TEMA 4.- Prueba de Aplicaciones Convencionales (3/4)

Calidad del Software

Dr. José Luis Abellán Miguel

Grado en Ingeniería Informática

Dr. José Luis Abellán Miguel - Tlf: (+34) 968 278 657
Universidad Católica San Antonio de Murcia www.ucam.edu

Índice

- Introducción
- Prueba de Caja Blanca
- Prueba de Caja Negra

Bibliografía

- Pressman, R. ***Ingeniería del Software: Un enfoque práctico***. 7ª edición. Madrid: McGraw Hill, 2010. ISBN: 9701054733 (disponible en la biblioteca UCAM) → **Capítulo 18**

Introducción

- Una vez generado el código fuente, el software debe probarse para descubrir (y corregir) tantos errores como sea posible antes de entregarlo al cliente.
  - La meta es diseñar una serie de **casos de prueba** que tengan una alta probabilidad de encontrar errores
- El software se prueba desde dos perspectivas diferentes:
  1. La lógica de programa interno se revisa usando técnicas de diseño de **casos de prueba de "caja blanca**"
  2. Los requerimientos de software se revisan usando técnicas de diseño de **casos de prueba de "caja negra**"
- La intención es encontrar el máximo número de errores con la mínima cantidad de esfuerzo y tiempo

Fundamentos de las Pruebas (1/3)

- ***Comprobabilidad del software:*** es saber con cuánta facilidad puede probarse un programa de cómputo
- Características que conducen a software comprobable:
  - ***Operatividad***. "Mientras mejor funcione mejor puede probarse."
    - Si un sistema se diseña e implementa teniendo como objetivo la calidad, relativamente pocos errores bloquearán la ejecución de las pruebas
  - ***Observabilidad***. "Lo que ve es lo que prueba."
    - Las entradas proporcionadas como parte de las pruebas producen distintas salidas. Los estados del sistema y las variables son visibles o consultables durante la ejecución. La salida incorrecta se identifica con facilidad.
  - ***Controlabilidad***. "Mientras mejor pueda controlar el software, más podrá automatizar y optimizar las pruebas."
    - Todo código es ejecutable a través de alguna combinación de entradas. El ingeniero de pruebas puede controlar directamente los estados del software, del hardware y las variables.

Fundamentos de las Pruebas (2/3)

- ***Descomponibilidad***. "Al controlar el ámbito de las pruebas, es posible aislar más rápidamente los problemas y realizar pruebas nuevas y más inteligentes."
  - El sistema de software se construye a partir de módulos independientes que pueden probarse de manera independiente.
- ***Simplicidad***. "Mientras haya menos que probar, más rápidamente se le puede probar."
  - El programa debe mostrar *simplicidad funcional*; *simplicidad estructural* (arquitectura es modular para limitar la propagación de fallos) y *simplicidad de código* (se adopta un estándar de codificación para facilitar la inspección y el mantenimiento).
- ***Estabilidad***. "Mientras menos cambios, menos perturbaciones para probar."
  - Los cambios al software son raros, se controlan cuando ocurren y no invalidan las pruebas existentes. El software se recupera bien de los fallos.
- ***Comprensibilidad***. "Mientras más información se tenga, se probará con más inteligencia."
  - El diseño arquitectónico y las dependencias entre componentes internos, externos y compartidos son bien comprendidos. La documentación técnica es accesible al instante, está bien organizada, es específica, detallada y precisa.

Fundamentos de las Pruebas (3/3)

❑ Atributos de una buena prueba

- *Una buena prueba tiene una alta probabilidad de encontrar un error*
  - Se debe comprender el software e intentar saber cómo puede fallar. De manera ideal, se prueban las clases de defectos.

    Una clase de defectos potenciales en una interfaz gráfica de usuario es el defecto para reconocer la posición adecuada del ratón. Se diseña entonces un conjunto de pruebas para revisar el ratón
- *Una buena prueba no es redundante*
  - El tiempo y los recursos de la prueba son limitados. No se trata de realizar una prueba que tenga el mismo propósito que otra. Cada una debe tener un propósito diferente
- *Una buena prueba debe ser "la mejor de la camada"*
  - Usar la prueba que tenga la mayor probabilidad de descubrir una clase de errores.
- *Una buena prueba no debe ser demasiado simple o demasiado compleja*
  - Aunque en ocasiones es posible combinar una serie de pruebas en un caso de prueba, los efectos colaterales posibles asociados con este enfoque pueden enmascarar errores. En general, cada prueba debe ejecutarse por separado.

Prueba de Caja Blanca (1/2)

- Filosofía de diseño de casos de prueba que usa la estructura de control descrita como parte del diseño a nivel de componentes para derivar casos de prueba.
- Al usar los métodos de prueba de caja blanca:
  - Garantizar que todas las rutas independientes dentro de un módulo se revisaron al menos una vez
  - Revisar todas las decisiones lógicas en sus lados verdadero y falso
  - Ejecutar todos los bucles en sus fronteras y dentro de sus fronteras operativas
  - Revisar estructuras de datos internas para garantizar su validez.

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (1/6)

- Debe introducirse una notación simple para la representación del flujo de control

Los constructos estructurados en gráfico de flujo forman:

Donde cada círculo representa una o más PDL no ramificadas o enunciados en código fuente

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (2/6)

- Ejemplo

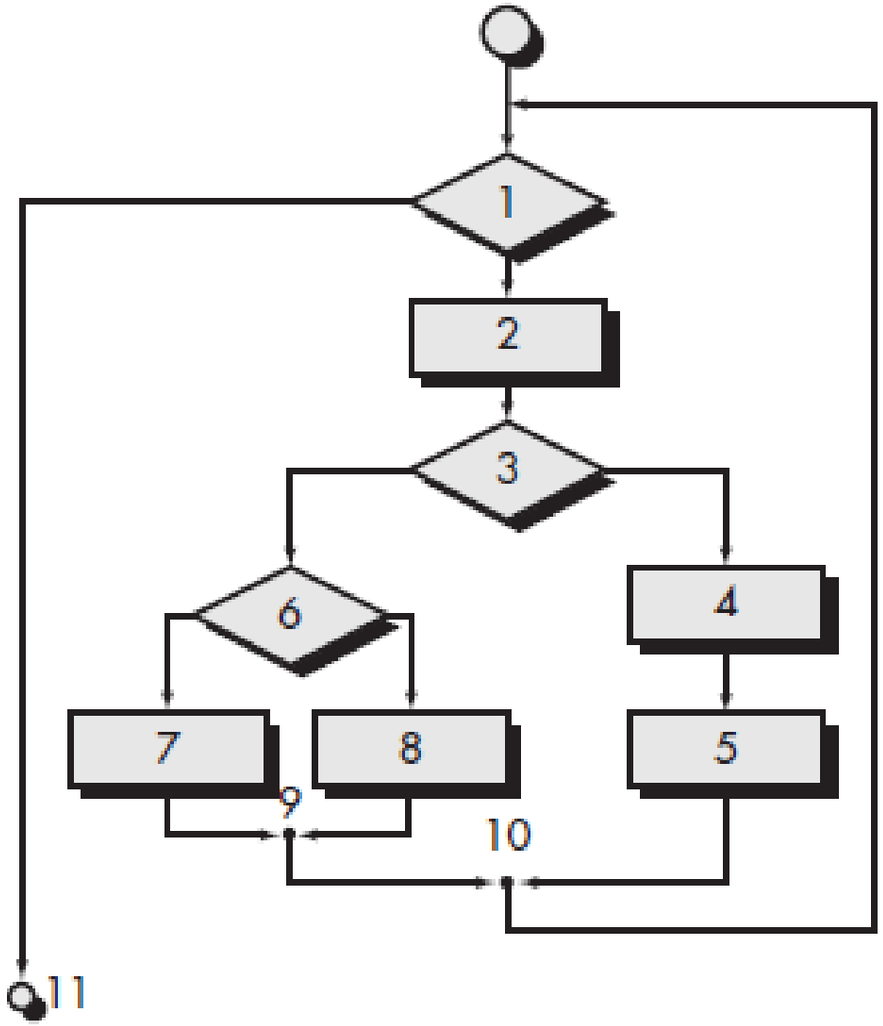


Diagrama de Flujo:
Estructura de control del programa

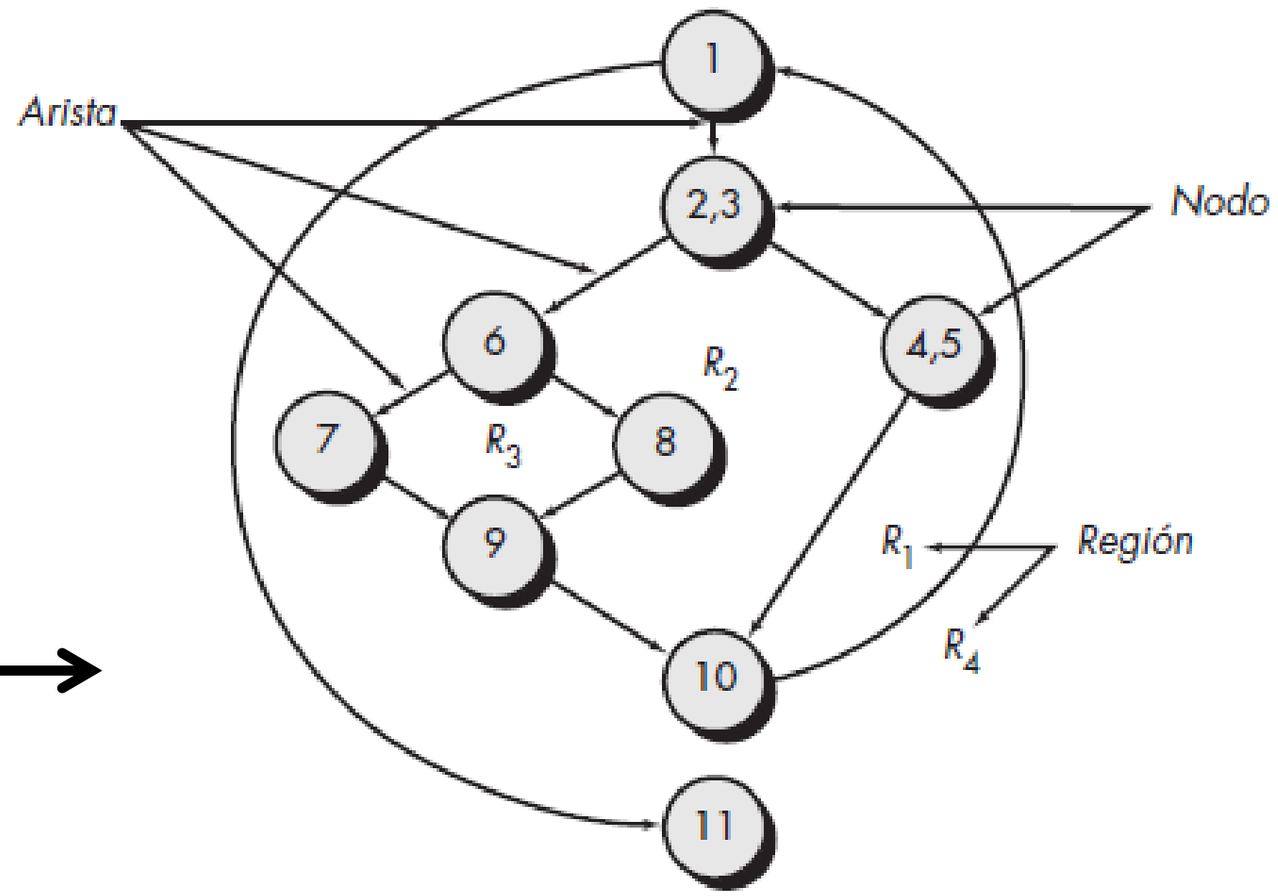


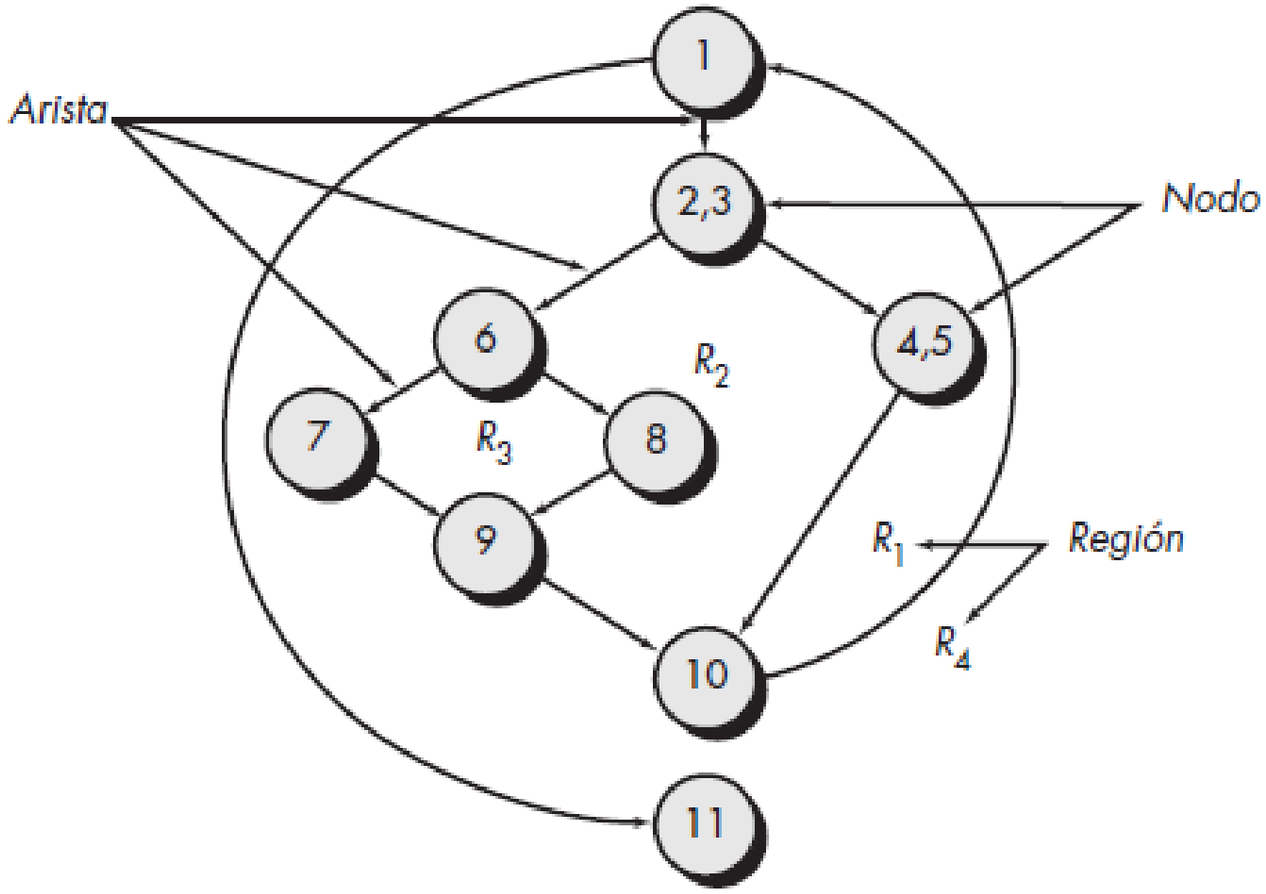
Gráfico de Flujo

Nodo: uno o más enunciados de procedimiento
Arista: flujo de control
Región: área cortada por arista

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (3/6)

- Rutas de programa independientes
  - Una ruta independiente es cualquiera que introduce al menos un nuevo conjunto de enunciados de procedimiento o una nueva condición en el programa
    - Al menos una arista que no se haya recorrido antes de definir la ruta



ruta 1: 1-11

ruta 2: 1-2-3-4-5-10-1-11

ruta 3: 1-2-3-6-8-9-10-1-11

ruta 4: 1-2-3-6-7-9-10-1-11

Conjunto básico: todo enunciado en el programa tendrá garantizada su ejecución al menos una vez, y cada condición se ejecutará en sus lados verdadero y falso

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (4/6)

- ¿Cómo saber cuántas rutas buscar?
  - Cálculo de la complejidad ciclomática.
- **La complejidad ciclomática**: cota superior sobre el número de casos de prueba que se requerirán para garantizar que cada enunciado en el programa se ejecuta al menos una vez
- **Tres maneras de calcular la complejidad ciclomática**:

1) El número de regiones del gráfico de flujo *G:*

$$R$$

2) La complejidad ciclomática $V(G)$ para un gráfico de flujo *G:*

$$V(G) = E - N + 2$$

E es el número de aristas y *N* el número de nodos del gráfico de flujo

3) La complejidad ciclomática $V(G)$ para un gráfico de flujo *G:*

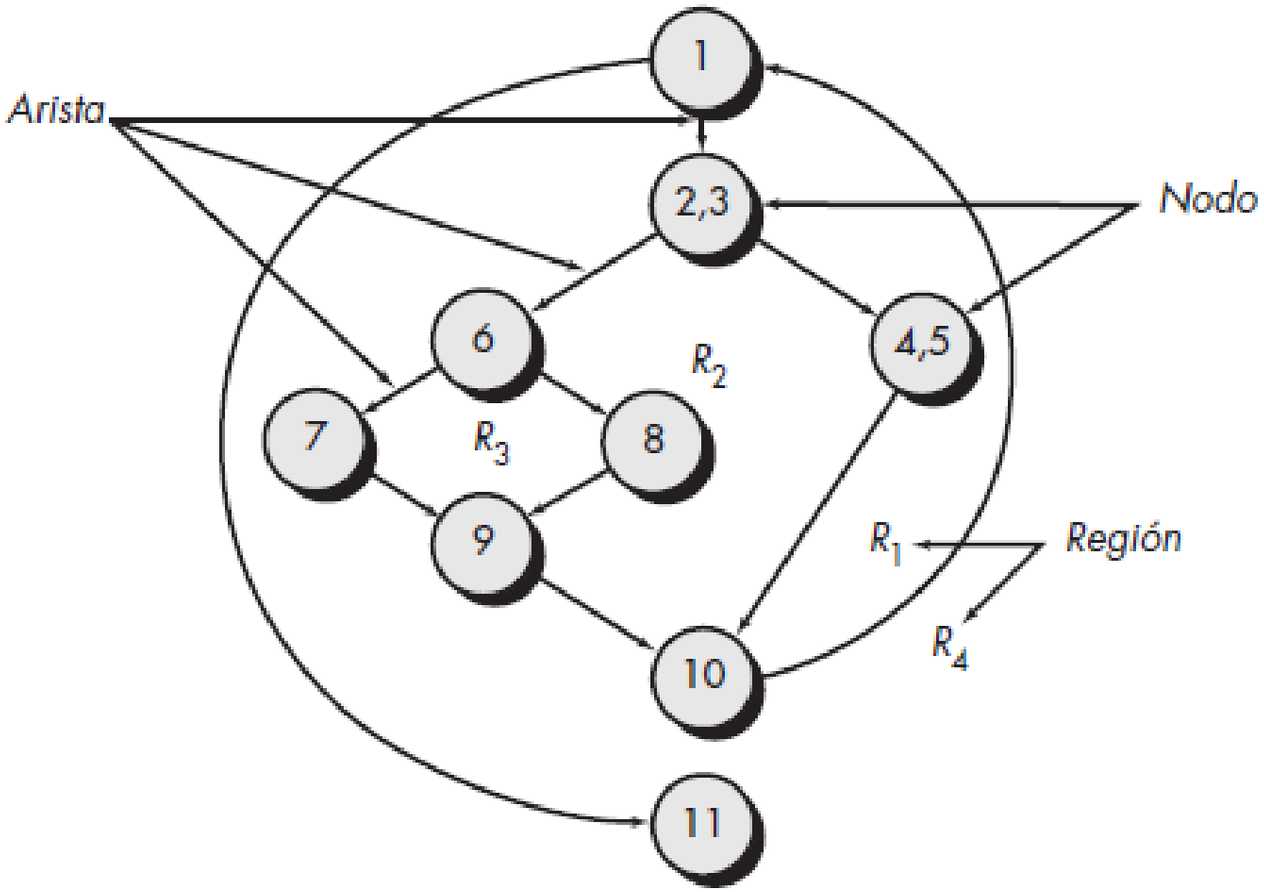
$$V(G) = P + 1$$

P es el número de nodos predicado contenidos en el gráfico de flujo *G*.

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (5/6)

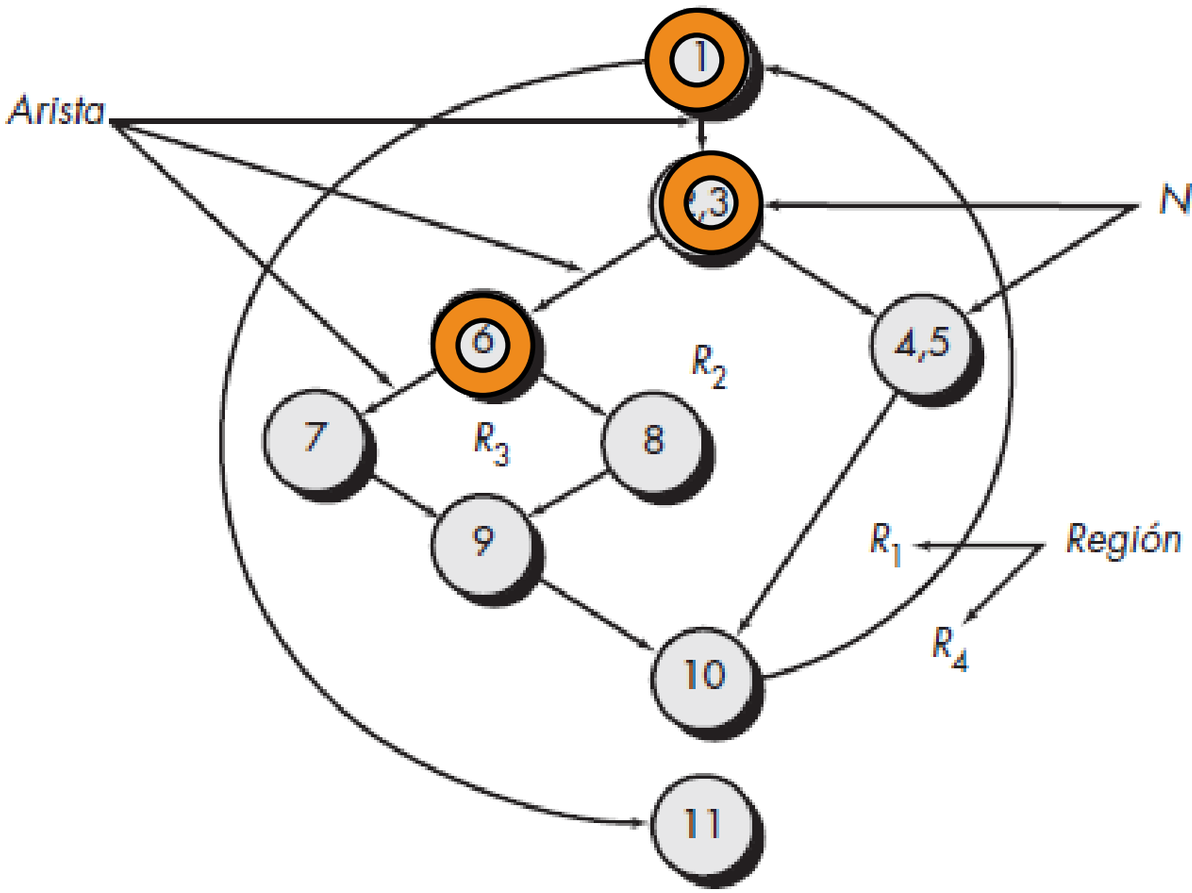
- **La complejidad ciclomática**
  - **Ejemplo:**



Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (5/6)

- **La complejidad ciclomática**
  - **Ejemplo:**



1. El gráfico de flujo tiene cuatro regiones.
2. $V(G)$ = 11 aristas − 9 nodos + 2 = 4.
3. $V(G)$ = 3 nodos predicado + 1 = 4.

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**
  - **Determine la complejidad Ciclomática**
  - **Determine un conjunto básico de rutas linealmente independientes**
  - **Plantee cómo preparar los casos de prueba que fuercen la ejecución de cada ruta en el conjunto básico**

```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
   minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
i = 1;
total.input = total.valid = 0;
sum = 0;
DO WHILE value[i] <> -999 AND total.input < 100
   increment total.input by 1;
   IF value[i] > = minimum AND value[i] < = maximum
         THEN increment total.valid by 1;
               sum = s sum + value[i]
         ELSE skip
   ENDIF
   increment i by 1;
ENDDO
IF total.valid > 0
   THEN average = sum / total.valid;
   ELSE average = -999;
13 ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**

```
PROCEDIMIENTO average;

* Este procedimiento calcula el promedio de 100 o menos
  números que se encuentran entre valores frontera; también
  calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
   minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
i = 1;
total.input = total.valid = 0;
sum = 0;
DO WHILE value[i] <> -999 AND total.input < 100
   increment total.input by 1;
   IF value[i] > = minimum AND value[i] < = maximum
         THEN increment total.valid by 1;
                sum = s sum + value[i]
         ELSE skip
   ENDIF
   increment i by 1;
ENDDO
IF total.valid > 0
   THEN average = sum / total.valid;
   ELSE average = -999;
13 ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**


```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
   minimum, maximum, sum IS SCALAR;
   TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100
      increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum
            THEN increment total.valid by 1;
                 sum = s sum + value[i]
            ELSE skip
      ENDIF
      increment i by 1;
   ENDDO
   IF total.valid > 0
      THEN average = sum / total.valid;
      ELSE average = -999;
13 ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

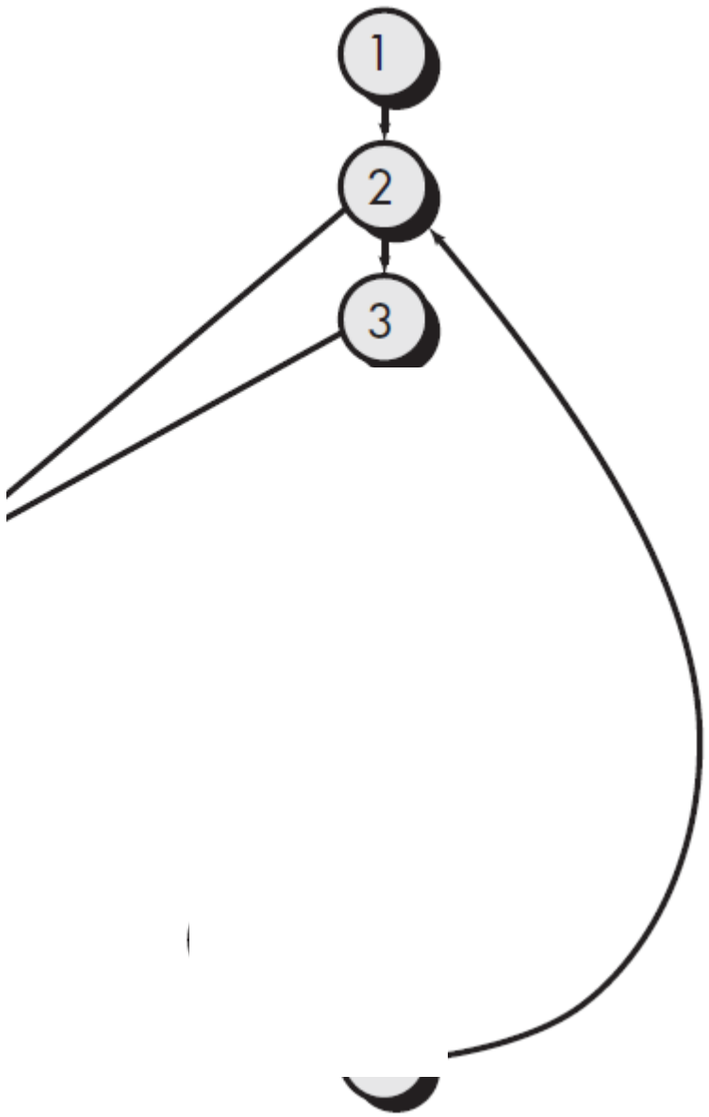
- **Ejercicio**
  - **Dibuje el gráfico de flujo**


```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
    minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
i = 1;                                          1
total.input = total.valid = 0;                  2
sum = 0;
DO WHILE value[i] <> -999 AND total.input < 100
    increment total.input by 1;
    IF value[i] > = minimum AND value[i] < = maximum
        THEN increment total.valid by 1;
             sum = s sum + value[i]
        ELSE skip
    ENDIF
    increment i by 1;
ENDDO
IF total.valid > 0
    THEN average = sum / total.valid;
    ELSE average = -999;                        13
ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**

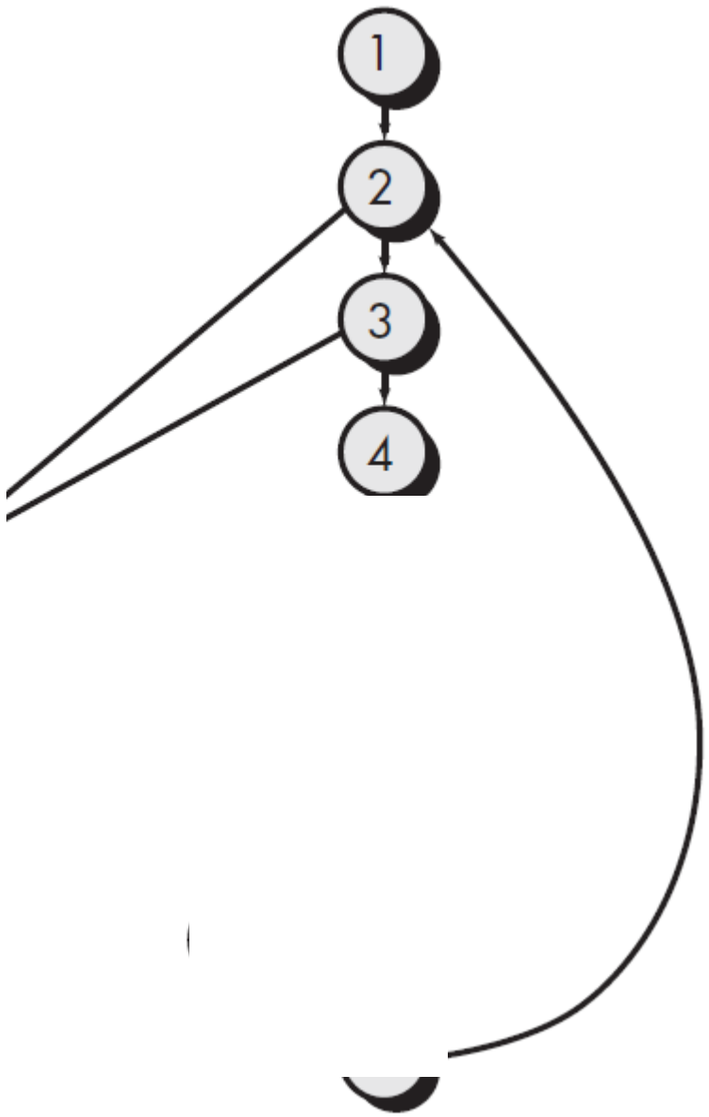



```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
    minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
i = 1;                                                    2
total.input = total.valid = 0;                    1
sum = 0;
DO WHILE value[i] <> -999 AND total.input < 100           3
    increment total.input by 1;
    IF value[i] > = minimum AND value[i] < = maximum
        THEN increment total.valid by 1;
             sum = s sum + value[i]
        ELSE skip
    ENDIF
    increment i by 1;
ENDDO
IF total.valid > 0
    THEN average = sum / total.valid;
    ELSE average = -999;
13 ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**

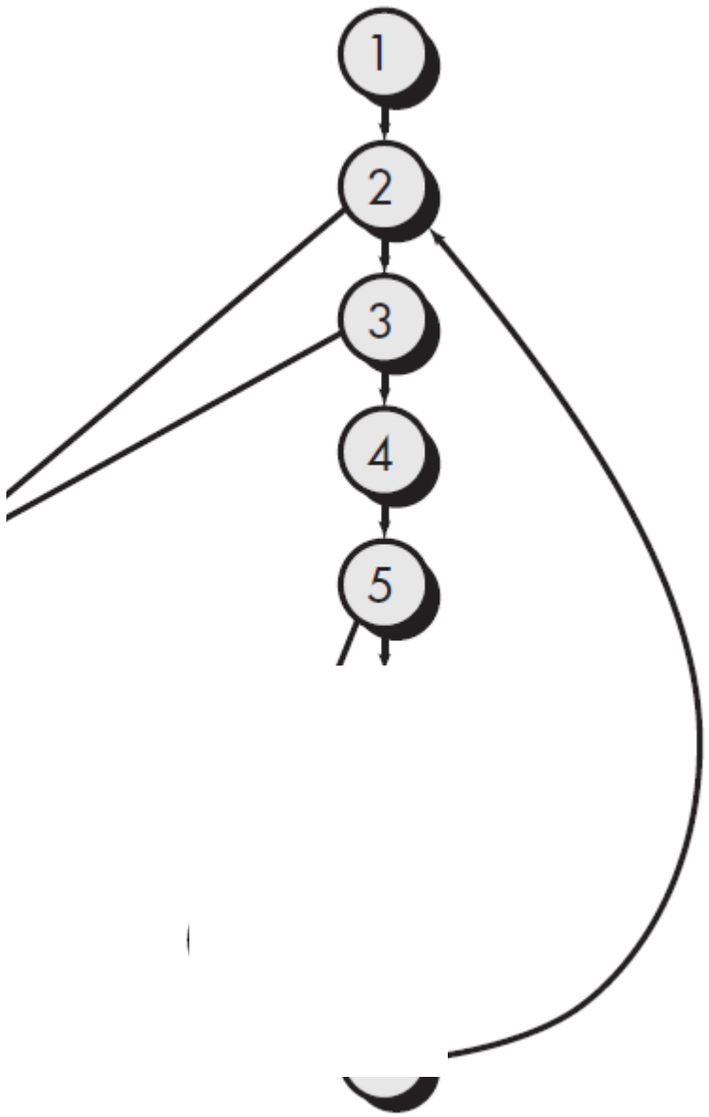



```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
    minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;        2
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100    3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum
            THEN increment total.valid by 1;
                 sum = s sum + value[i]
            ELSE skip
      ENDIF
      increment i by 1;
   ENDDO
   IF total.valid > 0
      THEN average = sum / total.valid;
      ELSE average = -999;
13 ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**

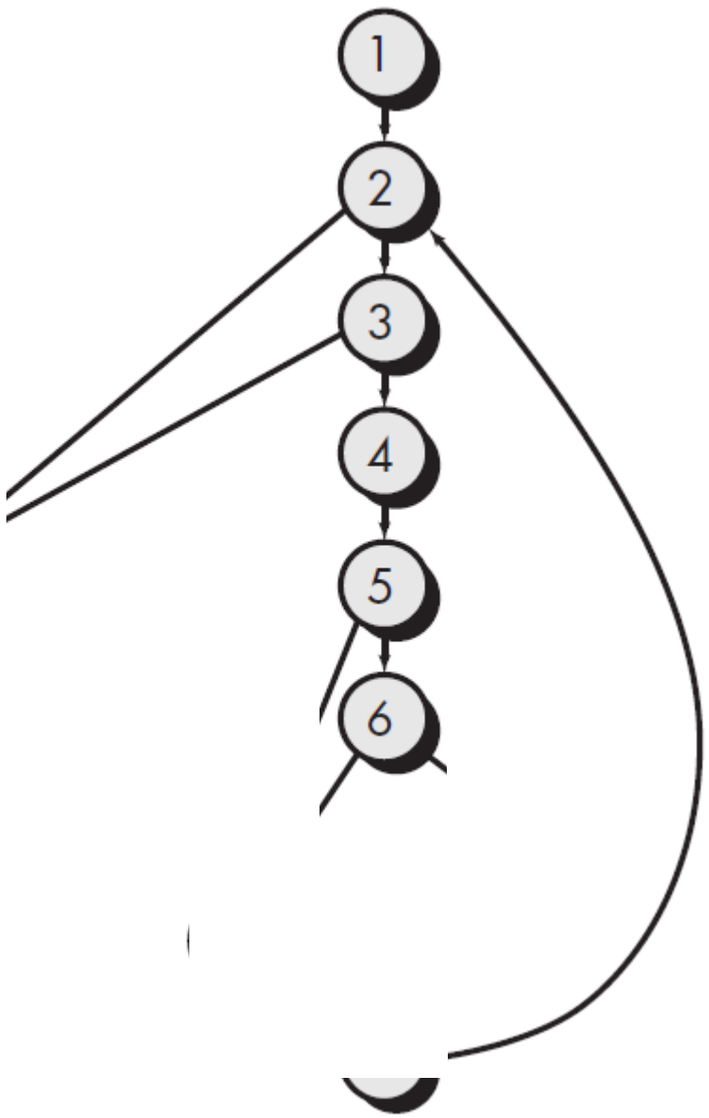



```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
    minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1   i = 1;
    total.input = total.valid = 0;        2
    sum = 0;
    DO WHILE value[i] <> -999 AND total.input < 100   3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum
5           THEN increment total.valid by 1;
                    sum = s sum + value[i]
            ELSE skip
      ENDIF
      increment i by 1;
    ENDDO
    IF total.valid > 0
      THEN average = sum / total.valid;
      ELSE average = -999;
13  ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**



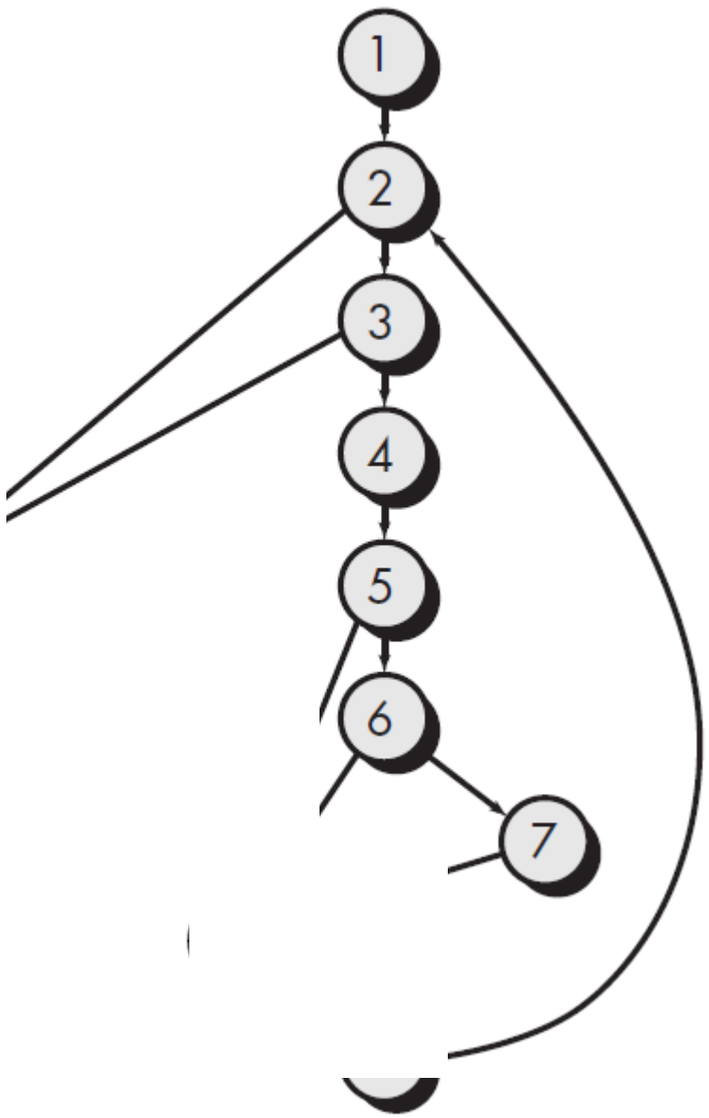

```
PROCEDIMIENTO average;

* Este procedimiento calcula el promedio de 100 o menos
  números que se encuentran entre valores frontera; también
  calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
    minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100      3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum  6
5        THEN increment total.valid by 1;
                sum = s sum + value[i]
         ELSE skip
      ENDIF
      increment i by 1;
   ENDDO
   IF total.valid > 0
      THEN average = sum / total.valid;
      ELSE average = -999;
13 ENDIF
END average
```

2

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**

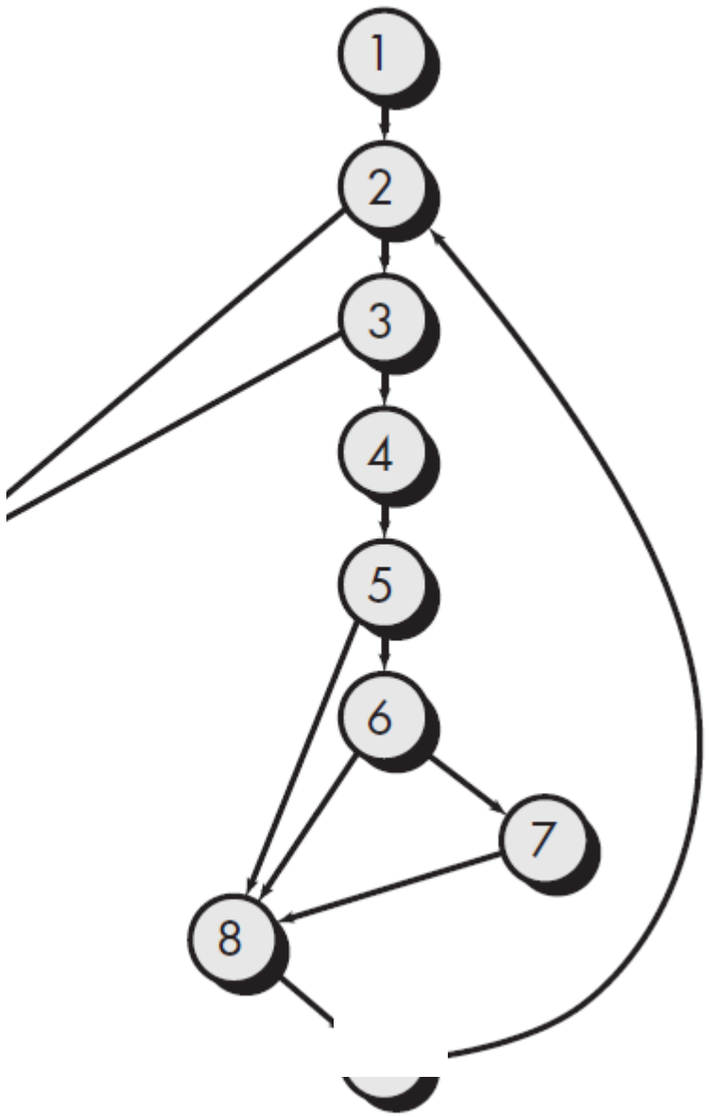



```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
     minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum
7        THEN increment total.valid by 1;
                  sum = s sum + value[i]
         ELSE skip
      ENDIF
      increment i by 1;
   ENDDO
   IF total.valid > 0
      THEN average = sum / total.valid;
      ELSE average = -999;
13 ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**

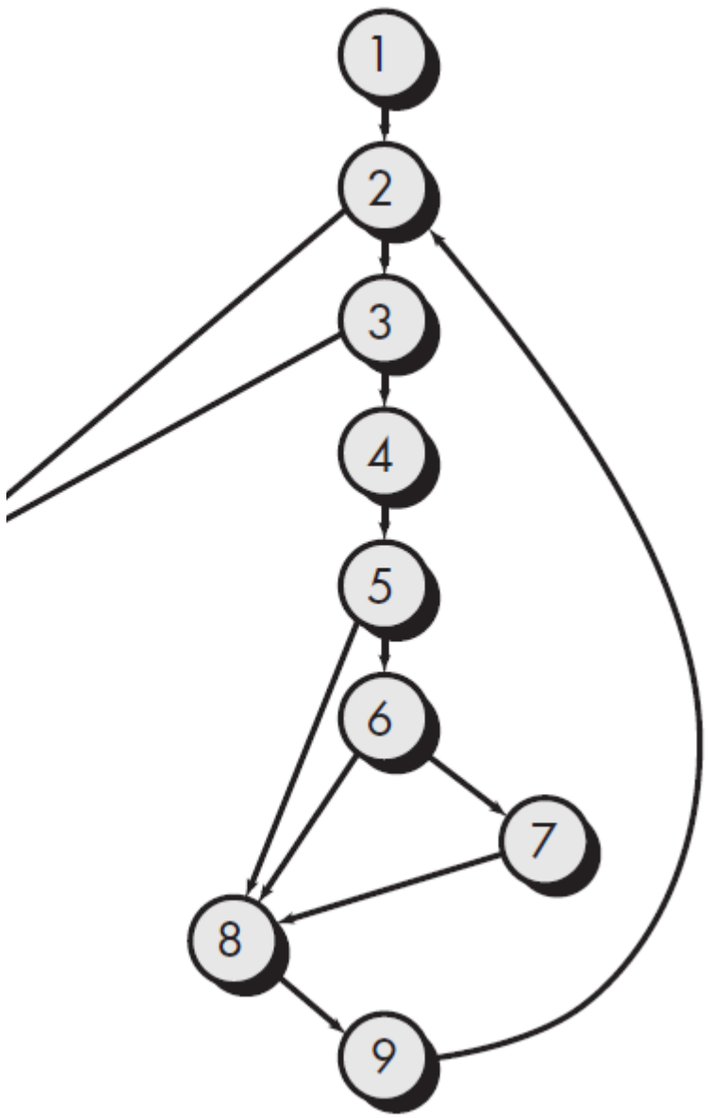



```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
     minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100      3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum  6
5        THEN increment total.valid by 1;
7                sum = s sum + value[i]
         ELSE skip
8     ENDIF
      increment i by 1;
   ENDDO
   IF total.valid > 0
      THEN average = sum / total.valid;
      ELSE average = -999;
13 ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**

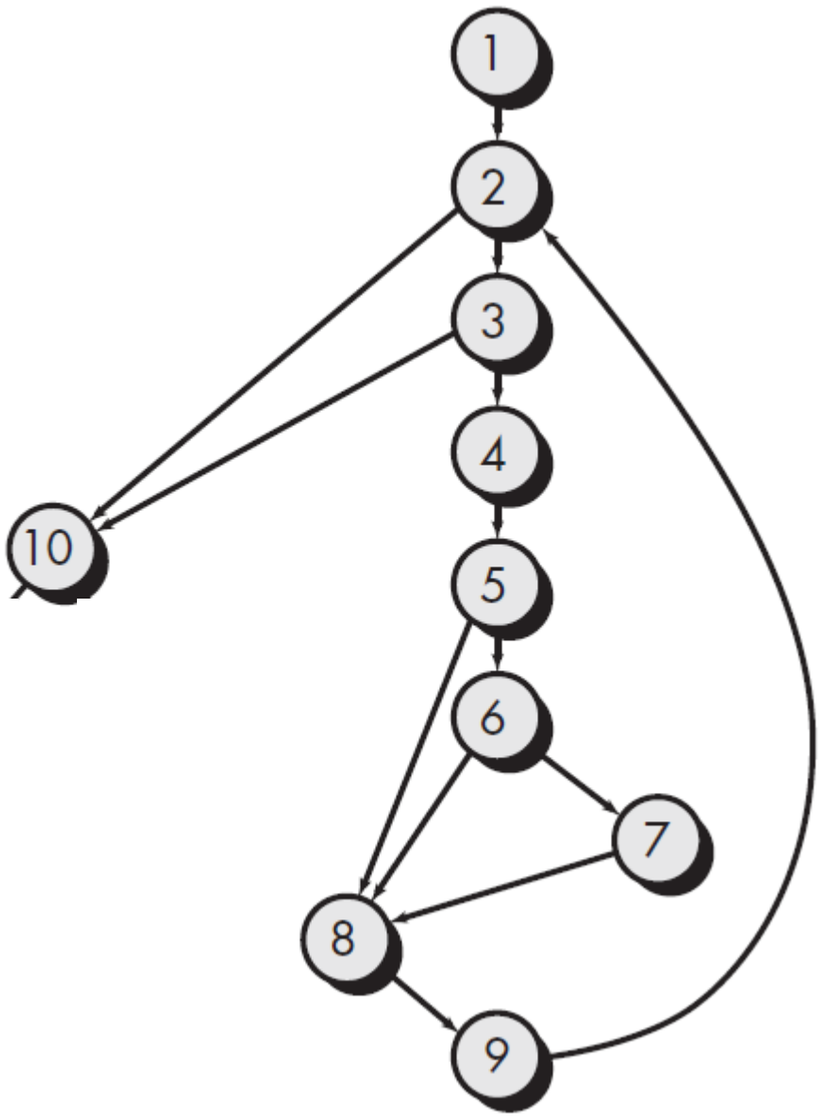



```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
     minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100      3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum   6
5        THEN increment total.valid by 1;
7                sum = s sum + value[i]
         ELSE skip
      ENDIF
8     increment i by 1;
9  ENDDO
   IF total.valid > 0
      THEN average = sum / total.valid;
      ELSE average = -999;
13 ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**

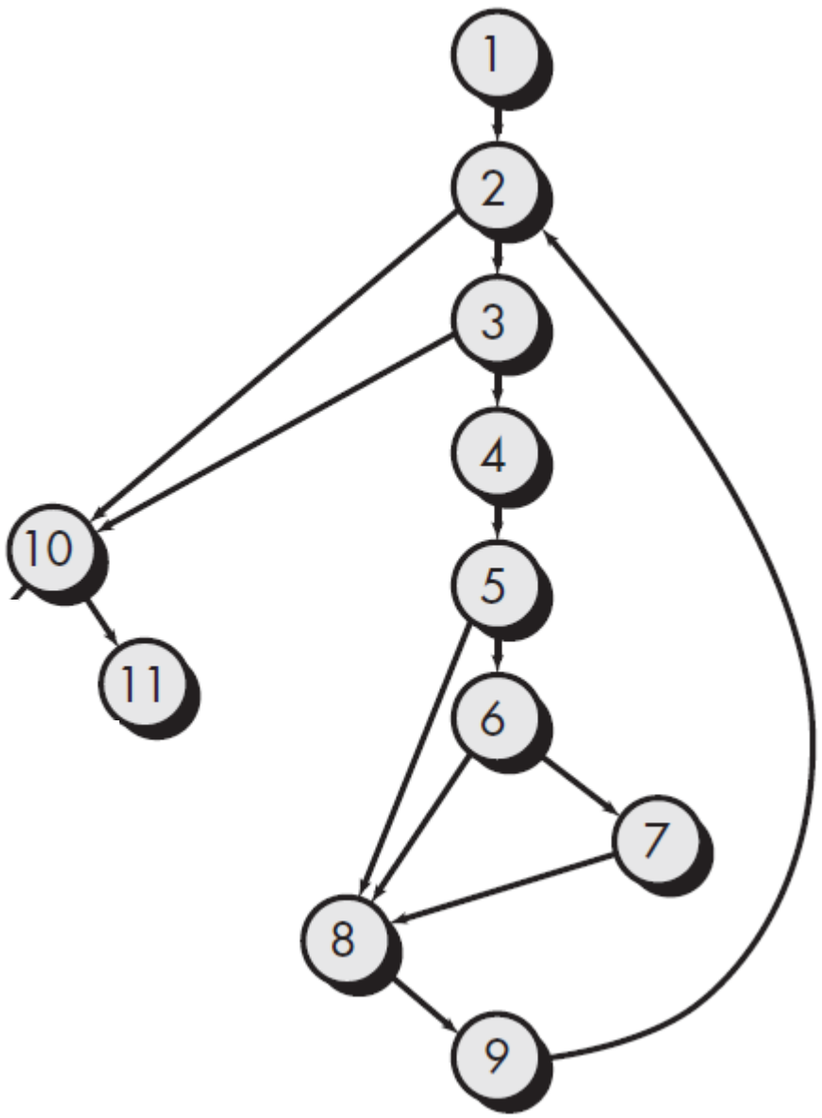



```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
     minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;        2
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100   3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum   6
5        THEN increment total.valid by 1;
7                sum = s sum + value[i]
         ELSE skip
8     ENDIF
      increment i by 1;
9  ENDDO
   IF total.valid > 0   10
      THEN average = sum / total.valid;
      ELSE average = -999;
13 ENDIF
END average
```

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**



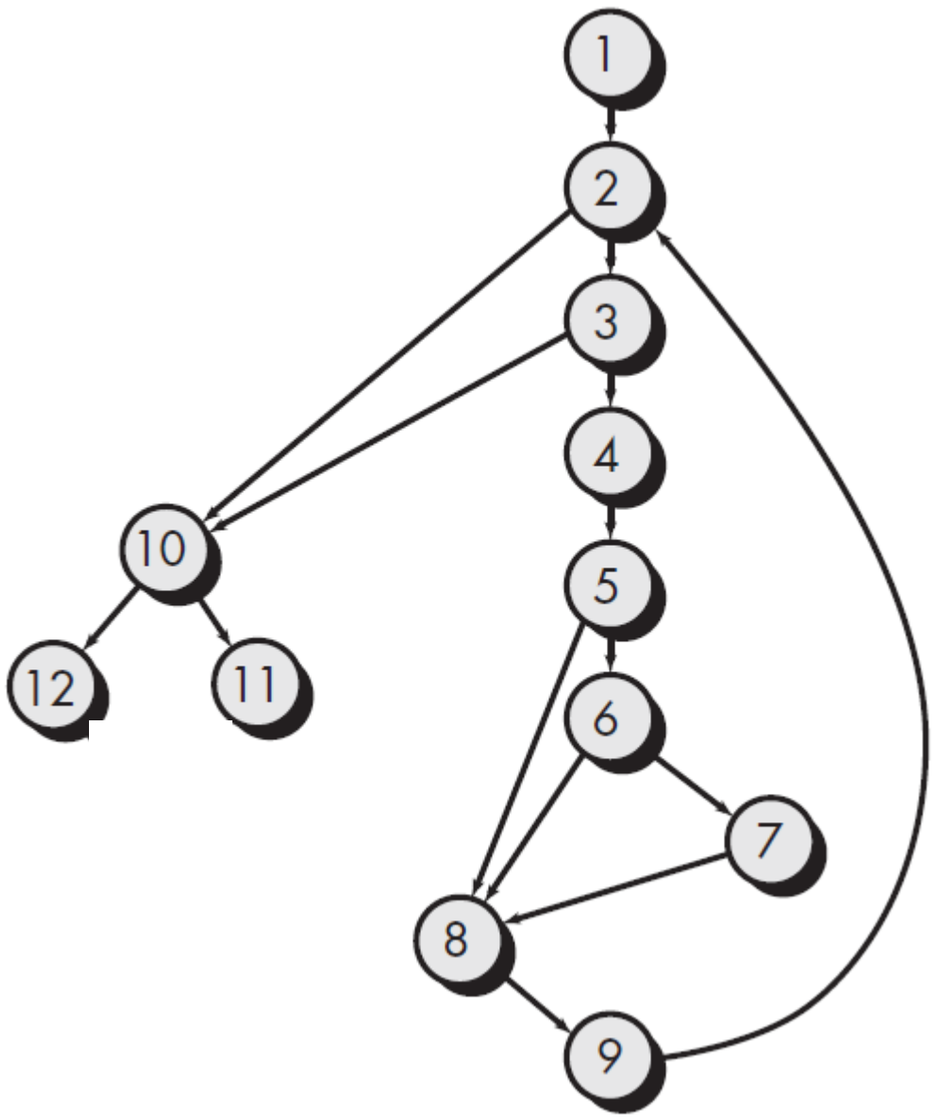

```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
     minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100      3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum  6
5        THEN increment total.valid by 1;
7               sum = s sum + value[i]
         ELSE skip
8     ENDIF
      increment i by 1;
9  ENDDO
   IF total.valid > 0    10
11    THEN average = sum / total.valid;
         ELSE average = -999;
13 ENDIF
END average
```

2

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**



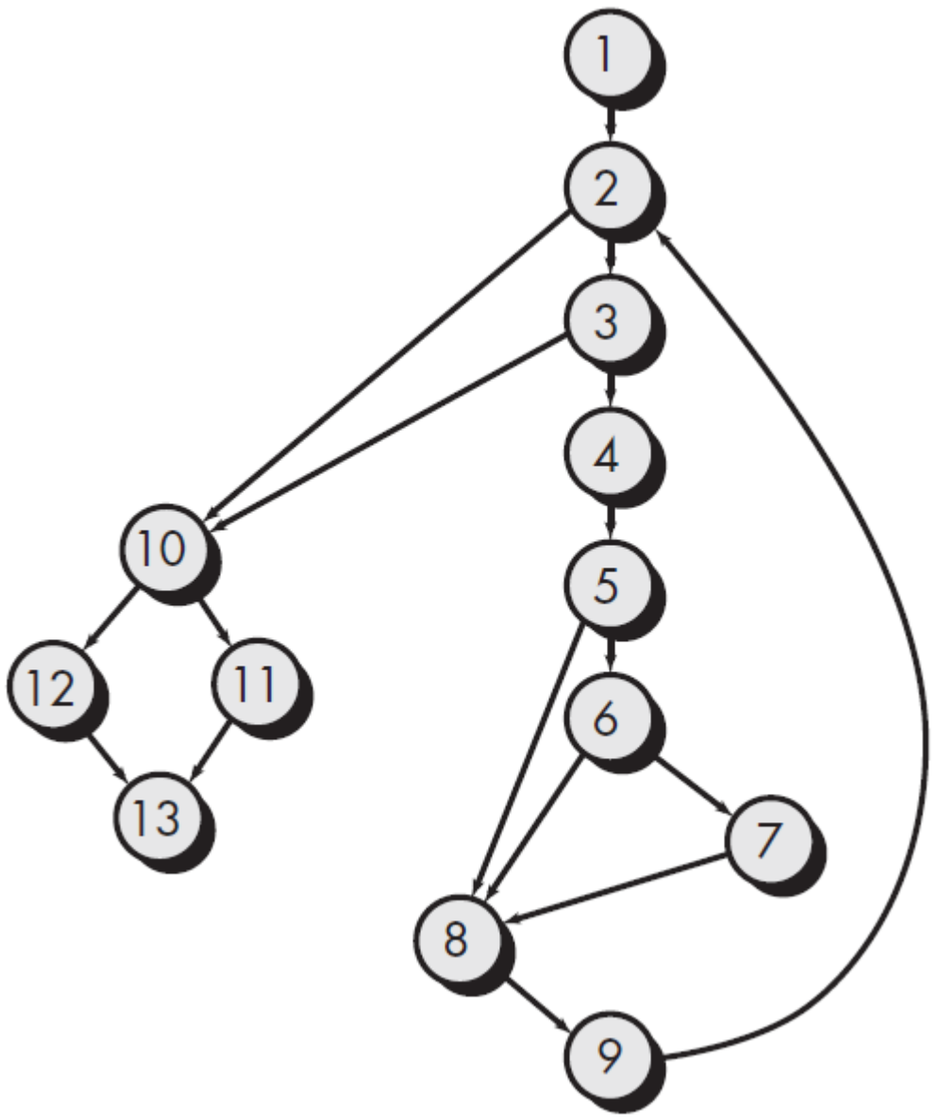

```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
     minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100      3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum  6
5        THEN increment total.valid by 1;
7               sum = s sum + value[i]
         ELSE skip
      ENDIF
8     increment i by 1;
9  ENDDO
   IF total.valid > 0                                   10
11    THEN average = sum / total.valid;
12    ELSE average = -999;
   ENDIF
END average
```

(2)

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**



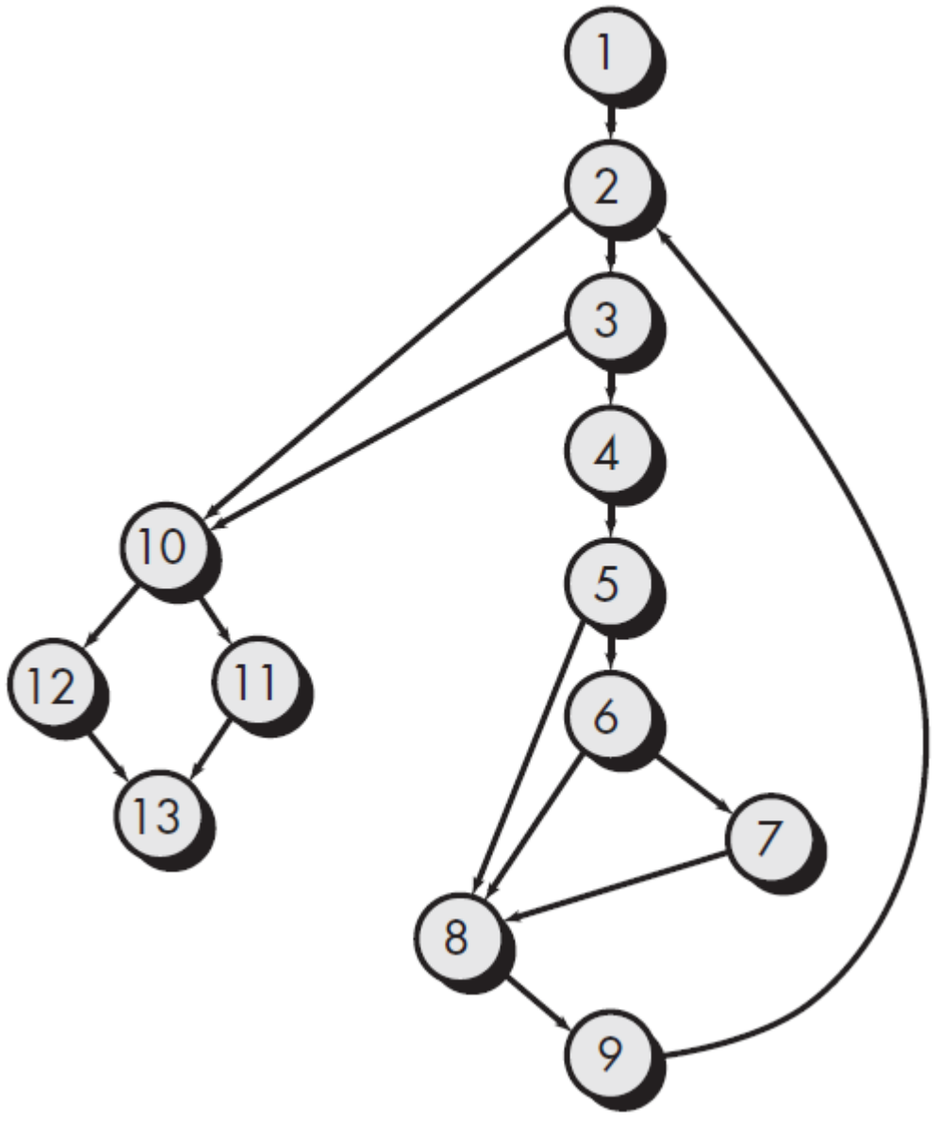

```
PROCEDIMIENTO average;

*  Este procedimiento calcula el promedio de 100 o menos
   números que se encuentran entre valores frontera; también
   calcula la suma y el número total válido.

INTERFACE RETURNS average, total.input, total.valid;
INTERFACE ACCEPTS value, minimum, maximum;

TYPE value[1:100] IS SCALAR ARRAY;
TYPE average, total.input, total.valid;
     minimum, maximum, sum IS SCALAR;
TYPE i IS INTEGER;
1  i = 1;
   total.input = total.valid = 0;
   sum = 0;
   DO WHILE value[i] <> -999 AND total.input < 100   3
4     increment total.input by 1;
      IF value[i] > = minimum AND value[i] < = maximum   6
5        THEN increment total.valid by 1;
7               sum = s sum + value[i]
         ELSE skip
      ENDIF
8     increment i by 1;
9  ENDDO
      IF total.valid > 0   10
11       THEN average = sum / total.valid;
12       ELSE average = -999;
13 ENDIF
END average
```

2

Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - Dibuje el gráfico de flujo
  - **Determine la complejidad Ciclomática**

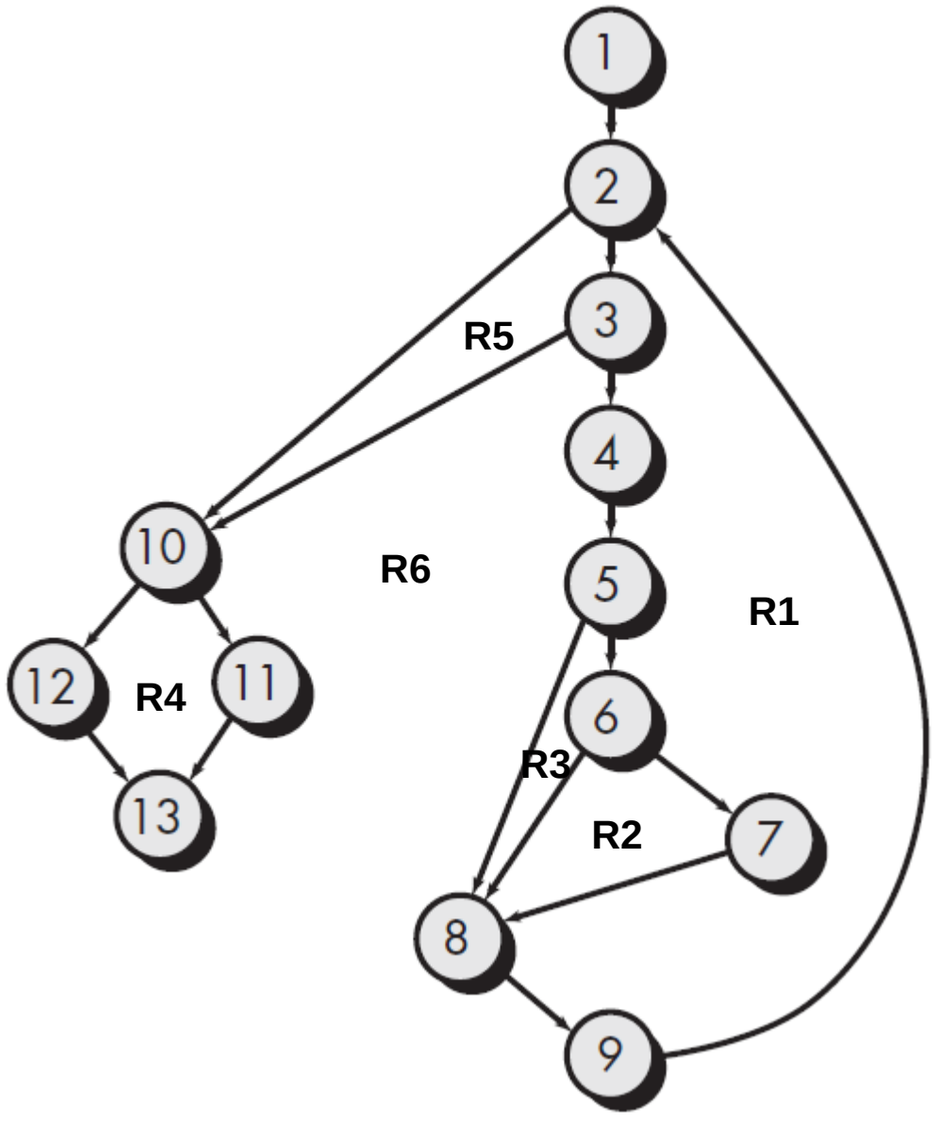


Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - Dibuje el gráfico de flujo
  - **Determine la complejidad Ciclomática**

$V(G) = 6$ regiones



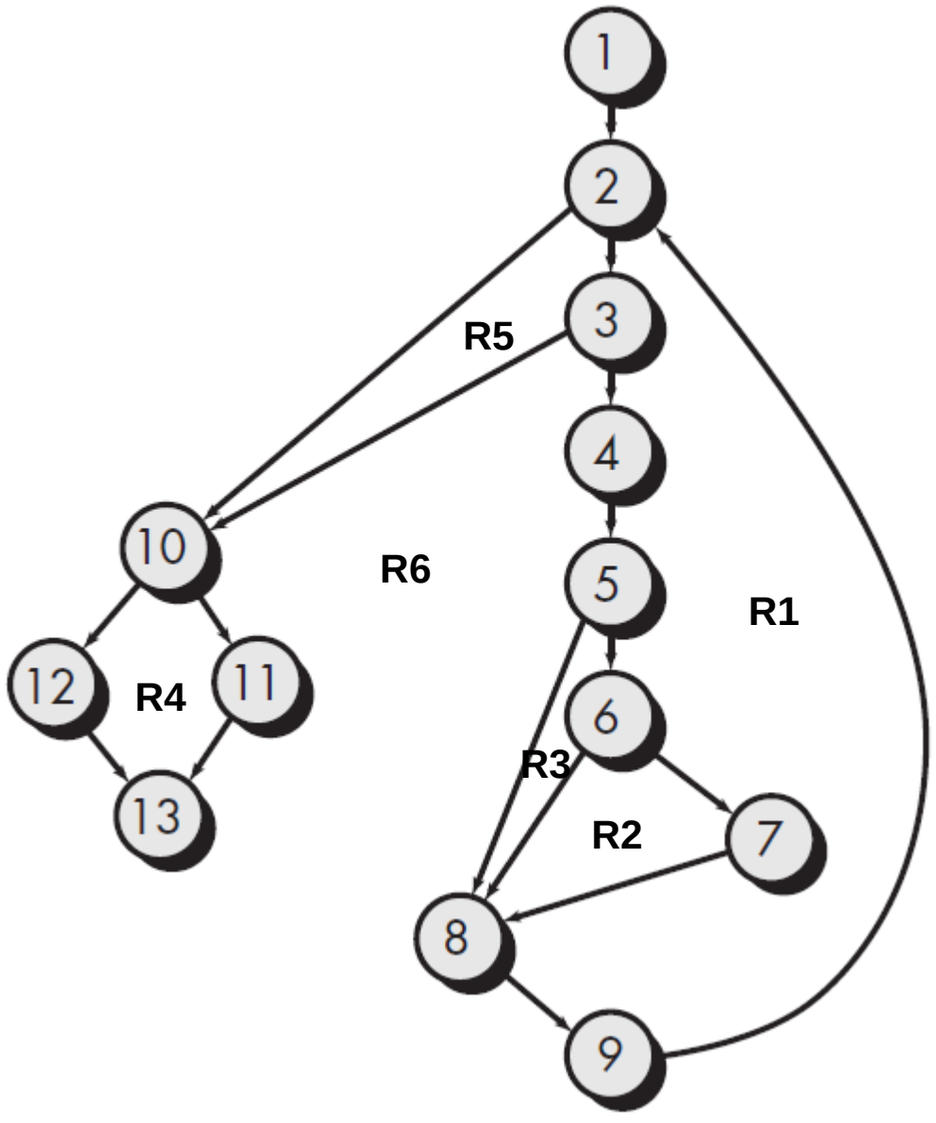
Prueba de Caja Blanca (2/2)

Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - Dibuje el gráfico de flujo
  - **Determine la complejidad Ciclomática**

$V(G) = 6$ regiones

$V(G) = 17$ aristas $-$ 13 nodos $+ 2 = 6$



Prueba de Caja Blanca (2/2)

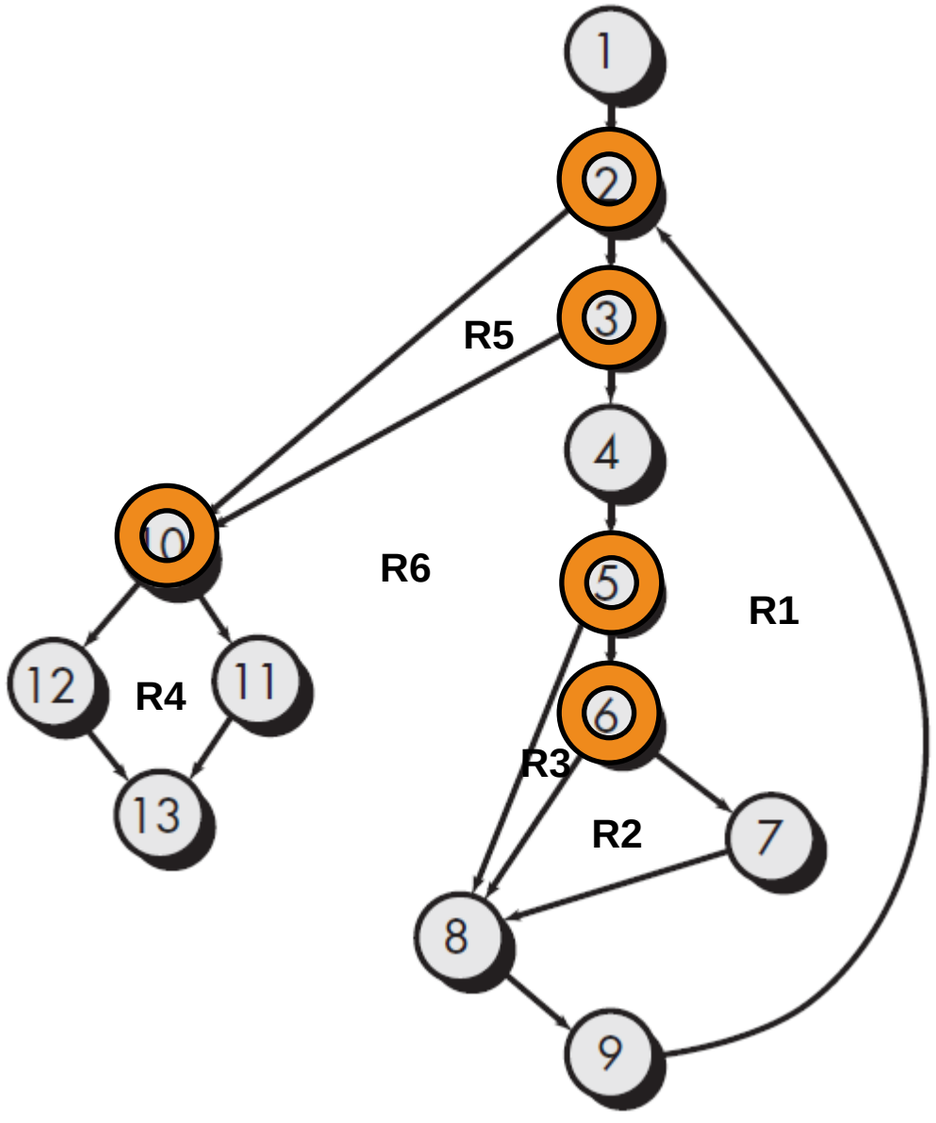
Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - Dibuje el gráfico de flujo
  - **Determine la complejidad Ciclomática**

$V(G)$ = 6 regiones

$V(G)$ = 17 aristas − 13 nodos + 2 = 6

$V(G)$ = 5 nodos predicado + 1 = 6



Prueba de Caja Blanca (2/2)
Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - Dibuje el gráfico de flujo
  - Determine la complejidad Ciclomática
  - **Determine un conjunto básico de rutas linealmente independientes**

ruta 1: 1-2-10-11-13

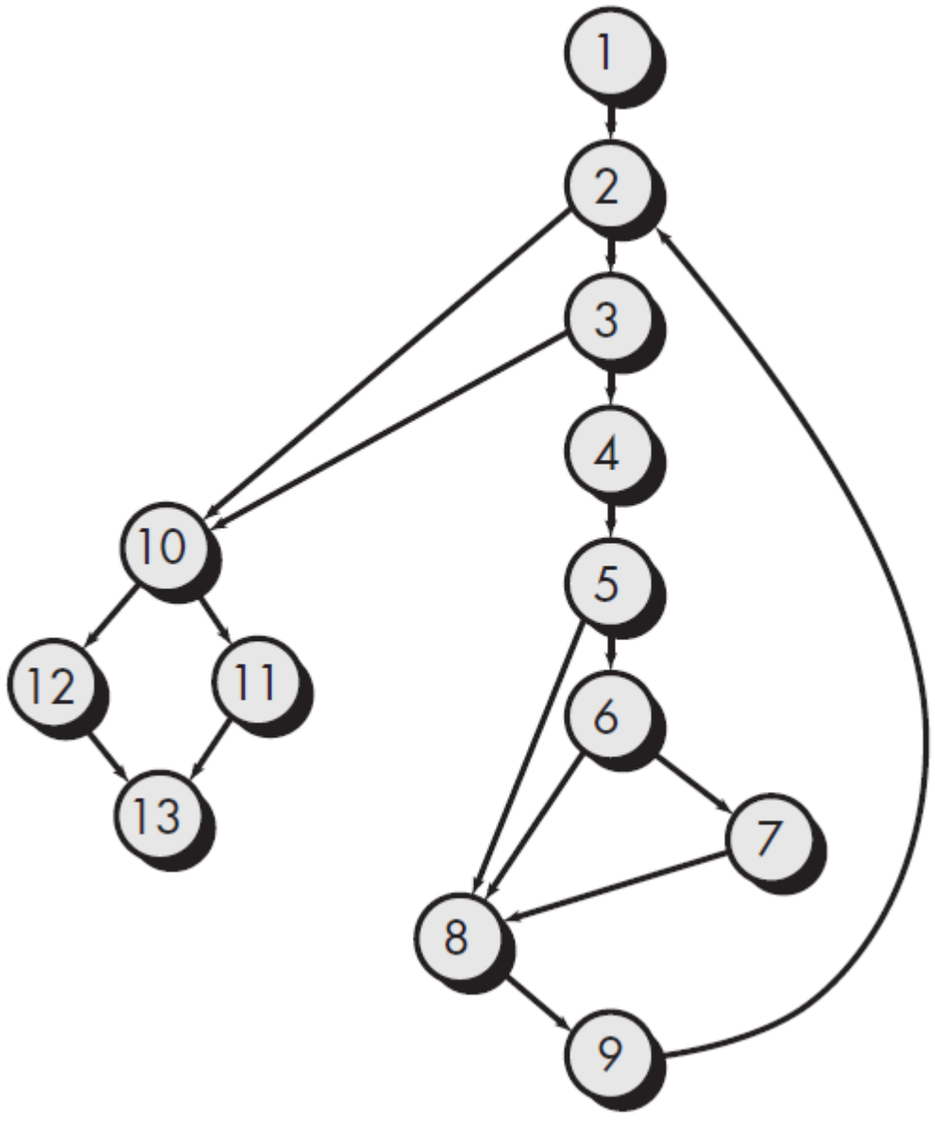
ruta 2: 1-2-10-12-13

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

ruta 4: 1-2-3-4-5-8-9-2-...

ruta 5: 1-2-3-4-5-6-8-9-2-...

ruta 6: 1-2-3-4-5-6-7-8-9-2-...



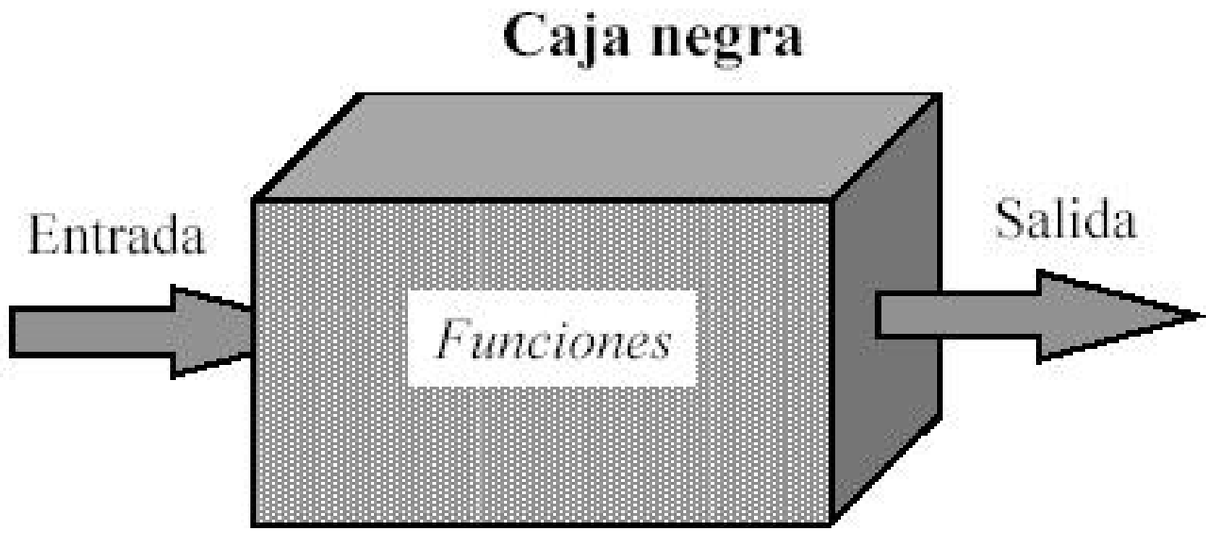
Prueba de Caja Blanca (2/2)
Prueba de Ruta Básica (6/6)

- **Ejercicio**
  - **Dibuje el gráfico de flujo**
  - **Determine la complejidad Ciclomática**
  - **Determine un conjunto básico de rutas linealmente independientes**
  - **Plantee cómo preparar los casos de prueba que fuercen la ejecución de cada ruta en el conjunto básico**

1. Los datos deben elegirse de modo que las condiciones en los nodos predicado se establezcan de manera adecuada conforme se prueba cada ruta
2. Cada caso de prueba se ejecuta y compara con los resultados esperados.
3. Una vez completados todos los casos de prueba, el examinador puede estar seguro de que todos los enunciados del programa se ejecutaron al menos una vez.

Prueba de Caja Negra (1/3)

- Se enfocan en los requisitos funcionales del software
  - Derivar conjuntos de condiciones de entrada que revisarán por completo todos los requerimientos funcionales para un programa.
- Las pruebas de caja negra no son una alternativa para las técnicas de caja blanca.
  - Es un enfoque complementario que es probable que descubra una clase de errores diferente que los métodos de caja blanca.



Prueba de Caja Negra (2/3)

- Intentan encontrar errores en las categorías siguientes:
  - Funciones incorrectas o faltantes
  - Errores de interfaz
  - Errores en las estructuras de datos o en el acceso a bases de datos externas
  - Errores de comportamiento o rendimiento
  - Errores de inicialización y terminación
- La prueba de caja negra tiende a aplicarse durante las últimas etapas de la prueba:
  - Unidad → Integración → Validación → Sistema

Prueba de Caja Negra (3/3)

- ¿Qué preguntas responden las pruebas de caja negra?
  - *¿Cómo se prueba la validez funcional?*
  - *¿Cómo se prueban el comportamiento y el rendimiento del sistema?*
  - *¿Qué clases de entrada harán buenos casos de prueba?*
    - *Escoger datos que ejerciten distintas funciones del software*
  - *¿El sistema es particularmente sensible a ciertos valores de entrada?*
  - *¿Cómo se aíslan las fronteras de una clase de datos?*
  - *¿Qué tasas y volumen de datos puede tolerar el sistema?*
  - *¿Qué efecto tendrán sobre la operación del sistema algunas combinaciones específicas de datos?*