PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE

PARA CUALQUIER PRODUCTO DE INGENIERÍA EXISTEN DOS ENFOQUES A LA HORA DE PROBAR UN PRODUCTO:

PRUEBAS DE CAJA BLANCA:

Se centra en el estudio minucioso de la operatividad de una parte del sistema considerando los detalles procedurales (la lógica del sistema).

PRUEBAS DE CAJA NEGRA:

Analiza principalmente la compatibilidad entre sí, en cuanto a las interfaces, de cada uno de los componentes del software (no tiene en cuenta la lógica del sistema).

ENFOQUES:

- **Caja blanca (como lo hace)**
- **Caja negra (que es lo que hace)**

PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

AL EMPLEAR LOS MÉTODOS DE PRUEBA DE CAJA BLANCA, EL INGENIERO DEL SOFTWARE PODRÁ DERIVAR EN CASOS DE PRUEBA QUE:

- **Garanticen que todas las rutas independientes del módulo se han ejecutado por lo menos una vez.**
- **Ejerciten los lados verdadero y falso de todas las decisiones lógicas.**
- **Ejecuten todos los bucles en sus límites y dentro de sus límites operacionales.**
- **Ejerciten estructuras de datos internos para asegurar su validez.**

PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

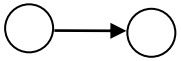
PRUEBA DE LA RUTA/ CAMINO BÁSICO

La *Prueba de la ruta básica*, es una técnica de caja blanca propuesta inicialmente por Tom McCabe. Este método permite a diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de *UN CONJUNTO BÁSICO DE CAMINOS DE EJECUCIÓN*. Los casos de prueba obtenidos de la ruta básica garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa.

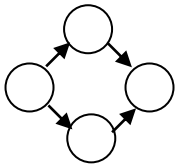
PASO 1. NOTACIÓN DE GRAFO DE FLUJO

Es una sencilla notación para la representación del flujo de control, denominada *grafo de flujo*, mediante esta notación cada construcción estructurada tiene su correspondiente símbolo en el grafo de flujo.

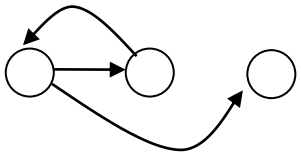
SECUENCIA



ALTERNATIVA



ITERATIVA PRE-CONDICIONAL

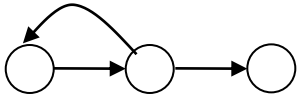


PROGRAMACIÓN I

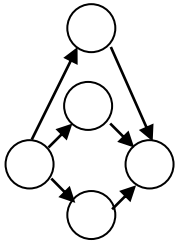
TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

ITERATIVA

POS-CONDICIONAL

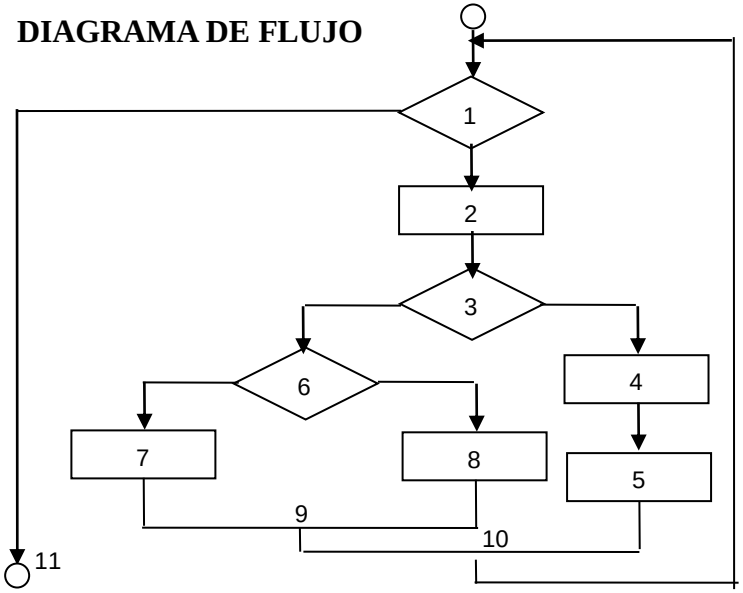


CASE

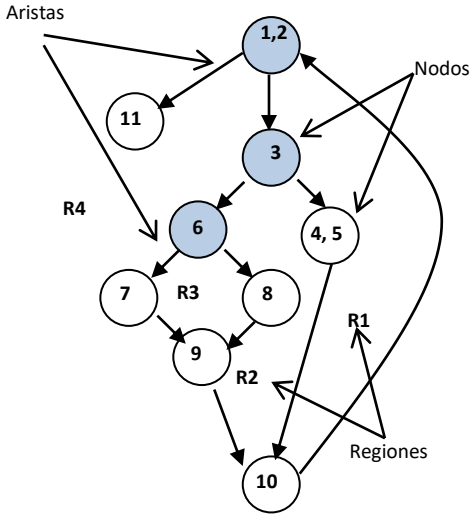


CADA CIRCULO REPRESENTA UNA O MAS SENTENCIAS.

DIAGRAMA DE FLUJO



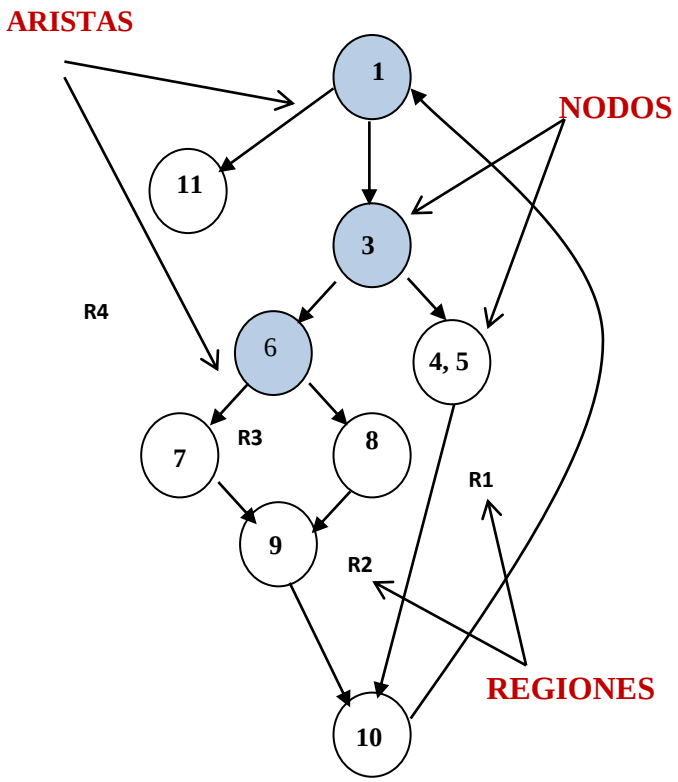
GRAFO DE FLUJO



PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

GRAFO DE FLUJO



Cada CÍRCULO, denominado *NODO* del grafo de flujo, representa una o mas sentencias procedimentales. Un solo nodo puede corresponder a una secuencia de *cuadros* de proceso y a un rombo de decisión.

Las flechas del grafo de flujo, denominadas *ARISTAS o ENLACES*, representan el flujo de control y son análogas a las flechas del diagrama de flujo.

Una arista debe terminar en un nodo, incluso aunque le nodo no represente ninguna sentencia procedimental.

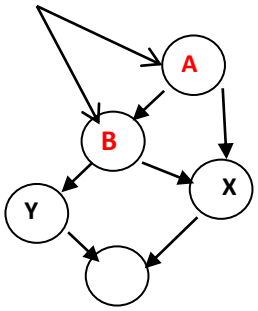
Las áreas delimitadas por aristas y nodos se denominan *REGIONES.* Cuando contabilizamos las regiones incluimos el área exterior al grafo, contando como otra región más.

PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

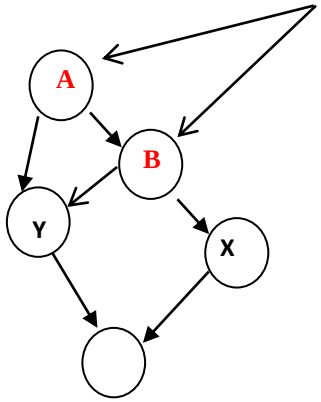
Cuando en un diseño procedimental se encuentran *CONDICIONES COMPUESTAS*, la generación del grafo de flujo se hace un poco mas complicada. Una condición compuesta se da cuando aparecen uno o más operadores lógicos (OR, AND) en una sentencia condicional. Cada nodo que contiene una *CONDICIÓN* se denomina *NODO PREDICADO* y esta caracterizado porque dos o mas aristas emergen de él

NODOS PREDICADOS



SI (A OR B)
ENTONCES X
SINO Y
FINSI

NODOS PREDICADOS



SI (A AND B)
ENTONCES X
SINO Y
FINSI

PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

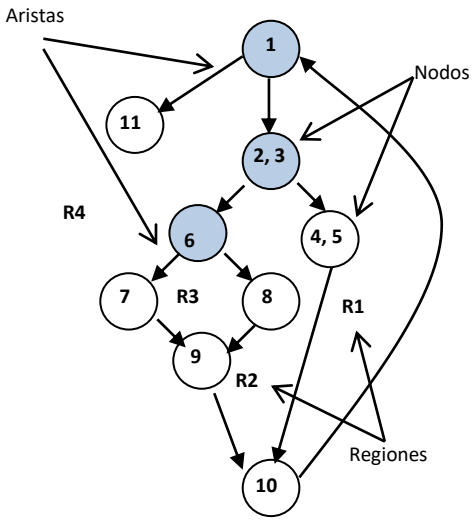
PASO 2. RUTAS INDEPENDIENTES DEL PROGRAMA

Un *CAMINO INDEPENDIENTE* es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición.

En términos de *GRAFO DE FLUJO*, un camino independiente esta constituido por lo menos por una arista que no haya sido recorrida anteriormente.

Por ejemplo, para el siguiente grafo de flujo, los caminos independientes son:

GRAFO DE FLUJO



CAMINO 1: 1, 11

CAMINO 2: 1, 2-3, 4-5, 10 . . .

CAMINO 3: 1, 2-3, 6, 7, 9, 10 . . .

CAMINO 4: 1, 2-3, 6, 8, 9, 10 . . .

CAMINO 5: 1, 2-3, 4-5, 10, 1, 11

EL 5 NO SE CONSIDERA UN CAMINO INDEPENDIENTE, ya que es simple mente una combinación de caminos ya especificados y no recorre ninguna nueva arista.

Los caminos 1, 2, 3 y 4 definidos anteriormente componen un CONJUNTO BÁSICO para el grafo, y se garantiza que se ejecutará al menos una vez cada sentencia del programa y que cada condición se habrá ejecutado en sus vertientes verdadera y falsa.

PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

COMPLEJIDAD CICLOMÁTICA

¿Como sabemos cuantos son los caminos a buscar?

El cálculo de la *COMPLEJIDAD CICLOMÁTICA* nos da la respuesta.

La complejidad ciclomática (V(G)) está basada en la teoría de grafos y nos da una métrica de software extremadamente útil, que aporta una valoración cuantitativa de la complejidad lógica de un programa.
Se calcula de tres formas:

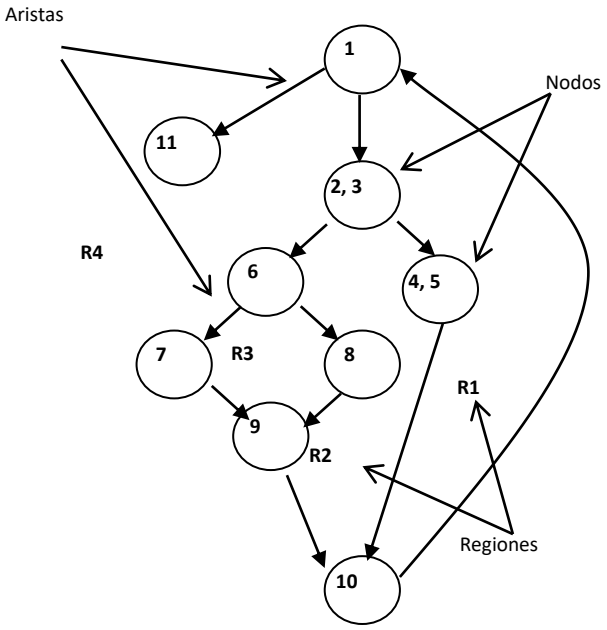
- **El número de regiones del grafo de flujo coincide con la complejidad ciclomática.**
- **V(G) = A – N + 2, donde A es el numero de aristas del grafo de flujo y N es el numero de nodos del mismo.**
- **V(G) = P + 1, donde P es el número de nodos predicado contenidos en el grafo de flujo G.**

PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

COMPLEJIDAD CICLOMÁTICA

GRAFO DE FLUJO



La complejidad ciclomática para este grafo es:

- **EL GRAFO DE FLUJO TIENE 4 REGIONES.**
- **V(G) = 11 ARISTAS – 9 NODOS + 2 = 4**
- **V(G) = 3 NODOS PREDICADO + 1 = 4**

LA COMPLEJIDAD CICLOMÁTICA DEL GRAFO ES 4.

Lo más notable es que el valor de V(G) proporciona el límite superior del número de rutas independientes que forman el conjunto básico, en consecuencia, ofrece un límite superior del número de pruebas que debe diseñarse y ejecutarse para garantizar la cobertura de todas las instrucciones del programa.

PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

PASO 3. OBTENCIÓN DE LOS CASOS DE PRUEBA

- **Es necesario seleccionar los datos de manera tal que se establezcan apropiadamente las condiciones de los nodos predicados, a medida que se prueba cada ruta.**
- **Cada caso de prueba se ejecuta y compara con los resultados esperados.**
- **Completada la prueba se puede estar seguro de que todas las instrucciones del programa se han ejecutado por lo menos una vez.**

Una vez ejecutados los casos de prueba y comparados los resultados obtenidos con los esperados, el responsable de la prueba podrá estar seguro de que todas las sentencias del programa se han ejecutado por lo menos una vez.

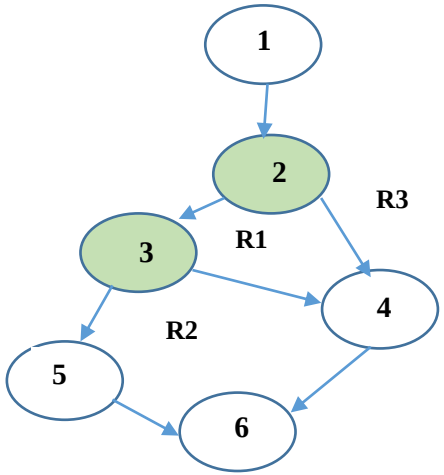
PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

EJEMPLO

Diseñar el conjunto de casos de prueba mediante el método de la complejidad ciclomática para el siguiente código:

```
R:=0;
SI ( X < 0 ) OR ( Y < 0)
        ENTONCES
        MOSTRAR  “X e Y deben ser positivos”
        ELSE
        R= ( x + y ) / 2
        MOSTRAR  “la media es: ” R
FINSI
```



COMPLEJIDAD CICLOMÁTICA

Regiones= 3

V(G)= Aristas-Nodos*+2 7-6+2= 3

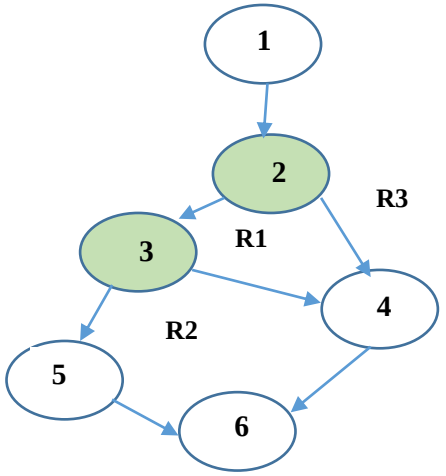
V(G)= Nodos predicados +1 2+1= 3

PROGRAMACIÓN I

TÉCNICAS DE PRUEBA DE SOFTWARE – PRUEBAS DE CAJA BLANCA

CAMINO 1: 1, 2, 4, 6
CAMINO 2: 1, 2, 3, 4, 6
CAMINO 3: 1, 2, 3, 5, 6

```
R:=0;
SI ( X < 0 ) OR ( Y < 0)
      ENTONCES
      MOSTRAR  "X e Y deben ser positivos"
      ELSE
      R= ( x + y ) / 2
      MOSTRAR  "la media es: " R
FINSI
```



Diseño de Casos de Prueba

CAMINO 1	X = -2	Y	SALIDA ESPERADA "x e y deben ser positivos"
CAMINO 2	x= 5	Y= -3	"x e y deben ser positivos"
CAMINO 3	X=4	Y=10	R=(X+Y)/2 "La media es" R

EJERCICIO

Leer 100 números enteros, mostrar el mensaje “Nulo” o “Negativo”, según corresponda y la cantidad de números entre 100 y 300.

```
Comienzo
i:=0;
MIENTRAS (i   <=  100 )
Leer Num
      SI  (Num  =  0)  ENTONCES
                  MOSTRAR  “Nulo”
                        ELSE
                     SI (Num   <  0) ENTONCES
                                    MOSTRAR “Negativo”
                                  ELSE
                                SI (Num  > = 100 AND Num <=  300) ENTONCES
                                                                  cant= cant +1
                                FINSI
                        FINSI
      FINSI
I=I+1
FINMIENTRAS
MOSTRAR  cant
FIN
```

BIBLIOGRAFIA

- Ingeniería de software. Un enfoque práctico; Roger Pressman, sesta edición
Cap. 5 pag. 122 a 125.; Cap. 14 pag. 419 a 438

- Ingeniería de software. Un enfoque práctico; Roger Pressman, séptima edición
Cap. 4 pag. 94 a 96.; Cap. 18