UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL TUCUMÁN

CÁTEDRA DE
INGENIERÍA DE SOFTWARE
Trabajo Práctico N° 3

GRUPO N° 22

Integrantes:

Díaz, Carlos Alberto

Paz Gramajo, Rita Ayelen

Pérez, Matías Leonel

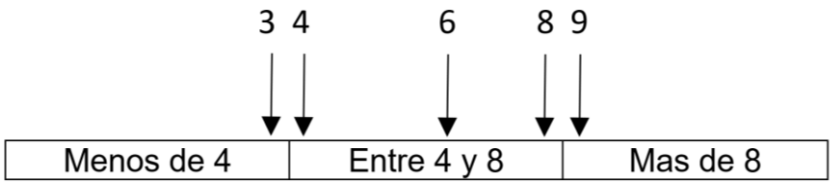
Sureda Granero, Pablo Gonzalo

Comisión: **4K1**

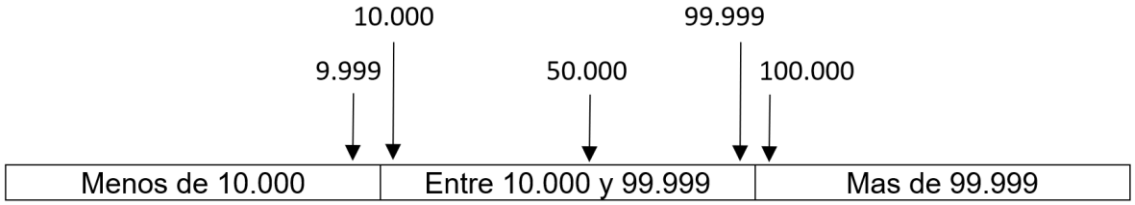
1. **Pruebas de partición**

a) Determinar las particiones de equivalencia para un programa, cuya especificación establece, que acepta de 4 a 8 entradas que son 5 dígitos enteros mayores que 10000.

Cantidad de valores de entrada

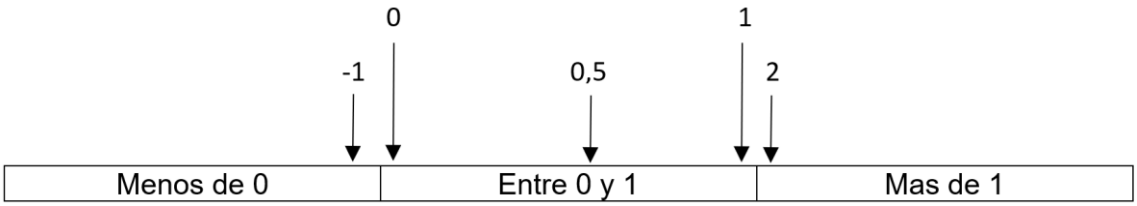


Valores de entrada



b)

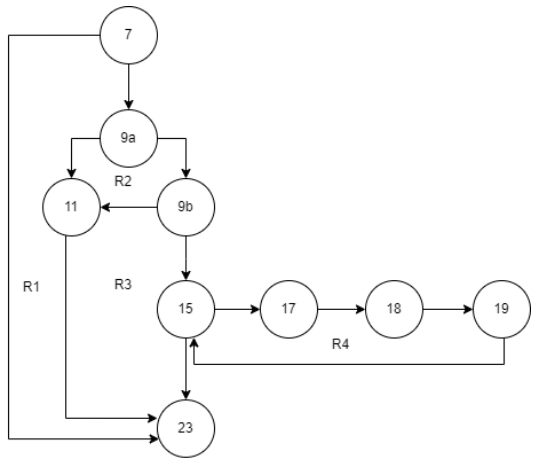
```
1   private static int fibonacci(int n)
2   {
3       int actual = 0;
4       int ant1, ant2;
5       ant1 = 1;
6       ant2 = 0;
7       if (n >= 0)
8       {
9           if ((n == 0) || (n == 1))
10          {
11              actual = n;
12          }
13          else
14          {
15              for (int i = 2; i <= n; i++)
16              {
17                  actual = ant1 + ant2;
18                  ant2 = ant1;
19                  ant1 = actual;
20              }
21          }
22      }
23      return actual;
24  }
```



2. Pruebas de caminos

Realizar el grafo de flujo para el código del apartado b) del punto 1 y para los siguientes métodos. Calcular la complejidad ciclomática asociada por los tres métodos.

1.b)

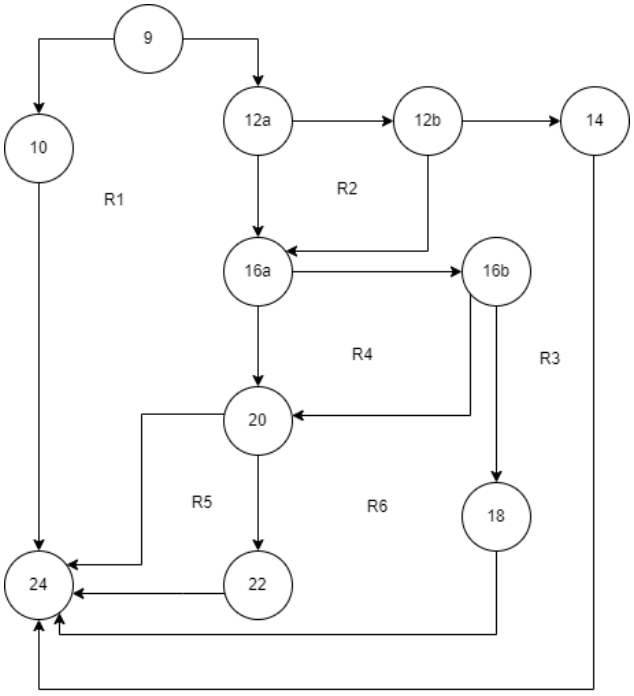


V(G) = NP +1 = 4 + 1 = **5**

V(G) = A–N+2 = 12 – 9 + 2 = **5**

V(G) = RC + 1 = 4 + 1 = **5**

```
1   public class ReglaDeDescuento
2   {
3       private static final double porcentajeMenor = 0.03d;
4       private static final double porcentajeIntermedio = 0.05d;
5       private static final double porcentajeMayor = 0.10d;
6
7       public double Calcular(double total)
8       {
9           if(total <= 0)
10              throw new IllegalArgumentException("El total debe ser mayor a 0");
11
12          if (total > 5000 && total <= 10000)
13          {
14              return total * porcentajeMenor;
15          }
16          if (total > 10000 && total <= 25000)
17          {
18              return total * porcentajeIntermedio;
19          }
20          else if (total > 25000)
21          {
22              return total * porcentajeMayor;
23          }
24          return 0;
25      }
26  }
```

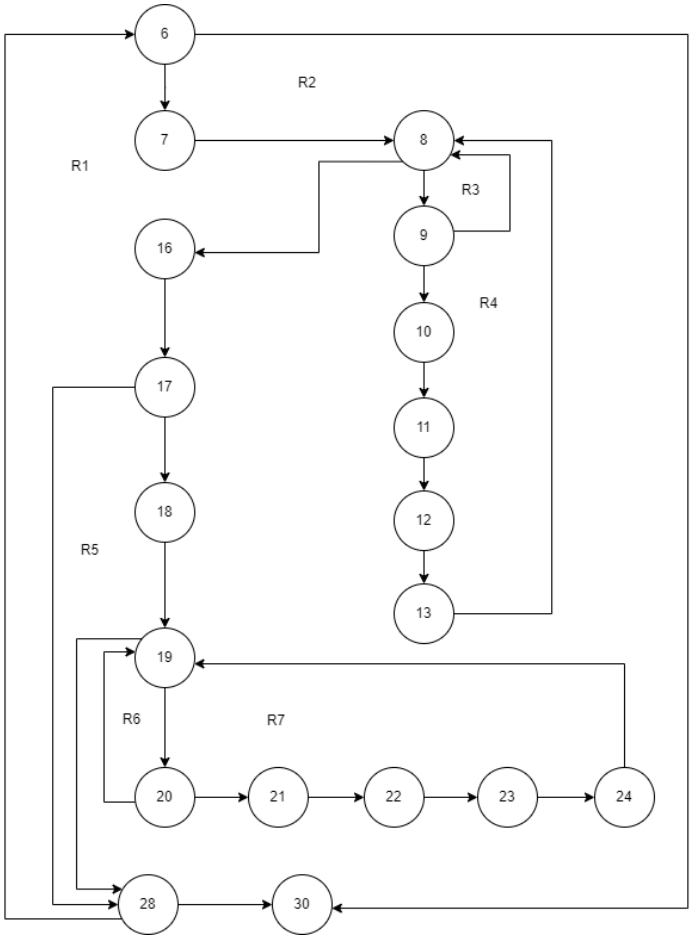


V(G) = NP +1 = 6 + 1 = **7**

V(G) = A–N+2 = 16 – 11 + 2 = **7**

V(G) = RC + 1 = 6 + 1 = **7**

```java
public static int[] cocktailSort(int[] numbers)
{
    boolean swapped = true;
    int i = 0;
    int j = numbers.length - 1;
    while(i < j && swapped){
        swapped = false;
        for(int k = i; k < j; k++){
            if(numbers[k] > numbers[k + 1]){
                int temp = numbers[k];
                numbers[k] = numbers[k + 1];
                numbers[k + 1] = temp;
                swapped = true;
            }
        }
        j--;
        if(swapped){
            swapped = false;
            for(int k = j; k > i; k--){
                if(numbers[k] < numbers[k - 1]){
                    int temp = numbers[k];
                    numbers[k] = numbers[k - 1];
                    numbers[k - 1] = temp;
                    swapped = true;
                }
            }
        }
        i++;
    }
    return numbers;
}
```



V(G) = NP +1 =7+1 = **8**

V(G) = A–N+2 = 25 – 19 + 2 = **8**

V(G) = RC + 1 = 7 + 1 = **8**

3. Pruebas de Unidad (unitarias)

Plantear las pruebas unitarias para la clase **ReglaDeDescuento**.

```
0 referencias
public class ReglaDeDescuentoTests
{
    [Fact]
    0 referencias
    public void Calcular_TotalMenorIgualA5_DeberiaLanzarExcepcion()
    {

        var regla = new ReglaDeDescuento();
        Assert.Throws<ArgumentException>(() => regla.Calcular(5));
    }

    [Fact]
    0 referencias
    public void Calcular_TotalEntre5000Y10000_DeberiaCalcularPorcentajeMenor()
    {
        var regla = new ReglaDeDescuento();
        double total = 7000;

        double resultado = regla.Calcular(total);

        Assert.Equal(total * 0.03d, resultado);
    }

    [Fact]
    0 referencias
    public void Calcular_TotalEntre10000Y25000_DeberiaCalcularPorcentajeIntermedio()
    {
        var regla = new ReglaDeDescuento();
        double total = 15000;

        double resultado = regla.Calcular(total);

        Assert.Equal(total * 0.05d, resultado);
    }

    [Fact]
    0 referencias
    public void Calcular_TotalMayorA25000_DeberiaCalcularPorcentajeMayor()
    {
        var regla = new ReglaDeDescuento();
        double total = 30000;

        double resultado = regla.Calcular(total);

        Assert.Equal(total * 0.10d, resultado);
    }
```

5. Pruebas de Versión (sistema)

Para el caso de uso Realizar Venta diseñar 2 (dos) casos de prueba. Los casos se deben preparar en la plantilla que se adjunta.

Caso de Prueba	
ID: 01	**Nombre:** Agregar artículos a la venta
Descripción: Agregar artículos a la venta	
Prioridad: Alta	**CU / HU:** Realizar venta
Módulo / Funcionalidad: Realizar venta	
Diseñado por:	**Fecha:**
Ejecutado por:	**Fecha:**

Precondiciones:

- El vendedor debe estar autenticado en el sistema.

Paso	**Acción**	**Resultado Esperado**	**Pasó/Falló**	**Comentarios**
1	Introducir código del artículo.	Visualizar descripción, marca, talle, color y cantidad disponible del artículo.		
2	Seleccionar talle y color.	Visualizar el detalle de la línea de venta.		

Caso de Prueba	
ID: 02	**Nombre:** Stock insuficiente al agregar artículo
Descripción: Cantidad de stock insuficiente al agregar artículos a la venta	
Prioridad: Alta	**CU / HU:** Realizar venta
Módulo / Funcionalidad: Realizar venta	
Diseñado por:	**Fecha:**
Ejecutado por:	**Fecha:**

Precondiciones:

- El vendedor debe estar autenticado en el sistema.

Paso	**Acción**	**Resultado Esperado**	**Pasó/Falló**	**Comentarios**
1	Introducir código del artículo.	Visualizar descripción, marca, talle, color y cantidad disponible del artículo.		
2	Seleccionar talle, color y cantidad mayor al stock disponible.	Visualizar un mensaje indicando "Stock insuficiente para la cantidad especificada”.		