aprenderaprogramar.com

Diseño top-down de algoritmos. Abstracción y aprehensión. Parte 2. (CU00226A)

Sección: Cursos

Categoría: Curso Bases de la programación Nivel II

Fecha revisión: 2024

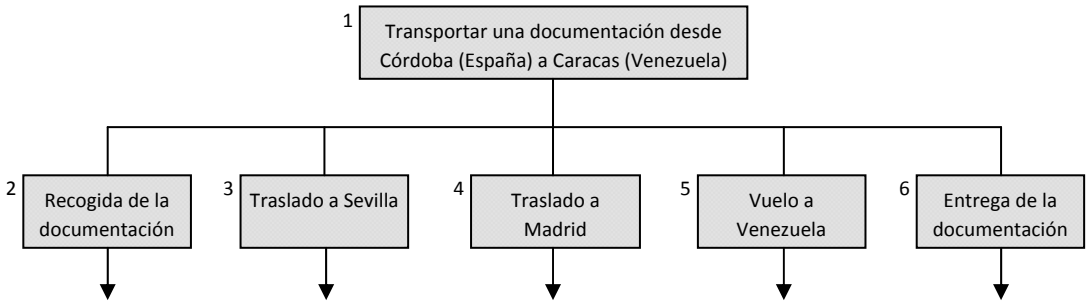
Autor: Mario R. Rancel

Resumen: Entrega nº 25 del Curso Bases de la programación Nivel II

24

ABSTRACCIÓN Y APREHENSIÓN. EL DISEÑO TOP-DOWN DE ALGORITMOS (CONTINUACIÓN)

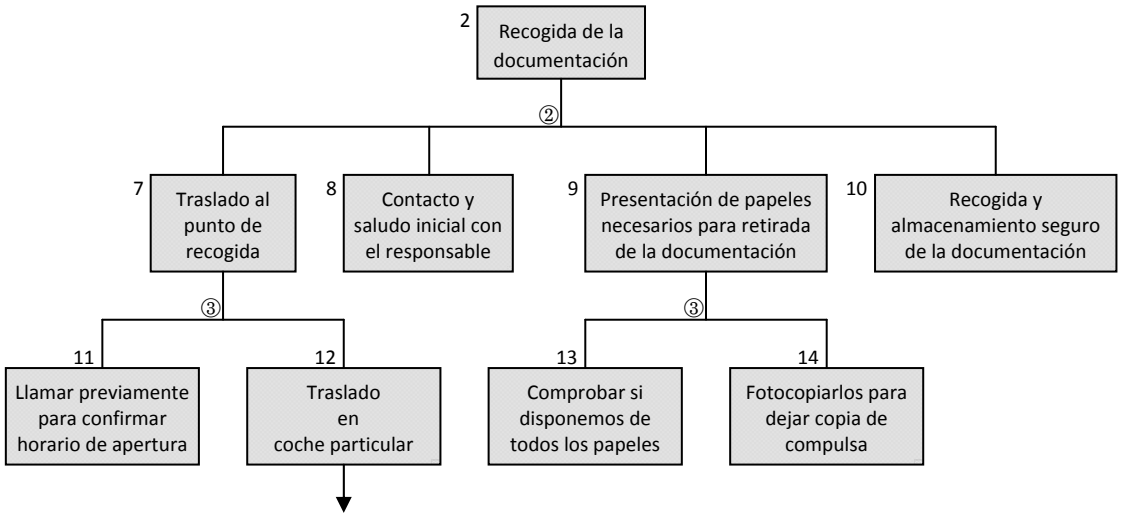
Para problemas largos y complejos el tamaño de un esquema descendente de contenidos se puede disparar, frente a lo cual usaremos herramientas de seguimiento similares a las que usamos en un diagrama de flujo. Supongamos un caso como el siguiente:



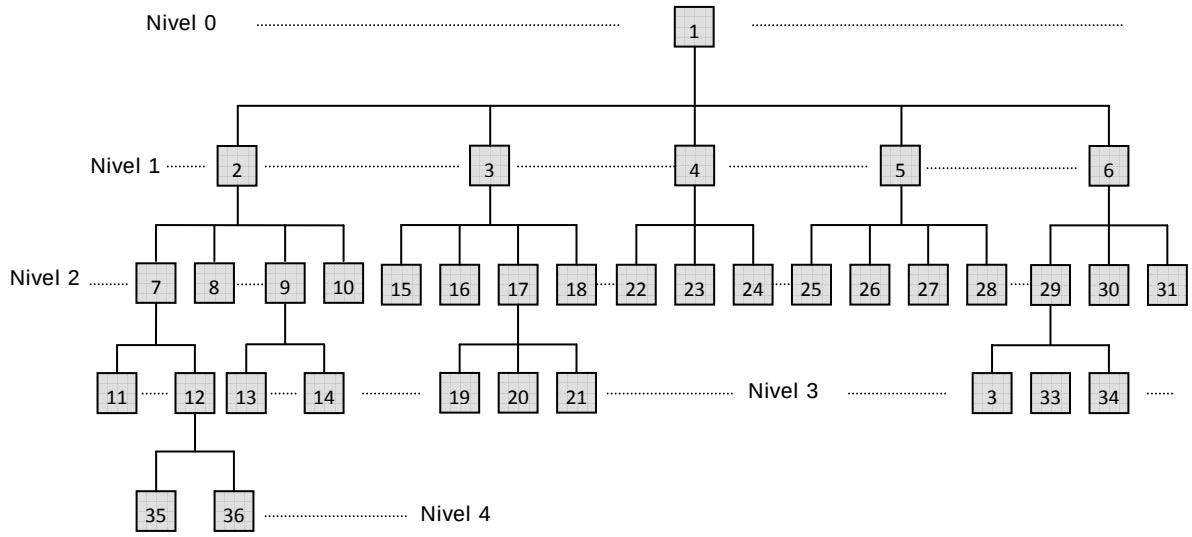
El problema ha sufrido una descomposición inicial y tenemos un esquema descendente de nivel 1, el cual es sintético del contenido del problema (menos sintético que a nivel 0, que sería la síntesis absoluta). Las subdivisiones no se han terminado, lo cual queda indicado con una flecha de continuidad. Se ha decidido no incluirlas por motivos de espacio o presentación.

El siguiente esquema recoge el desarrollo del nodo 2 (nivel 1) hasta el nivel 3.

Desarrollo del nodo 2 a nivel 3



Según este esquema toda la herencia del nodo 2 muere excepto la derivada del nodo 12, que se encuentra en el nivel 3. En otro esquema, supongamos que final para esta rama, tendríamos el desarrollo del nodo 12 a nivel 4, o a nivel 5, o a nivel 16, según la cantidad de ramificaciones que hubiera. Una vez desarrollados todos los nodos hasta un punto de no división, podríamos representar el problema como un árbol. Para este ejemplo el desarrollo final nos lleva a algo así:

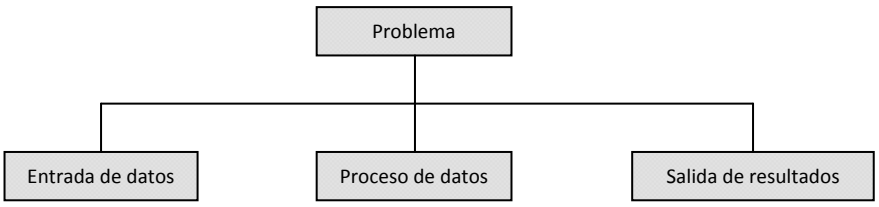


En este esquema la numeración refleja el orden en que se fueron construyendo los desarrollos de nodos. Se puede racionalizar de modo que exista siempre continuidad entre los nodos de un mismo nivel. A cada nodo se le puede asignar una descripción a modo de leyenda así como los datos, comentarios, desarrollos, etc. que se considere oportuno.

La visión en árbol puede ayudar a la planificación del trabajo. Por ejemplo, si tenemos un árbol con tres ramas iniciales de fuerte descomposición podemos pensar en organizar el grupo de programación de forma que existan tres equipos, cada uno atacando una rama. Por ejemplo, para 3 personas, una por rama, o para 6 personas, 2 por rama. También es posible atacar una rama muy larga y subdividida por tres puntos distintos, "olvidando" el resto de ramas.

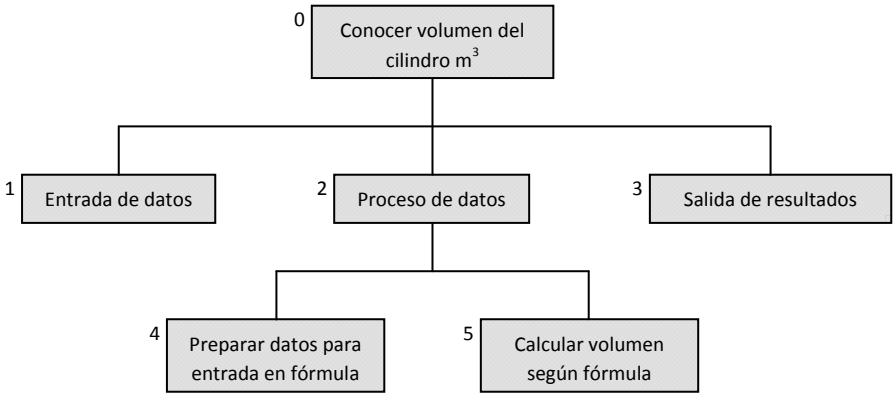
La decisión sobre la estrategia de ataque al árbol recaerá en la figura de una persona que coordina la programación. Normalmente será una persona con experiencia en estas tareas, o en su defecto, la persona con más experiencia del grupo. Si el grupo de programación es de sólo dos personas, una debe asumir el papel de coordinador para lograr desarrollos cohesionados en estilo, formato, estética, etc. Si el grupo es tan grande que no puede ser manejado por un coordinador sólo, éste designará subcoordinadores que le apoyen en su cometido.

Centrándonos en el uso del diseño *top – down* en programación, y hablando en abstracto, un problema suele constar de tres partes típicas:

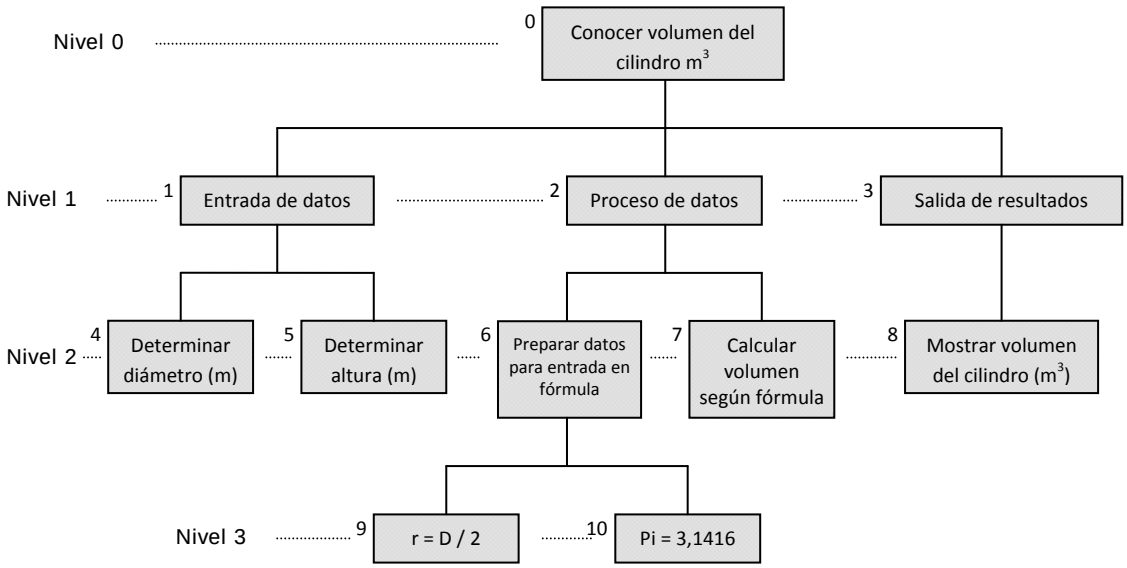


Los nodos del nivel 1 son coincidentes con los módulos que hemos utilizado en alguno de los programas de páginas anteriores. ¿Es una casualidad? Obviamente no; muchos programas responden a este patrón y en el caso de los muy sencillos, ni siquiera es necesario continuar con subdivisiones el

esquema descendente. Veamos la aplicación que tendría para el sencillo algoritmo de obtención del volumen de un cilindro conocidos su altura y su diámetro.



Repetimos que el nivel de desarrollo tiene que ser determinado por el programador. ¿Por qué el nodo 1 no se subdivide? Porque a nuestro criterio, el contenido del nodo 1 lo controlamos mentalmente, lo abarcamos sin problema. No necesitamos "ver más" para programarlo. Si lo desglosamos, estaremos ya prácticamente a nivel de instrucción elemental, y ya hemos dicho que preferimos el uso de los esquemas descendentes para ver la estructura del programa más que llegar a sus detalles. Sin embargo, si se desea, se puede proseguir el desglose. Vamos a hacerlo.

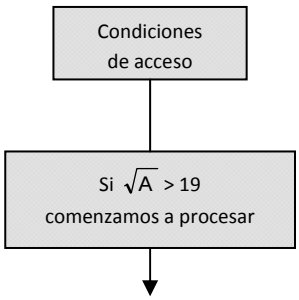


En este ejemplo vemos algunas cosas interesantes. En primer lugar, no se ha buscado tratar de usar un lenguaje directamente enfocado a la programación. En vez de "Determinar diámetro (*m*)", se podía haber escrito *Pedir D*, lo cual es más directo y más "estilo de programador". Sin embargo, no vemos muy razonable buscar un enfoque directo a la programación porque existen herramientas mucho más potentes para ello (pseudocódigo y diagramas de flujo). El objetivo del esquema descendente de contenidos es más el tener clara la estructura del programa, sus partes, para después programarlo, que tener un desarrollo parecido al código del programa. Así pues, dados estos objetivos, entendemos

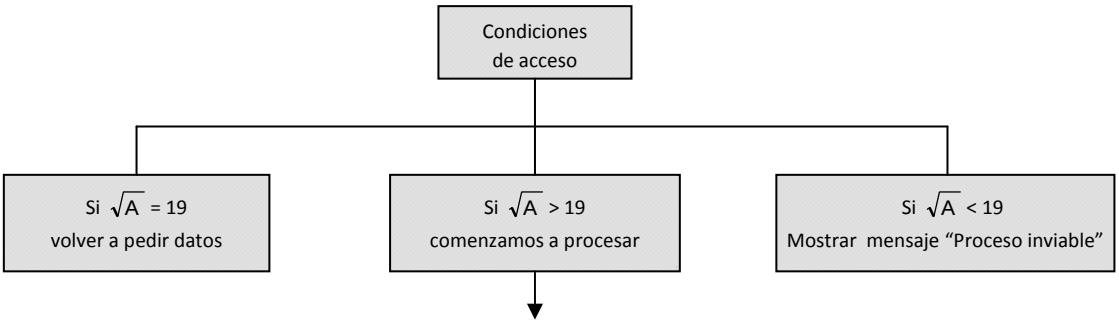
más claro escribir “Determinar diámetro (*m*)”, que nos informa de que hemos de programar la captura de un dato (diámetro) que se espera en metros, que escribir “*Pedir D*” que resulta poco descriptivo.

Otra cuestión que podemos ver es que dentro de los nodos de un nivel, algunos son relativamente abstractos y otros elementales. Por ejemplo, en el nivel 2 el nodo “Preparar datos para entrada en fórmula” goza de cierta abstracción, mientras que el nodo 8 “Mostrar volumen del cilindro” es de carácter elemental. Conclusión: el nivel en que se encuentra un nodo no necesariamente refleja su grado de abstracción, siendo este intrínseco al nodo sin importar su situación. Así, en un problema muy largo podríamos encontrarnos nodos de nivel 1 de carácter elemental y no subdivididos frente a nodos de nivel 24 de carácter fuertemente abstracto.

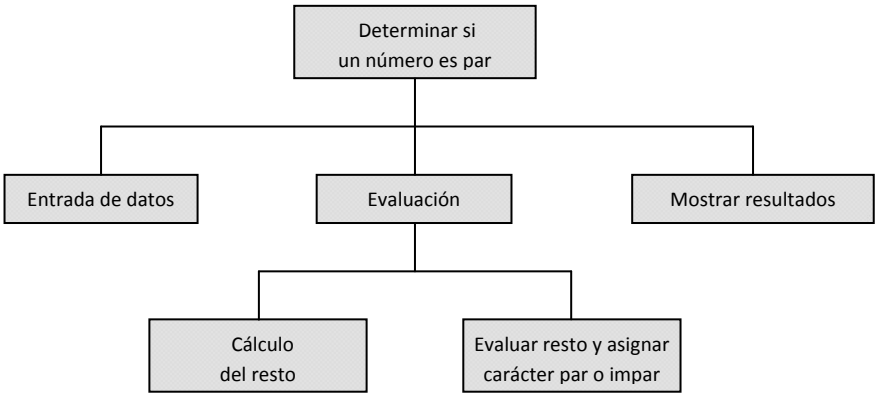
Por último, en relación al esquema anterior, vemos un nodo que da lugar a otro de inferior nivel sin que exista subdivisión. Esto obedece a abstracciones que constan de un único componente elemental o a procesos englobadores que comprenden un único elemento. Normalmente se trata de nodos abstractos cuya descendencia podría ser amplia, pero que por circunstancias específicas del programa engloban a un solo elemento. En este caso, “Salida de resultados” podría constar de distintos componentes, pero por circunstancias específicas del algoritmo sólo tiene una. Consideremos otro ejemplo:



En este ejemplo se supone que por circunstancias específicas del programa se estableció una única vía al entrar en el nodo de condiciones de acceso. Pero podía haber sido del tipo:



Consideremos otro caso, el del problema de determinar si un número es par o impar (cuyo algoritmo vimos al estudiar la instrucción *Si ... Entonces*, aunque recordemos que no habíamos introducido aún el concepto de módulo). El esquema descendente podría ser algo así:



Olvidándonos de que es un ejemplo y suponiendo que fuera un programa con cientos de líneas, aquí tendríamos la estructura del programa. A partir de esta visión de conjunto tendríamos que decidir cómo organizamos el código y los módulos. Un módulo irá usualmente asociado a un nodo que requiere un desarrollo de cierta extensión y complejidad. Vamos a suponer distintas organizaciones para nuestro ejemplo con el supuesto de cientos de líneas.

a) Algoritmo principal engloba a entrada de datos y muestra de resultados y llama a módulo *Evaluacion*. Módulo *Evaluacion* tiene desarrollo propio y llamada a módulos *CalculoResto* y *EvaluarResto*.

b) Algoritmo principal tiene su desarrollo más llamadas a módulos *Entradadedatos, Evaluacion* y *MostrarResultados.*

c) Algoritmo principal tiene su desarrollo y llama a tres módulos (*Entradadedatos, Evaluacion y MostrarResultados*) y a su vez el módulo *Evaluacion* llama a otros dos módulos (*CalculoResto* y *EvaluarResto*).

¿Cuál es la mejor opción? Cada programador es el que decide cómo estructura su programa. Tenemos diversos criterios que nos orientan como:

- Un módulo no debe ser demasiado extenso ni demasiado breve. En líneas generales, debemos buscar construir módulos de extensión media evitando los excesivamente largos o demasiado cortos. No obstante, la correcta organización y estructura del programa es un factor prioritario: no debemos agrupar contenidos inconexos ni dividir un proceso que se entiende mejor agrupado.
- Buscar la independencia del módulo. En general es preferible que cada módulo realice un único proceso claramente identificable.
- Buscar la facilidad de comprensión de la estructura. En ocasiones nos puede interesar crear o eliminar módulos con el único fin de obtener un programa con estructura de fácil comprensión.
- Tener en cuenta diversas orientaciones que establecimos al estudiar los módulos como 1) Evitar autollamadas 2) Uso de módulos genéricos 3) Buscar el bloqueo del producto 4)Otras.

Pero no existe ninguna fórmula del tipo “Todos los nodos del nivel 1 serán módulos del programa”. El programador y su experiencia habrán de decidir.

Próxima entrega: CU00227A

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=36&Itemid=60