Un algoritmo descrito en pseudocódigo deberá tener siempre la siguiente estructura:

```
ALGORITMO <nombre_algoritmo>;
CONSTANTES
    { Sección reservada para la declaración de datos constantes utilizados por el algoritmo }
    ...
VARIABLES
    { Sección reservada para la declaración de datos variables utilizados por el algoritmo }
    ...
{ Se puede incluir también la definición de acciones con nombre }

INICIO                                  { Comienzo del cuerpo principal de acciones del algoritmo }
    <ACCIÓN1>;
    <ACCIÓN2>;
    ...
    <ACCIÓNN>;
FIN                                     { Fin del cuerpo principal de acciones del algoritmo }
```

3.2. Representación de estructuras algorítmicas.

3.2.1. Primitivas.

- **Asignación**. En general el formato a utilizar es el siguiente: **<variable> ← <expresión>**
- **Entrada de datos**. Se representa de la siguiente manera:

Pseudocódigo:

LEA <variable>

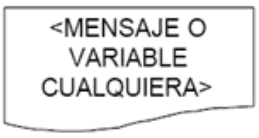
Diagrama de flujo:

– **Salida de datos**. Se representa de la siguiente manera:

Pseudocódigo:

ESCRIBA "MENSAJE CUALQUIERA"
ESCRIBA <variable>
ESCRIBA "La Variable es: ", <variable>

Diagrama de flujo:



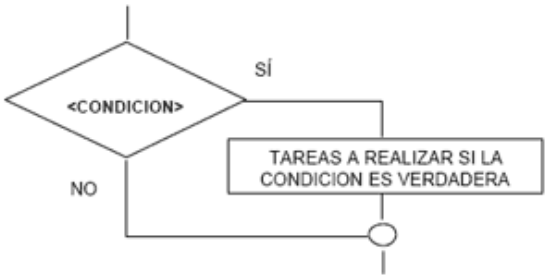
3.2.2. Estructuras de control alternativas.

– **Simples**. Se representa de la siguiente manera:

Pseudocódigo:

Si <condición> entonces
  Instrucción (es)
Fin-Si

Diagrama de flujo:

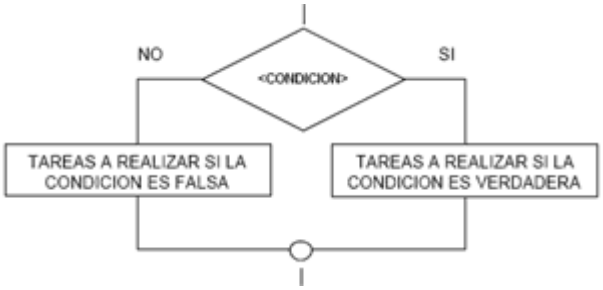


– **Dobles**. Se representa de la siguiente manera:

Pseudocódigo:

Si <condición> entonces
    Instrucción (es)
  Si no
    Instrucción (es)
Fin-Si

Diagrama de flujo:

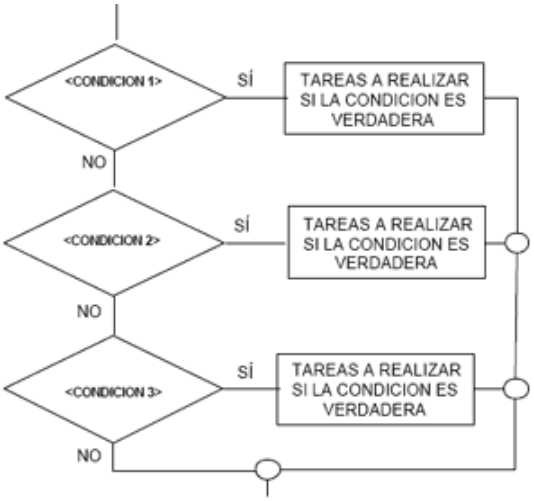


– **Múltiples**. Se representa de la siguiente manera:

Pseudocódigo:

Si <condición> entonces
  Instrucción(es)
 Si no
  Si <condición> entonces
    Instrucción(es)
   Si no
    .
    . Varias condiciones
    .

Diagrama de flujo:



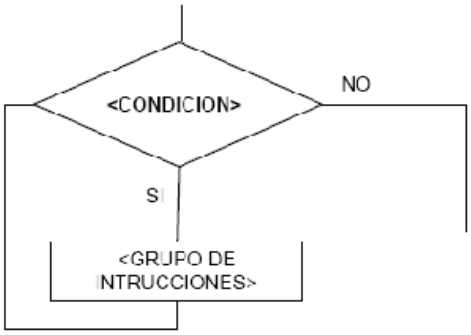
3.2.3. Estructuras de control iterativas.

- **Mientras que**. Se representa de la siguiente manera:

Pseudocódigo

```
Mientras Que <condición>
        Accion1
        Accion2
          .
          .
        AccionN
Fin-Mientras
```

Diagrama de flujo

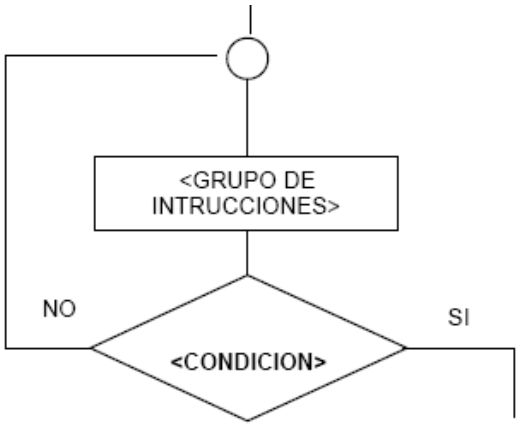


- **Repetir hasta**. Se representa de la siguiente manera:

Pseudocódigo

```
Repita
        Accion1
        Accion2
          .
          .
        AccionN
Hasta <condición>
```

Diagrama de flujo



- **Para-Hasta**. Se representa de la siguiente manera:

Pseudocódigo

```
Para <var> ← <exp1> hasta <exp2> paso <exp3> haga
        <Tareas a repetir>
Fin-Para
```

Diagrama de flujo