Curso 2020-2021

Boletín de ejercicios resueltos Tema 8

DISEÑO DE CASOS DE PRUEBA

ANTONIO GARRIDO, ELISEO MARZAL, SOLEDAD VALERO

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
anys
etsinf
Escuela Técnica Superior de Ingeniería Informática

Boletín de ejercicios resueltos Tema 9

Contenido

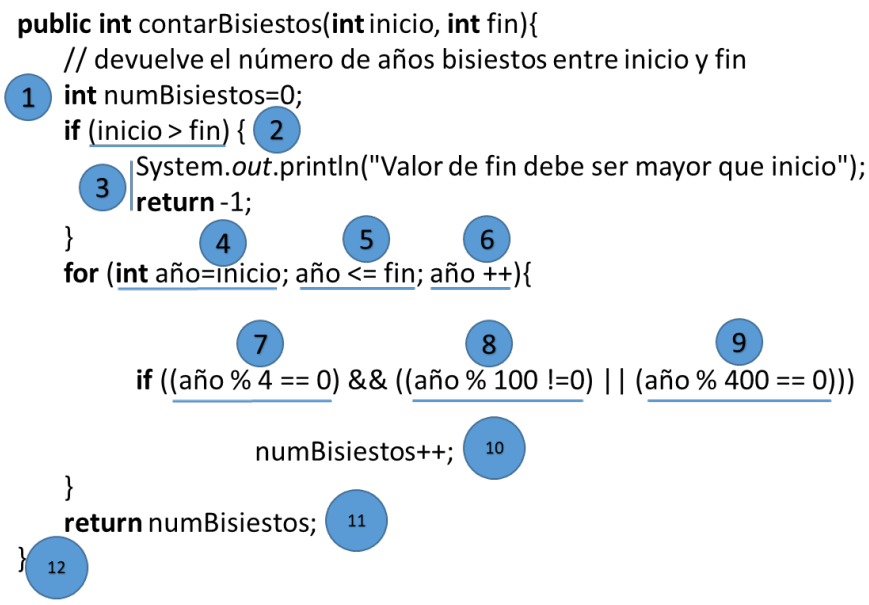
NOTA IMPORTANTE

En este boletín se ofrecen resultados de ejercicios sobre diseño de casos de prueba que plantean una solución a una colección de problemas. Para cada ejercicio se ofrece una posible solución, pero dicha solución no es única y, muy posiblemente, existirán otras soluciones igualmente válidas.

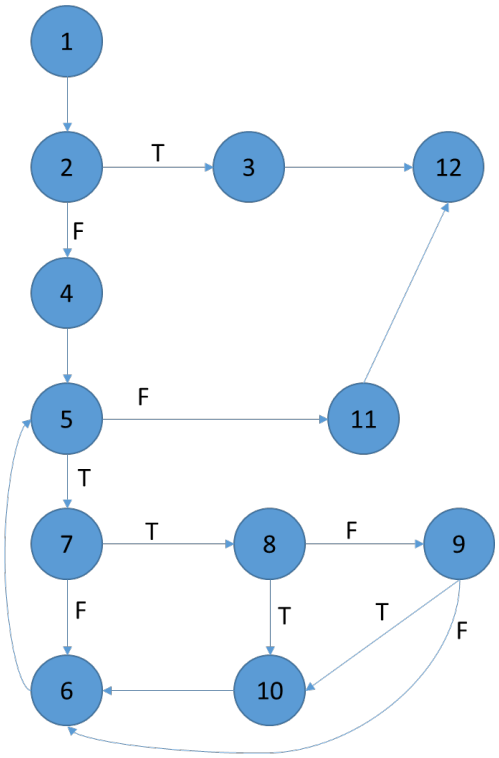
1. Contar años bisiestos

Diseñar los casos de prueba para el siguiente fragmento de código siguiendo la técnica del camino básico (dibuje el grafo de flujo, calcule la complejidad ciclomática, especifique los caminos independientes y los casos de prueba asociados a cada camino).

```
public int contarBisiestos(int inicio, int fin){
	// devuelve el número de años bisiestos entre inicio y fin
	int numBisiestos=0;
	if (inicio > fin) {
		System.out.println("Valor de fin debe ser mayor que inicio");
		return -1;
	}
	for (int año=inicio; año <= fin; año ++){
		if ((año % 4 == 0) && ((año % 100 !=0) || (año % 400 == 0)))
			numBisiestos++;
	}
	return numBisiestos;
}
```



NOTA. Para hacer el diagrama más intuitivo se han añadido las etiquetas 1-2, aunque se podían haber fusionado en una única etiqueta.



V(G) = 6

C1: 1 - 2 - 3 - 12

C2: 1 - 2 - 4 - 5 - 11 - 12

C3: 1 - 2 - 4 - 5 - 7 - 6 - 5 - 11 - 12

C4: 1 - 2 - 4 - 5 - 7 - 8 - 9 - 10 - 6 - 5 - 11 - 12

C5: 1 - 2 - 4 - 5 - 7 - 8 - 10 - 6 - 5 - 11 - 12

C6: 1 - 2 - 4 - 5 - 7 - 8 - 9 - 6 - 5 - 11 - 12

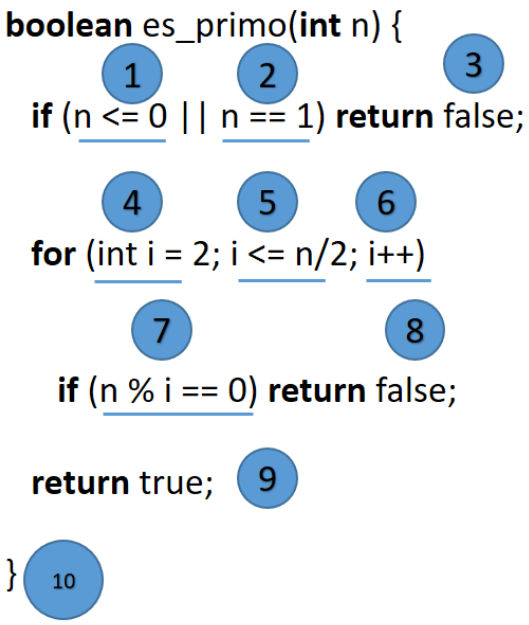
Casos de prueba:

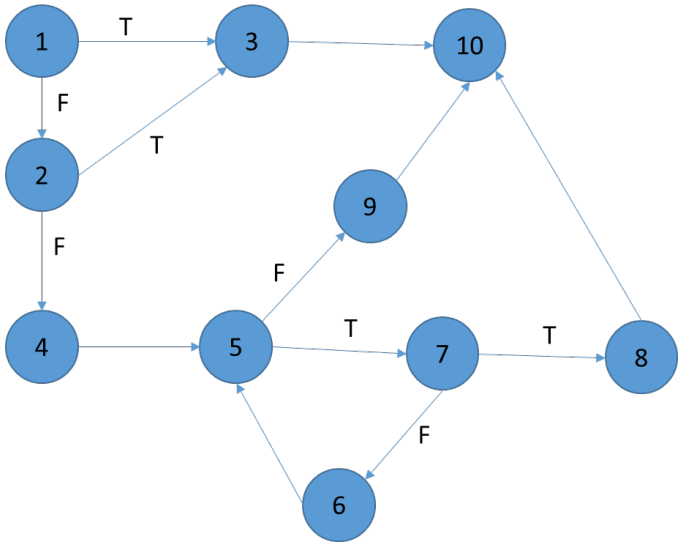
Camino	inicio	fin	retorno
C1	2040	2018	“Valor de fin debe ser mayor…” -1
C2	No existe caso de prueba, pues lo detectaría C1		
C3	2023	2023	0
C4	2000	2000	1
C5	2004	2004	1
C6	2100	2100	0

2. Número primo

Diseñar los casos de prueba para el siguiente fragmento de código siguiendo la técnica del camino básico (dibuje el grafo de flujo, calcule la complejidad ciclomática, especifique los caminos independientes y los casos de prueba asociados a cada camino).

```
boolean es_primo(int n) {
   if (n <= 0 || n == 1) return false;
   for (int i = 2; i <= n/2; i++)
      if (n % i == 0) return false;
   return true;
}
```





V(G) = 5

C1: 1 - 3 -10

C2: 1 - 2 - 3 - 10

C3: 1 - 2 - 4 - 5 - 9 - 10

C4: 1 - 2 - 4 - 5 - 7 - 8 - 10

C5: 1 - 2 - 4 - 5 - 7 - 6 - 5 - 9 – 10

Casos de prueba (se detecta claramente que no se calcula bien si un número es primo):

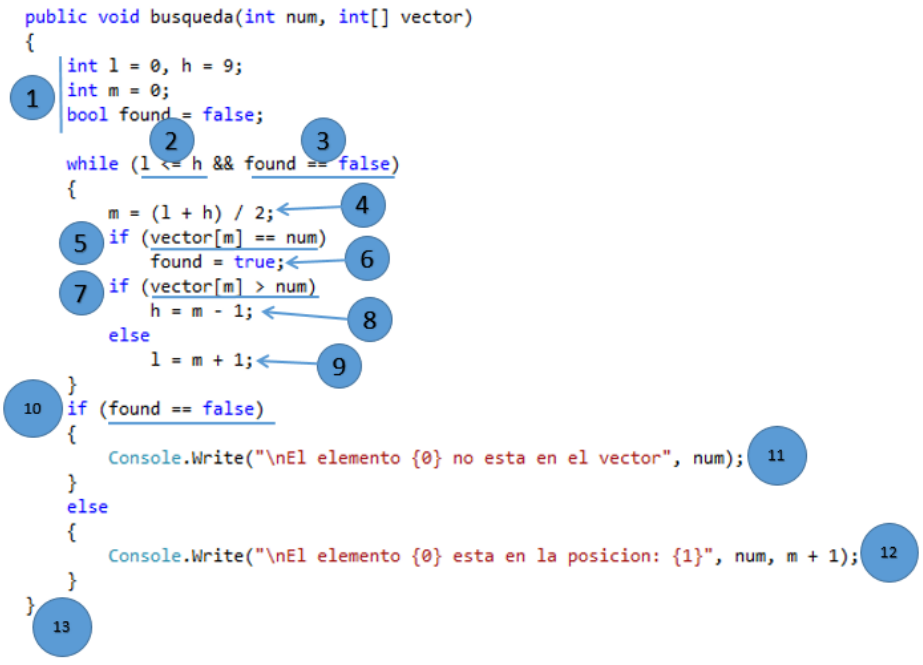
Camino	n	retorno
C1	0	false
C2	1	false
C3	No existe caso de prueba, pues lo detectaría C2	
C4	4	false
C5	No existe caso de prueba. Sería aconsejable probar casos de prueba adicionales a los de esta técnica donde se entre más veces en el bucle	

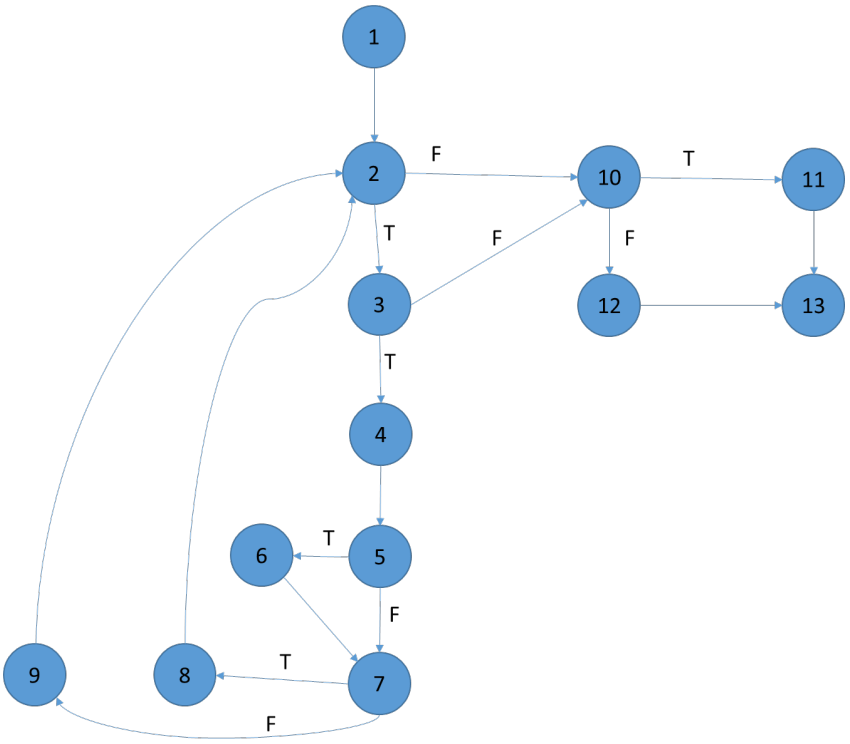
3. Búsqueda de un número

Diseñar los casos de prueba para el siguiente fragmento de código siguiendo la técnica del camino básico (dibuje el grafo de flujo, calcule la complejidad ciclomática, especifique los caminos independientes y los casos de prueba asociados a cada camino).

```
public void busqueda(int num, int[] vector)
{
    int l = 0, h = 9;
    int m = 0;
    bool found = false;

    while (l <= h && found == false)
    {
        m = (l + h) / 2;
        if (vector[m] == num)
            found = true;
        if (vector[m] > num)
            h = m - 1;
        else
            l = m + 1;
    }
    if (found == false)
    {
        Console.Write("\nEl elemento {0} no esta en el vector", num);
    }
    else
    {
        Console.Write("\nEl elemento {0} esta en la posicion: {1}", num, m + 1);
    }
}
```





V(G) = 6

C1: 1 - 2 - 10 - 11 - 13

C2: 1 - 2 - 10 - 12 - 13

C3: 1 - 2 - 3 - 10 - 12 - 13

C4: 1 - 2 - 3 - 4 - 5 - 6 - 7 - 9 - 2 - 3 - 10 - 12 - 13

C5: 1 - 2 - 3 - 4 - 5 - 7 - 8 - 2 - 10 - 11 - 13

Casos de prueba:

Camino	num	vector	salida por pantalla
C1	No existe caso de prueba pues inicialmente “l<=h” siempre		
C2	No existe caso de prueba pues inicialmente “l<=h” siempre		
C3	No existe caso de prueba pues inicialmente “found” vale false		
C4	5	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]	“El elemento 5 está en la posición 5”
C5	No existe caso de prueba. Sería aconsejable probar casos de prueba adicionales a los de esta técnica donde se entre más veces en el bucle para probar que un elemento no esté en el vector		

4. Número doble

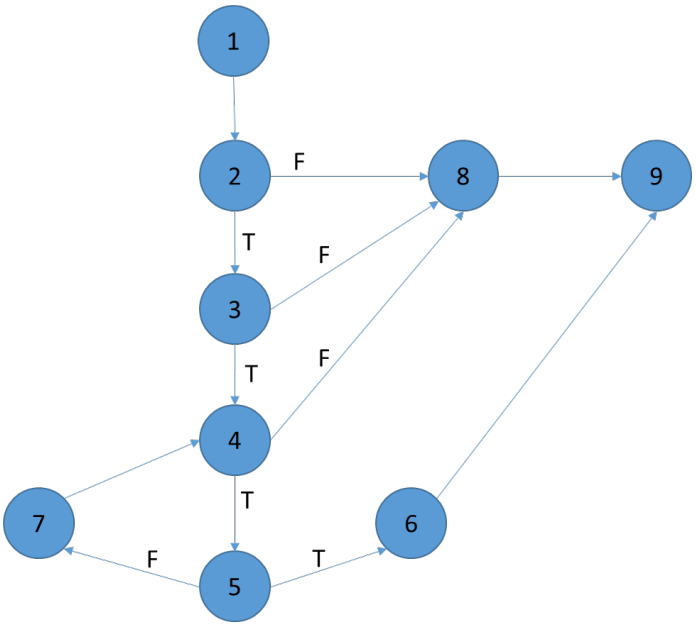
Diseñar los casos de prueba para el siguiente fragmento de código siguiendo la técnica del camino básico (dibuje el grafo de flujo, calcule la complejidad ciclomática, especifique los caminos independientes y los casos de prueba asociados a cada camino).

```
public bool even(int n, int INT_MIN, int INT_MAX)
{
    int k = INT_MIN;

    if ((INT_MIN <= n) && (n <= INT_MAX))
    {
        while (k <= INT_MAX)
        {
            if (n == 2 * k)
                return true;
            k++;
        }
    }
    return false;
}
```

```
public bool even(int n, int INT_MIN, int INT_MAX)
{
    int k = INT_MIN; 1
                  2                  3
    if ((INT_MIN <= n) && (n <= INT_MAX))
    {               4
        while (k <= INT_MAX)
        {             5
            if (n == 2 * k)
                return true; 6
            k++; 7
        }
    }
    return false; 8
} 9
```

NOTA. Para hacer el diagrama más intuitivo se han añadido las etiquetas 1-2, aunque se podían haber fusionado en una única etiqueta.



V(G) = 5

C1: 1 - 2 - 8 - 9

C2: 1 - 2 - 3 - 8 - 9

C3: 1 - 2 - 3 - 4 - 8 - 9

C4: 1 - 2 - 3 - 4 - 5 - 6 - 9

C5: 1 - 2 - 3 - 4 - 5 - 7 - 4 - 8 - 9

Casos de prueba:

Camino	n	INT_MIN	INT_MAX	retorno
C1	1	2	5	false
C2	6	2	5	false
C3	No existe caso de prueba pues para llegar a la etiqueta 4 INT_MIN <= INT_MAX			
C4	4	2	5	true
C5	3	3	3	false

5. Número doble con gestión de excepciones try..catch

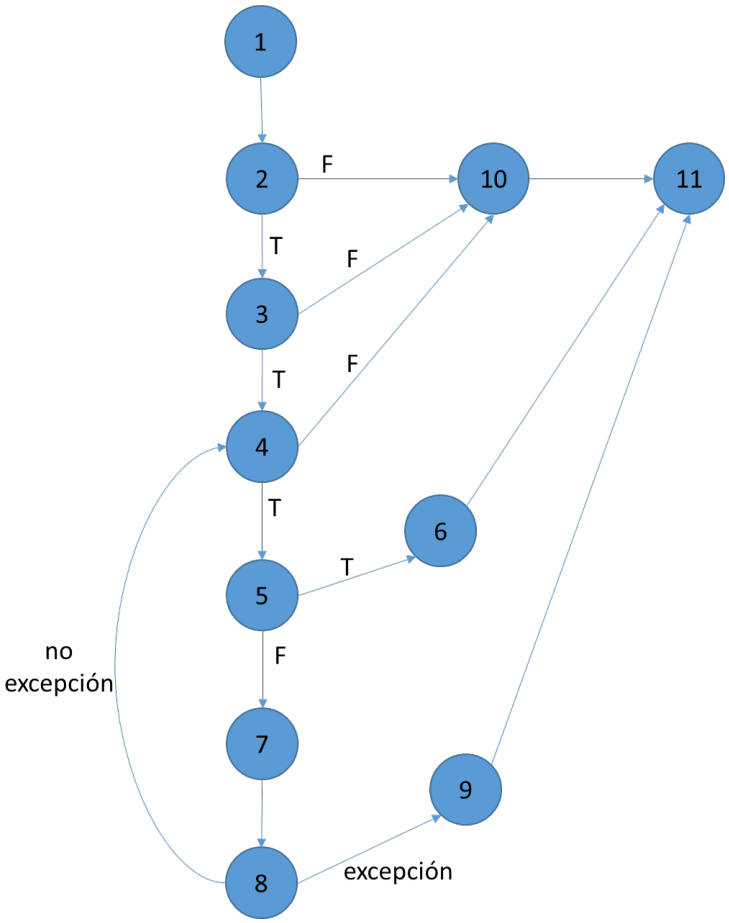
Para el siguiente ejercicio que gestiona excepciones, utilice la técnica del camino básico para dibujar el grafo de flujo, calcular la complejidad ciclomática y especificar los caminos independientes. **NOTA: no se piden los casos de prueba**.

```
public bool even(int n, int INT_MIN, int INT_MAX)
{
    int k = INT_MIN;

    if ((INT_MIN <= n) && (n <= INT_MAX))
    {
        while (k <= INT_MAX)
        {
            if (n == 2 * k)
                return true;
            k++;
            try
            {
                    comprobarValor(k);
            }
            catch
            {
                    return false;
            }
        }
    }
    return false;
}
```

```
 public bool even(int n, int INT_MIN, int INT_MAX)
 {
     int k = INT_MIN; 1
                      2                  3
     if ((INT_MIN <= n) && (n <= INT_MAX))
     {                4
         while (k <= INT_MAX)
         {            5
             if (n == 2 * k)
                 return true; 6
            k++; 7
            try
            {
                 comprobarValor(k); 8
            }
            catch
            {
                  return false; 9
            }
         }
     }
     return false; 10
 } 11
```

Importante: las instrucciones de un bloque try..catch se representan como nodos predicado (condiciones). Es decir, **cada instrucción que pueda disparar una excepción** entre el try..catch puede: 1) disparar dicha excepción, saltando al bloque catch o, 2) continuar con el flujo normal del programa ejecutando la siguiente instrucción.



V(G) = 6

C1: 1 - 2 - 10 - 11

C2: 1 - 2 - 3 - 10 - 11

C3: 1 - 2 - 3 - 4 - 10 - 11

C4: 1 - 2 - 3 - 4 - 5 - 6 - 11

C5: 1 - 2 - 3 - 4 - 5 - 7 - 8 - 9 - 11

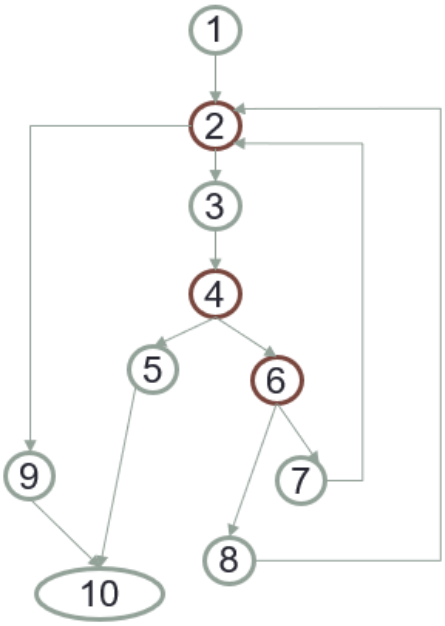
C6: 1 - 2 - 3 - 4 - 5 - 7 - 8 - 4 - 10 – 11

6. Búsqueda binaria

Diseñar los casos de prueba para el siguiente fragmento de código siguiendo la técnica del camino básico (dibuje el grafo de flujo, calcule la complejidad ciclomática, especifique los caminos independientes y los casos de prueba asociados a cada camino).

```
static public int search(char c, char []v)
        {
            int a, z, m;
            a = 0;
            z = v.Length - 1;
            while (a <= z)
            {
                m = (a + z) / 2;
                if (v[m] == c) {
                    return 1;
                }
                else if(v[m] < c)
                {
                    a = m + 1;
                }
                else
                {
                    z = m - 1;
                }
            }
            return 0;
        }
```

```
static public int search(char c, char []v)
        {
            int a, z, m;
1           a = 0;
            z = v.Length - 1;
2           while (a <= z)
            {
                m = (a + z) / 2;  3
4               if (v[m] == c) {
                    return 1;  5
                }
6               else if(v[m] < c)
                {
                    a = m + 1;  7
                }
                else
                {
                    z = m - 1;  8
                }
            }
            return 0;  9
10      }
```



V(G)= 4

Áreas = 4

Nodos Predicado = 3, luego V(G)=3+1=4

Nodos = 10, luego V(G)= 12-10+2 =4

Aristas = 12

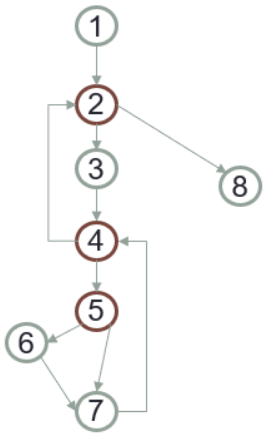
Camino	Entrada	Salida
{1,2,9,10} Cadena vacía	V=“” c=‘a’	0
{1,2,3,4,5,10} En el primer lugar	V=“a” c=‘a’	1
{1,2,3,4,6,7,2,9,10} Cadena con solo un carácter, menor que el objetivo	V=“a” c=‘b’	0
{1,2,3,4,6,8,2,9,10} Cadena con solo un carácter, mayor que el objetivo	V=“b” c=‘a’	0

7. Ordenación de forma ascendente

Aplique la técnica del camino básico al diseño del mínimo caso de prueba para probar el siguiente método estático que ordena un array de enteros de forma ascendente. (dibuje el grafo de flujo, calcule la complejidad ciclomática, especifique los caminos independientes y los casos de prueba asociados a cada camino).

```
static public void sort(int[] testArray)
{
            int tempValue;
            int i = 0;
            bool isSwapped = true;
            while (isSwapped)
            {   isSwapped = false;
                i++;
                Console.Out.WriteLine("Before "+i+" iteration :");
                Console.Out.WriteLine("");
                for (int j = 0; j < testArray.Length - i; j++)
                {
                    if (testArray[j] > testArray[j + 1])
                    {
                        tempValue = testArray[j];
                        testArray[j] = testArray[j + 1];
                        testArray[j + 1] = tempValue;
                        isSwapped = true;
                    }
                }
            }
}
```

```
static public void sort(int[] testArray)
{
  int tempValue;
1 int i = 0;
  bool isSwapped = true;
  while (isSwapped) 2
  {   isSwapped = false;
      i++;
  3 Console.Out.WriteLine("Before "+i+" iteration :");
      Console.Out.WriteLine("");
      for (int j = 0; j < testArray.Length - i; j++) 4 7
      {
            if (testArray[j] > testArray[j + 1]) 5
            {
                tempValue = testArray[j];
              6 testArray[j] = testArray[j + 1];
                testArray[j + 1] = tempValue;
                isSwapped = true;
             }
       }
   }
8}
```



V(G)= 4

Áreas = 4

Nodos Predicado = 3 → 3+1=4

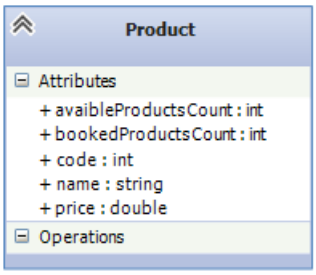
Nodos = 10 → 12-10+2 =4

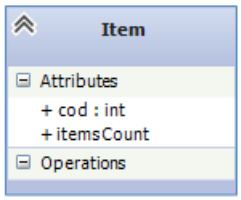
Aristas = 12

Path	**Input**	**Output**
{1,2,8} Cadena Vacía	No posible	No posible
{1,2,3,4,2,8} Vacía o una posición	[ ]	[]
{1,2,3,4,5,6,7,4,2,8} Dos posiciones ordenadas	[1,2]	[1,2]
{1,2,3,4,6,8,2,9,10} Dos posiciones desordenadas	[2,1]	[1,2]

8. Almacén

Aplique la técnica del camino básico al diseño del mínimo caso de prueba para probar el siguiente método estático que ordena un array de enteros de forma ascendente. (dibuje el grafo de flujo, calcule la complejidad ciclomática, especifique los caminos independientes y los casos de prueba asociados a cada camino).



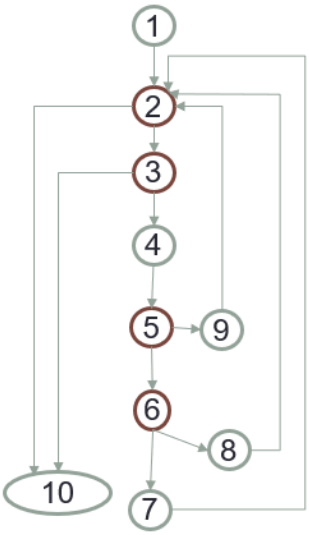



```
static public int bookItems(ArrayList products, Item item, out double cost, out string message)
{
    int j;
    Product product;
    j = 0;
    message = "Product not found";
    cost = 0;

    while ((j < products.Count ) && (message.Equals("Product not found")))
    { product = products[j] as Product;
        if (item.code == product.code)
        { if (item.itemsCount <= product.availableProductsCount)
            {
                cost = cost + item.itemsCount * product.price;
                product.availableProductsCount -= item.itemsCount;
                product.bookedProductsCount += item.itemsCount;
                message = "Product booked";
            }
            else
            {
                message = "Not enough products";
            }
        }
        else
        {
            j++;
        }
    }
    return j;
}
```

```
static public int bookItems(ArrayList products, Item item, out double cost, out string message)
{
    int j;
    Product product;
    j = 0;
    message = "Product not found";
    cost = 0;

    while ((j < products.Count ) && (message.Equals("Product not found")))
    { product = products[j] as Product;
        if (item.code == product.code)
        { if (item.itemsCount <= product.availableProductsCount)
            {
                cost = cost + item.itemsCount * product.price;
                product.availableProductsCount -= item.itemsCount;
                product.bookedProductsCount += item.itemsCount;
                message = "Product booked";
            }
            else
            {
                message = "Not enough products";
            }
        }
        else
        {
            j++;
        }
    }
    return j;
}
```



V(G)= 5

Áreas = 5

Nodos predicado= 4 → 4+1=5

Nodos = 10 → 13-10+2 =5

Aristas = 13

Caminos
C1:{1,2,10} Sin productos
C2: {1,**2,3**,10}
C3:{1,2,**3,4**,5,9,2,10} Solo un product que no es el deseado
C4:{1,2,3,4**,5,6**,8,2,10} Solo un product, que es el buscado pero del que no hay suficiente stock
C5:{1,2,3,4,5,**6,7**,2,310} Solo un product, que es el buscado y del que hay suficiente stock

	Entrada		**Salida**			
	Products	**Items**	**Return**	**Cost**	**Message**	**Products.out**
C1	[ ]	{code=2; itemsCount=5}	0	0.0	Product not found	Sin cambios
C2	No posible		No posible			
C3	[{code = 5; availableProductsCount =5; bookedProductsCount = 5; price = 10}]	{code=2; itemsCount=10}	1	0.0	Product not found	Sin cambios
C4	[{code = 2; availableProductsCount = 5; bookedProductsCount = 5; price = 10}]	{code=2; itemsCount=10}	0	0.0	Not enough products	Sin cambios
C5	[{code = 2; availableProductsCount =15; bookedProductsCount = 5; price = 10}]	{code=2; itemsCount=10}	0	100	Product booked	[{code = 2; availableProductsCount =5; bookedProductsCount = 15; price = 10}]

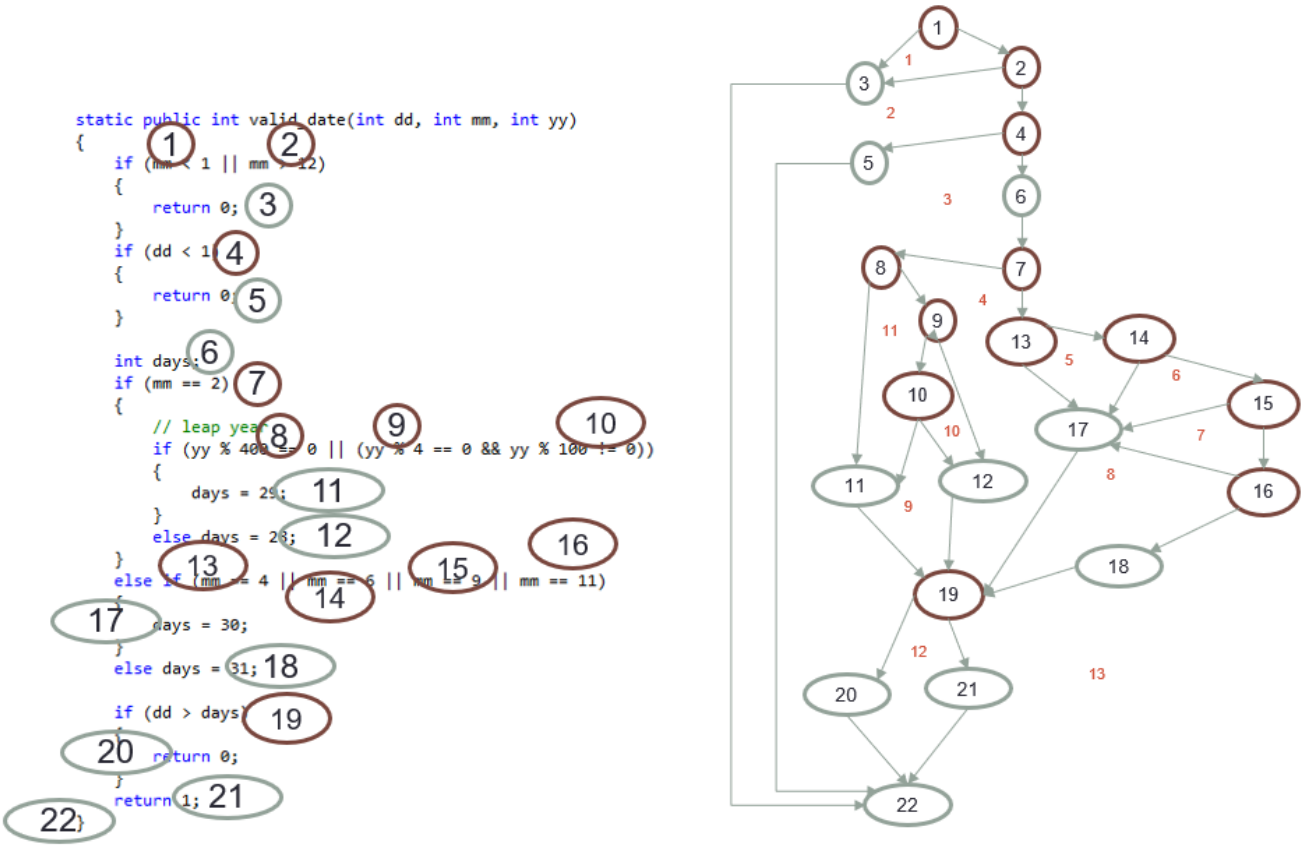
9. Validación de fecha

Aplique la técnica del camino básico al diseño del mínimo caso de prueba para probar el siguiente método estático que ordena un array de enteros de forma ascendente. (dibuje el grafo de flujo, calcule la complejidad ciclomática, especifique los caminos independientes y los casos de prueba asociados a cada camino).

```
static public int valid_date(int dd, int mm, int yy)
{
    if (mm < 1 || mm > 12)
    {
        return 0;
    }
    if (dd < 1)
    {
        return 0;
    }

    int days;
    if (mm == 2)
    {
        // leap year
        if (yy % 400 == 0 || (yy % 4 == 0 && yy % 100 != 0))
        {
            days = 29;
        }
        else days = 28;
    }
    else if (mm == 4 || mm == 6 || mm == 9 || mm == 11)
    {
        days = 30;
    }
    else days = 31;

    if (dd > days)
    {
        return 0;
    }
    return 1;
}
```



V(G)= 13

Áreas = 13

Nodos Predicado= 12 → 12+1=13

Nodos = 22 → 33-22+2 =13

Aristas = 33

	Camino	Entrada	Salida
1	{1,3,22} Mes <1	mm=-1; dd=any;yy=any	0
2	{1,**2,3**,22} Mes>12	mm=-13; dd=any; yy=any	0
3	{1,**2,4**,5,22} Mes válido. Dias <1	mm=1; dd=-1; yy=any	0
4	{1,2,**4,6**,7,8,11,19,20,22} Febrero, año divisible por 400 (bisiesto). Días>29	mm=2; dd=30; yy=2000;	0
5	{1,2,4,6,**7,13**,17,19,20,22} Abril, Días>30	mm=4; dd=31; yy=any	0
6	{1,2,4,6,7,**13,14**,17,19,20,22} Junio, Días>30	mm=6; dd=31; yy=any	0
7	{1,2,4,6,7,13,**14,15**,17,19,20,22} Sept, Días>30	mm=9; dd=31; yy=any	0

8	{1,2,4,6,7,13,14,**15,16**,17,19,20,22} Nov, Días>30	mm=11; dd=31; yy=any	0
9	{1,2,4,6,7,13,14,15,**16,18**,19,20,22} Dic, Días>30	mm=12; dd=32; yy=any	0
10	{1,2,4,6,7,13,14,15,**16,17**,19,21,22} Dic, Dias válidos	mm=12; dd=31; yy=any	1
11	{1,2,4,6,7**,8,9**,12,19,20,22} Febrero, año no divisible por 400, no divisible por 4. Días>28	mm=2; dd=31; yy=2005	0
12	{1,2,4,6,7,8,**9,10**,11,19,20,22} Febrero. Divisible por 4 y no por 100 (bisiesto). Días>29	mm=2; dd=30; yy=2004	0
13	{1,2,4,6,7,8,**9,10**,11,19,20,22} Febrero. No divisible por 400. Divisible por 4 y por 100 (no bisiesto). Días>28	mm=2; dd=30; yy=2100	0

10. CheckSum

Dado el siguiente fragmento de código:

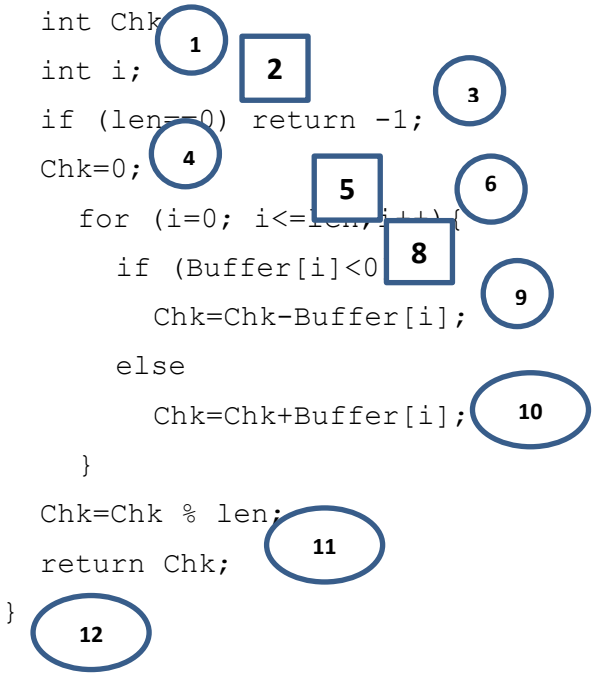
```
int CheckSum(ArrayList<int> Buffer, int len){
  int Chk;
  int i;
  if (len==0) return -1;
  Chk=0;
    for (i=0; i<=len;i++){
      if (Buffer[i]<0)
        Chk=Chk-Buffer[i];
      else
        Chk=Chk+Buffer[i];
    }
  Chk=Chk % len;
  return Chk;
}
```

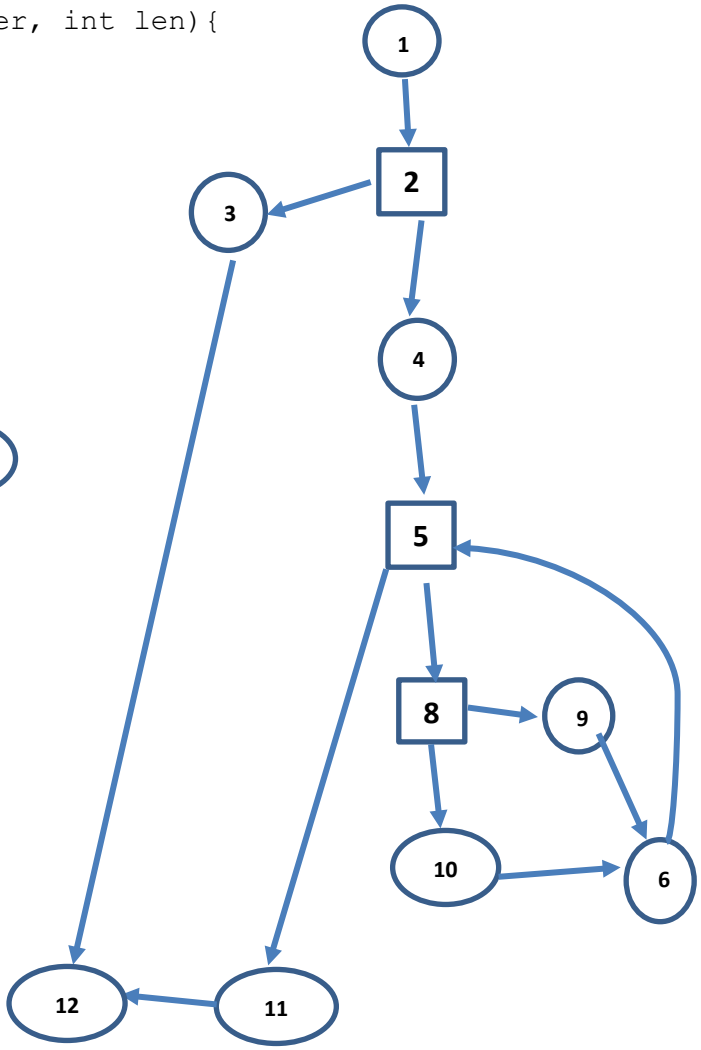
Se pide:

a) Obtener la complejidad ciclomática.
b) Obtener todos los caminos independientes.
c) Generar los casos de prueba.

a)

```
int CheckSum(ArrayList<int> Buffer, int len){
  int Chk;
  int i;
  if (len==0) return -1;
  Chk=0;
    for (i=0; i<=len;i++){
      if (Buffer[i]<0)
        Chk=Chk-Buffer[i];
      else
        Chk=Chk+Buffer[i];
    }
  Chk=Chk % len;
  return Chk;
}
```





CC = Nº Regiones = 4

CC=Nodos Predicado+1=3+1=4

CC=Aristas-Nodos+2=12-10+2=4

b)

Camino 1: 1-2-3-11

Camino 2: 1-**2-4**-5-11

Camino 3: 1-2-4-**5-8**-10-6-5-11

Camino 4: 1-2-4-5-**8-9**-6-5-11

c)

Camino	Entrada		Salida
	Buffer	Len	
1	[1]	0	-1
2	[1]	-1	0
3	[1]	1	1
4	[-1]	1	1

11. Calcula bonus (try..catch)

Dado el siguiente fragmento de código:

```
public double calculaBonus(string cod_empleado, int meses)
{
   int bonus = 0;
   try
   {
      int codigo = int.Parse(cod_empleado);
      if(codigo > 100)
      {
          bonus = 100 * meses;
      }
      else
      {
           bonus = 10;
       }
    }
    catch (FormatException e)
    {
        Console.WriteLine("cod_bonus no contiene un número" +
            e.Message);
    }

    catch (Exception e)
    {
         Console.WriteLine("Otro error" + e.Message);
    }
    return bonus;

 }
```

Se pide:

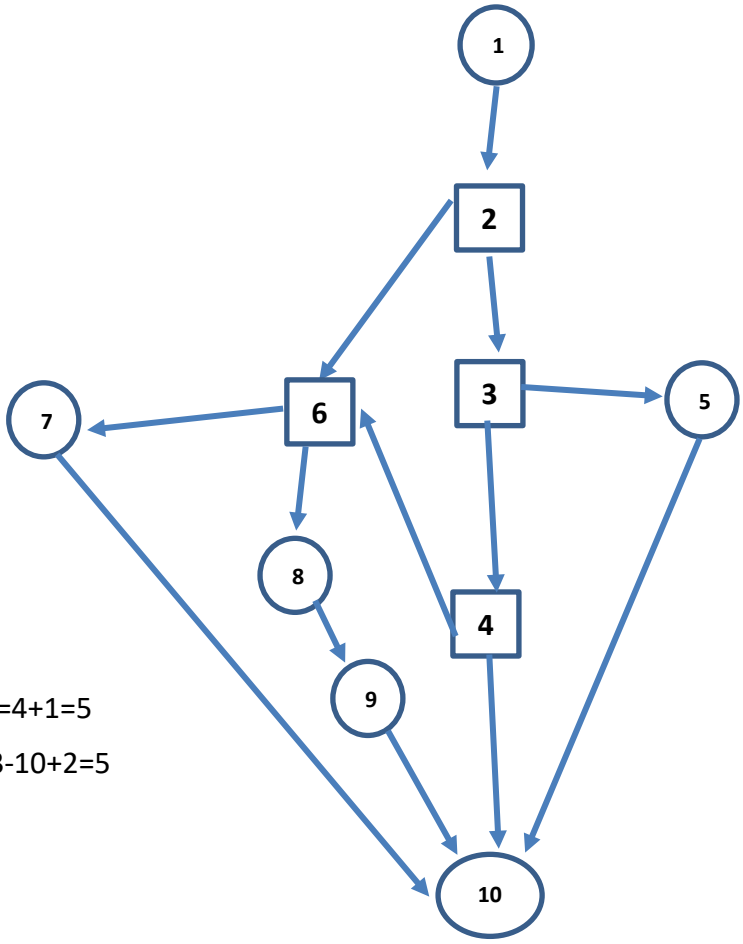
a) Obtener la complejidad ciclomática.
b) Obtener todos los caminos independientes.
c) Generar los casos de prueba.

a)

```
public double calculaBonus(string cod_empleado, int meses)
 {
    int bonus = 0;
    try
    {
       int codigo = int.Parse(cod_empleado);
       if(codigo > 100)
       {
           bonus = 100 * meses;
       }
       else
       {
            bonus = 10;
        }
     }
     catch (FormatException e)
     {
         Console.WriteLine("cod_bonus no contiene un número" +
             e.Message);
     }

     catch (Exception e)
     {
          Console.WriteLine("Otro error" + e.Message);
     }
     return bonus;

  }
```



CC = Nº Regiones = 5

CC=Nodos Predicado+1=4+1=5

CC=Aristas-Nodos+2=13-10+2=5

b)

Camino 1: 1-2-6-7-10

Camino 2: 1-2-3-4-10

Camino 3: 1-2-3-5-10

Camino 4: 1-2-6-8-9-10

Camino 5: 1-2-3-4-6-8-9-10

c)

Camino	Entrada		Salida
	Cod_empleado	meses	
1	"A"	2	Cod_bonus no contiene un número 0
2	"1000"	2	200
3	"90"	2	10
4			No hay entradas que permitan probar este camino
5	"1000"	Int.MaxValue	Otro error 0

12. Clasificación de profesores

Aplicando la técnica de pruebas de caja negra de partición equivalente, diseña los casos de prueba para el siguiente módulo software que clasifica a los individuos en función de las siguientes entradas:

- Código: cadena de 3 dígitos que no empieza por "00"
- Carácter de control: carácter válido de la 'a'..'z' o carácter '-'
- Tipo de persona: contratado o interino

Los valores de salida esperados serán:

- "S1", si el código válido es par y la persona es contratado.
- "S2", si el código válido es par y la persona es interino.
- "S3", si el código válido es impar y la persona es contratado.
- "S4", si el código válido es impar y la persona es interino.
- "S5", si el carácter de control es un '-'.
- "S6", en caso contrario.

Nota 1. Si pueden darse varios valores de salida, la salida de menor número es prioritaria (por ej. S1 es más prioritaria que S5).

Nota 2. Se pide definir las clases válidas/inválidas, técnica heurística utilizada y definir los casos de prueba.

Entrada	Clases válidas	Clases inválidas	Regla/heurística
Código	1. código en [010..999] y par 2. código en [010..999] e impar	3. código < 010 (o código de menos de 3 dígitos) 4. código > 999 (o código de más de 3 dígitos) 5. cadena que no sean dígitos 6. código que empieza por 00 (NOTA: al ser el código una cadena y no un número, se puede considerar también este caso)	Rango valores Condición booleana Clases menores
Carácter de control	7. carácter en [‘a’..’z’] 8. carácter =’-‘	9. carácter inválido	Conjunto finito valores Condición booleana
Tipo de persona	10. contratado 11. interino	12. otro tipo	Conjunto de valores admitidos

Casos de prueba:

Clases válidas cubiertas	Código	Carácter de control	Tipo de persona	Salida
1, 7, 10	100	‘b’	contratado	S1
2, 8, 11	101	‘-’	interino	S4

Clases inválidas cubiertas	Código	Carácter de control	Tipo de persona	Salida
3, 7, 10	05	‘b’	contratado	S6
4, 8, 10	1050	‘-‘	contratado	S5
5, 7, 10	a#1=	‘c’	contratado	S6
6, 8, 11	007	‘-‘	interino	S5
1, 9, 10	200	‘=’	contratado	S6
1, 8, 12	400	‘-‘	desconocido	S5

13. Clasificación de personas

Aplicando la técnica de pruebas de caja negra de partición equivalente, diseña los casos de prueba para el siguiente módulo software de clasificación con las siguientes entradas:

- Código: número de 4 dígitos que no empieza por 0 ni por 1
- Carácter de control: carácter válido de la A..Z
- Tipo de persona: profesor o alumno

Los valores de salida esperados serán:

- "S1", si el código válido es par y la persona es profesor.
- "S2", si el código válido es par y la persona es alumno.
- "S3", si el código válido es impar y la persona es profesor.
- "S4", si el código válido es impar y la persona es alumno.
- "S5", si el carácter de control no es un valor de la 'A'..'Z'.

Nota 1. Si pueden darse varios valores de salida, la salida de menor número es prioritaria (por ej. S1 es más prioritaria que S5).

Nota 2. Se pide definir las clases válidas/inválidas, técnica heurística utilizada y definir los casos de prueba.

Entrada	Clases válidas	Clases inválidas	Regla/heurística
Código	1. código en [2000..9999] y par 2. código en [2000..9999] e impar	3. código < 2000 4. código > 9999 5. número que no sean 4 dígitos	Rango valores Condición booleana Clases menores
Carácter de control	6. carácter en ['A'..'Z']	7. carácter no en ['A'..'Z']	Conjunto finito valores Condición booleana
Tipo de persona	8. profesor 9. alumno	10. otro tipo	Conjunto de valores admitidos

Casos de prueba:

Clases válidas cubiertas	Código	Carácter de control	Tipo de persona	Salida
1, 6, 8	2050	'A'	profesor	S1
2, 6, 9	2051	'B'	alumno	S4

Clases inválidas cubiertas	Código	Carácter de control	Tipo de persona	Salida
3, 6, 8	1980	'B'	profesor	Error. Salida no contemplada
4, 6, 9	10050	'B'	alumno	Error. Salida no contemplada
5, 6, 8	575	'C'	profesor	Error. Salida no contemplada
1, 7, 8	2020	'+'	profesor	S5
2, 6, 10	2021	'V'	asistente	Error. Salida no contemplada

14. Clasificación de personas extendido con la técnica AVL (Análisis de Valores Límite)

Extended el ejercicio anterior de "clasificación de personas" para trabajar con los valores límite y diseñad sus casos de prueba.

Entrada	Nuevas clases válidas	Nuevas clases inválidas
Código	11. código = 2000 12. código = 2001 13. código = 9998 14. código = 9999	15. código = 1999 16. código = 10000
Carácter de control	-	17. si el carácter se recuperara a partir de un string, string de tamaño 0: '' 18. si el carácter se recuperara a partir de un string, string de tamaño 2: '??'
Tipo de persona	-	-

Casos de prueba:

Clases válidas cubiertas	Código	Carácter de control	Tipo de persona	Salida
11, 6, 8	2000	‘B’	profesor	S1
12, 6, 8	2001	‘B’	profesor	S3
13, 6, 9	9998	‘H’	alumno	S2
14, 6, 9	9999	‘H’	alumno	S4

Clases inválidas cubiertas	Código	Carácter de control	Tipo de persona	Salida
15, 6, 8	1999	‘B’	profesor	Error. Salida no contemplada
16, 6, 8	10000	‘B’	profesor	Error. Salida no contemplada
1, 17, 8	2000	‘’ (si es **obligatorio** introducir un carácter, este caso de prueba no podría ejecutarse)	profesor	Dependiendo de cómo se interprete el carácter de control: S5 o Error. Salida no contemplada
2, 18, 9	2001	“AB” (si **solo** se puede introducir un carácter, este caso de prueba no podría ejecutarse)	alumno	Dependiendo de cómo se interprete el carácter de control: S5 o Error. Salida no contemplada

15. Cálculo de la nota

Un método que genera un listado recibe la siguiente información:

- Nombre del alumno, debe tener como mínimo un nombre y apellido.
- Grupo, son tres caracteres donde lo primero es una de las siguientes letras A, C o D y los otros dos caracteres son dos dígitos que pueden ir del 01 al 15.
- Nota teoría (NT), número sobre 10.
- Nota práctica (NP), número sobre 10.
- Nota trabajo (NTr), número sobre 8.

El método devuelve como salida la nota final en función de la fórmula NT*0.60+*NP*0.4+NTr*0.1. En caso contrario debe devolver error.

Se pide:

a) Crear la tabla de clases de equivalencia (indicando la regla heurística aplicada).
b) Generar los casos de prueba.

a)

Entrada	Clases válidas	Clases inválidas	Regla/heurística
Nombre	1. >= 2 palabras	2. <2 Palabras	Condición booleana
Grupo	3. 3 caracteres, con el primero A, y el resto dentro del rango ['01'-'15'] 4. 3 caracteres, con el primero 'C', y el resto dentro del rango ['01'-'15']. 5. 3 caracteres, con el primero 'D', y el resto dentro del rango ['01'-'15'].	8. < 3 caracteres 9. >3 caracteres 10. Otro valor 11. <01 12. >15 13. No es número	Nº finito de valores Conjunto de valores Rango de valores Booleana
Nota teoría	15. [0-10]	16. <0 17. >10 18. No es un número (NOTA: si el parámetro de entrada es de tipo int este caso no sería necesario)	Rango de valores Booleana
Nota Práctica	19. [0-10]	20. <0 21. >10 22. No es un número (NOTA: si el parámetro de entrada es de tipo int este caso no sería	Rango de valores Booleana

		necesario)	
Nota Trabajo	23. [0-8]	24. <0 25. >8 26. No es un número (NOTA: si el parámetro de entrada es de tipo int este caso no sería necesario)	Rango de valores Booleana

b) Casos de prueba:

Clases válidas cubiertas	Nombre	Grupo	NT	NP	NTr	Salida
1, 3, 15, 19, 23	Ana López	A12	10	10	8	10.8
1, 4, 15, 19, 23	Ana López	C09	5	5	4	5.4
1, 5, 15, 19, 23	Ana López	D05	0	0	0	0

Clases inválidas cubiertas	Nombre	Grupo	NT	NP	NTr	Salida
2, 3, 4, 7, 15, 19, 23	Ana	A12	10	10	8	Error
1, 8, 4, 7, 15, 19, 23	Ana López	A1	10	10	8	Error
1, 9, 4, 7, 15, 19, 23	Ana López	A001	10	10	8	Error
1, 10, 7, 15, 19, 23	Ana López	B01	10	10	8	Error
1, 11, 15, 19, 23	Ana López	A00	10	10	8	Error
1, 12, 15, 19, 23	Ana López	A20	10	10	8	Error
1, 13, 15, 19, 23	Ana López	AAA	10	10	8	Error
1, 3,16, 19, 23	Ana López	A05	-1	10	8	Error
1, 3, 17, 19, 23	Ana López	A05	12	10	8	Error
1, 3, 18, 19, 23	Ana López	A05	A	10	8	Error
1, 3, 15, 20, 23	Ana López	A05	10	-1	8	Error
1, 3, 15, 21, 23	Ana López	A05	10	12	8	Error
1, 3, 15, 22, 23	Ana López	A05	10	A	8	Error
1, 3, 15, 19, 24	Ana López	A05	10	10	-1	Error
1, 3, 15, 19, 25	Ana López	A05	10	10	12	Error
1, 3, 15, 19, 26	Ana López	A05	10	10	A	Error

16. Clasificación de productos

Un programa recibe como entrada un fichero de texto con las siguientes columnas:

- Product-number: un campo entero positivo menor a 256, con 3 dígitos.
- Product-code: un campo alfanumérico de 4 caracteres.
- Expiry-Month: representa los meses que transcurren hasta que el producto caduca; es un valor positive de dos dígitos (excepto 00).
- Sale: un campo de solo un carácter, que es "+" cuando el producto está en oferta o "-" en otro caso.

Se pide crear la tabla de clases de equivalencia (indicando la regla heurística aplicada) y generar los casos de prueba.

Entrada	**Clases Válidas**	**Clases inválidas**	**Heurísticas**
Product-number	(1) Número de 3 dígitos, en el rango [000,255]	(2) No numérico (3) <3 dígitos (4) >3 dígitos (5) <000 (6) >255	Conjunto finito de valores Clases menores Rango valores
Product-code	(7) Alfanumérico de 4 caracteres	(8) No Alfanumérico (9) <4 caracteres (10) >4 caracteres	Booleano Clases menores Conjunto finito de valores
Expiry-Month	(11) Número de 2 digitos en [01,99]	(12) No number (13) Less 2 digits (14) More 2 digits (15) ='00'	Booleano Conjunto finito de valores Clases menores Booleano
Sale	(16) '+' (17) '-'	(18) Other value ('*')	Conjunto de valores válidos

	Clases válidas	Entrada
T.C. Valid C.	(1)(7)(11)(16)	Product-number=‘001’ ; Product-code=‘code’; Expiry-Month=’24’; Sale =‘+’
	(1)(7)(11)(17)	Product-number=‘001’ ; Product-code=‘code’; Expiry-Month=’24’; Sale =‘-’

	Clases inválidas	Entrada
T.C. Invalid C.	(**2**)(7)(11)(16)	Product-number=‘aaa’; Product-code=‘code’; Expiry-Month=’24’; Sale =‘+’
	(**3**)(7)(11)(16)	Product-number=‘00’; Product-code=‘code’; Expiry-Month=’24’; Sale =‘+’
	(**4**)(7)(11)(16)	Product-number=‘0000’; Product-code=‘code’; Expiry-Month=’24’; Sale =‘+’
	(**5**)(7)(11)(16)	Product-number=‘-01’; Product-code=‘code’; Expiry-Month=’24’; Sale =‘+’
	(**6**)(7)(11)(16)	Product-number=‘257’; Product-code=‘code’; Expiry-Month=’24’; Sale =‘+’
	(1)(**8)**(11)(16)	Product-number=‘001’ ; Product-code=‘-*!+’; Expiry-Month=’24’; Sale =‘+’
	(1)(**9**)(11)(16)	Product-number=‘001’ ; Product-code=‘cod’; Expiry-Month=’24’; Sale =‘+’
	(1)(**10**)(11)(16)	Product-number=‘001’ ; Product-code=‘codes’; Expiry-Month=’24’; Sale =‘+’
	(1)(7)(**12)**(16)	Product-number=‘001’ ; Product-code=‘code’; Expiry-Month=‘mm’; Sale =‘+’
	(1)(7)(**13**)(16)	Product-number=‘001’ ; Product-code=‘code’; Expiry-Month=‘0’; Sale =‘+’
	(1)(7)(**14**)(16)	Product-number=‘001’ ; Product-code=‘code’; Expiry-Month=‘000’; Sale =‘+’
	(1)(7)(**15**)(16)	Product-number=‘001’ ; Product-code=‘code’; Expiry-Month=‘00’; Sale =‘+’
	(1)(7)(11)(**18**)	Product-number=‘001’ ; Product-code=‘code’; Expiry-Month=’24’; Sale =‘*’