Ingeniería del Software

TEMA 5: PRUEBAS

EJERCICIOS RESUELTOS

Ejercicio 1

Dada la siguiente función en lenguaje Java:

```
1  boolean fechaCorrecta (int dia, int mes, int año){
2    int limite;
3    boolean bisiesto;
4    if (año<2005){
5      bisiesto=((año%4==0)&&(año%100!=0))||(año%400==0) ;
6      switch (mes) {
7        case 1:
8        case 3:
9        case 5:
10       case 7:
11       case 8:
12       case 10:
13       case 12:limite=31;
14                break;
15       case 2:if (bisiesto)
16                 limite=29;
17              else limite=28;
18              break;
```

```
19          case 4:
20          case 6:
21          case 9:
22          case 11: limite=30;
23                   break;
24          default: return (false);
25       }
26       if ((dia>0) && (dia<=limite))
27          return (true);
28       else return (false);
29       }
30    else return (false);
31 }
```

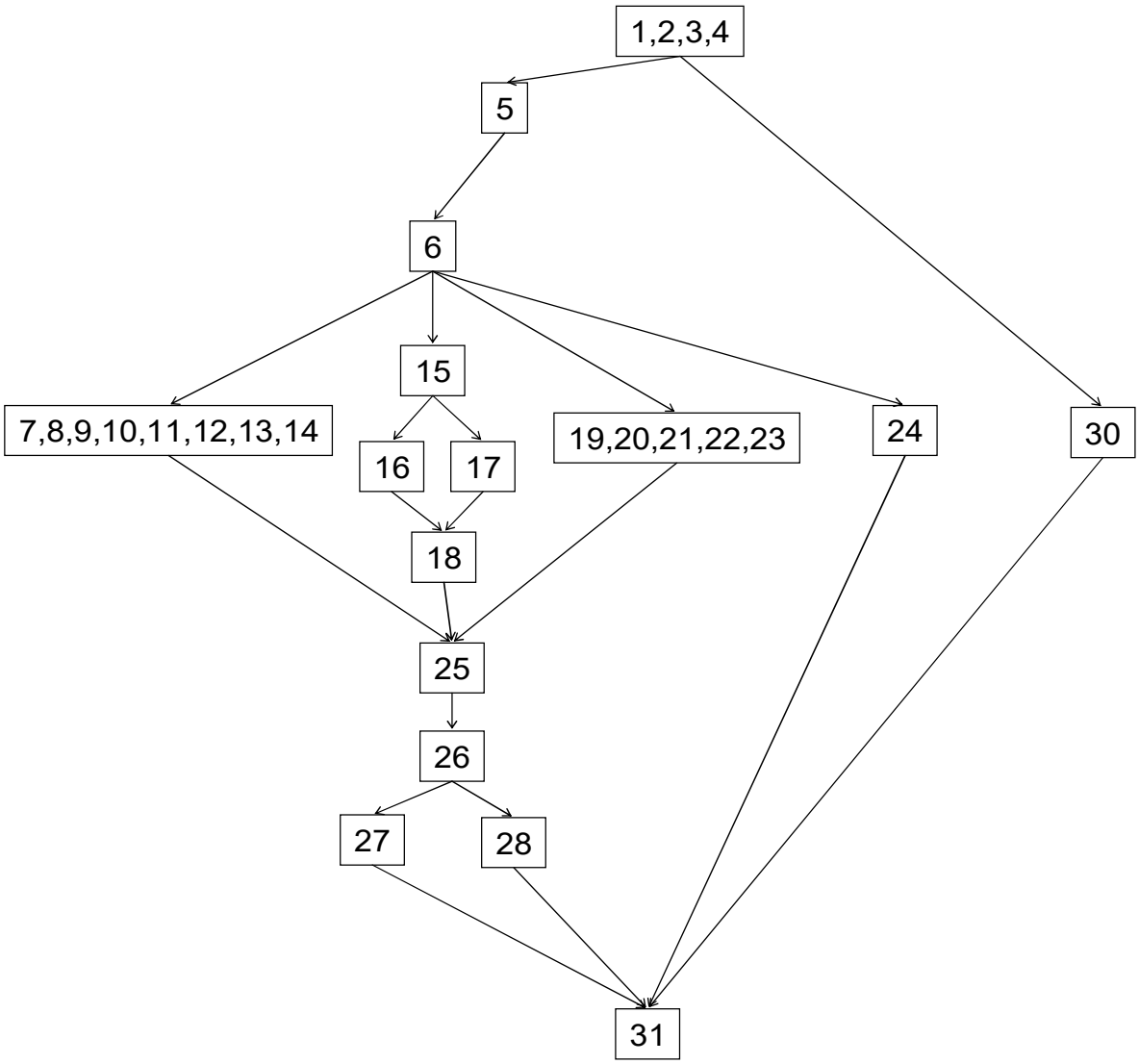
Se pide:

a) Dibujar el grafo de flujo y calcular la complejidad ciclomática. ¿Qué indica el valor obtenido?

b) Identifica los caminos básicos.

Solución:

a)



V(G) = 21 - 16 + 2 = 7

Indica el nº mínimo de casos de prueba necesarios para obtener cobertura de sentencias, es decir, determina el número de caminos básicos linealmente independientes del conjunto básico del programa.

a) Caminos básicos:

```
Camino 1: 1-4, 5, 6, 7-14, 25, 26, 27,31
Camino 2: 1-4, 5, 6, 15, 16, 18, 25, 26, 28, 31
Camino 3: 1-4, 5, 6, 15, 17, 18, 25, 26, 28, 31
Camino 4: 1-4, 5, 19-23, 25, 26, 27, 31
Camino 5: 1-4, 5, 19-23, 25, 26, 28, 31
Camino 6: 1-4, 5, 24, 31
Camino 7: 1-4, 30, 31
```

Ejercicio 2

El siguiente algoritmo en C++ calcula la distancia de Levenshtein, o distancia entre palabras, que es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra.

1	`#include <string>`
	`#include <vector>`
	`#include <algorithm>`

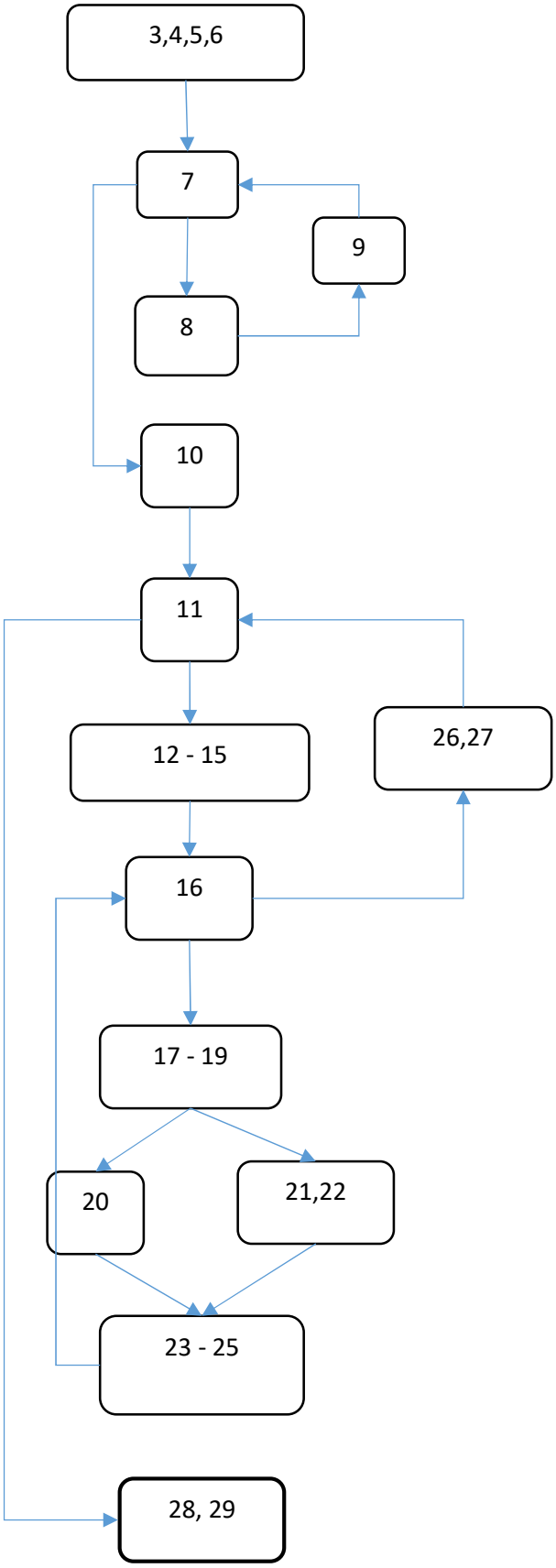
2	`using namespace std;`
3	`int levenshtein(const string &s1, const string &s2)`
4	`{`

5	`int N1 = s1.size();`
	`int N2 = s2.size();`
	`int i, j;`
	`vector<int> T(N2+1);`

6	**`i = 0;`**
7	`while(i <= N2){`
8	`T[i] = i;`
9	**`i++; }`**

```
10      i = 0;
11      while(i < N1)
12      {
13          T[0] = i+1;
14          int corner = i;
15          j = 0
16          while(j < N2)
17          {
18              int upper = T[j+1];
19              if ( s1[i] == s2[j] )
20                  T[j+1] = corner;
21              else
22                  T[j+1] = min(T[j], min(upper, corner)) + 1;
23              corner = upper;
24              j++;
25          }
26          i++;
27      }
28      return T[N2];
29  }
```

a) Dibuja el Grafo de Flujo utilizando la numeración de líneas que se propone para el algoritmo anterior.



b) Calcula la Complejidad Ciclomática al menos de 2 formas distintas.

```
V(G) = Aristas - Nodos  + 2 =  17 aristas - 14 nodos + 2 = 5

V(G) = Regiones Cerradas + 1  = 4 + 1 = 5
```

Ejercicio 3

Dado el siguiente fragmento de programa en java:

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

¿Qué caminos del grafo asociado satisfacen los siguientes casos de prueba en cobertura de condición?

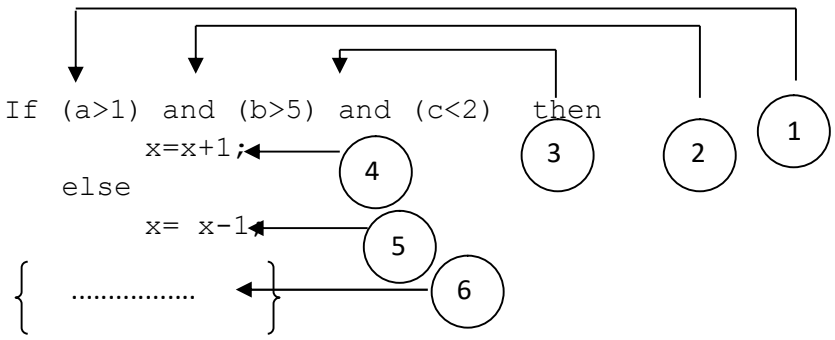
- {a=0, b=11, c=1}
- {a=4, b=4,c=4}
- {a=2, b=6, c=0}

Justifica tus respuestas.

Solución:

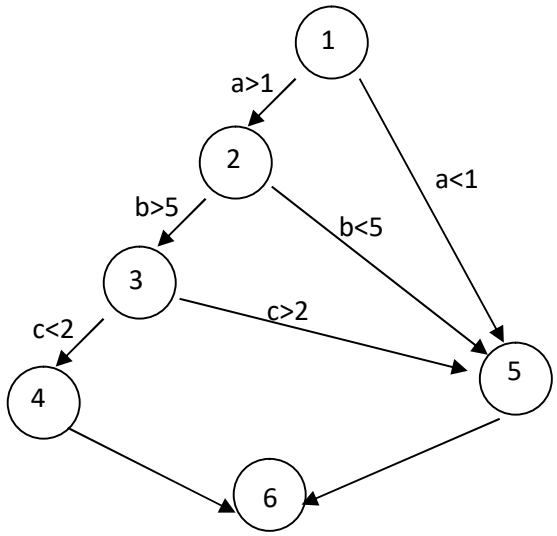
Primero calculamos el grafo de flujo en cobertura de condición y la complejidad ciclomática. De esta manera veremos con más facilidad cuales son los pasos que se ejecutan con los casos de prueba que nos dan.

1. Señalamos en el fragmento de código los nodos:



Incluimos un sexto nodo que hace referencia al código que supuestamente viene después (en el enunciado no lo determina). De esta manera podremos calcular la complejidad ciclomática sin errores.

2. Dibujamos el grafo de flujo:



3. Calculamos la complejidad ciclomática:

V(G)= 8 – 6 + 2 = 4 caminos independientes

Camino 1 ➔ 1 – 5 – 6
Camino 2 ➔ 1- 2 – 5 – 6
Camino 3 ➔ 1 – 2 – 3 – 5 –6
Camino 4 ➔ 1 – 2 –3 – 4 – 6

Con el caso de prueba {a=0, b=11, c=1} ejecutamos el primer camino independiente.
Con el caso de prueba {a=4,b=4,c=4} ejecutamos el segundo camino independiente.

Con el caso de prueba {a=2, b=6, c=0} ejecutamos el cuarto camino independiente.