



Ingeniería de Software II

Tipos de prueba

Prueba del software

- » La etapa de Prueba no es la primera instancia en que se localizan defectos.
- » Se ha visto que la revisión de requerimientos y el diseño contribuyen a descubrir los problemas (defectos) en las etapas tempranas.

2

Prueba del software

» ¿Qué significa que el software ha fallado?

El software no hace lo que especifican los requerimientos

» Posibles razones:

- *Especificación errónea.*
- *Requerimientos imposibles con las estructuras previstas.*
- *Defectos en diseño del sistema.*
- *Defectos en diseño del programa.*
- *Defectos en código.*

3

Tipos de Defecto

» Algorítmicos

Ej. : No inicializar variables

» De sintaxis

Ej. : Confundir un 0 por una O

» De precisión

Ej. : Fórmulas no implementadas correctamente

» De documentación

Ej. : Documentación no acorde con lo que hace el software

» De sobrecarga

Ej. : El sistema funciona bien con 100 usuarios pero no con 110.

Tipos de Defecto

» De capacidad

Ej. : El sistema funciona bien con ventas $< 1.000.000$

» De coordinación o sincronización

Ej.: Comunicación entre procesos con fallas

» De rendimiento

Ej.: Tiempo de respuesta inadecuado.

» De recuperación

Ej. : No volver a un estado normal luego de una falla

» De relación hardware-software

Ej.: Incompatibilidad entre componentes

» De estándares

Ej. : No cumplir con la definición de estándares y procedimientos

Clasificación ortogonal de Defectos

- » Los defectos se han organizado en categorías.
- » Primeramente se debe identificar si es un:
 - Defecto por omisión** (resulta cuando algún aspecto clave del código falta).
Ej: variable no inicializada.
 - Defecto de cometido** (resulta cuando algún aspecto es incorrecto).
Ej: variable inicializada con un valor erróneo.

6

Clasificación ortogonal de Defectos

»Pfleeger Cap. 8

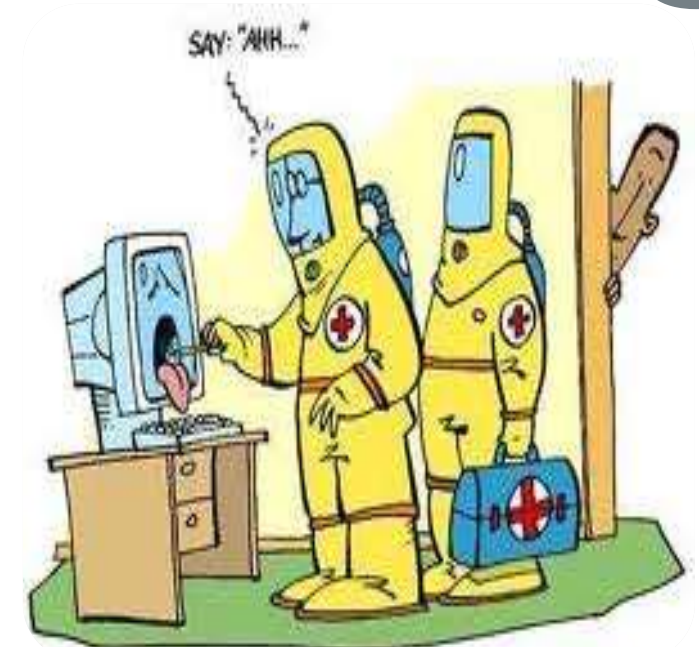
Tipo de defecto	Significado
Función	Afecta la capacidad, interfaces.
Interfaz	Afecta a la interacción con otros componentes.
Comprobación	Afecta la lógica del programa.
Asignación	Afecta la estructura de datos.
Sincronización	Involucra sincronización de recursos compartidos y de tiempo real.
Construcción	Ocurre debido a problemas en repositorios, gestión de cambios o control de versiones.
Documentación	Afecta a publicaciones.
Algoritmo	Involucra la eficiencia o exactitud de un algoritmo.

7

Prueba del software

- » ¿Cuál es el primer objetivo de la prueba?
- » Diseñar pruebas que saquen a la luz diferentes clases de errores, haciéndolo en la menor cantidad de tiempo y esfuerzo.
- » **¿Cuándo una prueba tiene éxito?**

Cuándo descubre errores



8

Objetivos y Beneficios de las Pruebas del Software

9

- » Descubrir errores antes que el software salga del ambiente de desarrollo
- » Detectar un error no descubierto hasta entonces.
- » Bajar los costos de corrección de errores en la etapa de mantenimiento

Principios de la Prueba

- » A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente.
- » Las pruebas deberían planificarse mucho antes de que empiecen.
- » Es aplicable el principio de Pareto. **El mismo dice que "el 80% de los errores de un software es generado por un 20% del código de dicho software, mientras que el otro 80% genera tan sólo un 20% de los errores".**

10



Principios de la Prueba

- » Las pruebas deberían empezar por «lo pequeño» y progresar hacia «lo grande»
- » Es importante asegurarse que se han aplicado (probado) todas las condiciones a nivel de componente.
- » Para ser más eficaces, las pruebas deberían ser realizadas por un equipo independiente.

11



¿Quién realiza las pruebas?

- » Varios factores justifican un equipo independiente de pruebas, entre ellos:
 - ❖ Evitar el conflicto entre la responsabilidad por los defectos y la necesidad de descubrir defectos
 - ❖ Llevar a cabo las pruebas concurrentemente con la codificación.
 - ❖ Los desarrolladores deben colaborar y corregir los errores.

12

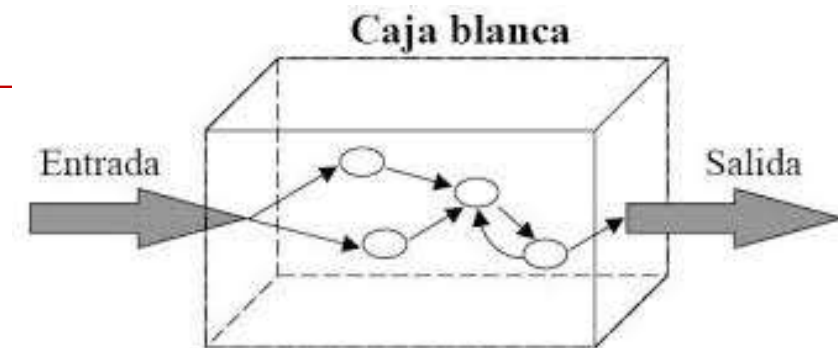
Pruebas del Software

- » Lamentablemente “La prueba **NO puede asegurar** la ausencia de defectos”
- » Se intentan buscar Casos de Prueba que permitan encontrar errores

13

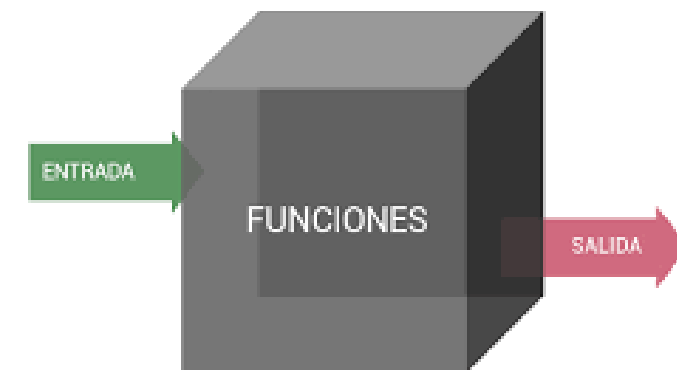
Tipos de Prueba del Software

» La prueba de **caja blanca** se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles.



14

» La prueba de **caja negra** se refiere a las pruebas que se llevan a cabo sobre la interfaz del software.



Tipos de Prueba

¿En qué momento se pueden definir los casos de prueba?

Ejemplo 1: Tengo una función del sistema que busca un número en una secuencia de números y devuelve la posición en la que se encuentra (-1 si no lo encuentra)

¿Qué valores podemos definir como casos de prueba para el ejemplo 1?
¿Por qué?

¿Cuántos valores debemos probar?

¿Cuándo lo podemos probar efectivamente?

15

¿Qué probarían en este código?
¿Por qué?

¿Qué valores debemos probar?

¿Cómo garantizamos que recorrimos todos los caminos posibles?

16

Ejemplo 2:

```
Procedure eliminarValor (var pri:
Lista; n:integer);
Var pos,ant:lista;
Begin
  pos:= pri; ant:= pri; ok:= false;
  while (pos <> nil) and (not ok)do
    if (pos^.datos = n) then ok:=
true
    else begin
      ant:=pos;
      pos:= pos^.sig;
    end;
```

```
if (ok=true) then
begin
  if (pos = pri) then
    pri:= pos^.sig
  else begin
    ant^.sig:= pos^.sig;
    dispose (pos);
  end;
End;
```



Caja Negra

Tipos de Prueba

Caja Negra (o Cerrada)

- » También denominada prueba de comportamiento, se centran en los requisitos funcionales del software.
- » Intenta descubrir diferentes tipos de errores que los métodos de caja blanca.

18



Tipos de Prueba Caja Negra (o Cerrada)

¿Qué errores busca las pruebas de
Caja Negra?



**Funciones
incorrectas**

Identificación de
errores en funciones
que son incorrectas.



**Errores de
interfaz**

Identificación de
errores en la interfaz
de usuario.



**Errores en
estructuras de
datos**

Identificación de
errores en
estructuras de datos
o bases de datos.



**Errores de
rendimiento**

Identificación de
errores relacionados
con el rendimiento.



**Errores de
inicialización**

Identificación de
errores de
inicialización y de
terminación.

Prueba De Partición Equivalente

- » El diseño de los casos de prueba se basa en una evaluación de las clases de equivalencia para una condición de entrada.
- » Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada.
- » Una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

20



Prueba De Partición Equivalente - Definición

Si una condición de entrada especifica un **rango**, se define una clase de equivalencia válida y dos no válidas.

Si una condición de entrada requiere un **valor específico**, se define una clase de equivalencia válida y dos no válidas.

Si una condición de entrada especifica un elemento de un **conjunto**, se define una clase de equivalencia válida y una no válida.

Si una condición de entrada es **lógica**, se define una clase de equivalencia válida y una no válida.

21

Prueba De Partición Equivalente

Ejemplo: Dar de alta un Jugquete

DAR DE ALTA JUGUETE :

Código:

Nombre:

Descripción:

Recomendaciones:

Género:

Edad:

Marca:

Stock: Stock mínimo:

Estado:

Dato	Tipo
Código	entero positivo
Nombre	string 20
descripción	string 256
Recomendaciones	string 512
Genero	enumerativo (masculino, femenino, no binario)
Edad	rango 0..120
Marca	string 25
Stock	entero
stock mínimo	entero positivo
Estado	enumerativo (normal, oferta, novedoso)

22

Identificación de las clases de equivalencia.

23

Condición de entrada	Clases de equivalencia válidas	Clases de equivalencia inválidas
código	0<= código <= 9999	código <0 código > 9999
nombre	1 a 20 caracteres	0 caracteres; mas de 20 caracteres;
descripción	0 a 256 caracteres	mas de 256 caracteres
recomendaciones	0 a 512 caracteres	mas de 512 caracteres
genero	masculino femenino	otra cadena de caracteres;
stock	Número entero	Caracteres no dígitos;

Análisis de Valores Límite (AVL)

- » Los errores tienden a darse más en los **límites** del campo de entrada que en el «centro».
- » Por ello, se ha desarrollado el análisis de valores límites (AVL) como técnica de prueba.
- » **Complementa a la partición equivalente.** En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL selecciona los casos de prueba en los «extremos» de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida

24

Caja Negra: Análisis de Valores Límite

- » Casos de prueba en los bordes de las clases:
- » Para una condición de **entrada de rango entre a y b** probar: a, b, <a y >b.
- » Para una **salida de rango entre a y b**: utilizar casos de prueba que generen valor de salida a y b
- » Probar las **estructuras de datos internas** en sus límites.

25



Caja Blanca

Tipos de Prueba

Caja Blanca (o Cristal o Abierta)

» Deriva casos de prueba de la estructura de control, para verificar detalles procedimentales. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.

27

ejerciten todas las decisiones lógicas .

ejecuten todos los bucles en sus límites..

ejerciten las estructuras internas de datos
para asegurar su validez.



Tipos de Prueba

Caja Blanca (o Cristal o Abierta)

»El flujo lógico de un programa a veces no es nada intuitivo, lo que significa que nuestras suposiciones intuitivas sobre el flujo de control y los datos nos pueden llevar a tener errores de diseño que sólo se descubren cuando comienza la prueba del camino.

28

»¿Por qué realizarlas?

»Los casos especiales son los más factibles de error

»Los errores tipográficos son aleatorios

»El flujo de control intuitivo es distinto del real

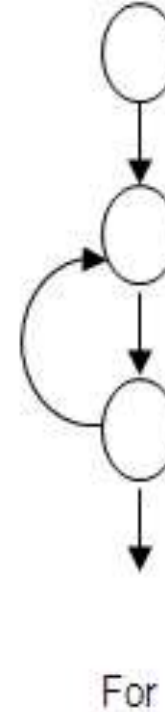
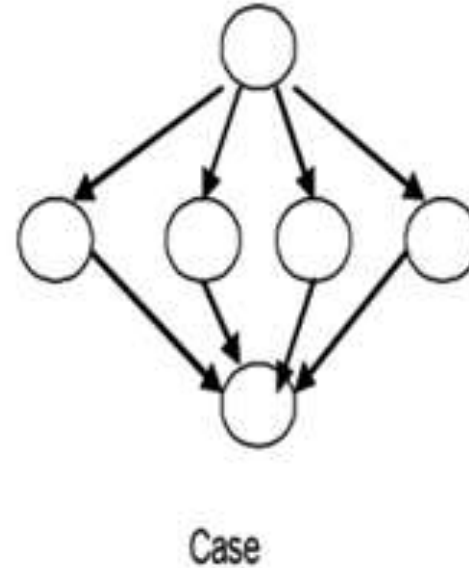
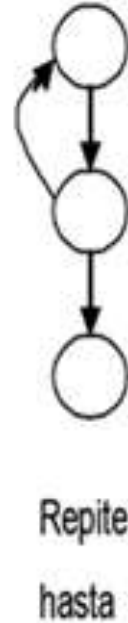
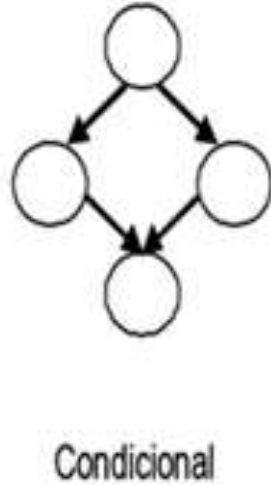
Prueba Del Camino Básico

- » Es una técnica propuesta por Tom McCabe. Permite al diseñador de casos de pruebas obtener una medida de la complejidad lógica y usarla como guía para la definición de caminos de ejecución.
- » Los casos de prueba obtenidos garantizan que se ejecuta al menos una vez cada sentencia del programa.

29

Prueba Del Camino Básico – Notación de grafo

Estructuras en forma de grafo de flujo:



30

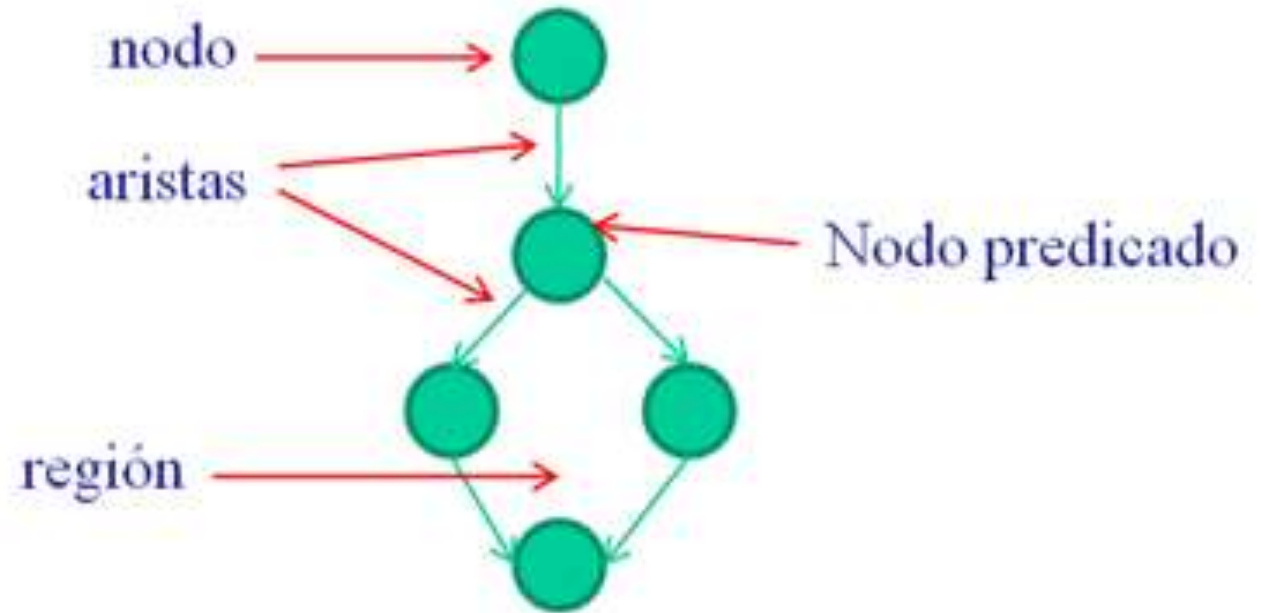
Prueba Del Camino Básico

» Cada círculo, denominado **nodo** del grafo de flujo, representa una o más sentencias procedimentales.

» Las flechas del grafo de flujo, denominadas **aristas** o representan flujo de control.

» Una arista debe terminar en un nodo.

Cada nodo que contiene una condición se denomina nodo predicado y está caracterizado porque dos o más aristas emergen de él.



Las áreas delimitadas por aristas y nodos se denominan regiones. Cuando contabilizamos las regiones incluimos el área exterior del grafo, contándolo como otra región más.

Prueba Del Camino Básico – Complejidad ciclomática

» La complejidad ciclomática

- » es una **métrica del software** que proporciona una medición cuantitativa de la complejidad lógica de un programa.
- » **define el número de caminos independientes** del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.
- » **Un camino independiente** es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición.

32

Prueba Del Camino Básico

» Complejidad ciclomática :

1. $V(g) = \text{Cantidad de regiones del grafo } \acute{o}$

2. $V(g) = A - N + 2 \acute{o}$

3. $V(g) = P + 1$

33

» La complejidad ciclomática debe medirse con las tres formulas de manera de verificar su exactitud.

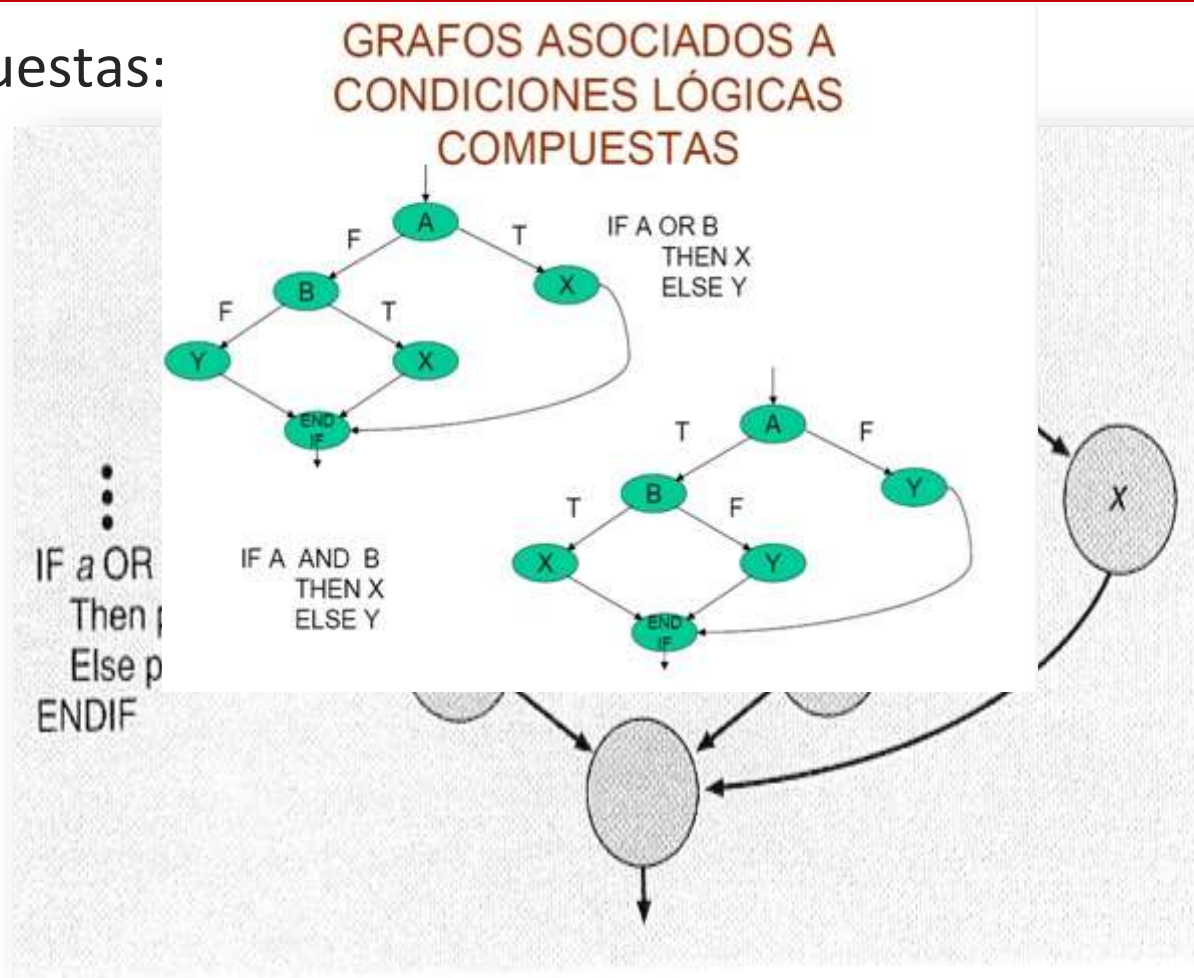
Prueba Del Camino Básico – Pasos para crear la prueba

1. Dibujar el grafo de flujo correspondiente.
2. Determinar la complejidad ciclomática.
3. Determinar un conjunto básico de caminos independientes.
4. Preparar los casos de prueba que forzarán la ejecución de cada camino del conjunto.
5. Ejecutar cada caso de prueba y comparar los resultados obtenidos con los esperados.

34

Prueba Del Camino Básico

»Condiciones compuestas:

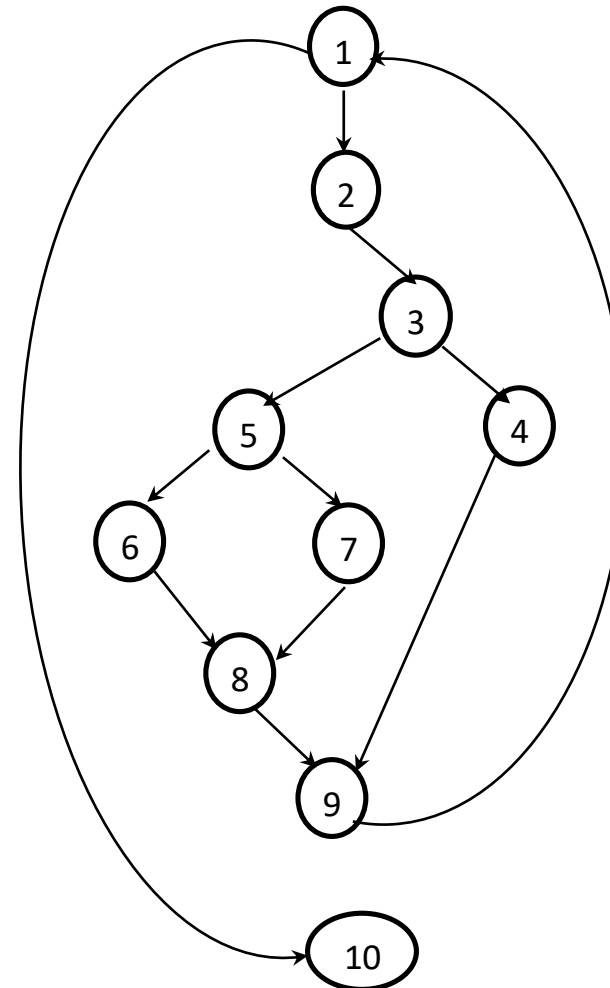


35

Prueba Del Camino Básico

Procedure Ordenar

```
(1) do while not eof() begin
(2)   leer registro;
(3)   if (campo 1 de registro = 0)
(4)     then procesar registro
           Guardar en buffer;
           Incrementar contador;
(5)   else if (campo 2 de registro = 0)
(6)     then reiniciar contador
(7)   else
           procesar registro;
           Guardar en archivo;
(8)   endif
(9) endif
(10) end;
```



36

Prueba Del Camino Básico

Regiones : 4

Nodos : 10

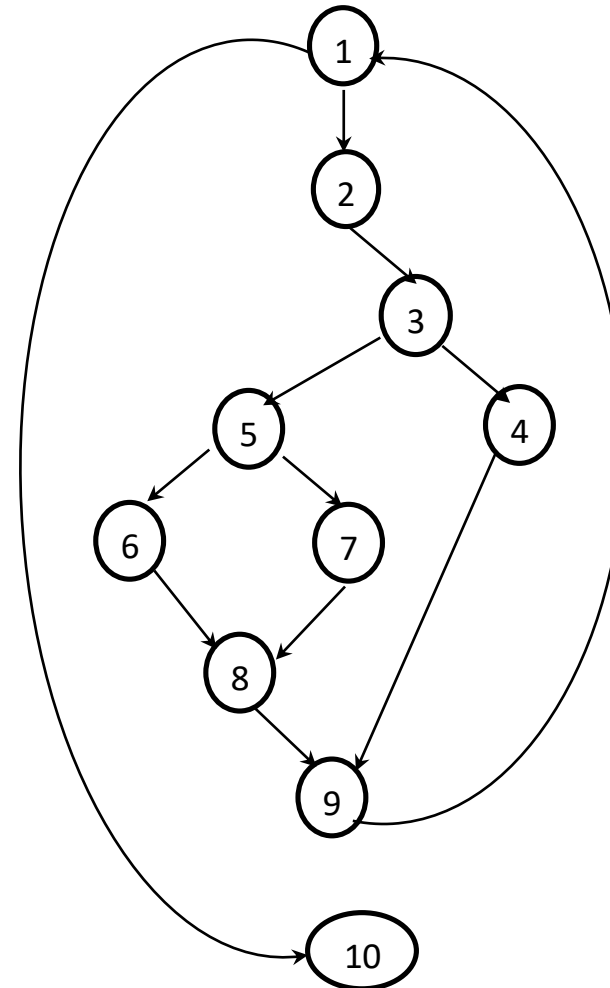
Aristas : 12

Nodos Predicados : 3

$$V(G) : A - N + 2 = 12 - 10 + 2 = 4$$

$$V(G) : NP + 1 = 3 + 1 = 4$$

$$V(G) : R = 4$$



37

Prueba Del Camino Básico

```
programejcaracteres;  
uses crt;  
var
```

```
  car: char; canta: integer; cantPal: integer;  
  priCar, ultCar: char;  
  cantCar: integer;
```

```
begin
```

```
  canta:= 0; cantPal:=0;  
  read(car);  
  while(car <> '.') do begin  
    cantPal:= cantPal + 1;  
    priCar:=car;  
    cantCar:=0;  
    while(car <> ' ') and (car <> '.') do begin  
      if(car = 'a') then  
        canta:= canta+1;  
      cantCar:= cantCar +1;  
      ultCar:=car;  
      read(car);  
    end;  
    while(car = ' ') do read(car);  
  end;  
  writeln('Cantidad de letras a ', canta);  
  writeln('Cantidad de palabras ',cantPal);
```

```
end.
```

2025

38

```

programejcaracteres;
uses crt;
var
  car: char; canta: integer; cantPal: integer; priCar, ultCar: char;
  cantCar: integer;
begin
  canta:= 0;
  cantPal:=0;
  read(car);

  while(car <> '.') do begin
    cantPal:= cantPal + 1;
    priCar:=car;
    cantCar:=0;

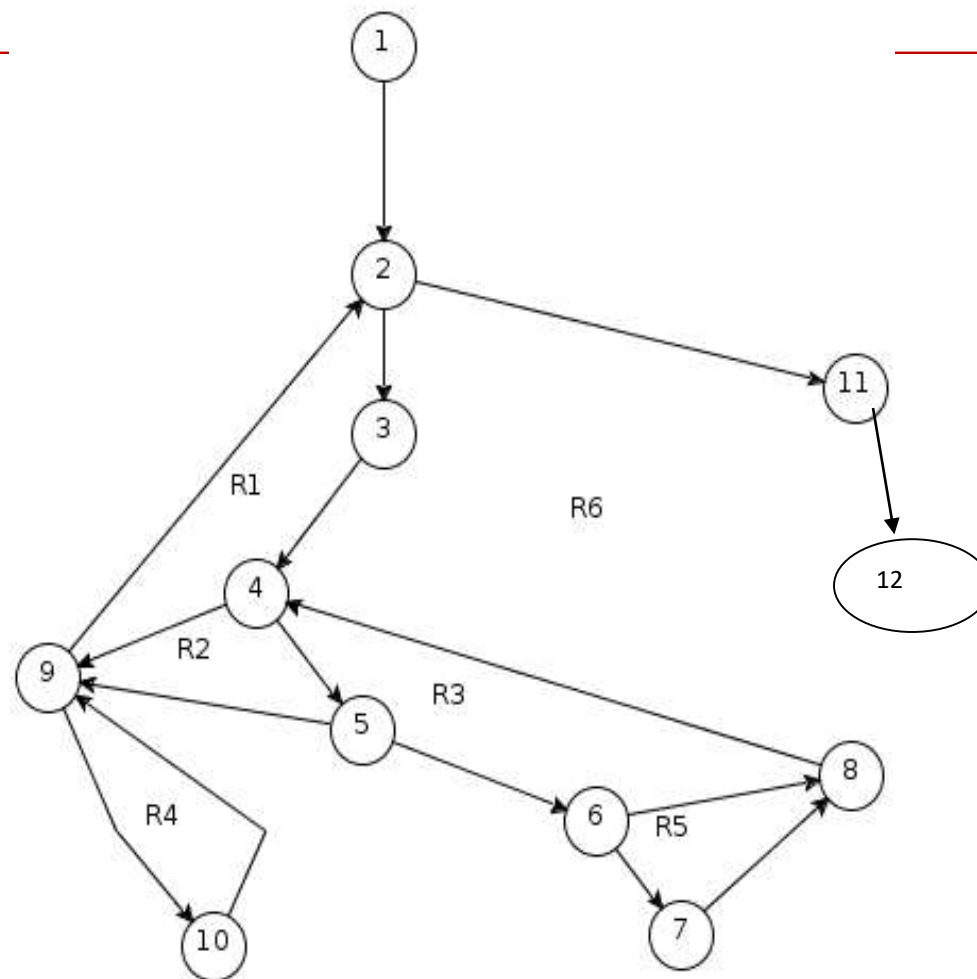
    while(car <> ' ') and (car <> '.') do begin
      if(car = 'a') then
        canta:= canta+1;
        cantCar:= cantCar +1;
        ultCar:= car;
        read(car);
      end;

      while(car = ' ') do read(car);
    end;

    writeln('Cantidad de letras a ', canta);
    writeln('Cantidad de palabras ',cantPal);
  end;
end;

```

co



39

Prueba Del Camino Básico – Complejidad ciclomática

Regiones : 6

Nodos : 12

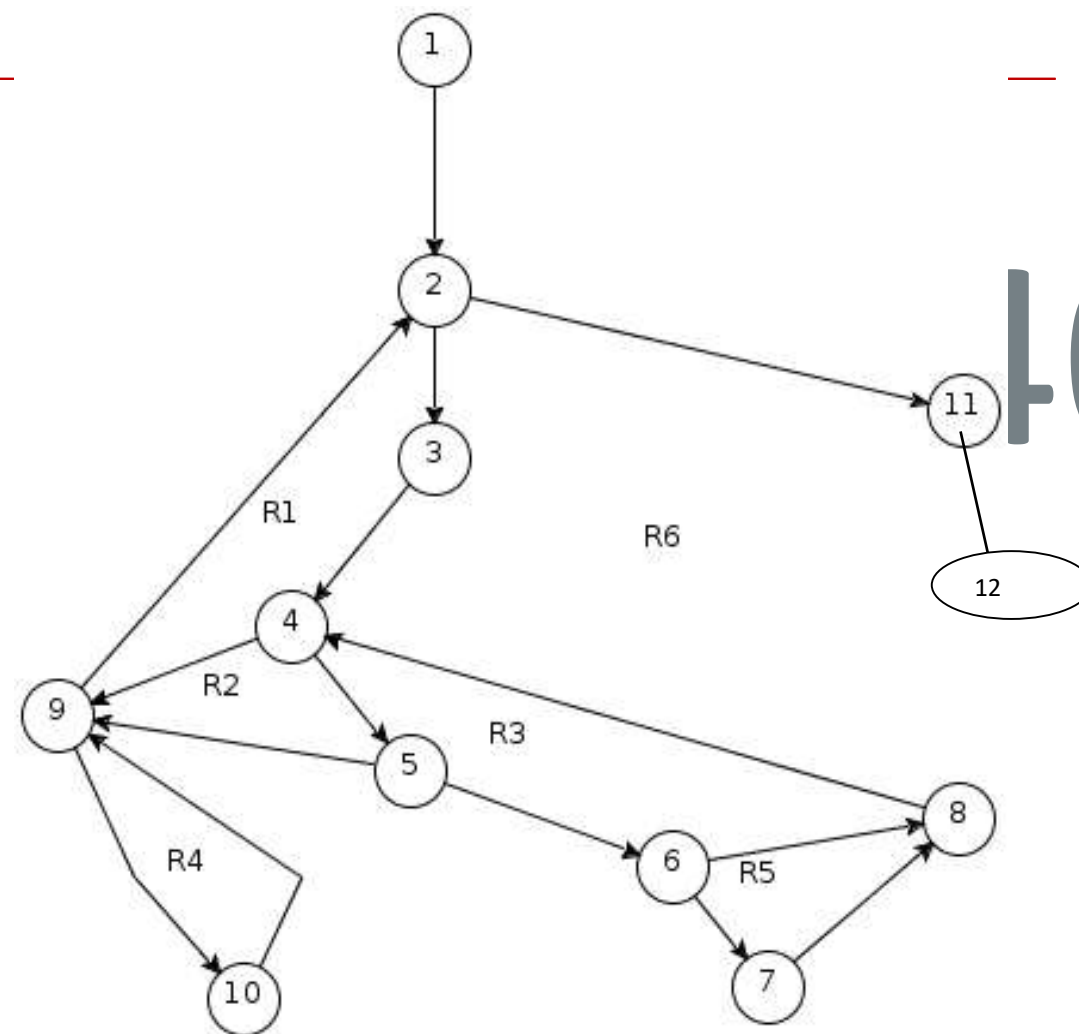
Aristas : 16

Nodos Predicados : 5

$$V(G) : A - N + 2 = 16 - 12 + 2 = \mathbf{6}$$

$$V(G) : NP + 1 = 5 + 1 = \mathbf{6}$$

$$V(G) : R = \mathbf{6}$$

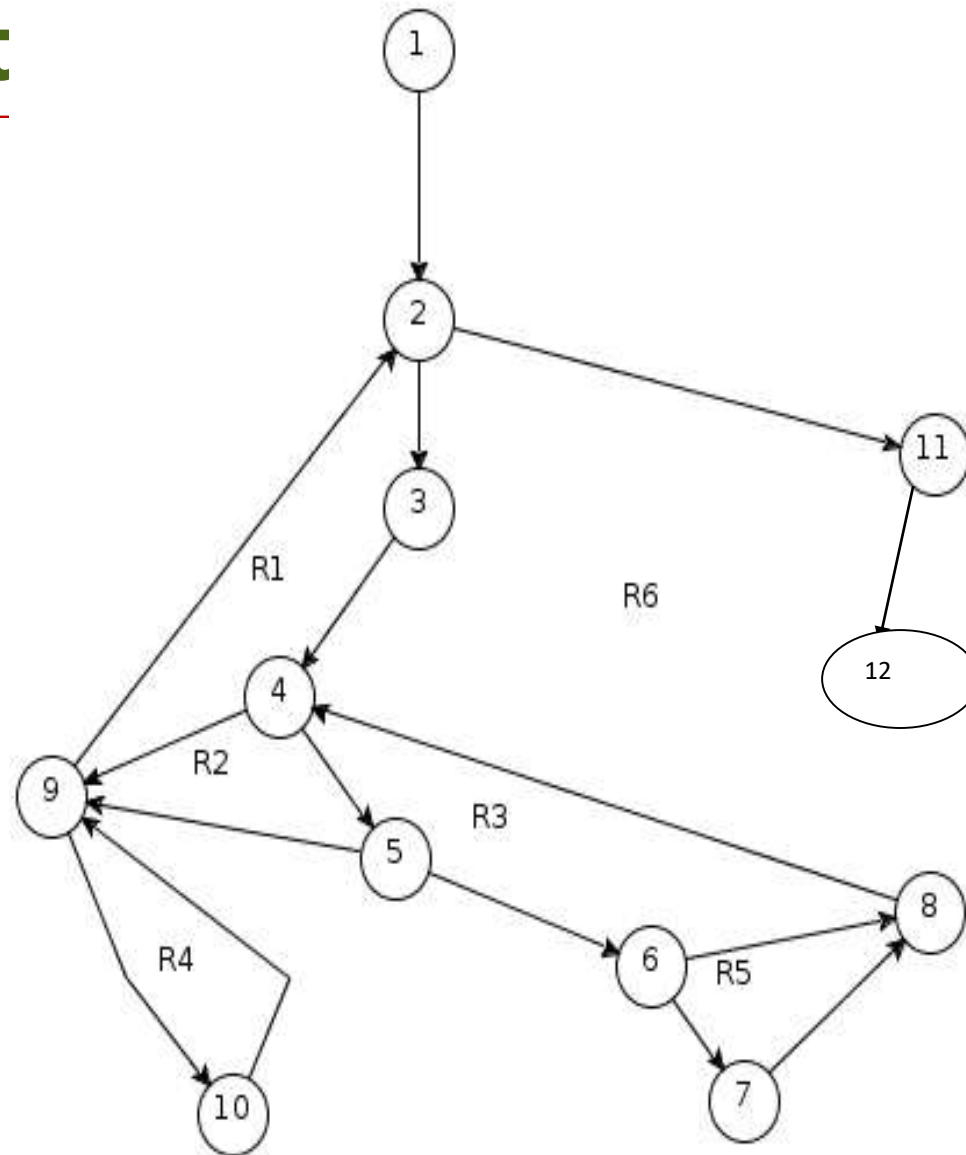


10

Prueba Del Camino Básico – Caminos linealmente independent

- »Camino 1: 1-2-11-12
- »Camino 2: 1-2-3-4-9-2-11-12
- »Camino 3: 1-2-3-4-9-10-9-2-11-12
- »Camino 4: 1-2-3-4-5-9-2-11-12
- »Camino 5: 1-2-3-4-5-6-8-4-9-2-11-12
- »Camino 6: 1-2-3-4-5-6-7-8-4-9-2-11-12

Cualquier otro camino que se quiera recorrer ya fue probado previamente en alguno de los 6 caminos



41

Prueba Del Camino Básico – Casos de prueba

»Caso de prueba del camino 1: 1-2-11-12

car= ‘.’

resultados esperados: canta = 0, cantPal=0

»Caso de prueba del camino 2: 1-2-3-4-9-2-11

car= ‘ ’

resultados esperados: canta = 0, cantPal=0

»Caso de prueba del camino

