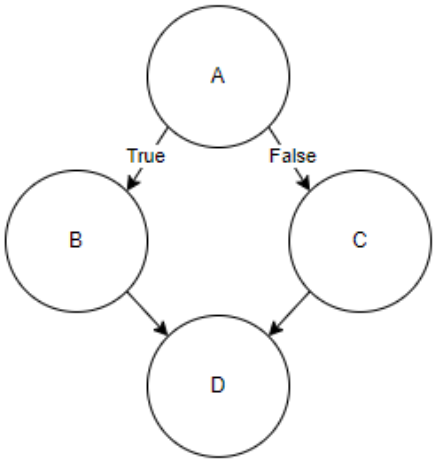
Práctica VI - Caja blanca

Programa 1

Código

```
If (a>1) and (b>5) and (c<2) then
        x=x+1;
else
        x= x-1;
end
```

Grafo



- A: representa el código hasta el if, hasta la bifurcación.
- B: Sentencia de condición cierta de A.
- C: Sentencia de condición falsa de A.
- D: Código Final.

Complejidad

V(G)= A - N + 2 = 4 - 4 + 2 = 2 (Aristas – Nodos + 2)
V(G) = nps + 1 = 1+1 = 2 (Nodos Predicado + 1)
Programa Simple

Conjunto básico de caminos

1: A -> B -> D
2: A -> C -> D

Casos de prueba

A	B	C	Camino	Resultado
2	6	1	1	x=x+1
1	6	1	2	x=x-1
2	5	1	2	x=x-1
2	6	2	2	x=x-1

Explicación

Se pueden ver dos caminos simples, uno donde una condición es cierta y otro donde la condición es falsa. Gracias a los casos de prueba y al código podemos ver que en el caso de que una de las variables no cumpla la condición seguirá el camino donde la condición es falsa.

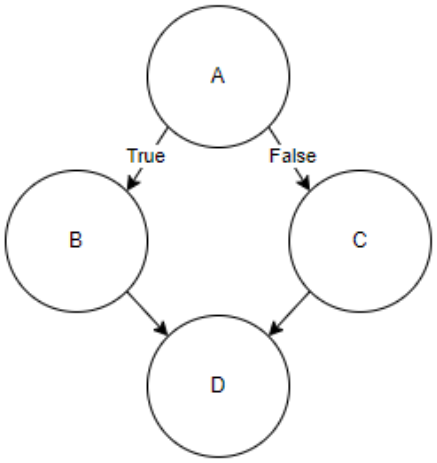
Nota: Hice así la condición porque me parece mucho más sencillo para realizar los casos de prueba ya que la condición en sí es una sola, siendo un conjunto de operadores lógicos.

Programa 2

Código

```
If (a>1) or (b>5) or (c<2)
        then x=x+1;
else
        x= x-1;
end
```

Grafo



- A: representa el código hasta el if, hasta la bifurcación.
- B: Sentencia de condición cierta de A.
- C: Sentencia de condición falsa de A.
- D: Código Final.

Complejidad

V(G)= A - N + 2 = 4 - 4 + 2 = 2 (Aristas – Nodos + 2)
V(G) = nps + 1 = 1+1 = 2 (Nodos Predicado + 1)
Programa Simple

Conjunto básico de caminos

1: A -> B -> D
2: A -> C -> D

Casos de prueba

A	B	C	Camino	Resultado
2	5	2	1	x=x+1
1	6	2	2	x=x+1
1	5	1	2	x=x+1
1	5	2	2	x=x-1

Explicación

Lo mismo que el programa 1 solo que con condiciones OR haciendo que en los casos de prueba el camino sea seleccionado si una o más condiciones son ciertas.

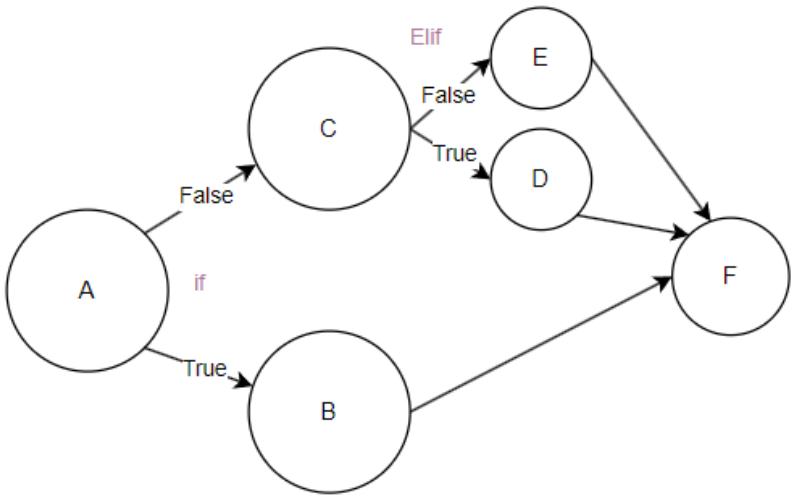
Programa 3

Código

```
Import java.io.*;
Public class Maximo {
        public static void main (String args[]) throws IOException
        {
                BufferedReader entrada = new BufferedReader (new InputStreamReader (System.in));
                int x,y,z,max;
                System.out.println("Introduce x,y,z: ");
                x = Integer.parseInt (entrada.readLine());
                y = Integer.parseInt (entrada.readLine());
                z = Integer.parseInt (entrada.readLine());
                if (x>y && x>z)
                        max = x;
                else{
                        if (z>y) max = z; max = y; else
        } System.out.println ("El máximo es "+ max);
        } //main
}
```

Pide números y los guarda en las variables x y z, según las condiciones le asigna a max el valor de una de las variables, las condiciones intentan ver cual de las variables es mayor.

Grafo



- A: representa el código hasta el if, incluyendo las lecturas.
- B: Sentencia de condición cierta de A.
- C: Sentencia de condición falsa de A, además representa un else con un if.
- D: Sentencia de condición cierta del if en C.
- E: Sentencia de condición cierta del if en C.
- F: Resto del código y finalización.

Complejidad

V(G)= A - N + 2 = 7 - 6 + 2 = 3 (Aristas – Nodos + 2)
V(G) = nps + 1 = 2 + 1 = 3 (Nodos Predicado + 1)
Programa Simple

Conjunto básico de caminos

1: A -> B -> F (x es mayor)
2: A -> C -> D -> F (z es mayor)
3: A -> C -> E -> F (y es mayor)

Casos de prueba

X	Y	Z	Camino	Resultado
5	1	1	1	max = x
1	1	5	2	max = z
1	5	1	3	max = y
1	1	1	3	max = y

Explicación

Podemos ver por los casos de prueba que cumple con su función de ver cual es la variable máxima, en el caso de que todas sean iguales se asigna a max el valor de y

Programa 4

Código

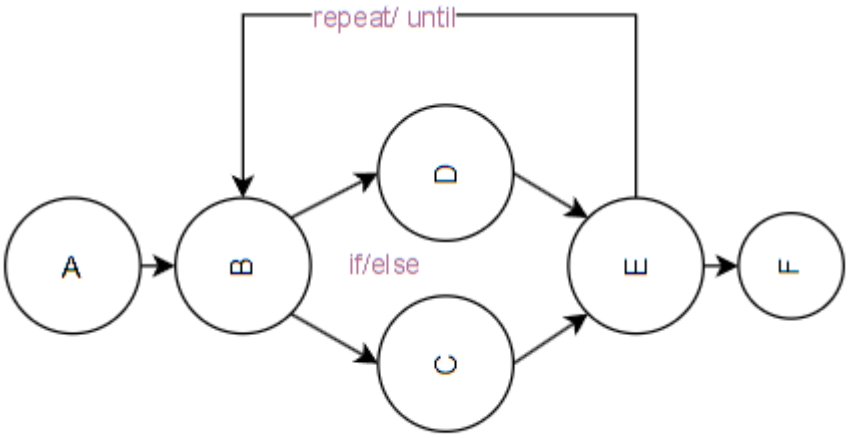
```
function obtener_media : real ;

var
      n, suma, conta, suma2, total_num : integer ;

begin
      read( n ) ;
      repeat
            if (n >= 20 and n <= 50) then
                  suma := suma + n ;
                  conta := conta + 1
            else
                  suma2 := suma2 + n ;
            total_num := total_num + 1 ;
            read (n) ;
      until n = 0 ;
      obtener_media := suma / conta ;
      write (total_num, suma2) ;
end ;
```

Lee un número, si es mayor que 20 y menor que 50 a una variable suma el valor de n, además se le incrementa un contador, si no es cierta la condición a suma2 se le añade n, después se incrementa otro contador diferente, luego vuelve a leer, todo hasta que el usuario incluya un 0.
Después del bucle se obtiene la media con suma y su contador y después se escribe el contador 2 y la suma2 (No entiendo esta sintaxis)

Grafo



- A: representa el código hasta el if dentro del repeat.
- B: Es el repeat + es el if.
- C: Sentencia de condición cierta de B
- D: Sentencia de condición falsa del if en B.
- E: Representa el resto del código después de la condición de B + la condición del bucle.
- F: Resto del código una vez salido del bucle y finalización.

Complejidad

V(G)= A - N + 2 = 7 - 6 + 2 = 3 (Aristas – Nodos + 2)
V(G) = nps + 1 = 2 + 1 = 3 (Nodos Predicado + 1)
Programa Simple

Conjunto básico de caminos

1: AB -> C -> E -> F
2: AB -> D -> E -> F
3: AB -> C -> E -> B -> C -> E -> F
3: AB -> D -> E -> B -> D -> E -> F

Casos de prueba

N,{N iteraciones}	Camino	Resultado media	Resultado otros
25, 0	1	media = 25 / 1	total_num = 1 | suma2 = 0
10, 0	2	media = 0 / 0	total_num = 1 | suma2 = 10
25,25,0	3	media = 50 / 2	total_num = 2 | suma2 = 0
10,10,0	4	media = 0 / 0	total_num = 2 | suma2 = 20

Explicación

Se puede ver que si no hay valores entre 20 y 50 no se incrementa el contador, resultando en una división por 0. En los demás casos se calcula la media.

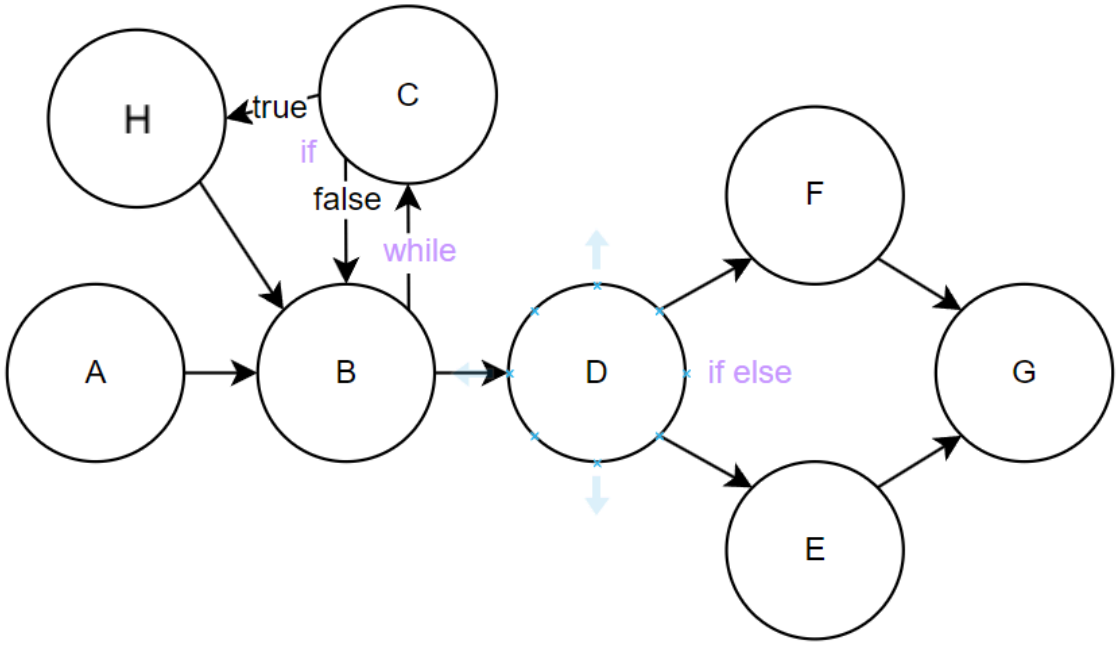
Programa 5

Código

```
public static double ReturnAverage(int value[],
                        int AS, int MIN, int MAX){
  /*
   Function: ReturnAverage  Computes the  average
   of all  those  numbers in the  input array  in
   the  positive  range  [MIN, MAX]. The  maximum
   size  of the array is AS. But, the  array size
   could be smaller than AS in which case the end
   of input is represented by -999.
  */
     int i, ti, tv, sum;
     double av;
     i = 0; ti = 0; tv = 0; sum = 0;
     while (ti < AS && value[i] != -999) {
         ti++;
         if (value[i] >= MIN && value[i] <= MAX) {
            tv++;
            sum = sum + value[i];
         }
         i++;
     }
     if (tv > 0)
        av = (double)sum/tv;
     else
        av = (double) -999;
     return (av);
}
```

Función que recibe un array, su tamaño y dos valores, un mínimo y un máximo. Recorre el array con un bucle y si este valor está entre Min y Max incrementa el contador y lo añade a la suma.
Una vez realizado el bucle si se han realizado sumas, es decir, el contador se ha incrementado se hace la media, si no se le asigna al valor de media -999 y se devuelve el valor.

Grafo



- A: representa el código..
- B: Es el while.
- C: Condición if dentro del while.
- H: Sentencia de condición cierta del if en C.
- D: If después del bucle.
- E: Sentencia de condición cierta del if en D.
- F: Sentencia de condición falsa del if en D.
- G: Resto del código una vez salido del bucle y finalización.

Complejidad

V(G)= A - N + 2 = 10 - 8 + 2 = 4 (Aristas – Nodos + 2)
V(G) = nps + 1 = 3 + 1 = 4 (Nodos Predicado + 1)
Programa Simple

Conjunto básico de caminos

1: A -> B -> D -> E -> G // No se puede cumplir
2: A -> B -> D -> F -> G
3: A -> B -> C -> B -> D -> E -> G // No se puede cumplir
4: A -> B -> C -> B -> D -> F -> G
5: A -> B -> C -> H -> B -> D -> E -> G
6: A -> B -> C -> H -> B -> D -> F -> G //No se puede cumplir

Casos de prueba

array	AS	Min,Max	Camino	Resultado
{}	0	10,10	2	av = -999
{1,2,3,4,5}	5	0,5	5	av = 3
{1,2,3,4,5}	5	10,10	4	av = -999
{1,2,0,7,5}	5	1,10	5	av = 3
{1,2,3,4,5}	10	1,5	5	av=3

Explicación

Podemos ver que para que en el if se cumpla la condición se debe de pasar por el if del while y viceversa. Esto hace que algunos de los caminos no se puedan cumplir en ningún caso.

Programa 6

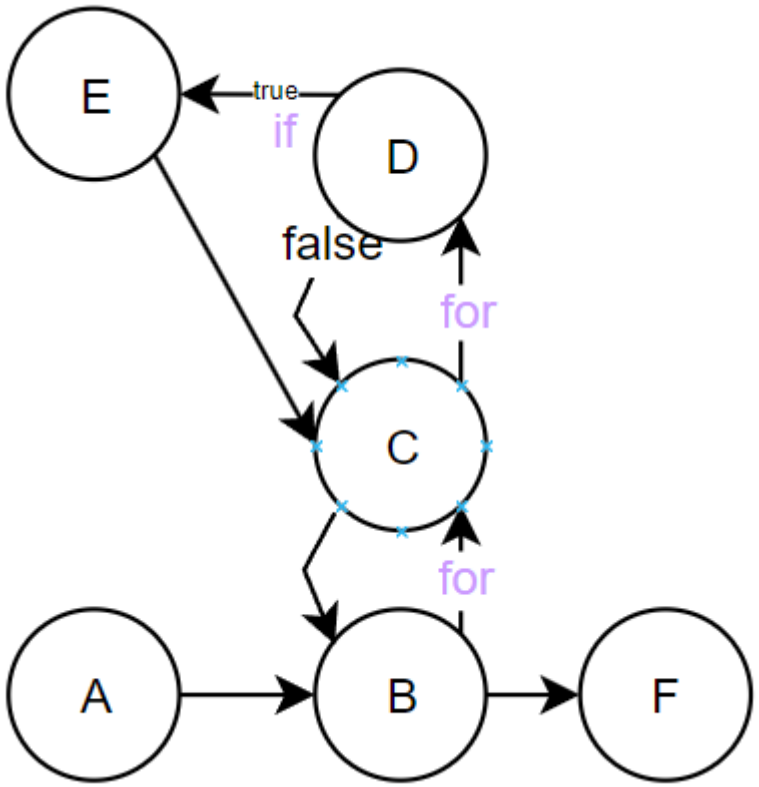
Código

```
Calcula cohesión (int nt1, nt2; String tok1[ ],
tok2[ ])

1 numAdh = 0
2 Para i de 0 hasta nt1-1
3     Para j de 0 hasta nt2-1
4         Si tok1[i]=tok2[j] entonces
5             numAdh = numAdh +1
6 total = tok1 + tok2 – numAdh
7 cohesión = numAdh / total
8 regresa cohesión
```

Función que toma 2 valores enteros y dos strings. Se hacen dos bucles for para comprobar cuántas veces un carácter de un string coincide con el carácter del otro string, esto incrementa un contador. Después se calcula un total y se calcula la cohesión la cual se retorna.

Grafo



- A: representa el inicio del código.
- B: Es el bucle for.
- C: For dentro del for de B.
- D: If dentro de los for anidados.
- E: Sentencia de condición cierta del if en D.
- F: Resto del código una vez salido del bucle y finalización.

Complejidad

V(G)= A - N + 2 = 8 - 6 + 2 = 4 (Aristas – Nodos + 2)
V(G) = nps + 1 = 3 + 1 = 4 (Nodos Predicado + 1)
Programa Simple

Conjunto básico de caminos

1: A -> B -> F
2: A -> B -> C -> D -> C -> B -> F
3: A -> B -> C -> D -> E -> C -> B -> F

Casos de prueba

nt1,nt2	tok1	tok2	Camino	Resultado
3,9	“tok”	“tttoookkk”	3	9/3
0,0	“”	“”	1	0/0
3,3	“tok”	“raw”	2	0/6

Explicación

Como se puede ver en los casos de prueba si el tamaño es 0 no pasará por los for lo que resultará en una división por 0. En los demás calcula el resultado como esperado.

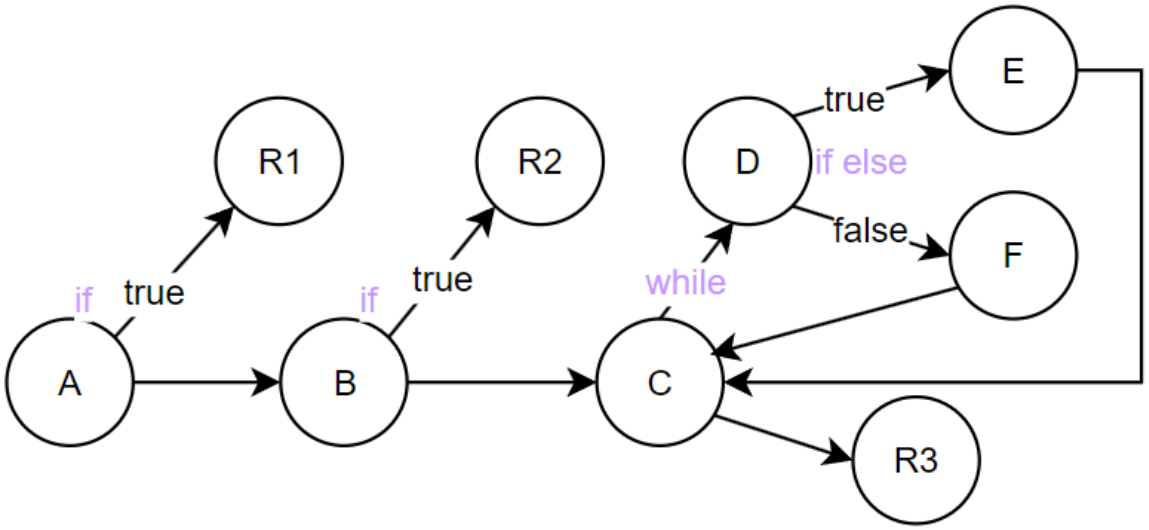
Programa 7

Código

```
1  lee x, y
2  Si (x<=0) o (y<=0) entonces
3      Escribe “deben ser no negativos;
4      regresa -1
5  Si (x=1) o (y=1) entonces
6       regresa 1
7  Mientras (x <> y)
8       Si (x>y) entonces
9             x = x - y
10     de otro modo y = y – x
11 regresa x
```

Lee dos números, comprueba si no son negativos, devuelve -1 si es negativo, 1 si ambos valores son 1 y mientras sean diferentes si x es mayor se le resta y, si y es mayor se le resta x. Devuelve x.

Grafo



- A: representa el inicio del código hasta el primer if.
- B: Es el segundo if.
- C: Bucle while.
- D: Código dentro del while.
- E: Sentencia de condición cierta del if en D.
- F: Sentencia de condición falsa del if en D.
- R1: Sentencia de condición cierta en A + retornar valor.
- R2: Sentencia de condición cierta en B + retornar valor.
- R3: Final del código de la función después del while y retornar valor.

Complejidad

V(G)= A - N + 2 = 10(+2 que faltan) - 9 + 2 = 5 (Aristas – Nodos + 2)
V(G) = nps + 1 = 4 + 1 = 5 (Nodos Predicado + 1)
Nota: Al retornar en 3 ocasiones esto hace que las aristas de los condicionales no vuelvan a la ejecución normal dado que los return rompen con esta.
Programa Simple

Conjunto básico de caminos

1: A -> R1
2: A -> B -> R2
3: A -> B -> C -> R3
4: A -> B -> C -> D -> E -> C -> R3
5: A -> B -> C -> D -> F -> C -> R3

Casos de prueba

x	y	Camino	Return
-1	1	1	-1
1	-1	1	-1
-1	-1	1	-1
1	8	2	1
8	1	2	1
1	1	2	1
3	3	3	3
4	2	4	2
2	4	5	2

Explicación

Se puede ver que cumple con con la ejecución esperada, solamente teniendo en cuenta que los return en el caso 1 y 2 rompen con la ejecución de la función.

Programa 8

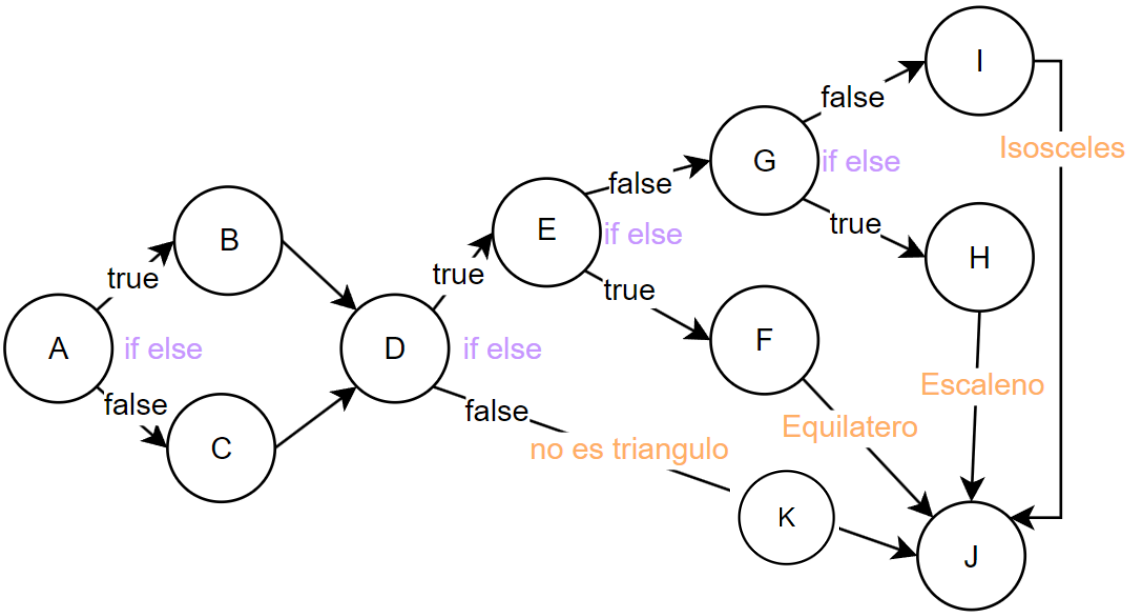
Código

need to decide whether to associate nodes with nonexecutable statements such as variable and type declarations; here we do not.

```
1. Program triangle2 'Structured programming version of simpler specification
2. Dim a,b,c As Integer
3. Dim IsATriangle As Boolean
       'Step 1: Get Input
4. Output("Enter 3 integers which are sides of a triangle")
5. Input(a,b,c)
6. Output("Side A is ",a)
7. Output("Side B is ",b)
8. Output("Side C is ",c)
       'Step 2: Is A Triangle?
9. If (a < b + c) AND (b < a + c) AND (c < a + b)
10.     Then IsATriangle = True
11.     Else IsATriangle = False
12 EndIf
       'Step 3: Determine Triangle Type
13. If IsATriangle
14.     Then   If (a = b) AND (b = c)
15.                Then Output ("Equilateral")
16.                Else    If (a ≠ b) AND (a ≠ c) AND (b ≠ c)
17.                            Then   Output ("Scalene")
18.                            Else   Output ("Isosceles")
19.                        EndIf
20.            EndIf
21.     Else   Output("Not a Triangle")
22. EndIf
23. End triangle2
```

Pide tres valores y comprueba si es un triángulo, luego determina el tipo de triángulo que es.

Grafo



- A: representa el inicio del código hasta el primer if.
- B: Sentencia de condición cierta del if en A.
- C: Sentencia de condición falsa del if en A.
- D: Segundo If.
- E: Sentencia de condición cierta del if en D, también el if que comprueba los equiláteros.
- K: Sentencia de condición falsa del if en D.
- F: Sentencia de condición cierta en E.
- G: Sentencia de condición cierta en E, también el if que controla si es escaleno o isósceles.
- H:Sentencia de condición verdadera del if en D.
- I:Sentencia de condición falsa del if en D.
- J: Resto del código después de los if y final.

Complejidad

V(G)= A - N + 2 = 14 - 11 + 2 = 5 (Aristas – Nodos + 2)
V(G) = nps + 1 = 4 + 1 = 5 (Nodos Predicado + 1)
Programa Simple

Conjunto básico de caminos

1: A -> C -> D -> K -> J
2: A -> B -> D -> E -> F -> J
3: A -> B -> D -> E -> G -> H -> J
4: A -> B -> D -> E -> G -> I -> J
// Siempre que pase por C pasará por K y siempre que pasa por B pasa por E

Casos de prueba

A	B	C	Camino	Return
10	2	2	1	No es triángulo
2	10	2	1	No es triángulo
2	2	10	1	No es triángulo
2	2	2	2	Equilátero
2	4	3	3	Escaleno
2	3	3	4	Isósceles

Explicación

Se puede ver gracias a los casos de prueba que siempre que no sea triángulo, pasará por un camino y siempre que lo sea pasará por otro, es decir que hay caminos que aunque existan, no se pueden alcanzar.