**PRUEBAS DE CAJA BLANCA**

**PARTE 1. DESCRIPCIÓN DE TÉCNICAS**

Se examinan los caminos lógicos del sistema haciendo diversos recorridos. Sus objetivos son:

* Ejecutar por lo menos una vez todos los caminos independientes de cada módulo.
* Ejecutar todas las decisiones lógicas en sus valores verdadero y falso.
* Ejecutar todos los bucles en sus límites.
* Ejercitar todas las estructuras de datos internas para comprobar su validez.

**Justificación de esfuerzos**

* Los errores lógicos y las suposiciones incorrectas son inversamente proporcionales a la probabilidad de que se ejecute un camino del programa.
* A menudo se piensa que un camino lógico tiene pocas posibilidades de ejecutarse cuando, de hecho, se puede ejecutar de forma regular.
* Los errores mecanográficos son aleatorios.
* Los errores se esconden en los rincones y se aglomeran en los límites.

**1. Prueba del Camino Básico.** Permite obtener una medida de la complejidad lógica del diseño procedimental y usarla para definir un conjunto básico de caminos de ejecución que garanticen que cada sentencia del programa se recorre al menos una vez.

Sirve para representar el flujo de control lógico. Se basa en la representación del código a probar a través de círculos y arcos que indican el flujo de control. Cada círculo representa una o más sentencias. Las estructuras en forma de grafo de flujo son:

**Secuencia IF While Do-While**

**Case**

Los nodos representan condiciones o secuencias de instrucciones.

Las aristas representan flujo de control del programa.

Cada nodo que contiene una condición se denomina nodo predicado y se caracteriza porque de él salen dos o más aristas. Una de las características para comprobar si un grafo de flujo está bien diseñado es que los únicos nodos de los que pueden partir dos aristas son los nodos predicados.

Si analizamos el siguiente flujograma:

**1**

**2**

**3**

**4**

**5**

**10**

**9**

**8**

**7**

**6**

**1**

**1**

Tendríamos el siguiente grafo:

**R1**

**R4**

**Región**

**11**

**2**

**1**

**3**

**4**

**5**

**R**

**2**

**R**

**3**

**6**

**9**

**8**

**7**

**10**

Nodo Predicado

Nodo

Predicado

**1.1. Complejidad Ciclomática.** Define el número de caminos independientes del conjunto básico de un módulo y por tanto nos proporciona una cota superior del número máximo de pruebas que se han de realizar para garantizar que cada sentencia se ejecuta al menos una vez.

Un camino independiente es cualquier camino del programa que incorpora frente a otros caminos un nuevo conjunto de sentencias o una nueva condición, es decir, ha de recorrer al menos una arista no recorrida hasta entonces.

Un conjunto de caminos componen un conjunto básico de caminos si al recorrerlos se recorre al menos una vez cada sentencia del programa y cada condición por tanto se habrá ejecutado en su parte verdadera y falsa. Un conjunto básico de caminos no es único.

***El número de caminos que componen un conjunto básico de caminos viene determinado por la complejidad ciclomática***, que se calcula como:

C(G) = Número de regiones del grafo.

C(G) = (Número de aristas – Número de nodos) + 2 = (A – N) + 2

C(G) = Número de nodos predicado + 1 = P + 1

Complejidad ciclomática del grafo de ejemplo:

C(G) = Existen cuatro regiones = 4

C(G) = (13 aristas - 11 nodos) + 2 = 4

C(G) = 3 nodos predicado + 1 = 4 (Los nodos predicado son el nodo 2, el nodo 4 y el nodo 6).

En el ejemplo anterior, los caminos básicos son:

Camino 1: 1 – 2 - 11

Camino 2: 1 – 2 – 3 – 4 – 5 – 10 – 2 - 11

Camino 3: 1 – 2 – 3 – 4 – 6 – 7 – 9 – 10 – 2 - 11 Camino 4: 1 – 2 – 3 – 4 – 6 – 8 – 9 – 10 – 2 - 11

**1.2. Generación de casos de prueba.**

1. Diseñar el grafo de flujo.
2. Calcular la complejidad ciclomática del grafo obtenido.
3. Determinar un conjunto básico de caminos linealmente independientes.
4. Diseñar los casos de prueba que permitan la ejecución de cada camino básico.
5. Ejecutar cada paso de prueba y comparar con los resultados esperados.

**1.3. Matriz del Grafo.** El mecanismo de prueba del camino básico es susceptible de ser automatizado. A tal efecto se utiliza una representación del grafo de flujo consistente en una matriz de grafo. Es una matriz cuadrada cuya longitud es el número de nodos del grafo de flujo. Cada fila y columna se corresponden a un nodo específico y el contenido de la matriz son las aristas del grafo, estas pueden llevar un peso de enlace que da información sobre el flujo de control. Puede ser 0 ó 1, indicando si existe conexión entre los nodos o se le puede asociar a cada arista propiedades más interesantes como:

* Probabilidad de que la arista se recorra.
* Tiempo invertido en recorrer la arista.
* Recursos requeridos durante el recorrido (memoria,...).

Ejemplo: Si la matriz del grafo contiene 0 ó 1 indicando si existe o no conexión entre los nodos, a la matriz resultante se la denomina matriz de conexiones.

A partir de la matriz se puede calcular la complejidad ciclomática y el conjunto básico de caminos.

Del ejemplo que venimos trabajando:

**Conectado**

**al Nodo**

**Nodo**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | | **1** |  | 1 |  |  |  |  |  |  |  |  |  | | **2** |  |  | 1 |  |  |  |  |  |  |  | 1 | | **3** |  |  |  | 1 |  |  |  |  |  |  |  | | **4** |  |  |  |  | 1 | 1 |  |  |  |  |  | | **5** |  |  |  |  |  |  |  |  |  | 1 |  | | **6** |  |  |  |  |  |  | 1 | 1 |  |  |  | | **7** |  |  |  |  |  |  |  |  | 1 |  |  | | **8** |  |  |  |  |  |  |  |  | 1 |  |  | | **9** |  |  |  |  |  |  |  |  |  | 1 |  | | **10** |  | 1 |  |  |  |  |  |  |  |  |  | | **11** |  |  |  |  |  |  |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | **conex** |  |  |  |  | | 1 | - | 1 | = | 0 | | 2 | - | 1 | = | **1** | | 1 | - | 1 | = | 0 | | 2 | - | 1 | = | **1** | | 1 | - | 1 | = | 0 | | 2 | - | 1 | = | **1** | | 1 | - | 1 | = | 0 | | 1 | - | 1 | = | 0 | | 1 | - | 1 | = | 0 | | 1 | - | 1 | = | 0 | |  |  |  |  |  | |  |  |  | **T** | **3** | |

Complejidad ciclomática = **T + 1** = 3 + 1 = 4

**2. Prueba de bucles.**

Los bucles junto a las decisiones son fundamentales en el desarrollo algorítmico, por lo que es importante el aplicar técnicas que verifiquen la corrección de los mismos. La prueba de bucles es una técnica de prueba de Caja Blanca que se centra en la validez de las construcciones de bucles.

Se pueden definir cuatro clases:

BUCLES SIMPLES: Se les debe aplicar las siguientes pruebas :

1. Saltar el bucle.
2. Pasar una vez.
3. Pasar dos veces.
4. Pasar j veces con j<n, siendo n el número máximo de pasos permitidos.
5. Pasar n-1, n, n+1 veces por el bucle.

BUCLES ANIDADOS: Si intentamos generalizar el proceso de prueba de bucles simples a un conjunto de bucles anidados, el número de posibles pruebas podría llegar a ser impracticable.

Existe una metodología para aplicar eficientemente esta prueba :

1. Comenzar con el bucle más interior. El resto se dejan en valores mínimos.

1. Se aplican las pruebas de bucles simples al bucle más interior, mientras se mantienen los bucles exteriores en sus valores mínimos. Añadir otras pruebas para los valores fuera de rango o para valores excluidos.

1. Progresar hacia fuera, llevando a cabo las pruebas para el siguiente bucle manteniendo los bucles exteriores en sus valores mínimos y los interiores en sus valores típicos.

1. Repetir el proceso hasta que todos los bucles hayan sido probados.

BUCLES CONCATENADOS: Se pueden probar igual que los bucles simples si son independientes entre sí y siguiendo la técnica de prueba de bucles anidados si existe cualquier relación entre los diferentes bucles concatenados. Por ejemplo, si hay dos bucles concatenados y se usa el contador del bucle 1 como valor inicial del bucle 2, entonces los bucles no son independientes; luego en este caso, se aplicará el enfoque para bucles anidados.

BUCLES NO ESTRUCTURADOS: Este tipo de bucles son construcciones no deseables y lo que se debe de intentar en la medida de lo posible es el rediseñarlos para que se ajusten a las reglas de la programación estructurada.

**PARTE 2. ALGUNOS EJEMPLOS.**

1. Subprograma que calcula la diferencia entre 2 valores horarios

|  |
| --- |
| **void diferencia\_horaria (int h1, int m1, int h2, int m2, int \*dh, int \*dm)**  **{**  **if (h1 <= h2)**  **{**  **if(m1 <= m2)**  **{**  **\*dh = h2 - h1;**  **\*dm = m2 - m1;**  **} else**  **{**  **\*dh = (h2 - h1) - 1;**  **\*dm = (60+m2) - m1;**  **} } else {**  **if (m1 <= m2)**  **{**  **\*dh = (h2 + 24) - h1;**  **\*dm = m2 - m1;**  **} else**  **{**  **\*dh = (h2 + 23) - h1;**  **\*dm = (60+m2) - m1;**  **}**    **}**  **cout<<"Diferencia : "<< \*dh <<" Horas , "<< \*dm <<" Minutos"; }** |

**11**

**12**

**3**

**SÍ**

**NO**

**4**

**5**

**6**

**8**

**9**

**10**

**7**

**NO**

**SÍ**

**S**

**Í**

**NO**

**2**

**1**

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **Paso 1:** Dibujar el grafo (se pueden apoyar en el diagrama de flujo o hacerlo directamente) | | |  | | --- | | **Paso 2:** Cálculo de la complejidad ciclomática    C(G) = Cantidad de regiones = 4  C(G) = (A – N) + 2 = (14 – 12) + 2 = 4  C(G) = Cantidad nodos predicado + 1  Nodos predicado (NP) son: 2 , 3 y 7  C(G) Total = 4 | |
| **R1**    **R**  **2**    **R**  **3**    **1**    **2**    **3**    **5**    **4**    **6**    **7**    **9**    **8**    **10**    **11**    **12**    **R**  **4**    **[1]**    **[**  **2**  **]**    **[**  **3**  **]** |
| |  | | --- | | **Paso 3:** Determinar el Conjunto de Caminos  Básicos Linealmente Independientes    C1 : 1 – 2 – 3 – 4 – 6 – 11 - 12  C2 : 1 – 2 – 3 – 5 – 6 – 11 - 12  C3 : 1 – 2 – 7 – 8 – 10 – 11 - 12 C4 : 1 – 2 – 7 – 9 – 10 – 11 - 12 | |

**Paso 4:** Diseñar los casos de prueba

|  |  |  |  |
| --- | --- | --- | --- |
| Camino | Valores de Entrada | Valor Esperado | Valor obtenido |
| 1 | Hora Inicial: 10:35 Hora Final: 14:50 | Diferencia: 4 horas 15 minutos |  |
| 2 | Hora Inicial: 12:55 Hora Final: 13:40 | Diferencia: 0 horas 45 minutos |  |
| 3 | Hora Inicial: 22:10 Hora Final: 00:50 | Diferencia: 2 horas 40 minutos |  |
| 4 | Hora Inicial: 23:50 Hora Final: 00:21 | Diferencia: 0 horas 31 minutos |  |

**Paso 5:** Ejecutar la prueba

1. Subprograma que genera un vector que contiene valores **NO múltiplos** del vector recibido

|  |
| --- |
| **void NO\_multiplos(int \*V, int n, int val, int \*Z, int \*tz)**  **{ int i, D; i = 0; while( i<n )**  **{**  **D = (V[i]/val)\*val;**  **if(V[i] != D)**  **{**  **Z[\*tz]=V[i]; //insertamos el elemento en la nueva posicion**  **\*tz = \*tz + 1; //abrimos un nuevo espacio**  **} i++;**  **}**  **}** |

**SÍ**

**3**

**2**

**1**

**7**

**5**

**6**

**4**

**NO**

**SÍ**

**NO**

**8**

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **Paso 1:** Dibujar el grafo (se pueden apoyar en el diagrama de flujo o hacerlo directamente) | | |  | | --- | | **Paso 2:** Cálculo de la complejidad ciclomática    C(G) = Cantidad de regiones = 3  C(G) = (A – N) + 2 = (9 – 8) + 2 = 3  C(G) = Cantidad nodos predicado + 1  Nodos predicado (NP) son: 2 y 4  C(G) Total = 3 | |
| **R**  **1**    **1**    **2**    **3**    **R**  **2**    **[**  **2**  **]**    **5**    **6**    **4**    **7**    **R**  **3**    **[1]**    **8** |
| |  | | --- | | **Paso 3:** Determinar el Conjunto de Caminos  Básicos Linealmente Independientes    C1 : 1 – 2 – 8  C2 : 1 – 2 – 3 – 4 - 5 – 6 – 7 - 2 – 8  C3 : 1 – 2 – 3 – 4 – 6 – 7 - 2 – 8 | |

**Paso 4:** Diseñar los casos de prueba

|  |  |  |  |
| --- | --- | --- | --- |
| Camino | Valores de Entrada | Valor Esperado | Valor obtenido |
| 1 | Vector de entrada vacío | Vector de salida vacío |  |
| 2 | Vector con NO múltiplos  unicamente    Valor = 5    {1, 2, 3, 28, 4, 6, 7, 8, 9,  11 } | Vector de salida IGUAL al  vector de entrada        {1, 2, 3, 28, 4, 6, 7, 8, 9,  11 } |  |
| 3 | Vector con múltiplos  unicamente    Valor = 5    {15, 25, 30, 40, 55, 60, 75,  85, 95, 105} | Vector de salida vacío |  |

**Paso 5:** Ejecutar la prueba

1. Programa que invierte un número entero desde cero hasta 999999999999999999

|  |
| --- |
| **main() { long long N, M, NI; long long D, R; char resp='s';**    **cout<<"\nPrograma que invierte un numero. Desea iniciar? (S/N): "; cin>>resp; getchar(); resp=toupper(resp);**    **while(resp!='N')**  **{**  **cout<<"Digite el Numero : "; cin>>N;**   1. **= N;**   **NI = 0;**    **if( (N>=0) && (N <= 999999999999999999 ) )**  **{**  **while(N!=0)**  **{**  **D = N/10;**  **R = N-(D\*10);**  **NI = (NI\*10) + R;**   1. **= D;**   **}**  **cout<<"El numero original es : "<<M<<endl; cout<<"El numero invertido es : "<<NI<<endl;**  **} else**  **cout<<"NO VALIDO. Fuera de rango"<<endl;**    **cout<<"\nDesea ingresar otro numero? (S/N): "; cin>>resp; getchar(); resp=toupper(resp); }**  **cout<<"\nPROGRAMA FINALIZADO";**    **}** |

**4**

**NO**

**SÍ**

**9**

**8**

**5**

**6**

**SÍ**

**NO**

**7**

**10**

**1**

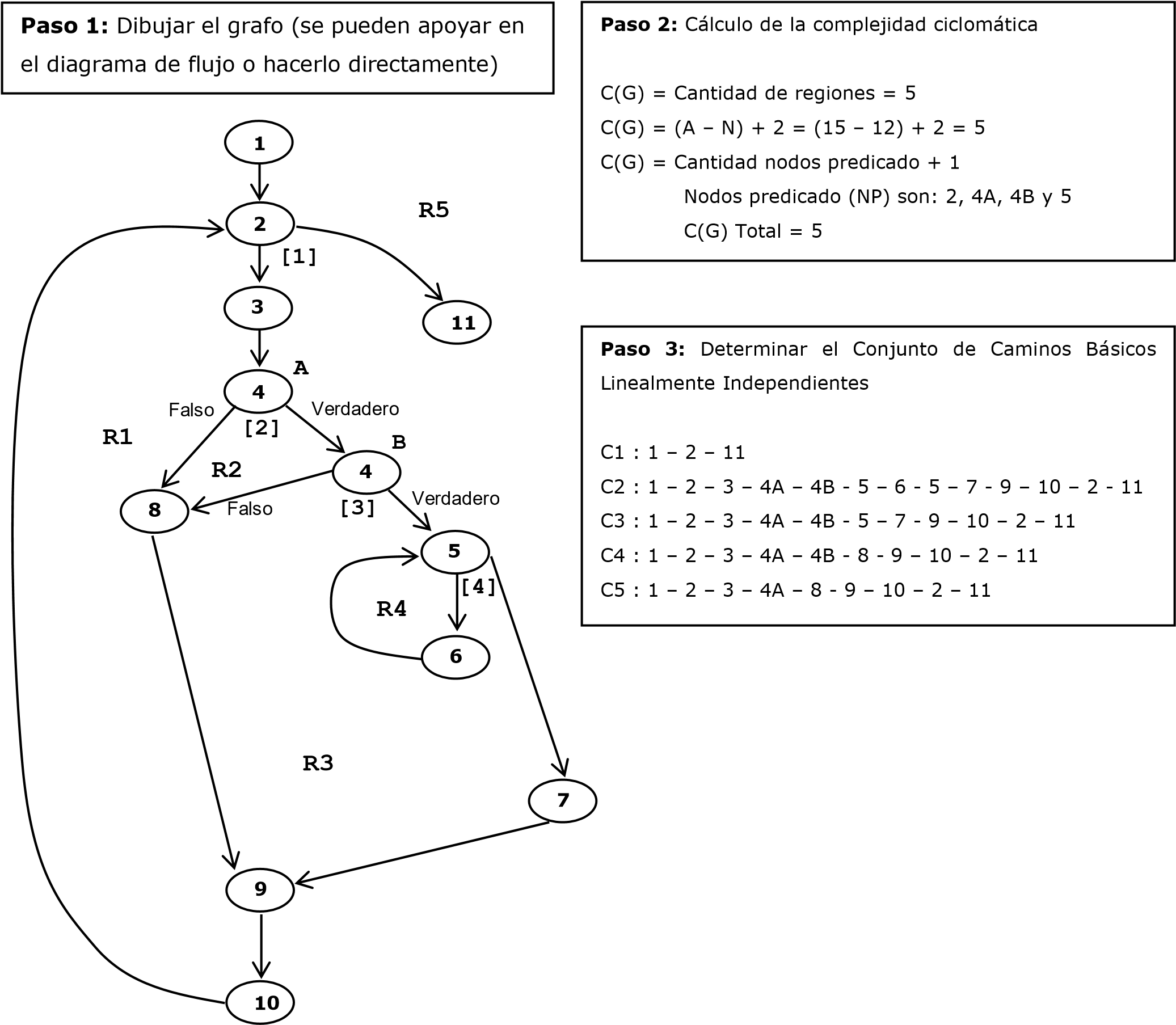
**SÍ**

**3**

**2**

**NO**

**11**



**Paso 4:** Diseñar los casos de prueba

|  |  |  |  |
| --- | --- | --- | --- |
| Camino | Valores de Entrada | Valor Esperado | Valor obtenido |
| 1 | Respuesta = ‘N’ | Programa finalizado sin efectuar proceso |  |
| 2 | Respuesta = ‘S’ Y Número en rango | Número invertido Y  Pregunta al usuario si desea otro número |  |
| 3 | Respuesta = ‘S’ Y Número = 0 | Número invertido = 0 Y  Pregunta al usuario si desea otro número |  |
| 4 | Respuesta = ‘S’ Y Número mayor al límite superior del rango | “Valor no válido. Fuera de rango” Y  Pregunta al usuario si desea otro número |  |
| 5 | Respuesta = ‘S’ Y Número menor al límite inferior del rango | “Valor no válido. Fuera de rango” Y  Pregunta al usuario si desea otro número |  |

**Paso 5:** Ejecutar la prueba

-------------------------------------------------------------------------------------------------------------------------- Thomas McCabe, establece en sus trabajos los siguientes valores de referencia:

<= 10, métodos sencillos, sin mucho riesgo.

> 10, <= 20, métodos medianamente complejos, con riesgo moderado.

> 20, <= 50, métodos complejos, con alto riesgo.

> 50, métodos inestables, de altísimo riesgo.

------------------------------------------------ FIN DEL DOCUMENTO