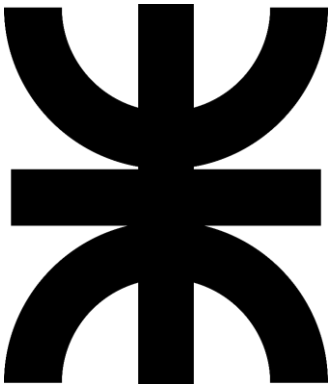
INGENIERÍA DE SOFTWARE

UNIVERSIDAD NACIONAL DE TUCUMAN

FACULTAD REGIONAL TUCUMAN



Trabajo Práctico Nº3

ESTRATEGIAS PARA CASOS DE PRUEBA, AUTOMATIZACIÓN DE PRUEBAS DE ACEPTACIÓN, PRUEBAS UNITARIAS Y PRUEBAS DE SISTEMA

FECHA DE PRESENTACIÓN:

DOCENTES

ING. MABEL TORRES

ING. FRANCISCO VICENTE

COMISIÓN 4K2 - GRUPO Nº 14

INTEGRANTES

MARIA BELÉN TERÁN NOUGUÉS – 52.639

SANTIAGO NEHUÉN SOSA – 52.624

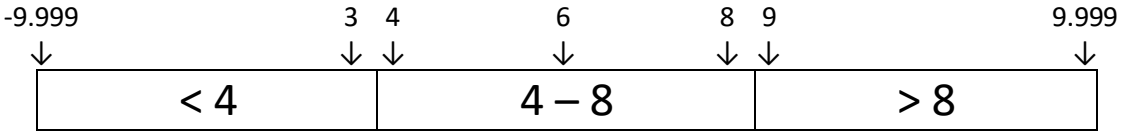
JULIO MARTÍN CALISAYA – 35.757

JAVIER ROZA – 42.274

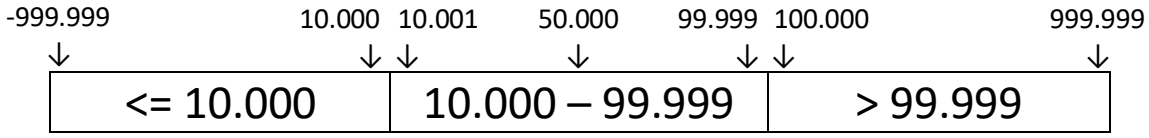
Ejercicio 1. Prueba de particiones

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 particiones según número de entradas y valores de prueba:



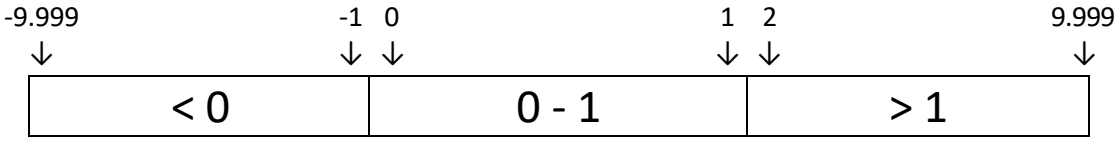
Particiones según el valor de las entradas y valores de prueba:



b) Basados en el siguiente código:

```
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  }
```

Se pueden armar las siguientes particiones de pruebas con sus correspondientes valores:

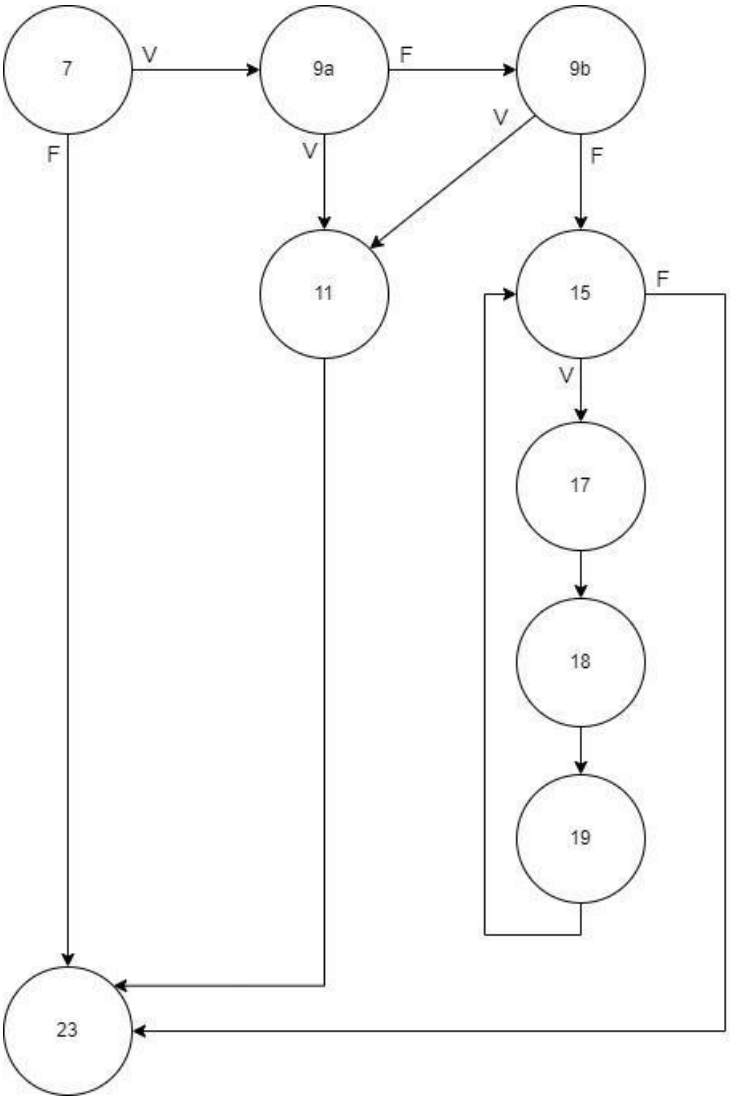


De dichas pruebas, se esperan los siguientes resultados:

Partición	Valor de Prueba	Resultado Esperado
< 0	-9.999	0
	-1	0
0-1	0	0
	1	1
> 1	2	1
	9.999	890.489.442

Ejercicio 2. Prueba de caminos

a) Para el código del punto 1-b su grafo es:



Su complejidad ciclomática es:

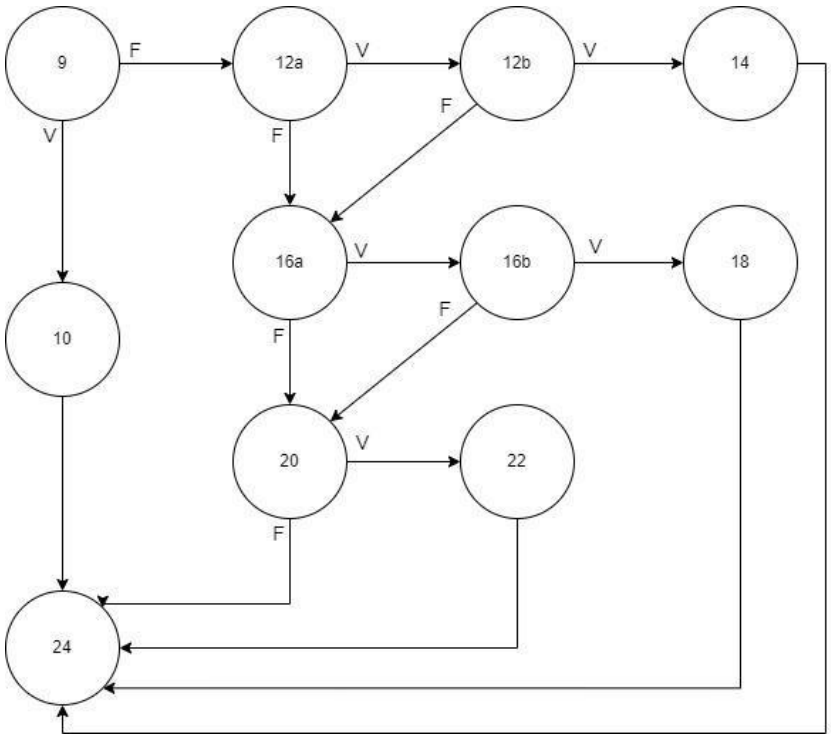
- Método 1: Nodos de condición + 1 = 4 + 1 = 5
- Método 2: Aristas – Nodos + 2 = 12 – 9 + 2 = 5
- Método 3: Regiones cerradas + 1 = 4 + 1 = 5

Los posibles caminos a seguir son:

Camino	Valor de prueba	Resultado esperado
7 – 23	-1	0
7 – 9a – 11 – 23	0	0
7 – 9a – 9b – 11 – 23	1	1
7 – 9a – 9b – 11 – 15 – 17 – 18 – 19 – 23	2	1
7 – 9a – 9b – 11 – 15 – 23	5	5

b) Para el siguiente código su grafo es:

```
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 }
```



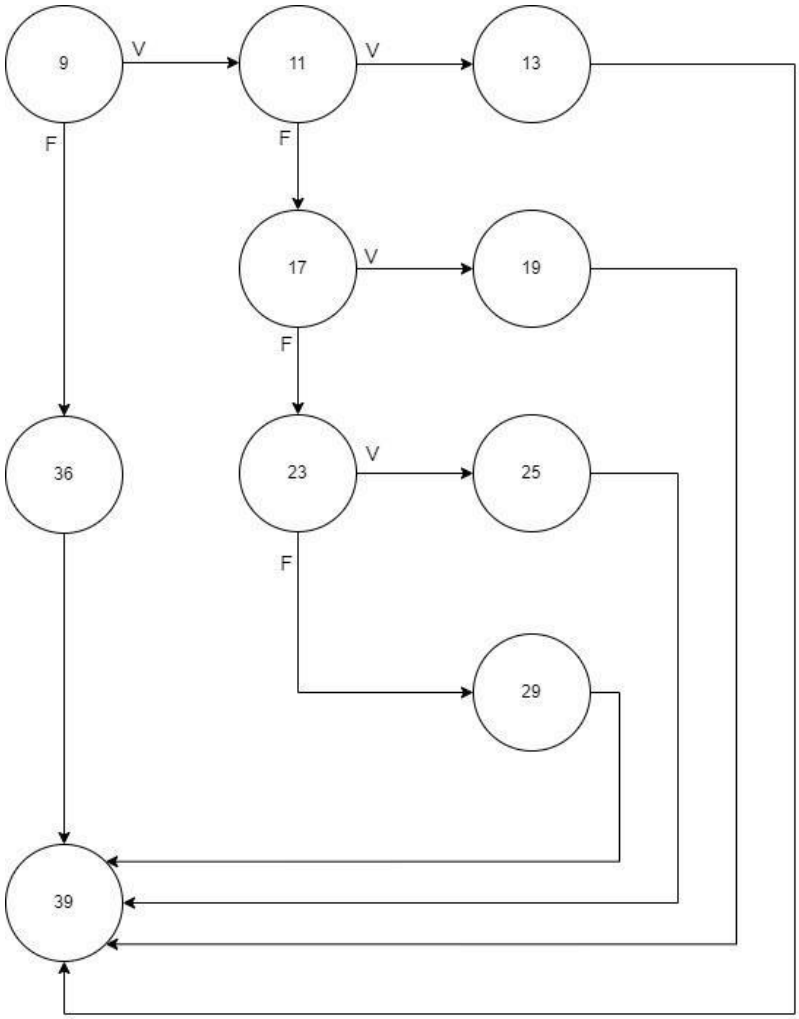
Su complejidad ciclomática es:

- Método 1: Nodos de condición + 1 = 6 + 1 = 7
- Método 2: Aristas – Nodos + 2 = 16 – 11 + 2 = 7
- Método 3: Regiones cerradas + 1 = 6 + 1 = 7

Esta complejidad podría reducirse aplicando los siguientes cambios al código:

```
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        private double calcular(double total)
8        {
9             if (total > 0)
10            {
11                  if (total <= 5000)
12                  {
13                        return 0;
14                  }
15                  else
16                  {
17                        if (total <= 10000)
18                        {
19                              return total * porcentajeMenor;
20                        }
21                        else
22                        {
23                              if (total <= 25000)
24                              {
25                                    return total * porcentajeIntermedio;
26                              }
27                              else
28                              {
29                                    return total * porcentajeMayor;
30                              }
31                        }
32                  }
33            }
34            else
35            {
36                  throw new IllegalArgumentException("El valor debe ser mayor a 0");
37            }
38       }
39  }
```

El grafo de esta nueva solución es:



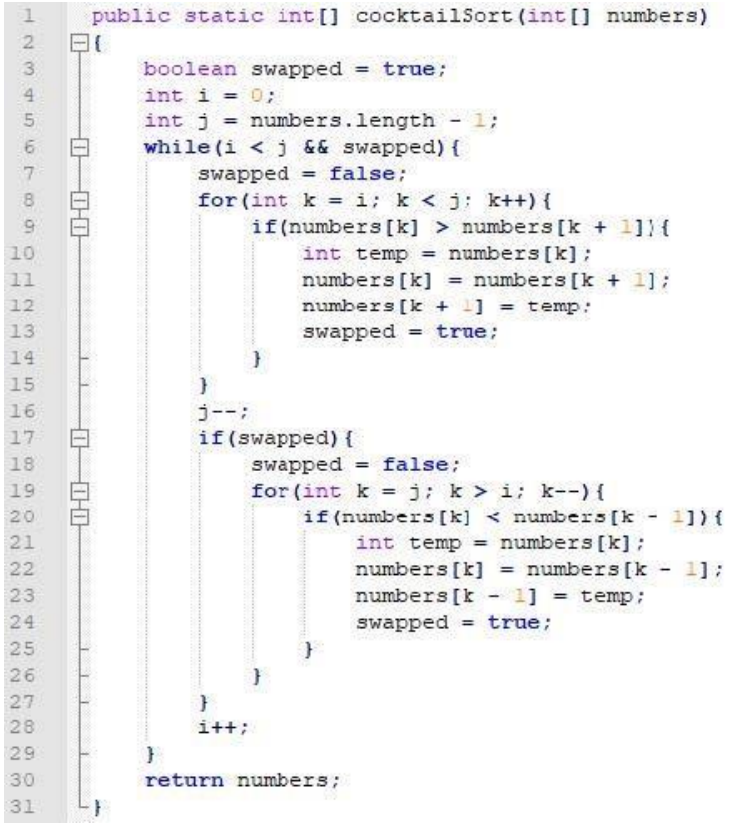
Su complejidad ciclomática es:

- Método 1: Nodos de condición + 1 = 4 + 1 = 5
- Método 2: Aristas – Nodos + 2 = 13 – 10 + 2 = 5
- Método 3: Regiones cerradas + 1 = 4 + 1 = 5

Los posibles caminos a seguir son:

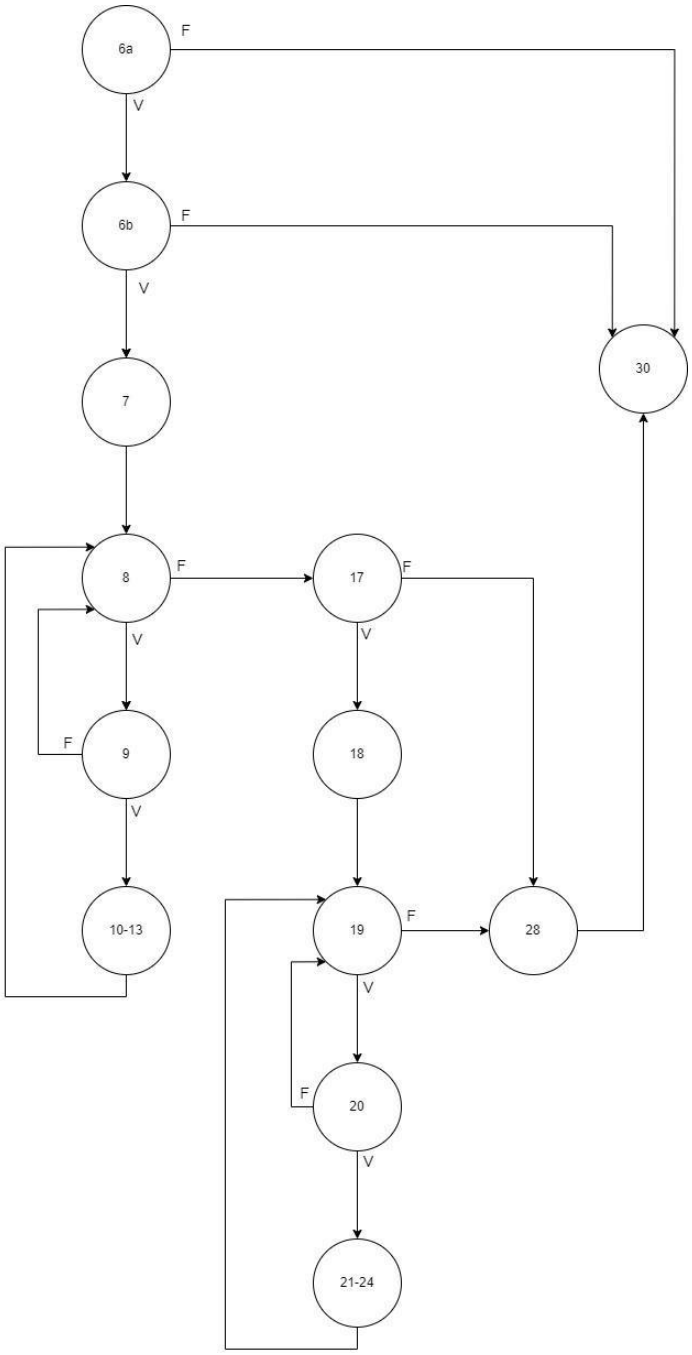
Camino	Valor de prueba	Resultado esperado
9 – 36 - 39	0	“El total debe ser mayor a 0”
9 – 11 – 13 – 39	2.500	0
9 – 11 – 17 – 19 – 39	7.500	225
9 – 11 – 17 – 23 – 25 – 39	15.000	750
9 – 11 – 17 – 23 – 29 – 39	30.000	3.000

c) Para el siguiente fragmento de código:



```
1  public static int[] cocktailSort(int[] numbers)
2  {
3      boolean swapped = true;
4      int i = 0;
5      int j = numbers.length - 1;
6      while(i < j && swapped){
7          swapped = false;
8          for(int k = i; k < j; k++){
9              if(numbers[k] > numbers[k + 1]){
10                 int temp = numbers[k];
11                 numbers[k] = numbers[k + 1];
12                 numbers[k + 1] = temp;
13                 swapped = true;
14             }
15         }
16         j--;
17         if(swapped){
18             swapped = false;
19             for(int k = j; k > i; k--){
20                 if(numbers[k] < numbers[k - 1]){
21                     int temp = numbers[k];
22                     numbers[k] = numbers[k - 1];
23                     numbers[k - 1] = temp;
24                     swapped = true;
25                 }
26             }
27         }
28         i++;
29     }
30     return numbers;
31 }
```

Su grafo es:



Su complejidad ciclomática es:

- Método 1: Nodos de condición + 1 = 7 + 1 = 8
- Método 2: Aristas – Nodos + 2 = 19 – 13 + 2 = 8
- Método 3: Regiones cerradas + 1 = 7 + 1 = 8

Ejercicio 3. Pruebas de Unidad (unitarias).

```
ReglaDeDescuento regla = new ReglaDeDescuento();

@ParameterizedTest

@ValueSource(doubles = {0,-1})

public void calcularDescuentoConNumeroCeroMenorQueCeroYTireExcepcion(double total){

 /*
```

```
        try{
                var pruebaTotal = regla.Calcular(total);
                fail();
    }catch(Exception e){
            assertEquals(e.getMessage(), "El total debe ser mayor a 0" );
    }
 */

  assertThrows(IllegalArgumentException.class,
    () => {regla.Calcular(total);});

//recibe un tipo de exception y un ejecutable
}
@Test
public void calcularDescuentoConNumeroEntre0y5000(){
        double total = 750d;
        var pruebaTotal =regla.Calcular(total);
        assertEquals(pruebaTotal, 0d);
}
@Test
public void calcularDescuentoConNumeroEntre5000y10000(){
        double total = 7500d;
        var pruebaTotal =regla.Calcular(total);
        assertEquals(pruebaTotal, 225d);
}
@Test
public void calcularDescuentoConNumeroEntre10000y25000(){
        double total = 25000d;
        var pruebaTotal =regla.Calcular(total);
        assertEquals(pruebaTotal, 1250d);
}
@Test
public void calcularDescuentoConNumeroMayorDe25000(){
        double total = 30000d;
        var pruebaTotal =regla.Calcular(total);
        assertEquals(pruebaTotal, 3000d);
}
```

5–

Plantilla para caso de prueba

Caso de Prueba		
ID: V001	**Nombre: Realizar Venta – Cliente Normal**	
Descripción: Verificar el proceso de venta para un cliente normal (Consumidor Final) que adquiere uno o varios productos.		
Prioridad: Alta	**CU / HU: Realizar Venta**	
Módulo / Funcionalidad: Ventas		
Diseñado por: Sosa, Terán Nougués, Calisaya, Rozas		**Fecha: 14/11/2023**
Ejecutado por: Sosa, Terán Nougués, Calisaya, Rozas		**Fecha: 14/11/2023**

Precondiciones:

El vendedor esta autenticado y registrado en el sistema

El número de comprobante esta almacenado en el sistema

Se dispone de stock suficiente para los productos involucrados

Paso	Acción	Resultado Esperado	Pasó / Falló	Comentarios
1	El cliente se presenta en el punto de venta.	El Vendedor inicia una nueva venta.		
2	El Vendedor registra los productos que el cliente desea comprar.	Los productos se agregan correctamente a la venta.		
3	Se selecciona el método de pago como "Efectivo".	El monto total se calcula correctamente.		
4	Se completa la venta.	Se emite el comprobante correspondiente y se actualiza el stock.		

ID: Identificador | CU: Caso de Uso | HU: Historia de Usuario

Caso de Prueba		
ID: V002	**Nombre: Realizar Venta - Cliente con Condición Tributaria**	
Descripción: Verificar el proceso de venta para un cliente con condición tributaria distinta a "Consumidor Final".		
Prioridad: Alta	**CU / HU: Realizar venta**	
Módulo / Funcionalidad: ventas		
Diseñado por: Sosa, Terán Nougués, Calisaya, Rozas		**Fecha: 14/11/2023**
Ejecutado por: Sosa, Terán Nougués, Calisaya, Rozas		**Fecha: 14/11/2023**

Precondiciones:

- **El Vendedor está autenticado y registrado en el sistema.**
- **El número de comprobante está almacenado en el sistema.**
- **Se dispone de stock suficiente para los productos involucrados.**

Paso	Acción	Resultado Esperado	Pasó / Falló	Comentarios
1	**El cliente con condición tributaria diferente a "Consumidor Final" se presenta en el punto de venta.**	Se asocia al cliente el tipo de comprobante según su condición tributaria.		
2	**El Vendedor registra los productos que el cliente desea comprar.**	**Los productos se agregan correctamente a la venta.**		
3	**Se selecciona el método de pago como "Tarjeta".**	Se inicia el proceso de pago con tarjeta.		
4	**Se completa la venta.**	**Se emite el comprobante correspondiente y se actualiza el stock.**		

ID: Identificador | CU: Caso de Uso | HU: Historia de Usuario