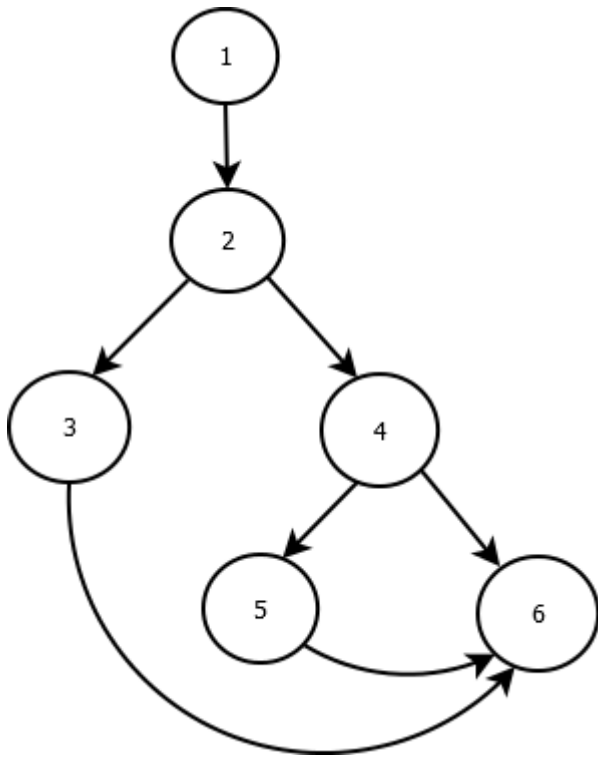


### EJERCICIO 1. Consultar positivo



- Cantidad de regiones: 3
- Cantidad de nodos: 7
- Cantidad de aristas: 8
- Complejidad ciclomática:
  - a)  $V(G) = a - n + 2 = 7 - 6 + 2 = 3$
  - b)  $V(G) = r = 3$
  - c)  $V(G) = c + 1 = 2 + 1 = 3$
- Caminos posibles:
  - a) 1, 2, 3, 4, 7
  - b) 1, 2, 3, 5, 6, 7
  - c) 1, 2, 3, 5, 7

## EJERCICIO 2. Factura eléctrica

La clase tiene los siguientes problemas:

- Cuando el consumo es igual o menor que 0, es decir, 0 o un valor negativo, va a entender que el precio va a ser de 9 centimos ya que cumple la condicion de  $\leq 300$ , pero lo logico es que el precio sea de 0 centimos.

```
63     @Test
64     public void test7() {
65         Factura factura = new Factura();
66         factura.setConsumokWh(-1);
67         int resultadoEsperado = 0;
68         int resultadoObtenido = factura.calcularPrezokWh();
69         Assertions.assertEquals(resultadoEsperado, resultadoObtenido);
70     }
71     @Test
72     public void test8() {
73         Factura factura = new Factura();
74         factura.setConsumokWh(-1);
75         int resultadoEsperado = 9;
76         int resultadoObtenido = factura.calcularPrezokWh();
77         Assertions.assertEquals(resultadoEsperado, resultadoObtenido);
78     }
79 }
```

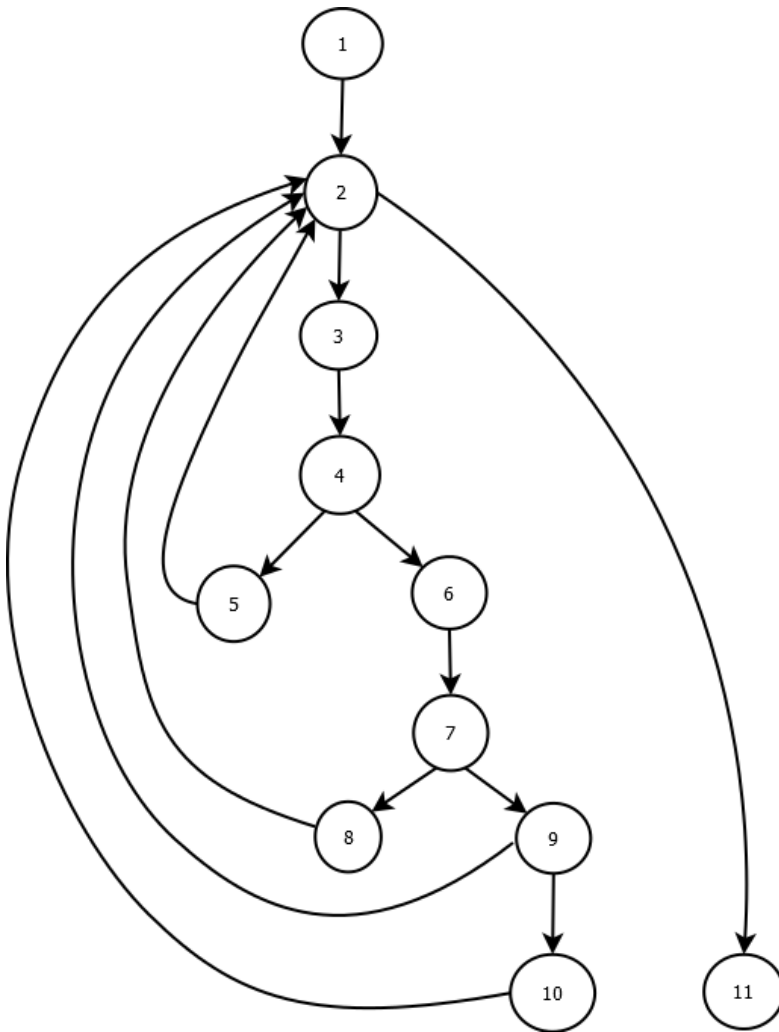
- Cuando el consumo es mayor que 2000, va a entender que el precio va a ser de 0 centimos ya que no cumple ninguna condicion. Lo logico es que a partir de 2000 de consumo, se establezca un precio, por ejemplo 4 centimos.

```
80     @Test
81     public void test9() {
82         Factura factura = new Factura();
83         factura.setConsumokWh(consumokWh: 2500);
84         int resultadoEsperado = 0;
85         int resultadoObtenido = factura.calcularPrezokWh();
86         Assertions.assertEquals(resultadoEsperado, resultadoObtenido);
87     }
88
89     @Test
90     public void test10() {
91         Factura factura = new Factura();
92         factura.setConsumokWh(consumokWh: 2500);
93         int resultadoEsperado = 4;
94         int resultadoObtenido = factura.calcularPrezokWh();
95         Assertions.assertEquals(resultadoEsperado, resultadoObtenido);
96     }
97 }
```

Ahora realizadas las pruebas, corrigo el codigo, añadiendo lo siguiente:

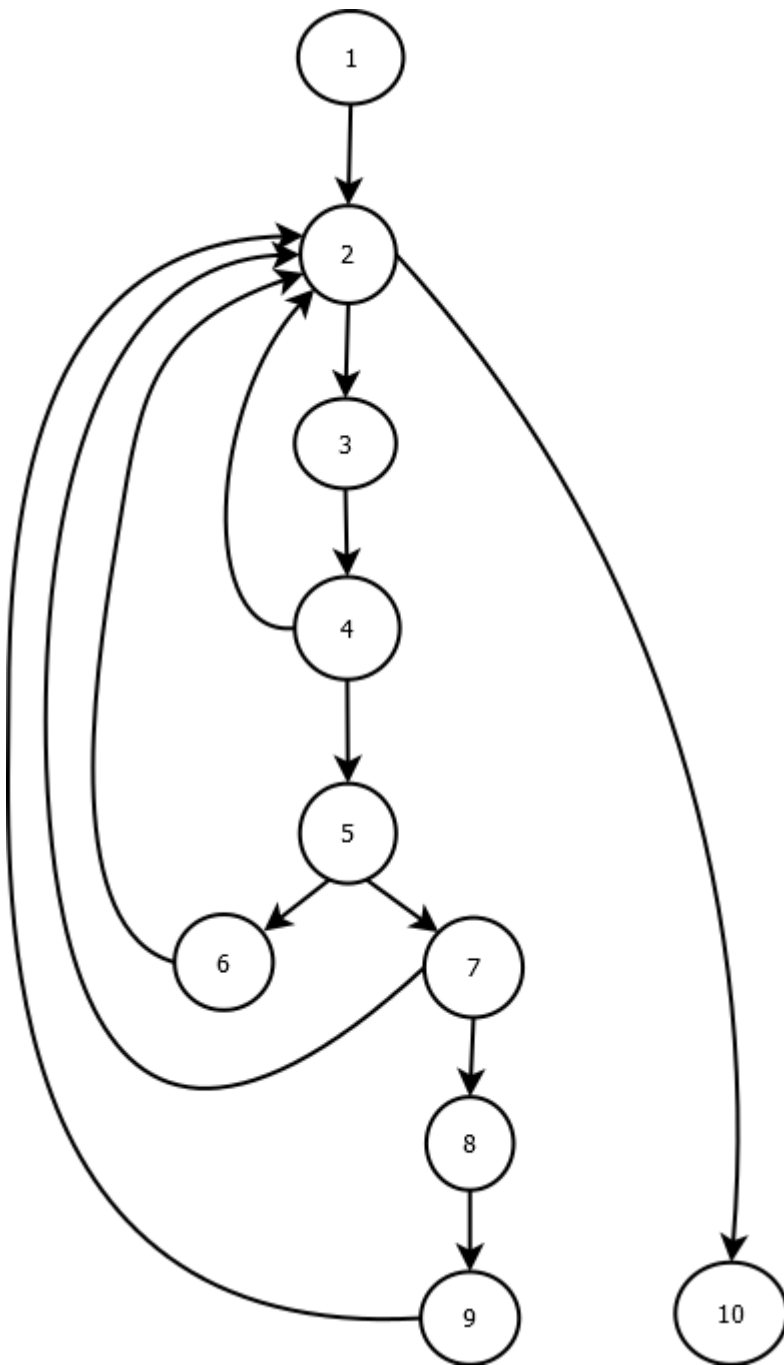
```
public int calcularPrezokWh(){
    int prezokWh=0;
    if(consumokWh<=0){
        return prezokWh;
    }else if (consumokWh<=300){
        prezokWh=9;
    }else if (consumokWh<=600){
        prezokWh=8;
    }else if (consumokWh<=1000){
        prezokWh=6;
    }else if (consumokWh<=2000){
        prezokWh=5;
    }else if (consumokWh>2000){
        prezokWh=4;
    }
    return prezokWh;
}
```

### EJERCICIO 3. Arrays



- Cantidad de regiones: 5
- Cantidad de nodos: 11
- Cantidad de aristas: 14
- Complejidad ciclomática:
  - a)  $V(G) = a - n + 2 = 14 - 11 + 2 = 5$
  - b)  $V(G) = r = 5$
  - c)  $V(G) = c + 1 = 4 + 1 = 5$
- Caminos posibles:
  - a) 1, 2, 11
  - b) 1, 2, 3, 4, 5, 2, 11
  - c) 1, 2, 3, 4, 6, 7, 8, 2, 11
  - d) 1, 2, 3, 4, 6, 7, 9, 10, 2, 11
  - e) 1, 2, 3, 4, 6, 7, 9, 11

#### EJERCICIO 4. Acrónimos



- Cantidad de regiones: 5
- Cantidad de nodos: 10
- Cantidad de aristas: 13
- Complejidad ciclomática:
  - a)  $V(G) = a - n + 2 = 13 - 10 + 2 = 5$
  - b)  $V(G) = r = 5$
  - c)  $V(G) = c + 1 = 4 + 1 = 5$
- Caminos posibles:
  - a) 1, 2, 10
  - b) 1, 2, 3, 4, 2, 10
  - c) 1, 2, 3, 4, 5, 6, 2, 10
  - d) 1, 2, 3, 4, 5, 7, 2, 10
  - e) 1, 2, 3, 4, 5, 7, 8, 9, 2, 10

## EJERCICIO 5. Pruebas unitarias con Junit

Una vez creado el código y haciendo comprobaciones, encuentre un problema en mi código, el cual, al intentar dividir cualquier número entre el número divisor "0", nos da un error. Para solucionar esto creo una excepción.

Código:

```
package repaso;

public class Calculadora {

    public int suma(int a, int b) {
        return a + b;
    }

    public int resta(int a, int b) {
        return a - b;
    }

    public int multiplicacion(int a, int b) {
        return a * b;
    }

    public int division(int a, int b) throws Exception {
        if (b == 0) {
            throw new Exception(message: "No se puede dividir entre cero");
        }
        return a / b;
    }

}
```

## Pruebas con JUnit:

The screenshot displays an IDE interface with the following components:

- Left Panel (Test Results):** Shows a tree view of test results. The root is 'junit' (38ms), which contains 'repaso' (38ms). Under 'repaso' is 'CalculadoraTest' (8.0ms), which contains five test methods: 'testSuma()' (0.0ms), 'testResta()' (3.0ms), 'testMultiplicacion()' (0.0ms), 'testDivision()' (0.0ms), and 'testDivisionPorCero()' (5.0ms). The 'FacturaTest' (30ms) is also listed but is not expanded.
- Center Panel (Source Code):** Displays the source code for 'CalculadoraTest.java'. The code includes package declarations, imports, and five test methods: 'testSuma()', 'testResta()', 'testMultiplicacion()', 'testDivision()', and 'testDivisionPorCero()'. The 'testDivisionPorCero()' method is highlighted with a red circle, indicating a failure.
- Right Panel (Test Run Log):** Shows a list of test runs. The first two runs are for 'testDivisionPorCero()' and both failed with the exception 'java.lang.Exception: No se puede dividir entre cero'. The subsequent three runs are for 'testDivision()' and all passed.

```
1 package repaso;
2 import org.junit.Test;
3 import static org.junit.Assert.*;
4
5 public class CalculadoraTest {
6
7     @Test
8     public void testSuma() {
9         Calculadora calculadora = new Calculadora();
10        int resultado = calculadora.suma(a: 2, b: 3);
11        assertEquals(5, resultado);
12    }
13
14    @Test
15    public void testResta() {
16        Calculadora calculadora = new Calculadora();
17        int resultado = calculadora.resta(a: 5, b: 2);
18        assertEquals(3, resultado);
19    }
20
21    @Test
22    public void testMultiplicacion() {
23        Calculadora calculadora = new Calculadora();
24        int resultado = calculadora.multiplicacion(a: 4, b: 3);
25        assertEquals(12, resultado);
26    }
27
28    @Test
29    public void testDivision() throws Exception {
30        Calculadora calculadora = new Calculadora();
31        int resultado = calculadora.division(a: 10, b: 2);
32        assertEquals(5.0, resultado, 0);
33    }
34
35    @Test
36    public void testDivisionPorCero() throws Exception {
37        Calculadora calculadora = new Calculadora();
38        int resultado = calculadora.division(a: 10, b: 0);
```

java.lang.Exception: No se puede dividir entre cero at repaso.CalculadoraTest.testDivisionPorCero(CalculadoraTest.java:38)

Test run at 3/10/2023, 11:06:49 PM  
testDivisionPorCero()  
Test run at 3/10/2023, 11:05:54 PM  
Test run at 3/10/2023, 11:03:06 PM  
Test run at 3/10/2023, 11:02:39 PM  
Test run at 3/10/2023, 11:02:32 PM  
Test run at 3/10/2023, 11:02:23 PM