International Institute of Information Technology, Bangalore

CS 731: Software Testing

Submitted by,

Sani Ranpariya **MT2021116**

Vishwajeet Deulkar **MT2021154**

Under the Guidance of

Prof. Meenakshi D Souza

Contents

1. Overview

The goal of this project is to understand and perform practical aspects of testing. We have used Control Flow Graph criteria for testing on a sample code that covers the edge and prime path for all the functions in the code.

2. Code Description

We have used the Project Scientific Calculator. It is a web-based application which is built using Java, HTML,CSS & JavaScript. It has modules given below.

1. Mathematical Operations which includes - Factorial, Log, Square-root, Power.
2. Area calculation of various shapes - Circle, Square, Rectangle, Traingle.
3. Quadratic equation solver.
4. Temperature unit converter.
5. Area unit converter.
6. Time unit converter.
7. Length unit converter.
8. Weight unit converter.

Source Code link : Scientific Calculator

3. Testing Strategy Used

Control Flow Graph

Control flow testing is a software testing technique that uses the control flow technique and it is represented as white box testing. Control flow testing is a testing strategy in a structured manner that depicts the execution order of the statements or instructions given. It is used to develop test cases of a program, where the tester selects a large portion of the program to test and to set the testing path. The whole test cases are represented in the form of a control flow graph like node, edge, and are mostly used in unit testing.

Testing tools used

We have used JUnit to design our test cases. JUnit is used as a testing framework for unit testing in Java programming language.

4. Test Case Description

As our code has decision statements and loops that are nested, we used prime paths to pull out all possible ways of covering nested loops. In this project all test cases are designed using edge coverage and prime paths coverage.

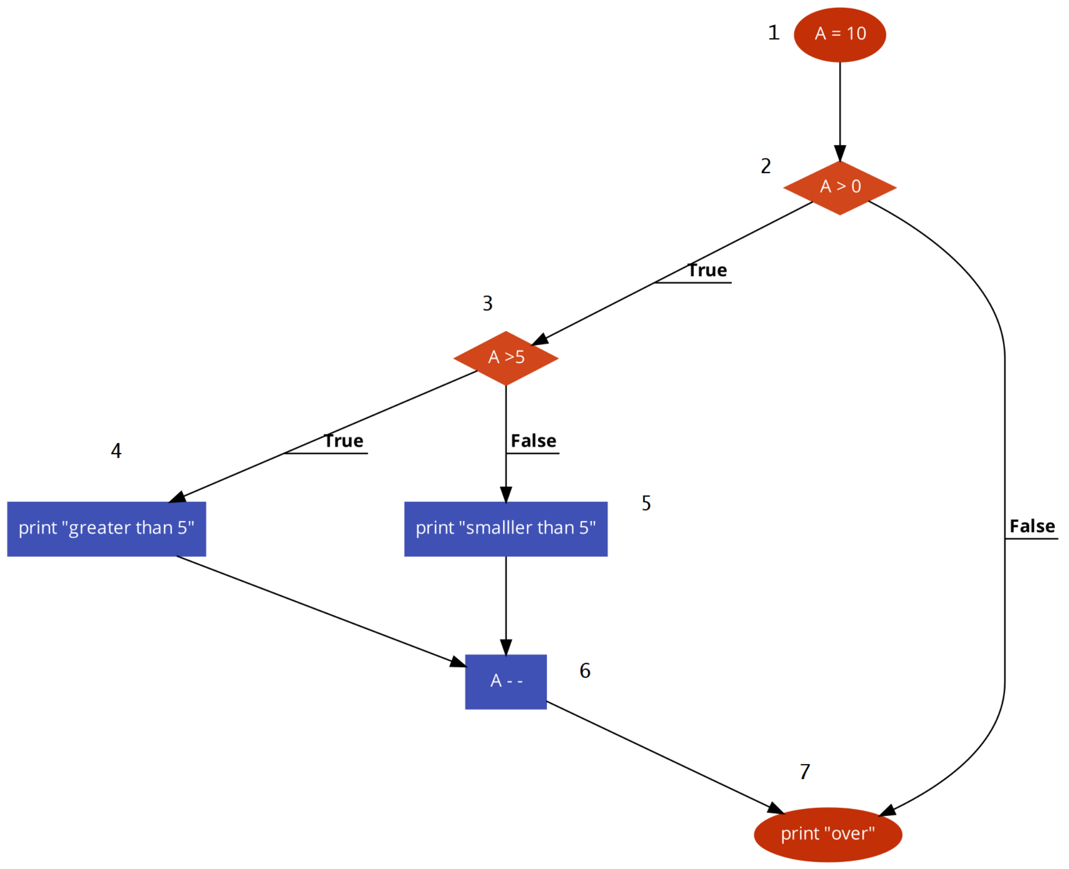
Edge Coverage : It answers the question of, has every edge in the control-flow graph been executed.

Prime Path coverage : A prime path is basically a simple path and it does not appear as a sub-path of any other simple path.

Example :
Sample pseudo Code :

```
A = 10;
if( A > 0){
  if(A >5){
    print "greater than 5";
  }
  else{
    print "smallller than 5";
  }
  A--;
}
print "over";
```

CFG :



Edge Coverage :
3 test paths are needed for Edge Coverage
[1,2,3,5,6,7]
[1,2,3,4,6,7]
[1,2,7]

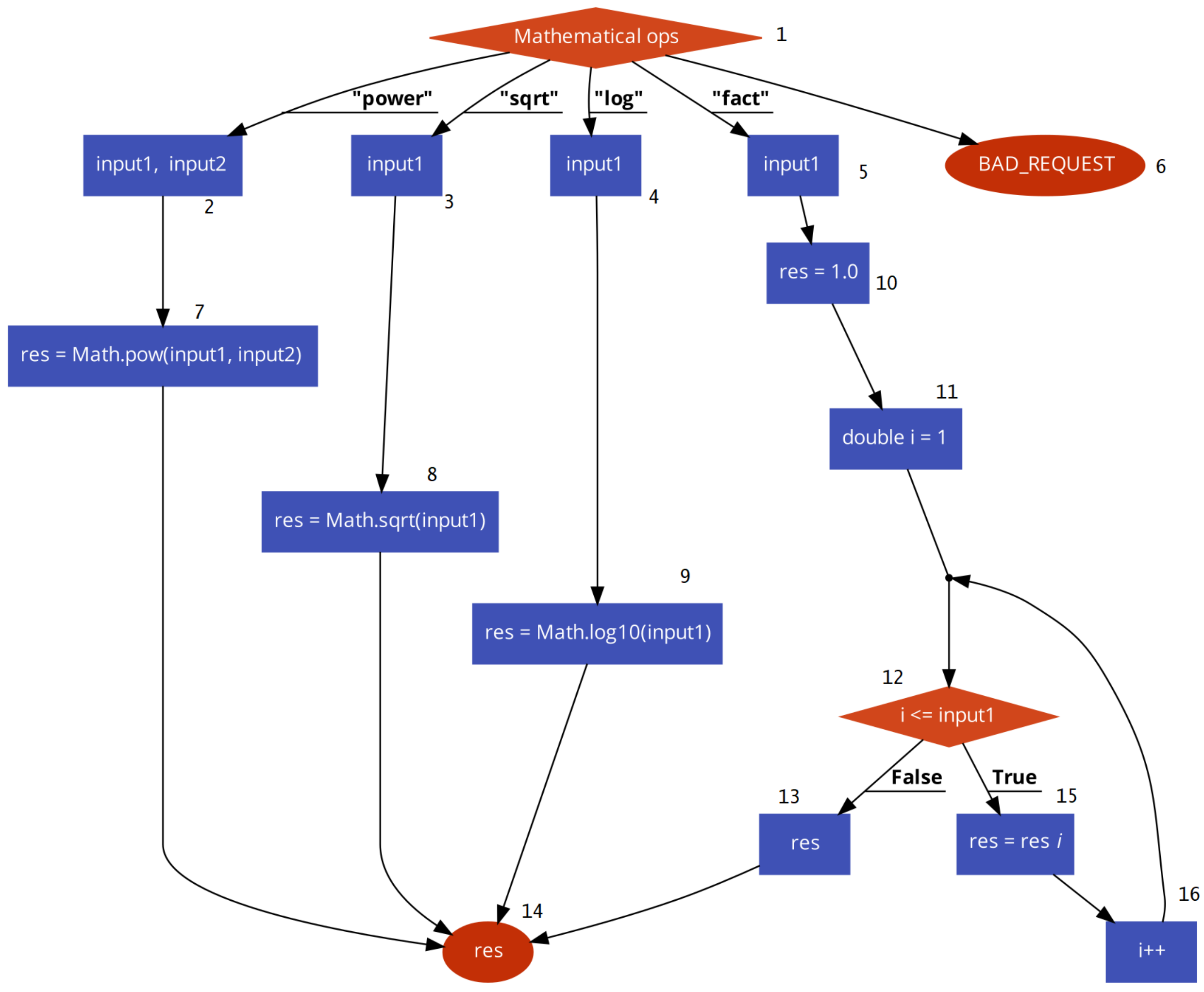
Prime Path Coverage :

Test Paths	Test Requirements that are toured by test paths directly
[1, 2, 7]	[1, 2, 7]
[1, 2, 3, 4, 6, 7]	[1, 2, 3, 4, 6, 7]
[1, 2, 3, 5, 6, 7]	[1, 2, 3, 5, 6, 7]

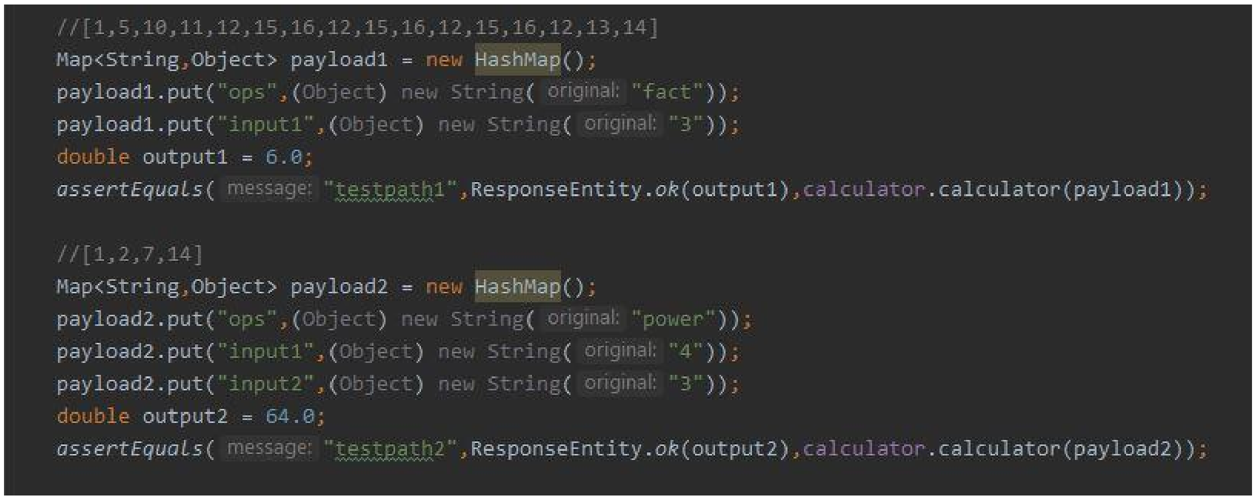
5. Control Flow graph for the code used

I. Mathematical Operations

CFG



Sample Test Cases



```
//[1,5,10,11,12,15,16,12,15,16,12,15,16,12,13,14]
Map<String,Object> payload1 = new HashMap();
payload1.put("ops",(Object) new String( original: "fact"));
payload1.put("input1",(Object) new String( original: "3"));
double output1 = 6.0;
assertEquals( message: "testpath1",ResponseEntity.ok(output1),calculator.calculator(payload1));

//[1,2,7,14]
Map<String,Object> payload2 = new HashMap();
payload2.put("ops",(Object) new String( original: "power"));
payload2.put("input1",(Object) new String( original: "4"));
payload2.put("input2",(Object) new String( original: "3"));
double output2 = 64.0;
assertEquals( message: "testpath2",ResponseEntity.ok(output2),calculator.calculator(payload2));
```

Edge Coverage

6 test paths are needed for Edge Coverage
[1,5,10,11,12,15,16,12,15,16,12,13,14]
[1,5,10,11,12,13,14]
[1,4,9,14]
[1,3,8,14]
[1,2,7,14]
[1,6]

Prime Path Coverage

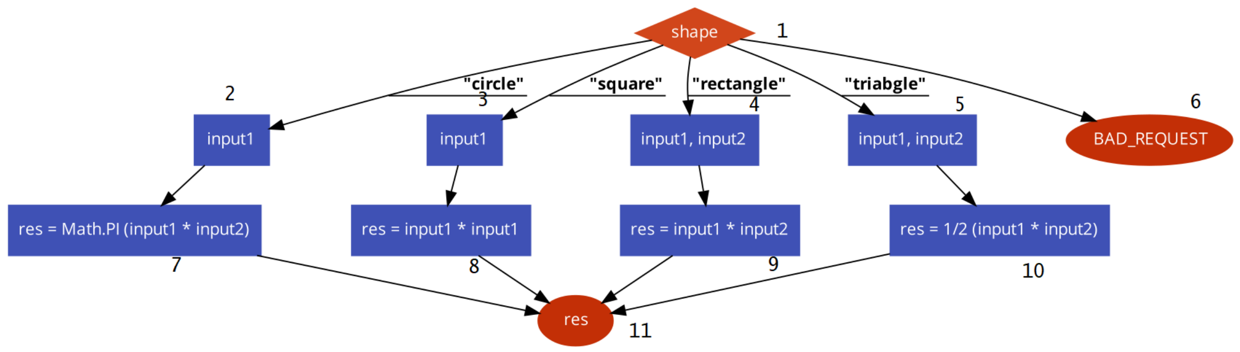
7 test paths are needed for Prime Path Coverage.

Test Paths	Test Requirements that are toured by test paths directly
[1,5,10,11,12,15,16,12,15,16,12,15,16,12,13, 14]	[1,5,10,11,12,15,16], [15,16,12,13,14], [15,16,12,15], [16,12,15,16], [12,15,16,12]
[1,6]	[1,6]
[1,2,7,14]	[1,2,7,14]
[1,3,8,14]	[1,3,8,14]

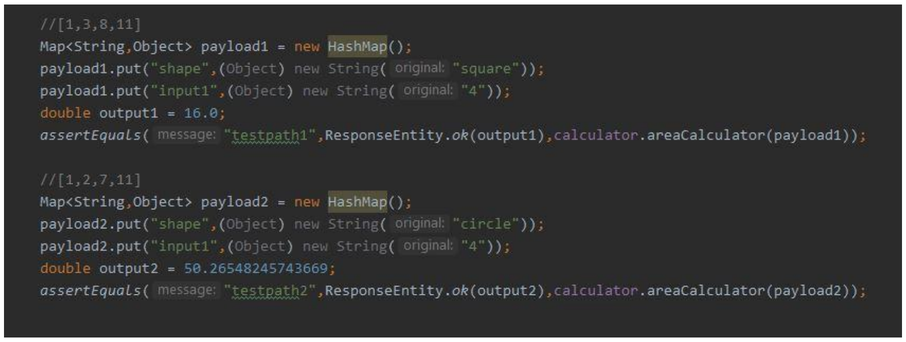
[1,4,9,14]	[1,4,9,14]
[1,5,10,11,12,15,16,12,13,14]	[1,5,10,11,12,15,16], [15,16,12,13,14], [12,15,16,12]
[1,5,10,11,12,13,14]	[1,5,10,11,12,13,14]

II. Area Calculations

CFG



Sample Test Cases



```
//[1,3,8,11]
Map<String,Object> payload1 = new HashMap();
payload1.put("shape",(Object) new String( original: "square"));
payload1.put("input1",(Object) new String( original: "4"));
double output1 = 16.0;
assertEquals( message: "testpath1",ResponseEntity.ok(output1),calculator.areaCalculator(payload1));

//[1,2,7,11]
Map<String,Object> payload2 = new HashMap();
payload2.put("shape",(Object) new String( original: "circle"));
payload2.put("input1",(Object) new String( original: "4"));
double output2 = 50.26548245743669;
assertEquals( message: "testpath2",ResponseEntity.ok(output2),calculator.areaCalculator(payload2));
```

Edge Coverage

5 test paths are needed for Edge Coverage
[1,5,10,11]

[1,4,9,11]
[1,3,8,11]
[1,2,7,11]
[1,6]

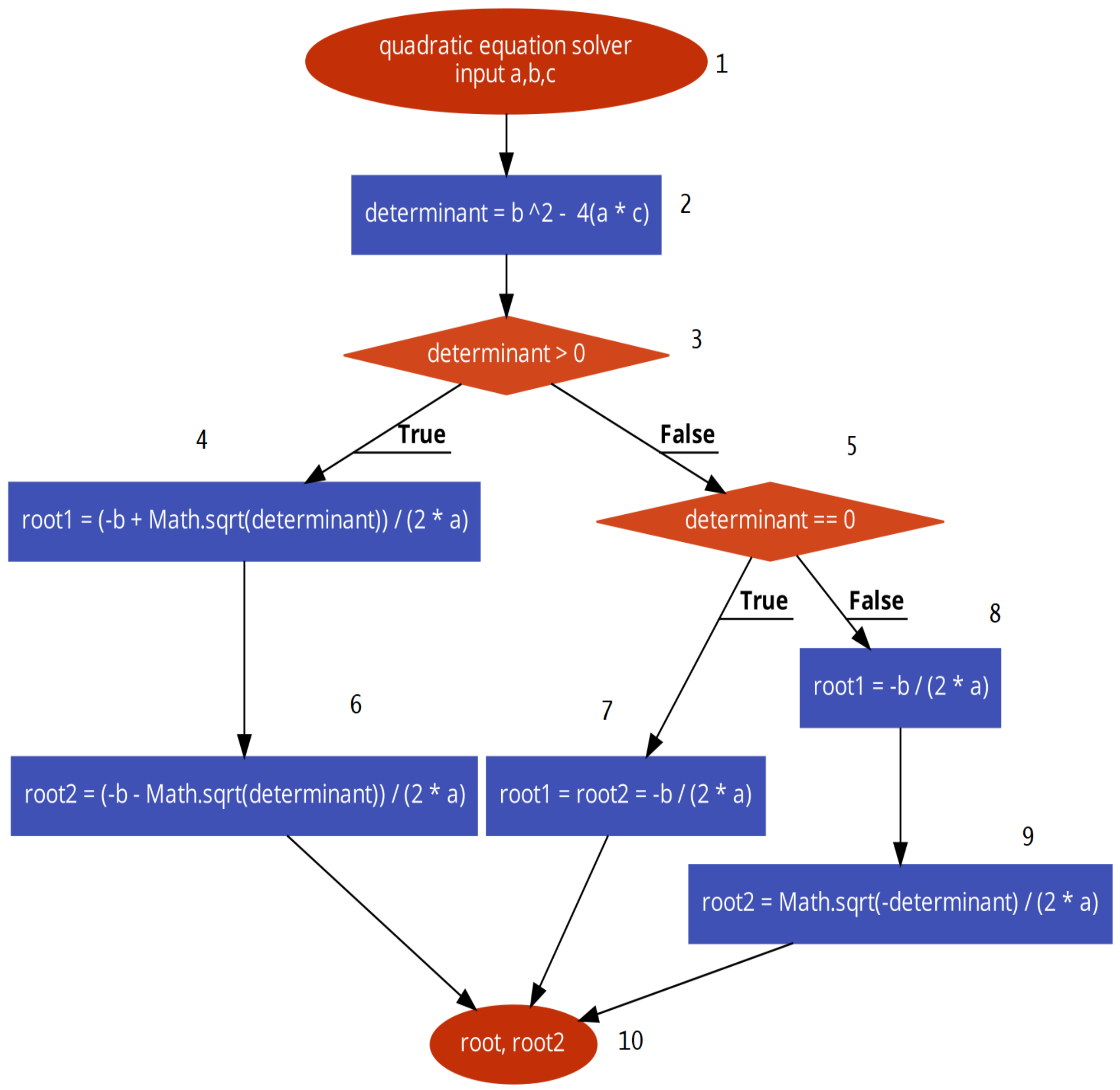
Prime Path Coverage

5 test paths are needed for Prime Path Coverage using the prefix graph algorithm

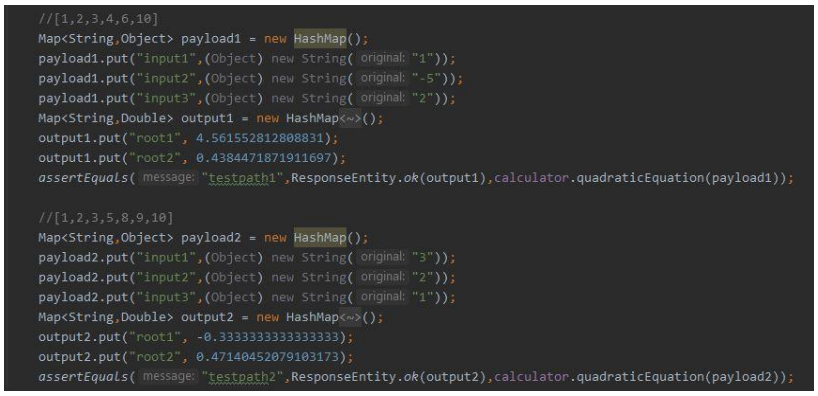
Test Paths	Test Requirements that are toured by test paths directly
[1,6]	[1,6]
[1,3,8,11]	[1,3,8,11]
[1,2,7,11]	[1,2,7,11]
[1,4,9,11]	[1,4,9,11]
[1,5,10,11]	[1,5,10,11]

III. Quadratic Equation Solver

CFG


quadratic equation solver
input a,b,c
1
determinant = b ^2 - 4(a * c)
2
determinant > 0
3
True
False
4
root1 = (-b + Math.sqrt(determinant)) / (2 * a)
5
determinant == 0
True
False
8
root1 = -b / (2 * a)
6
7
root2 = (-b - Math.sqrt(determinant)) / (2 * a)
root1 = root2 = -b / (2 * a)
9
root2 = Math.sqrt(-determinant) / (2 * a)
root, root2
10

Sample Test Cases



```java
//[1,2,3,4,6,10]
Map<String,Object> payload1 = new HashMap();
payload1.put("input1",(Object) new String( original: "1"));
payload1.put("input2",(Object) new String( original: "-5"));
payload1.put("input3",(Object) new String( original: "2"));
Map<String,Double> output1 = new HashMap<~>();
output1.put("root1", 4.561552812808831);
output1.put("root2", 0.4384471871911697);
assertEquals( message: "testpath1",ResponseEntity.ok(output1),calculator.quadraticEquation(payload1));

//[1,2,3,5,8,9,10]
Map<String,Object> payload2 = new HashMap();
payload2.put("input1",(Object) new String( original: "3"));
payload2.put("input2",(Object) new String( original: "2"));
payload2.put("input3",(Object) new String( original: "1"));
Map<String,Double> output2 = new HashMap<~>();
output2.put("root1", -0.3333333333333333);
output2.put("root2", 0.47140452079103173);
assertEquals( message: "testpath2",ResponseEntity.ok(output2),calculator.quadraticEquation(payload2));
```

Edge Coverage

3 test paths are needed for Edge Coverage
[1,2,3,5,8,9,10]
[1,2,3,5,7,10]
[1,2,3,4,6,10]

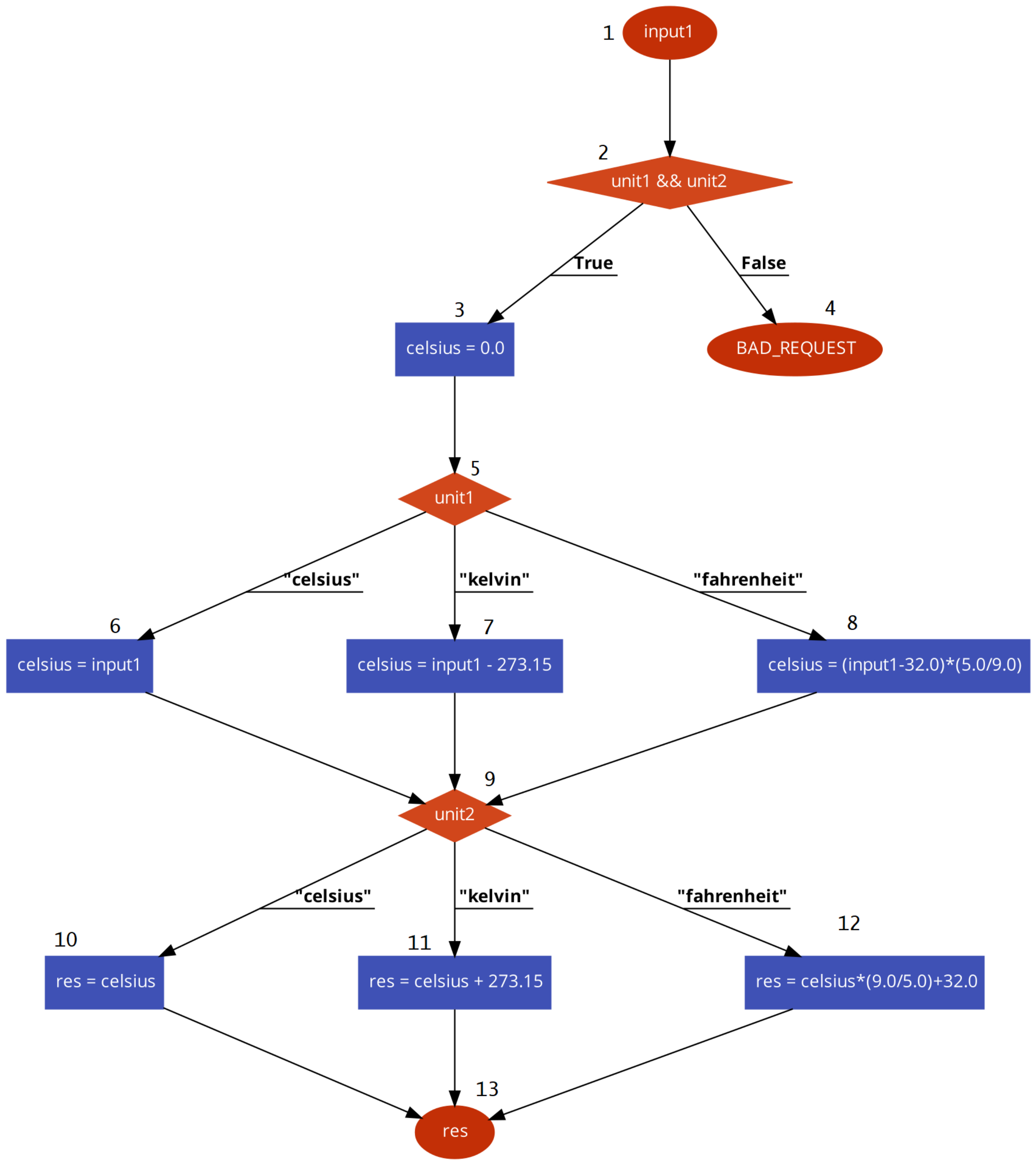
Prime Path Coverage

3 test paths are needed for Prime Path Coverage using the prefix graph algorithm

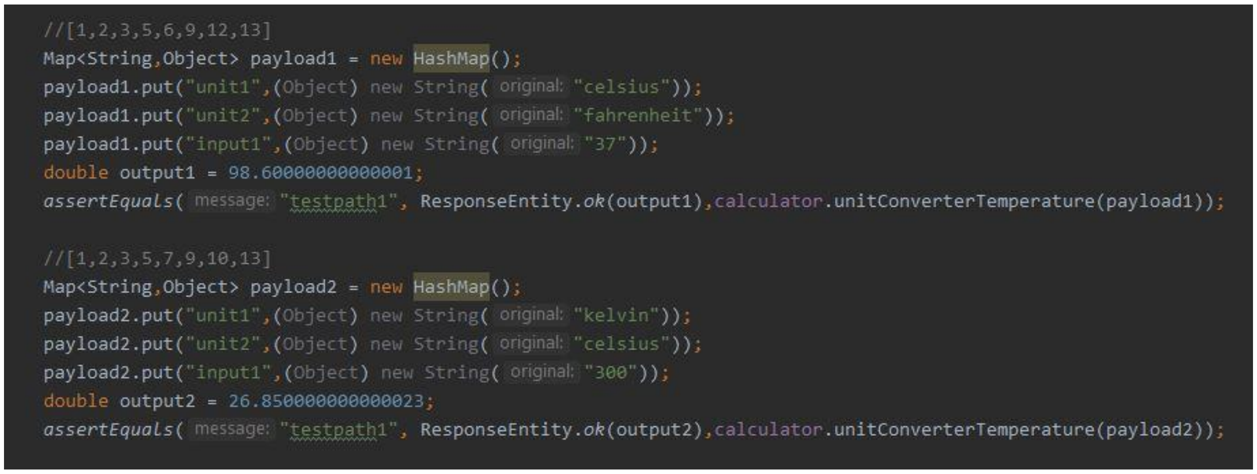
Test Paths	Test Requirements that are toured by test paths directly
[1,2,3,5,7,10]	[1,2,3,5,7,10]
[1,2,3,4,6,10]	[1,2,3,4,6,10]
[1,2,3,5,8,9,10]	[1,2,3,5,8,9,10]

IV. Temperature Converter

CFG



Sample Test Cases



```
//[1,2,3,5,6,9,12,13]
Map<String,Object> payload1 = new HashMap();
payload1.put("unit1",(Object) new String( original: "celsius"));
payload1.put("unit2",(Object) new String( original: "fahrenheit"));
payload1.put("input1",(Object) new String( original: "37"));
double output1 = 98.60000000000001;
assertEquals( message: "testpath1", ResponseEntity.ok(output1),calculator.unitConverterTemperature(payload1));

//[1,2,3,5,7,9,10,13]
Map<String,Object> payload2 = new HashMap();
payload2.put("unit1",(Object) new String( original: "kelvin"));
payload2.put("unit2",(Object) new String( original: "celsius"));
payload2.put("input1",(Object) new String( original: "300"));
double output2 = 26.850000000000023;
assertEquals( message: "testpath1", ResponseEntity.ok(output2),calculator.unitConverterTemperature(payload2));
```

Edge Coverage

4 test paths are needed for Edge Coverage

[1,2,3,5,8,9,12,13]
[1,2,3,5,7,9,11,13]
[1,2,3,5,6,9,10,13]
[1,2,4]

Prime Path Coverage

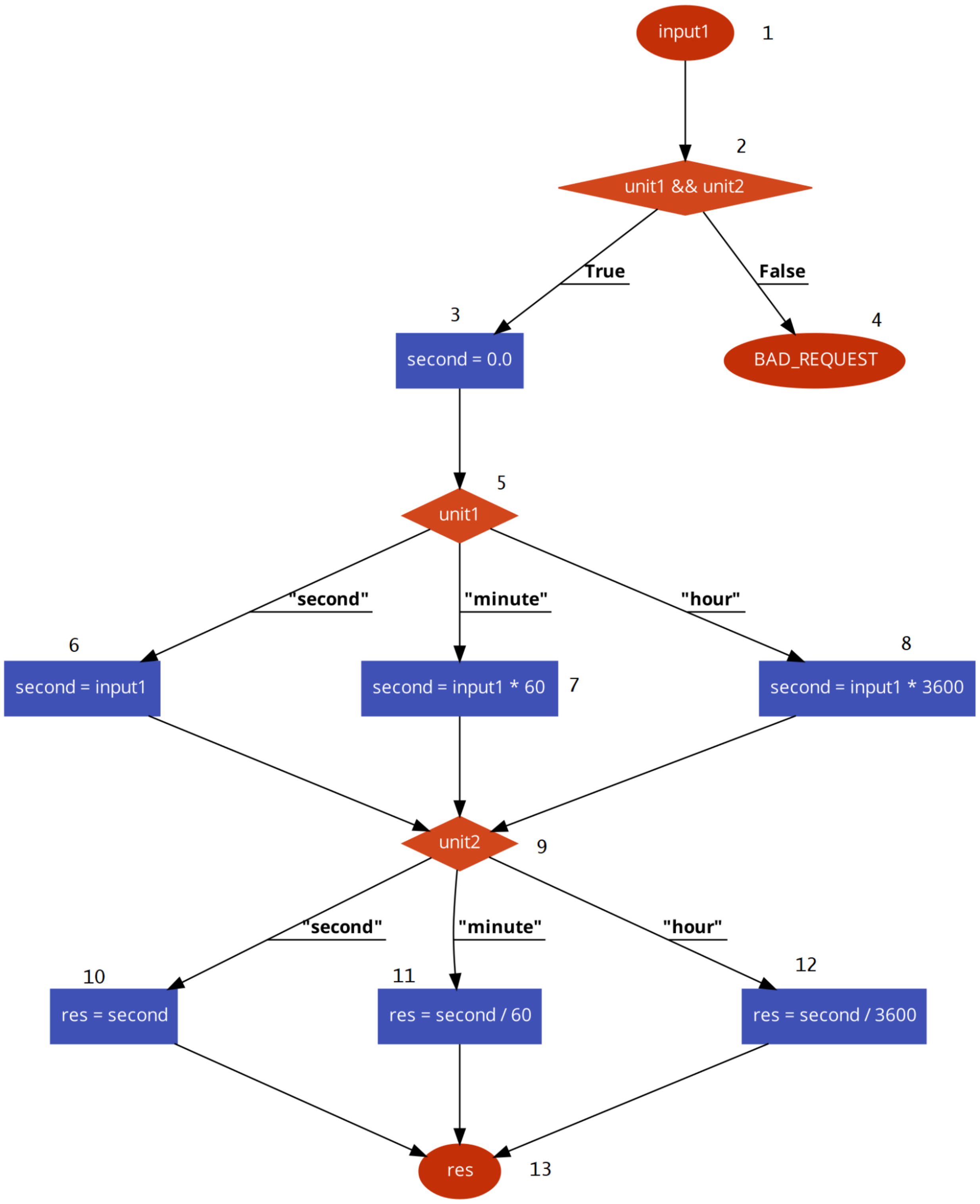
10 test paths are needed for Prime Path Coverage using the prefix graph algorithm

Test Paths	Test Requirements that are toured by test paths directly
[1,2,4]	[1,2,4]
[1,2,3,5,6,9,12,13]	[1,2,3,5,6,9,12,13]
[1,2,3,5,7,9,10,13]	[1,2,3,5,7,9,10,13]
[1,2,3,5,6,9,11,13]	[1,2,3,5,6,9,11,13]
[1,2,3,5,6,9,10,13]	[1,2,3,5,6,9,10,13]
[1,2,3,5,7,9,11,13]	[1,2,3,5,7,9,11,13]
[1,2,3,5,8,9,11,13]	[1,2,3,5,8,9,11,13]

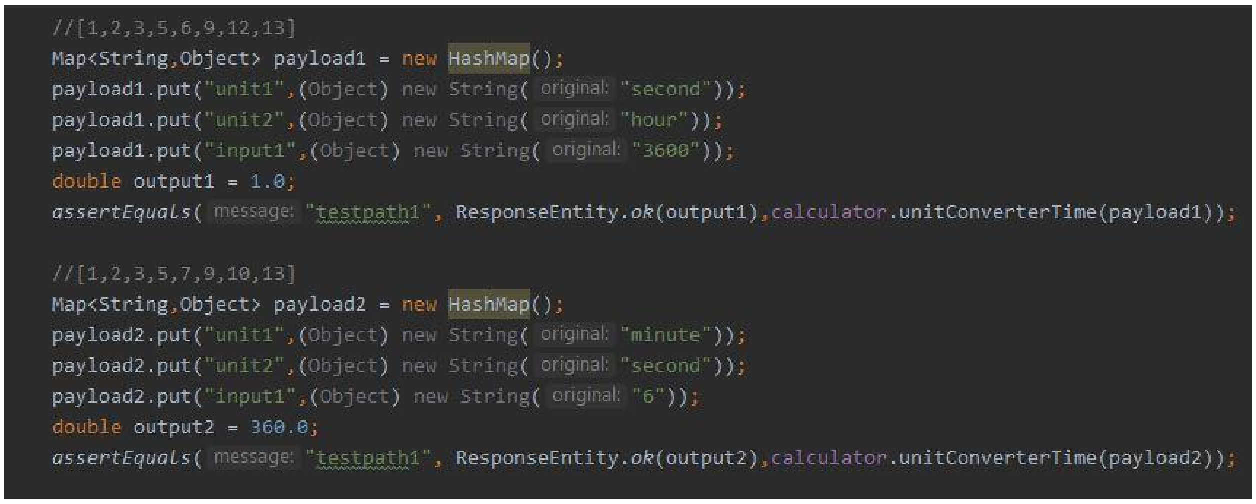
[1,2,3,5,8,9,12,13]	[1,2,3,5,8,9,12,13]
[1,2,3,5,8,9,10,13]	[1,2,3,5,8,9,10,13]
[1,2,3,5,7,9,12,13]	[1,2,3,5,7,9,12,13]

V. Time Converter

CFG



Sample Test Cases


```
//[1,2,3,5,6,9,12,13]
Map<String,Object> payload1 = new HashMap();
payload1.put("unit1",(Object) new String( original: "second"));
payload1.put("unit2",(Object) new String( original: "hour"));
payload1.put("input1",(Object) new String( original: "3600"));
double output1 = 1.0;
assertEquals( message: "testpath1", ResponseEntity.ok(output1),calculator.unitConverterTime(payload1));

//[1,2,3,5,7,9,10,13]
Map<String,Object> payload2 = new HashMap();
payload2.put("unit1",(Object) new String( original: "minute"));
payload2.put("unit2",(Object) new String( original: "second"));
payload2.put("input1",(Object) new String( original: "6"));
double output2 = 360.0;
assertEquals( message: "testpath1", ResponseEntity.ok(output2),calculator.unitConverterTime(payload2));
```

Edge Coverage

4 test paths are needed for Edge Coverage
[1,2,3,5,8,9,12,13]
[1,2,3,5,7,9,11,13]
[1,2,3,5,6,9,10,13]
[1,2,4]

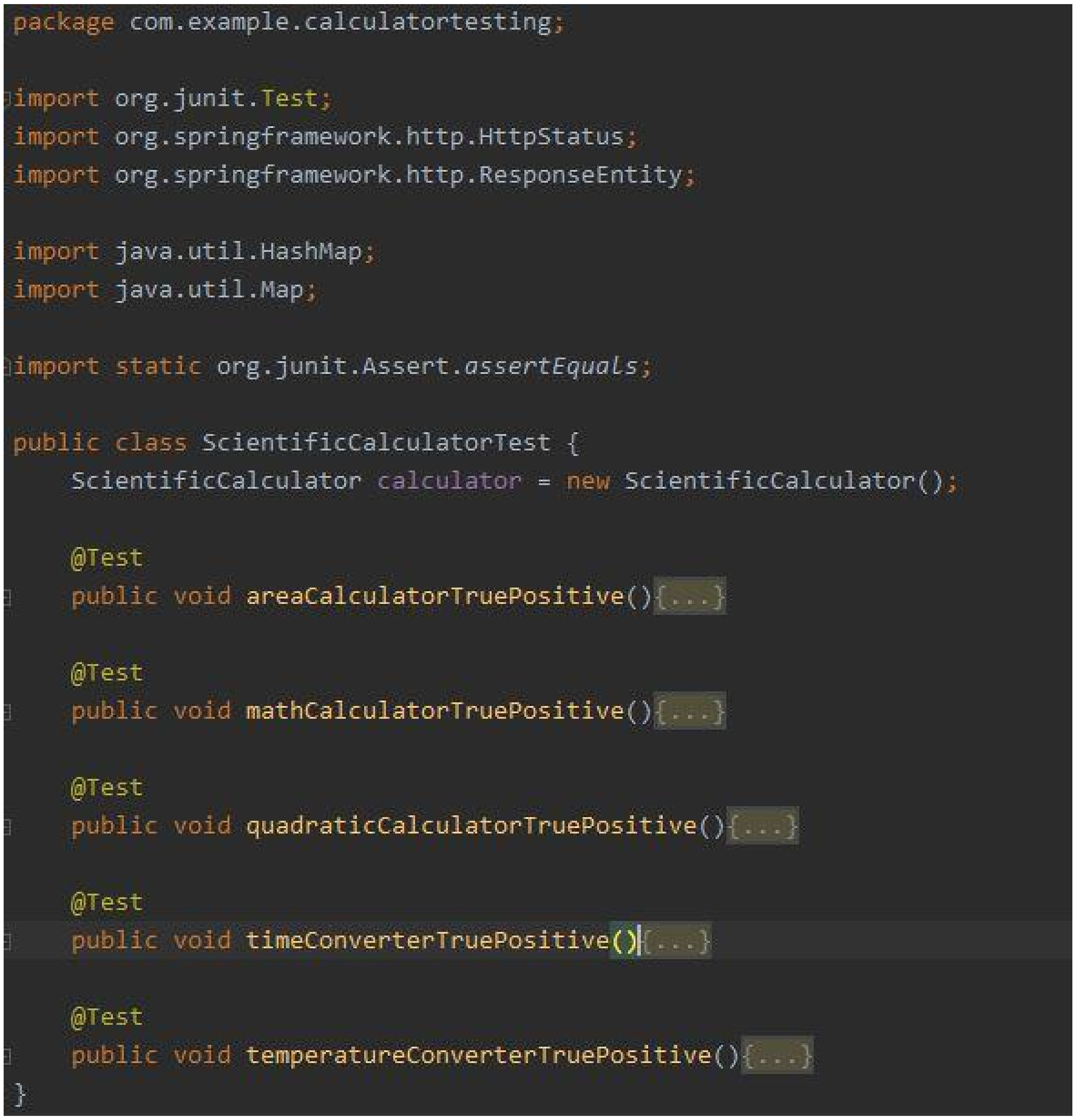
Prime Path Coverage

10 test paths are needed for Prime Path Coverage using the prefix graph algorithm

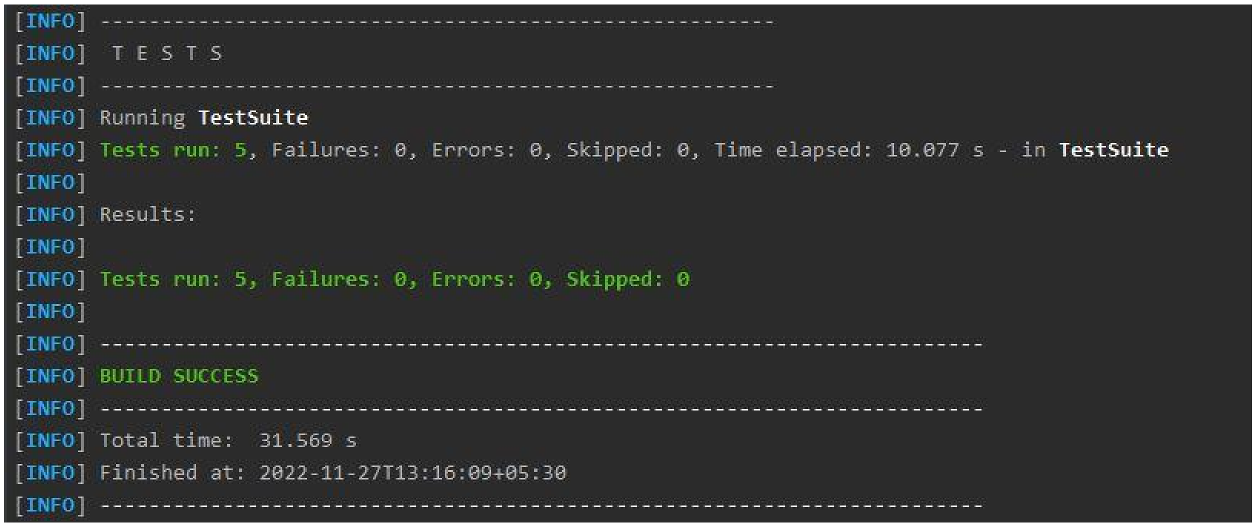
Test Paths	Test Requirements that are toured by test paths directly
[1,2,4]	[1,2,4]
[1,2,3,5,6,9,12,13]	[1,2,3,5,6,9,12,13]
[1,2,3,5,7,9,10,13]	[1,2,3,5,7,9,10,13]
[1,2,3,5,6,9,11,13]	[1,2,3,5,6,9,11,13]
[1,2,3,5,6,9,10,13]	[1,2,3,5,6,9,10,13]

[1,2,3,5,7,9,11,13]	[1,2,3,5,7,9,11,13]
[1,2,3,5,8,9,11,13]	[1,2,3,5,8,9,11,13]
[1,2,3,5,8,9,12,13]	[1,2,3,5,8,9,12,13]
[1,2,3,5,8,9,10,13]	[1,2,3,5,8,9,10,13]
[1,2,3,5,7,9,12,13]	[1,2,3,5,7,9,12,13]

6. All of Test Case Modules



```
package com.example.calculatortesting;

import org.junit.Test;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

import java.util.HashMap;
import java.util.Map;

import static org.junit.Assert.assertEquals;

public class ScientificCalculatorTest {
    ScientificCalculator calculator = new ScientificCalculator();

    @Test
    public void areaCalculatorTruePositive(){...}

    @Test
    public void mathCalculatorTruePositive(){...}

    @Test
    public void quadraticCalculatorTruePositive(){...}

    @Test
    public void timeConverterTruePositive(){...}

    @Test
    public void temperatureConverterTruePositive(){...}
}
```



```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running TestSuite
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 10.077 s - in TestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  31.569 s
[INFO] Finished at: 2022-11-27T13:16:09+05:30
[INFO] ------------------------------------------------------------------------
```

7. Team members Contribution

Sani Ranpariya : Implemented the source code on which testing is to be performed, Design Test cases for the functions in code based on CFG.

Vishwajeet Deulkar : Design Test cases for the functions in code based on CFG, Creating Control flow graphs for the functions.