ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

"¿TU SOFTWARE ES CORRECTO? PRUEBALO"

AUTORIA
INMACULADA VILLÉN ALTAMIRANO

TEMÁTICA
¿TU SOFTWARE ES CORRECTO? PRUEBALO

ETAPA
ESO, BACHILLERATO, CICLO DE GRADO MEDIO Y SUPERIOR

Resumen

En un proyecto software pequeño, mediano o grande, no se puede escuchar o leer las especificaciones, a continuación empezar a programar, hacerlo rápidamente (1 o 2 días) y el cliente ya lo tiene funcionando. Esto es lo que piensa mucha gente, pero el problema es que los informáticos no tienen una varita mágica para hacer los programas y que estos funcionen, sin ningún error y a la primera. La ingeniería del software se encarga de esta labor: seguir unos pasos y el último, antes de entregar al cliente, realizar las pruebas del software

Palabras clave

Ingeniería del software, pruebas de la caja blanca y de la caja negra, depuración, grafo, codificación.

1. INTRODUCCIÓN.

El proceso de ingeniería del software se puede ver como una espiral donde se dan cada una de las fases necesarias para el desarrollo del software. Inicialmente, la ingeniería del sistema define el papel del software y conduce al análisis de los requisitos del software, donde se establece el campo de la información, la función, el comportamiento, el rendimiento, las restricciones y los criterios de validación del software. Al movernos hacia el interior de la espiral, llegamos al diseño y, por último, a la codificación. Para desarrollar software, damos vueltas en espiral a través de una serie de flujos o líneas que disminuyen el nivel de abstracción en cada vuelta.

También podemos imaginar una estrategia para la prueba del software si nos movemos hacia fuera de la espiral. La "prueba unidad" comienza en el vértice de la espiral y se centra en cada unidad o módulo. La prueba avanza, al movernos hacia afuera de la espiral, hasta llegar a la "prueba de integración", donde el foco de atención es el diseño y la construcción de la arquitectura del software. Dando otra vuelta por la espiral hacia afuera, encontramos la "prueba de validación", donde se validan los requisitos establecidos como parte del análisis de requisitos del software, comparándolos con el sistema que ha sido construido. Finalmente, llegamos a la "prueba del sistema" en la que se prueban como un todo el software y otros elementos del sistema. Por tanto, como se puede observar conforme más hacia afuera vamos mayor es el alcance de cada una de las pruebas.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

2. FUNDAMENTOS DE LA PRUEBA DEL SOFTWARE.

La prueba del software es un elemento crítico para la garantía de la calidad del software que se pretende desarrollar. Las pruebas representan el último repaso tanto de las especificaciones, como del diseño y como de la codificación.

Las pruebas se realizan después de obtener el código pero su diseño puede venir planificado desde las primeras etapas de desarrollo de dicho software.

La prueba es un proceso que se enfoca sobre la lógica interna del software, y sobre las funciones externas, es decir la prueba realizará un repaso por todas y cada una de las partes del software.

2.1. DEFINICIONES Y OBJETIVOS:

- La prueba es un proceso de ejecución de un programa con la intención de descubrir un error. Es por tanto un proceso destructivo donde lo que buscamos es encontrar los puntos débiles del software con el fin de observar las partes que no funcionan correctamente.
- Un buen caso de prueba es aquél que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces.

2.2 OBJETIVO.

Como objetivo general de fase de pruebas del software se considera el diseño de pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de esfuerzo y tiempo posible.

Un objetivo secundario, aunque igual de importante, es que las pruebas demuestren hasta que punto las funciones del software desarrollado están de acuerdo con las especificaciones realizadas anteriormente y hasta que punto se cumplen los requerimientos de rendimiento.

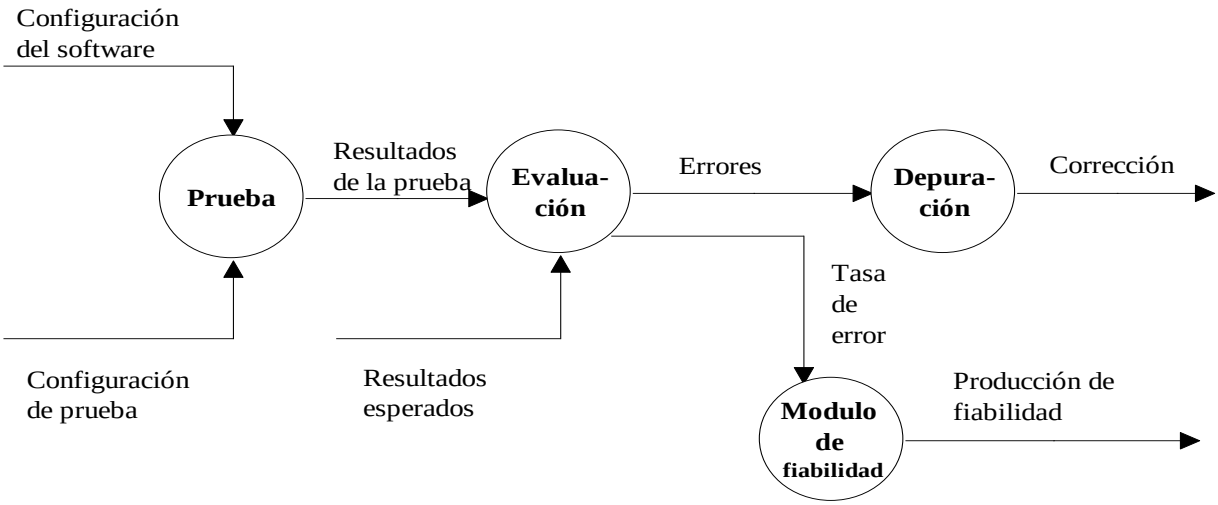
2.3. LIMITACIÓN.

Sin embargo, las pruebas del software no pueden asegurar la ausencia de defectos en el producto final. Mediante las pruebas nunca se puede asegurar que se vaya a llegar a un sistema libre o ausente de errores. Únicamente se puede demostrar que existen defectos en el software, no que no existen.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

3. PROCESO DE PRUEBA.

El flujo de información para el proceso de prueba sigue el siguiente esquema:



A medida que se van recopilando y evaluando los resultados de la prueba puede ocurrir:

- Se encuentran con regularidad errores serios.
- El funcionamiento del software parece ser correcto.
- La prueba no descubre errores.

4. DISEÑO DE CASOS DE PRUEBA.

El diseño de casos de prueba no es una tarea simple ya que un buen diseño de casos de prueba puede exigir tanto esfuerzo casi como el diseño, aunque en la mayoría de los casos este esfuerzo no se le dedique.

Cualquier sistema software puede ser probado de las siguientes formas:

- conociendo la función específica para lo que fue diseñado el producto, se pueden realizar pruebas que demuestren que cada función es completamente operativa. Estas pruebas se llaman ***pruebas de la caja negra***, sólo se considera la entrada y la salida.

- conociendo el funcionamiento del producto, se pueden desarrollar pruebas que aseguren que las operaciones internas se ajustan a las especificaciones y que todos los componentes internos se han comprobado de forma adecuada. Estas pruebas se llaman ***pruebas de la caja blanca***.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

5. PRUEBAS DE LA CAJA BLANCA.

Son, en general, métodos de diseño de casos de prueba que utilizan la estructura de control del diseño procedimental para derivar casos de prueba.

Permiten examinar la estructura interna del programa diseñando casos de prueba que examinen la lógica del programa.

Con este tipo de pruebas se puede obtener:

- casos de prueba que garanticen que se ejecuta por lo menos una vez todos los caminos independientes de cada módulo.
- casos de prueba que garantizan que se ejercitan todas las decisiones lógicas en sus caras verdadero (V) y falso (F).
- casos de prueba que garantizan que se ejecutan todos los bucles en sus límites y con sus límites operacionales.
- casos de prueba que garanticen que se ejecuta la estructura de datos internas para asegurar su validez.

Existen diversos tipos de prueba de caja blanca. En esta ocasión se examinarán dos tipos de los más importantes:

- Prueba del camino básico.
- Prueba de bucles.

5.1. PRUEBA DEL CAMINO BASICO.

Este método fue diseñado por McCabe el cual:

- Permite al diseñador de casos de prueba obtener una medida de la complejidad lógica del diseño procedimental.
- Utilizar la medida anterior como guía para definir un conjunto básico de caminos de ejecución que recorrerán cada arista del programa al menos un vez.
- Para ejecutar cada camino se fija un caso de prueba.

Los casos de prueba derivados con el método de McCabe garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

5.1.1. GRAFO DE FLUJO O GRAFO DEL PROGRAMA.

McCabe usa una notación para indicar el flujo de control dentro del procedimiento que se considere.

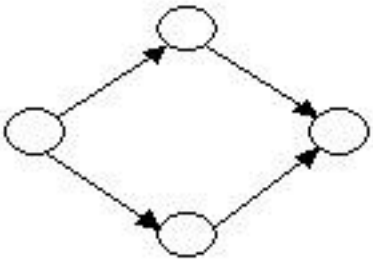
Existen tres tipos de elementos:

- Nodos
- Aristas.
- Regiones.

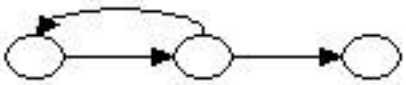
Las construcciones estructuradas se representan de la siguiente forma:

Secuencia

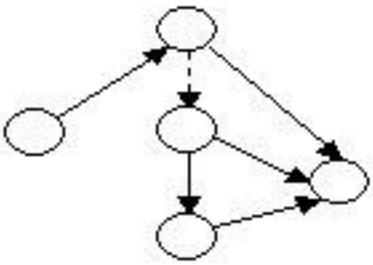
If – Then - Else



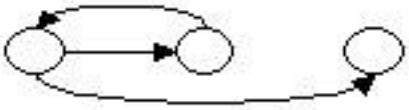
Until



Case



While



Cada círculo es un **nodo** y puede representar una o varias sentencias. Las flechas se llaman **aristas** y representan el flujo de control. Siempre una arista debe acabar en un nodo.

Grafo de flujo se denomina al grafo que se obtiene al representar las sentencias de un programa o módulo mediante la notación indicada.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

5.1.2. COMPLEJIDAD CICLOMÁTICA.

La complejidad ciclomática de un grafo es una medida que nos proporciona la información necesaria para conocer cuantos casos de prueba serán necesarios diseñar para que queden comprobados todos y cada uno de los flujos de control y sentencias del programa.

Por tanto esta medida proporciona el número de caminos independientes en el grafo de flujo asociado al programa, lo cual nos indica el límite superior para el número de casos de prueba que hay que realizar para asegurar que se ejecute al menos una vez cada sentencia, considerando como camino independiente cualquier camino del procedimiento o programa que introduce por lo menos un nuevo conjunto de sentencias de procesamiento o una nueva condición.

Un camino independiente al resto de caminos ya probados es el que se mueve por una arista del grafo de flujo que no haya sido recorrida por los caminos anteriores.

Por tanto, la complejidad ciclomática se puede definir o calcular de la siguiente manera:

1) El número de regiones del grafo de flujo.
2) La complejidad ciclomática V(G) de un grafo de flujo G se define como:

$$V(G) = E – N + 2$$

donde E es el número de aristas del grafo de flujo y N es el número de nodos del grafo.

3) La complejidad ciclomática V(G), de un grafo de flujo G también se define como.

$$V(G) = P + 1$$

donde P es el número de nodos predicados contenidos en el grafo de flujo G. Un nodo predicado es aquel que posee una condición y se caracteriza por que de él salen dos o más aristas.

5.1.2.1. EJEMPLO.

Veamos un ejemplo donde se calcula la complejidad ciclomática asociada al siguiente procedimiento, en el cual se realiza la búsqueda de un elemento en dos vectores con el fin de indicar si existe ese elemento en cualquiera de los dos vectores. El mensaje de elemento encontrado se visualizará tantas veces como ocurrencias se encuentren. Además en el vector segundo se deberá inicializar todos sus elementos:

```
procedimiento buscar_elemento (vector1 ,vector2, elemento, longitud)
inicio
 contador = 1
 mientras (contador <= longitud)
hacer
 mostrar_pantalla(“Procesando posición: ”,  contador)
```

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

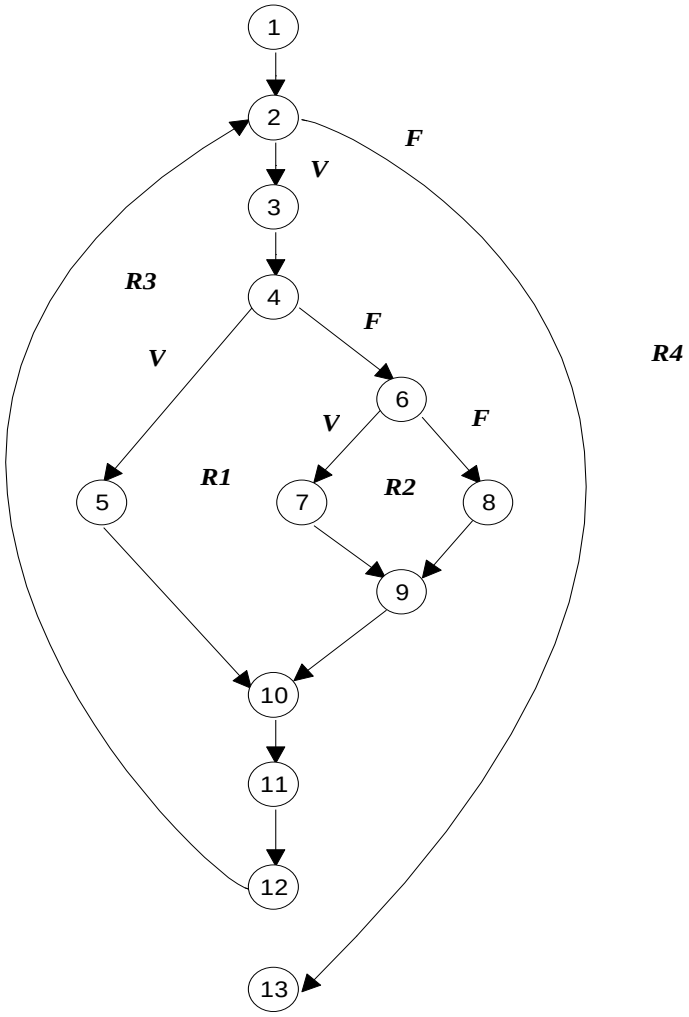
```
        si vector1[contador] = elemento
        entonces
                mostrar_pantalla(“Coincidencia Encontrada posición:”, contador)
        sino
                si vector2[contador] = elemento
                entonces
                        mostrar_pantalla(“Coincidencia Encontrada posición:”,
                                                                contador)
                sino
                   vector2[contador] = 0
                finsi
        finsi
        contador = contador + 1
  finmientras
fin
```

El primer paso es dibujar el grafo de flujo del procedimiento. Para ello primero etiquetamos dentro del código los nodos que se encuentran en el procedimiento, Cada nodo está formado por una o varias sentencias que determinan un mismo flujo de control..

```
    procedimiento buscar_elemento (vector1 ,vector2, elemento, longitud)
    inicio
1:  contador = 1
2:  mientras (contador <= longitud)
3:  hacer
    mostrar_pantalla(“Procesando posición: ”,  contador)
4:          si vector1[contador] = elemento
5:          entonces
                mostrar_pantalla(“Coincidencia Encontrada posición:”, contador)
6:          sino   si vector2[contador] = elemento
7:                 entonces
                        mostrar_pantalla(“Coincidencia Encontrada posición:”,
                                                                contador)
8:                 sino
                      vector2[contador] = 0
9:                 finsi
10:         finsi
11:         contador = contador + 1
12: finmientras
13:   fin
```

Haciendo uso de la notación indicada y de los nodos que se han establecido podemos observar el siguiente grafo de flujo.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008



El cálculo de la complejidad ciclomática se puede llevar a cabo a través de las tres formas indicadas anteriormente. Veamos como se calcularía por medio de cada una de las expresiones:

A- El número de regiones del grafo de flujo.

Como se indica en el grafo de flujo el número de regiones cerradas son tres. Además se suma una región adicional que se corresponde con una región genérica donde se incluye todo el grafo de flujo.

V(G) = 4.

B- La complejidad ciclomática V(G) de un grafo de flujo G se define como:

V(G) = E – N + 2

donde E es el número de aristas del grafo de flujo y N es el número de nodos del grafo.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

En nuestro caso tenemos que:

Número de aristas: E = 15
Número de nodos: N = 13

V(G) = 15 – 13 + 2 = 4

1) La complejidad ciclomática V(G), de un grafo de flujo G también se define como.

V(G) = P + 1

donde P es el número de nodos predicados contenidos en el grafo de flujo G. Un nodo predicado es aquel que posee una condición y se caracteriza por que de él salen dos o más aristas.

En nuestro caso el número de nodos predicados es el siguiente:

Número de nodos predicado: P = 3

V(G) = 3 +1 = 4

Como se puede observar, sea cual sea el método elegido para el cálculo de la complejidad ciclomática el resultado obtenido es siempre el mismo.

El hecho de que la complejidad ciclomática V(G) sea 4 indica que con cuatro casos de prueba que recorran los cuatro caminos linealmente independientes se podrá comprobar toda la lógica interna del procedimiento.

Los caminos independientes que se obtengan a partir del grafo no tienen por que ser siempre los mismos, lo que sí es necesario es que en cada camino se incluya una nueva arista no visitada hasta entonces.

Un conjunto de caminos independientes sería:

Camino 1: 1-2-13
Aristas Nuevas Recorridas: 1-2, 2-13
Camino 2: 1-2-3-4-5-10-11-12-2-13
Aristas Nuevas Recorridas: 1-2, 2-3, 3-4, 4-5, 5-10, 10-11, 11-12, 12-2
Camino 3: 1-2-3-4-6-7-9-10-11-12-2-13
Aristas Nuevas Recorridas: 4-6, 6-7, 7-9, 9-10
Camino 4: 1-2-3-4-6-8-9-10-11-12-2-13
Aristas Nuevas Recorridas: 6-8, 8-9.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

Se puede observar que cada camino introduce nuevas aristas. El camino:

1-2-3-4-5-10-11-12-2-3-4-6-7-9-10-11-12-2-13

Aunque es un camino totalmente diferente a los indicados anteriormente no se considera camino independiente, ya que simplemente es una combinación de caminos ya especificados, y no recorre ninguna nueva arista.

5.1.3. PASOS PARA EL DISEÑO DE CASOS DE PRUEBA.

Dado un procedimiento o programa los pasos a seguir para definir los casos de prueba asociados al mismo son los siguientes:

1. Se parte del diseño o del código.
2. Se dibuja el correspondiente grafo de flujo partiendo de dicho diseño o código por medio de las notaciones anteriores.
3. Determinar la complejidad ciclomática del grafo de flujo resultante.
4. Fijar un conjunto básico de caminos linealmente independientes.
5. Determinar los casos de prueba que permitan la ejecución de cada camino del conjunto anterior.
6. Ejecutar cada caso de prueba y comparar con los resultados esperados.

5.1.4. Ejemplo de prueba del camino básico.

Veamos un ejemplo que calcula la media de hasta 100 números que se encuentren entre límites. También calcula el total de entradas y el total de válidos.

```
Procedimiento media (valor, mínimo, máximo)
inicio
    i = 1
    total_entradas = total_validos = 0
    suma = 0
    mientras valor[i] ≠ -999  Y  total_entrada < 100
hacer
    total_entrada = total_entrada + 1
    si  valor[i] >= minimo  Y  valor[i] <= maximo
    entonces
            total_valido = total_valido + 1
            suma = suma + valor[i]
    finsi
    i = i + 1
finmientras
si   total_valido > 0
entonces    media = suma / total_valido
sino   media = -999
```

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

```
finsi
fin
```

PASO 1: Se parte del código descrito anteriormente.

PASO 2:

Para ello etiquetamos cada bloque de sentencias secuencial con el mismo flujo de control y cada condición.

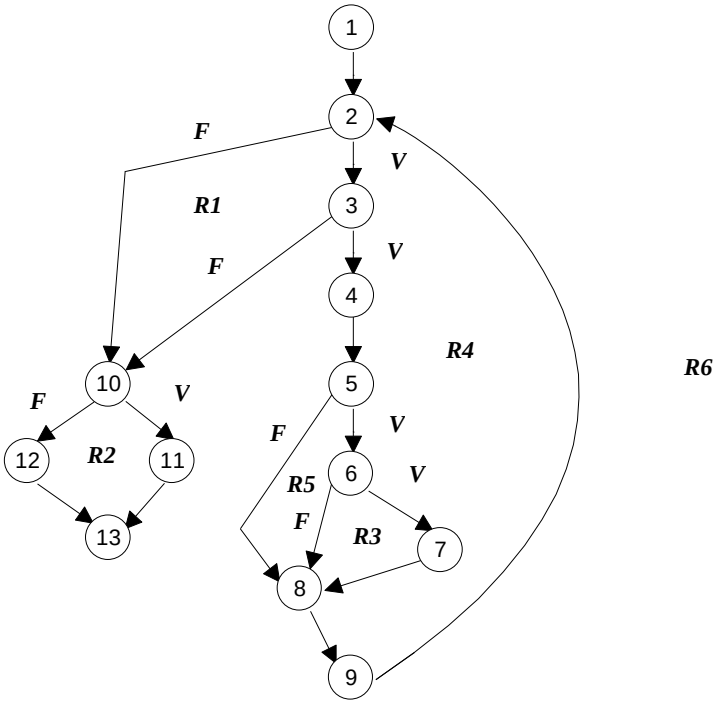
```
Procedimiento media (valor, mínimo, máximo)
Inicio
1:          i = 1
      total_entradas = total_validos = 0
            suma = 0
                              2:                  3:
      mientras valor[i] ≠ -999  Y  total_entrada < 100
         hacer
4:                  total_entrada = total_entrada + 1

                                  5:                  6:
            si  valor[i] >= minimo  Y  valor[i] <= maximo
            entonces
7:                          total_valido = total_valido + 1
                  suma = suma + valor[i]
8:                finsi
            i = i + 1
9:          finmientras

                              10:
         si  total_valido > 0
11:         entonces   media = suma / total_valido
12:         sino   media = -999
13:         finsi
fin
```

A continuación dibujamos el grafo de flujo, partiendo del código del procedimiento:

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008



PASO 3:

Una vez representado el grafo de flujo se obtiene el valor de la complejidad ciclomática asociada a dicho grafo por cualquiera de las expresiones que se han indicado.

V(G) = 6 regiones.
V(G) = 17 aristas – 13 nodos + 2 = 6
V(G) = nodos predicado + 1 = 5 + 1 = 6

PASO 4:

A continuación se fija un conjunto básico de caminos linealmente independientes conociendo que sólo se necesitarán 6 caminos linealmente independientes para asegurarnos que se pasa por todos los caminos del grafo y con ello por todas las instrucciones del código.

Un ejemplo de caminos linealmente independientes es el siguiente:

Camino 1: 1-2-10-11-13
Camino 2: 1-2-10-12-13
Camino 3: 1-2-3-10-11-13
Camino 4: 1-2-3-4-5-8-9-2- …. -> cualquier camino elegido ya es válido como por ejemplo el 2-10-11-13
Camino 5: 1-2-3-4-5-6-8-9-2- … -> “”

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

Camino 6: 1-2-3-4-5-6-7-8-9-2 … -> “”

PASO 5:

El siguiente paso sería sacar los distintos casos de prueba para cada uno de los caminos.

Caso de prueba del camino 1:

Camino 1: 1-2-10-11-13

Para el camino 1, se coge un valor de k que tenga una entrada válida donde k < i, y también se tiene en cuenta que el valor de (i) = -999. Para que se de este camino se tienen que dar estos valores, lo cual sólo se puede hacer si este camino está incluido en otro camino mayor que permita hacer total_valido > 0, ya que en caso contrario no podría ocurrir. Los resultados esperados son una media correcta y entradas válidas.

- Valor[k] = entrada válida, con k<i, definida abajo.
- Valor[i] = -999, donde 2 <= i <= 100
- Resultados esperados:

Medida correcta sobre los n valores y totales adecuados

- Nota: no se puede probar por sí sola: debe ser probada como parte de los caminos 4,5 y 6.

Caso de prueba del camino 2:

Camino 2: 1-2-10-12-13

Para el camino 2 es igual que el anterior, pero en este caso existe un valor, el primero, igual a -999, es decir, no se entra en el bucle ***mientras***. Se recorre el grafo por el nodo 12 que indica que la media es nula. Este camino se puede recorrer de forma independiente, no hace falta recorrer los otros caminos.

- valor[1] = -999
- Resultados esperados:

Media = -999; otros totales con sus valores iniciales.

Caso de prueba del camino 3

Camino 3: 1-2-3-10-11-13

Para probar el camino 3 vemos que no se cumple la segunda condición del ***mientras***. Se probará intentando procesar un número de entradas mayores de 100. Los primeros 100 valores deben de ser válidos y los resultados esperados son los mismos que en el camino 1, donde se ejecuta ***mientras*** para 100 entradas y después sale de él. El camino 3 también puede ser 1-2-3-10-12-13, pero sólo uno de los dos.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

- Intento de proceder 101 o más valores.
- Los primeros 100 valores deben ser válidos.
- Resultados Esperados:

Medida correcta sobre los n valores y totales adecuados

Caso de prueba del camino 4:

Camino 4: 1-2-3-4-5-8-9-2- …. -> cualquier camino elegido ya es válido como por ejemplo el 2-10-11-13

Para el camino 4, en el ***si*** dentro del ***mientras*** existe una salida. Valor[i] es menor que mínimo. El diseño de este caso de prueba tiene como objetivo comprobar un valor de entrada menor que el mínimo establecido. Resultado: media correcta entre límites mínimo y máximo. Sale del ***si*** cuando valor[k] < mínimo.

- valor[i] = entrada valida con i < 100
- valor[k] < minimo para k<i
- Resultados Esperados:

Media correcta sobre los valores y totales adecuados.

Caso de prueba del camino 5:

Camino 5: 1-2-3-4-5-6-8-9-2- …

Para el camino 5, hay que comprobar un valor de k mayor que el valor de máximo establecido. Resultado: media correcta entre mínimo y máximo y número de entradas.

- valor[i] = entrada valida con i <100
- valor[k] > maximo para k<=i
- Resultados Esperados:

Media correcta sobre los n valores y totales adecuados.

Caso de prueba del camino 6:

Camino 6: 1-2-3-4-5-6-7-8-9-2 …

Para el camino 6 existen n valores que cumplen las condiciones del procedimiento. Se obtienen medias y entradas adecuadas. Es el caso más general.

- valor[i] = entrada válida con i < 100
- Resultados Esperados:

Media correcta sobre los n valores y totales adecuados.

ISSN 1988-6047 DEP. LEGAL: GR 2922/2007 Nº 9 – AGOSTO DE 2008

Autoría

Inmaculada Villén Altamirano
Córdoba
· E-MAIL: inma_villen@yahoo.es