Pruebas unitarias en Java Cubrimiento de código con Codecover y CoView

Un nuevo método merge, que solo contiene un CLI no ejecutable y 4 ítems para las pruebas de ciclos.

En el artículo anterior presentamos la herramienta CoView y descubrimos que el método merge de nuestro ejemplo contiene varios caminos no ejecutables.
Realizando un análisis de dicho método se descubre que al salir del while siempre se hace falsa la condición del arreglo cuyo último elemento es menor al del otro arreglo. Es decir, si el elemento a[length-1] es menor que el elemento b[length-1] entonces se saldrá del while debido a que la condición i<a.length se hará falsa; el otro caso es inverso.
Esto indica que desde el momento que se entra al método se puede conocer cuál es el arreglo que quedará por recorrerse luego de ejecutado el while. En base a este conocimiento desarrollamos un nuevo método merge que se presenta en la Figura 1.

```
public static int[] merge(int[] primero, int[] ultimo) {
    if (primero.length == 0)
        return ultimo;
    if (ultimo.length == 0)
        return primero;
    if (primero[primero.length-1] > ultimo[ultimo.length-1]){
        int[] aux = primero;
        primero = ultimo;
        ultimo = aux;
    }
    int[] c = new int[primero.length + ultimo.length];
    int iUltimo = 0;
    int iPrimero = 0;
    int iC = 0;

    do {
        if (ultimo[iUltimo] < primero[iPrimero]){
            c[iC] = ultimo[iUltimo];
            iC++;
            iUltimo++;
        }
        else {
            c[iC] = primero[iPrimero];
            iC++;
            iPrimero++;
        }
    } while(iPrimero < primero.length);
    do {
        c[iC] = ultimo[iUltimo];
        iC++;
        iUltimo++;
    } while(iUltimo<ultimo.length);
    return c;
}
```

Figura 1 – Nuevo método merge

Casos de prueba

Se ejecutan los mismos cuatro casos de prueba de los artículos anteriores y se observa el cubrimiento de ciclos y de caminos linealmente independientes del nuevo método merge. La Tabla 1 presenta estos cuatro casos de prueba; se presentan los dos array de entrada (primero y ultimo) y el resultado esperado al ejecutar el caso de prueba.

Caso de Prueba	Entrada		Resultado Esperado
	Primero	Ultimo	
CP 1	{1, 2, 3, 4, 5, 6}	{7, 8, 9, 10, 11}	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10,11}
CP 2	{1, 2, 5, 11}	{3, 4, 8, 10}	{1, 2, 3, 4, 5, 8, 10, 11}
CP 3	{}	{8}	{8}
CP 4	{4, 5}	{3}	{3, 4, 5}

Tabla 1 – Los cuatro casos de prueba iniciales

Cubrimiento de ciclos con CodeCover

El método merge contiene dos ciclos. Ambos ciclos son del tipo do-while por lo que cada uno contiene dos ítems candidatos, resultando en cuatro ítems candidatos en total. Los ítems candidatos son las opciones de ejecución de los ciclos. Cada ciclo do-while, durante la corrida de un caso de prueba, puede ejecutarse una única vez o más de una vez (estos son casos interesantes y distintos desde el punto de vista de las pruebas).

En el método merge anterior se tenían nueve ítems candidatos. Por lo que, desde el punto de vista del cubrimiento de ciclos, el método nuevo tiene una menor complejidad.

La Tabla 2 presenta qué ítems cubre cada caso de prueba. El conjunto de casos de prueba compuesto por los casos presentados obtiene un cubrimiento completo de ciclos.

	Primer do-while del método		Segundo do-while del método	
Casos de prueba	Ítem Uno	Ítem Muchos	Ítem Uno	Ítem Muchos
CP 1	X	Cubre	X	Cubre
CP 2	X	Cubre	Cubre	X
CP 3	X	X	X	X
CP 4	Cubre	X	X	Cubre
Total Conjunto Casos de Prueba	**Cubre**	**Cubre**	**Cubre**	**Cubre**

Tabla 2 – Cubrimientos de ciclos con los 4 casos iniciales

Caminos Linealmente Independientes con CoView

La Figura 2 presenta el grafo de flujo de control del método merge y las 7 zonas en las que queda dividido. La cantidad de zonas es el número ciclomático y por lo tanto la cantidad de Caminos Linealmente Independientes (CLI).
La herramienta CoView genera automáticamente un conjunto de CLI. Cada bifurcación del código (if, while, do-while, etc.) puede tomar el valor True o False. Son estas bifurcaciones las que determinan cada camino en el código.

El método merge tiene 6 bifurcaciones; 3 if al comienzo, un primer do-while, un if dentro del primer do-while y otro do-while al final. Los caminos los representaremos con los valores de verdad que toman esas bifurcaciones al ejecutarse el mismo.
Los caminos determinados por la herramienta son los siguientes:

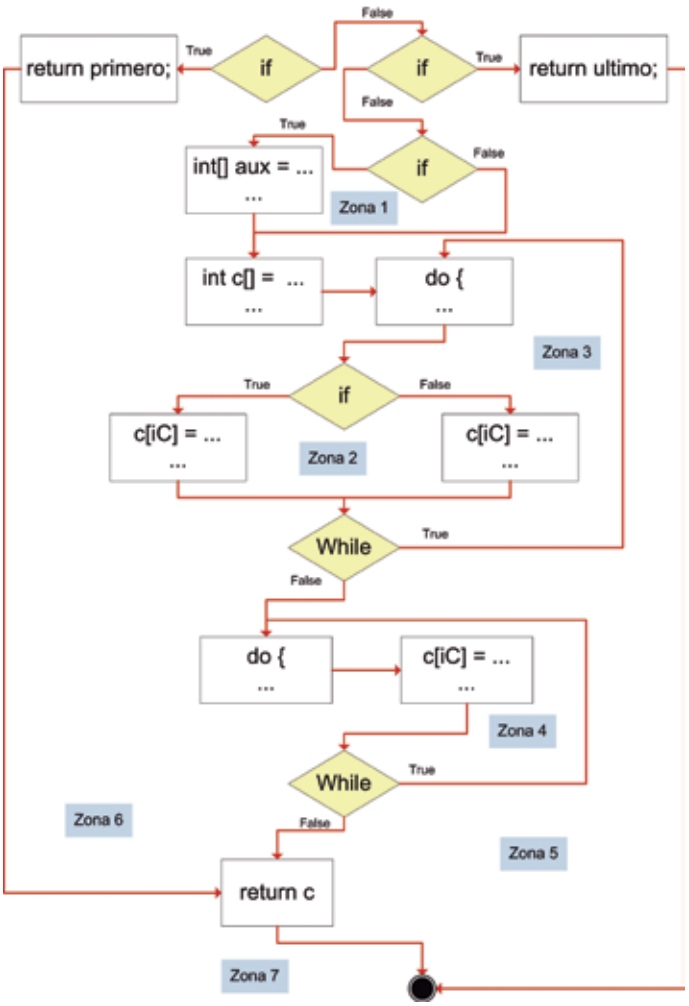

Figura 2 – Grafo de flujo de control del método merge

T_, FT_, FFTTTT, FFFTTT, FFTFTT, FFTTFT, FFTTTF. Los dos primeros caminos tienen truncadas las bifurcaciones debido a que en esos casos el método se abandona sin ejecutar el resto de las bifurcaciones. Estos dos casos se dan en los dos primeros if.

Como ejemplo se presenta en la Figura 1 el camino FFTTTT. La herramienta muestra con verde las bifurcaciones que toman el valor true, con rojo las que toman el valor false y con gris las sentencias que se ejecutan en el camino.
La forma que acabamos de presentar para identificar un camino es en realidad una simplificación. En el ejemplo FFTTTT se ejecutan dos veces los cuerpos de ambos do-while. En la segunda ejecución del primer do-while, el if interno se evalúa en false. Esto lo sabemos mirando la Figura 1 ya que las líneas de código del else están pintadas con gris pero la condición del if interno al do-while está pintada de verde.
También lo sabemos porque CoView presenta de otra forma el camino; que en este caso lo determina de forma exacta y unívoca. A continuación se presenta con esta nueva notación el mismo camino que la Figura 1 (FFTTTT):

Formato: [número línea código][decisión]:[evalúa a]

[2] primero.length==0 : FALSE
[4] ultimo.length==0 : FALSE
[6] primero[primero.length-1]>ultimo[ultimo.length-1] : TRUE
[17] ultimo[iUltimo]<primero[iPrimero] : TRUE
[26] iPrimero<primero.length : TRUE
[17] ultimo[iUltimo]<primero[iPrimero] : FALSE
[26] iPrimero<primero.length : FALSE
[31] iUltimo<ultimo.length : TRUE
[31] iUltimo<ultimo.length : FALSE

Con esta notación queda totalmente determinado el camino a recorrer. En el ejemplo, se muestra que la línea de código17 (el if dentro del while) una vez se evalúa en true y la segunda vez que se ejecuta se evalúa en false (como habíamos determinado anteriormente analizando la Figura 1).

Cubriendo los CLI con casos de prueba

Al ejecutar nuestro conjunto de 4 casos de prueba solamente el camino T_ es cubierto. Este camino se cubre con el caso de prueba número 3.

Entonces, para lograr un 100% de cubrimiento de CLI, se debe realizar un análisis de cada uno de los 6 caminos que no fueron cubiertos y generar un caso de prueba para cada uno de esos caminos.
Por ejemplo, para el camino FT_ debemos generar un caso en el cual el array de nombre primero no debe ser vacío y el array de nombre ultimo sí debe ser vacío. El resultado esperado de este caso debe ser igual al array primero.
Realizando este tipo de análisis para cada uno de los caminos que faltan cubrir se generan los 6 casos de prueba necesarios para alcanzar un 100% de cubrimiento de CLI. Estos se presentan en la Tabla 3.

Caso de Prueba	Entrada		Resultado Esperado
	Primero	Ultimo	
FT_	{1, 2, 3}	{}	{1, 2, 3}
FFTTTT	{1, 3, 4}	{2}	{1, 2, 3, 4}
FFFTTT	{2}	{1, 3, 4}	{1, 2, 3, 4}
FFTFTT	{3, 4}	{1, 2}	{1, 2, 3, 4}
FFTTFT	El camino determinado no es ejecutable		
FFTTTF	{1, 3}	{2}	{1, 2, 3}

Tabla 3 – Casos de prueba para cubrir los CLI

El camino FFTTFT no es ejecutable. Dejamos al lector que identifique el motivo y que encuentre una forma de cambiar el método merge para que permita tener todos sus caminos ejecutables (si es posible).
La herramienta CoView permite seleccionar caminos a ser excluidos del criterio. Por ejemplo, en este caso es razonable excluir del criterio el camino que no es ejecutable. De esta manera, con los casos de prueba presentados, se cubre el 100% de los CLI que son ejecutables.

En este artículo presentamos un nuevo método merge, que solamente contiene un CLI no ejecutable y 4 ítems para las pruebas de ciclos. Desde el punto de vista de las pruebas de CLI y de ciclos este método es mejor que el anterior.
También presentamos un conjunto de pruebas que logra cubrir al 100% tanto el criterio de ciclos con la herramienta CodeCover como el criterio de CLI (considerando solamente los caminos ejecutables) con la herramienta CoView.

Diego Vallespir
Fernando Marotta
Carmen Bogado

Grupo de Ingeniería de Software, UdelaR
gris@fing.edu.uy