aprenderaprogramar.com

Algoritmos genéricos en pseudocódigo. Verificación funcional y verificación total para detectar errores. (CU00233A)

Sección: Cursos

Categoría: Curso Bases de la programación Nivel II

Fecha revisión: 2024

Autor: Mario R. Rancel

Resumen: Entrega nº 32 del Curso Bases de la programación Nivel II

24

ALGORITMOS GENÉRICOS. VERIFICACIÓN FUNCIONAL Y VERIFICACIÓN TOTAL

Cuando en los cursos básicos de fundamentos de programación se habla de "Conocer el problema a resolver" encontramos enunciados como éstos, en relación con el planteamiento de objetivos: "Ya desde este momento conviene empezar a pensar en resolver problemas genéricos o con variables no fijadas". También: "Obtener potencialidad implica que el programa sea capaz de resolver el mayor número de casos y variantes posibles. El programa hay que abrirlo y no cerrarlo". Sobre esta idea se ha estado incidiendo de forma directa o indirecta a lo largo de todo el curso. El concepto de programación genérica se reforzó, una vez más, cuando se introdujo la idea de módulo genérico. El uso de algoritmos genéricos, entendidos éstos como aquellos que admiten un amplio rango de valores o datos de entrada y, en función de éstos, dan lugar a unos resultados, es positivo desde todos los puntos de vista excepto, quizás, el de lograr la verificación del algoritmo.

Hasta ahora hemos estado viendo procesos de verificación basados en unos determinados datos de entrada y unos determinados datos de salida. Considérese ahora que se tenga que probar una situación con infinitas posibilidades de resultados. Esto nos lleva a que no podamos realizar la verificación total del algoritmo, entendida ésta como la prueba de que para cualquiera de los datos de entrada el resultado del algoritmo es siempre correcto. Pero no sólo el caso de posibilidades infinitas, como podría ser el conjunto de números reales o enteros, nos lleva a esta situación. Bastará el hecho de que el número de pruebas sea elevado para que realizar una verificación total sea inviable o inoportuno, ya que incluso utilizando un ordenador para probar el algoritmo el tiempo requerido es excesivo. Por otro lado, la verificación total no es un objetivo que haya que buscar como si del Santo Grial se tratara. Es una posibilidad que en algunas ocasiones nos puede interesar mientras que en otras carece de sentido. ¿Cómo proceder entonces a verificar un algoritmo genérico? En general, realizando cierto número de pruebas para las que los resultados esperables son conocidos (o al menos se conoce el entorno aproximado en que deben estar). El número de pruebas a realizar se establecerá:

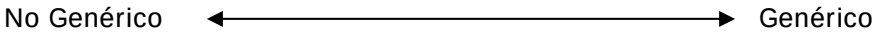
a) En cantidad suficiente para abarcar un rango amplio de valores de entrada, incluyendo valores especiales (como el cero, un máximo, un mínimo, etc.). Obviamente si el programa está enfocado a no procesarse ante ciertos valores de entrada no admisibles, dichos valores no habrá que probarlos porque no tiene sentido.
b) De acuerdo con la experiencia del programador.

Cada prueba se desarrolla con alguna de las técnicas de verificación que hemos visto. La verificación de un algoritmo genérico mediante un número limitado de pruebas la denominamos verificación funcional en contraposición a la verificación total. La verificación funcional no asegura que el algoritmo vaya a funcionar en cualquier circunstancia, pero sí puede determinar que el resultado esperable es correcto con alto grado de probabilidad para un rango de valores dado.

Veamos un ejemplo. En primer lugar definiremos un algoritmo genérico a partir del ejemplo que veníamos usando:

```
1. Desde i = 1 hasta 5 Hacer
        2. Desde j = 1 hasta 5 Hacer
                3. A = i * j
                4. Mostrar A
        5. Siguiente j
6. Siguiente i
```

```
1. Desde i = a hasta b Hacer
        2. Desde j = m hasta n Hacer
                3. A = i * j
                4. Mostrar A
        5. Siguiente j
6. Siguiente i
```



Se supone que todas las variables están definidas como de tipo entero. ¿Cómo enfocar la prueba de este algoritmo? Trataremos de realizar pruebas con datos representativos del conjunto de posibilidades que se pueden presentar. El proceso es similar al que se usa en estrategias de resolución para problemas con resolución intuida pero método paso a paso a determinar. Los "casos" a probar serán los necesarios o los que creamos necesarios buscando representatividad de situaciones, circunstancias especiales conocidas y sencillez. Estos términos debemos ya conocerlos, por lo que omitimos su explicación. Vamos a pasar a realizar una elección de casos para el ejemplo que nos ocupa.

Caso	Valores	Representatividad	Circunstancias especiales	Sencillez
1	a = 1 b = 3 m = 1 n = 3	Enteros positivos, valores pequeños	No	Sí
2	a = 500 b = 502 m = 1200 n = 1202	Enteros positivos, valores grandes	No	Sí
3	a = –2 b = 2 m = –2 n = 2	Paso de negativos a positivos	Interviene el cero	Sí
4	a = –3 b = –1 m = –3 n = –1	Enteros negativos, valores pequeños	No	Sí
5	a = –500 b = –498 m = –1200 n = –1198	Enteros negativos, valores grandes	No	Sí
6	a = 0 b = 0 m = 0 n = 0	Todos ceros	Todos ceros	Sí
7	a = 0 b = 3 m = 0 n = 2	Interviene el cero	Interviene el cero	Sí

¿Son muchos o pocos casos y están bien o mal elegidos? Depende del programador al que se le pregunte. Dados dos programadores enfrentados al mismo problema lo más probable es que:

- El número de casos varíe.
- Los casos para uno representativos pueden no serlo tanto para el otro.
- Si tienen un mínimo de experiencia, llegarán ambos a una verificación igual de buena.

Cada uno de los casos será verificado a través de alguna de las técnicas estudiadas. Si durante las verificaciones se constata algún problema o no hay entera satisfacción, se puede ampliar el número de casos. Veamos lo que serían las verificaciones de los casos planteados con tablas de variables.

CASO 1

Paso bucle 01 (i)	Paso bucle 02 (j)	Valor de A
1	1 2 3	1 2 3
2	1 2 3	2 4 6
3	1 2 3	3 6 9

CASO 2

Paso bucle 01 (i)	Paso bucle 02 (j)	Valor de A
500	1200 1201 1202	600000 600500 601000
501	1200 1201 1202	601200 601701 602202
502	1200 1201 1202	602400 602902 603404

CASO 3

Paso bucle 01 (i)	Paso bucle 02 (j)	Valor de A
–2	–2 –1 0 1 2	4 2 0 – 2 – 4
–1	–2 –1 0 1 2	2 1 0 –1 –2
0	–2 –1 0 1 2	0 0 0 0 0
1	–2 –1 0 1 2	–2 –1 0 1 2
2	–2 –1 0 1 2	–4 –2 0 2 4

CASO 4

Paso bucle 01 (i)	Paso bucle 02 (j)	Valor de A
−3	−3 −2 −1	9 6 3
−2	−3 −2 −1	6 4 2
−1	−3 −2 −1	3 2 1

CASO 5

Paso bucle 01 (i)	Paso bucle 02 (j)	Valor de A
−500	−1200 −1199 −1198	600000 599500 599000
−499	−1200 −1199 −1198	598800 598301 597802
−498	−1200 −1199 −1198	597600 597102 596604

CASO 6

Paso bucle 01 (i)	Paso bucle 02 (j)	Valor de A
0	0	0

CASO 7

Paso bucle 01 (i)	Paso bucle 02 (j)	Valor de A
0	0 1 2	0 0 0
1	0 1 2	0 1 2
2	0 1 2	0 2 4
3	0 1 2	0 3 6

Hay que tener en cuenta que los resultados de las pruebas son interpretables sólo en función de los objetivos planteados para el programa y de la existencia de errores que puedan impedir su ejecución. Hay tres situaciones posibles:

a) El programa se ejecuta y el resultado es conforme.
b) El programa se ejecuta pero el resultado es no conforme.
c) El programa no se ejecuta (da error).

Hablaremos de los tipos de errores y su gestión más adelante. En este momento nos estamos centrando en los posibles fallos en el diseño del algoritmo y no en los que puedan aparecer en conjunto al ejecutar el programa. El diseño es no satisfactorio cuando da lugar a resultados no deseados o cuando da lugar a una imposibilidad de ejecución porque, por ejemplo, produzca indeterminaciones matemáticas, la asignación de un valor a una variable que no coincide con el tipo declarado, etc.

De las verificaciones realizadas sobre el doble anidamiento se desprende que parecen no existir problemas para la ejecución (no hay indeterminaciones matemáticas, todos los números son enteros...) pero ¿El diseño es o no satisfactorio? Para responder a esta pregunta primero tendremos que responder: ¿Cuál era el objetivo del algoritmo? Si la verificación trataba simplemente de comprobar que el programa admite un amplio rango de valores de entrada, sin importar el resultado, podemos decir que el diseño es satisfactorio. Si el objetivo era obtener un resultado cualquiera pero siempre positivo, el caso 3 pone de manifiesto que el diseño del algoritmo no es satisfactorio. Si fuera así habría que proceder al rediseño, bien modificando la estructura actual o bien olvidando ésta y creando una nueva.

Una pregunta que podemos hacernos es: ¿Es posible que se produzcan situaciones $a > b$ o $m > n$? Esto depende del proceso de entrada de datos. Si lo consideramos "aleatorio" (por ejemplo al antojo del usuario) obviamente la situación sí es posible. La consecuencia sería que el bucle no se llega a ejecutar con lo cual *A* no modifica su valor previo ni se muestra valor alguno (ver "Uso y mal uso de la instrucción *Desde ... Siguiente*"). Habrá que responder a la pregunta: ¿Es esto aceptable en nuestro programa? Puede que lo sea o puede que no. Habrá que estudiar las implicaciones que se derivan de ello en un contexto dado.

La representación sería para por ejemplo *a = 3 b = 1 m = 4 n = 2 (caso 8)*

CASO 8

Paso bucle 01 (i)	Paso bucle 02 (j)	Valor de A
—	—	— [No se llega a entrar en el proceso]

Próxima entrega: CU00234A

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