International Institute of Information Technology, Bangalore. 26/C, Electronic City, Hosur Road, Bengaluru – 560 100

Project Report: **CFG based Testing using Junit and jacoco.**

Team Member 1: Amitesh Anand MT2019117
Team Member 2: Sujindar Selvaraj MT2019143
M.Tech CSE 2019-21
2020-CS 731-Software Testing

INTRODUCTION

The control flow graph is widely used in statistical analysis and in programs where it is essential to visualize and conceptualize the flow of execution in the program. Right after its development by F.E.Allen, it has been in extensive use for understanding and testing purposes. The basic characters can be seen in the following diagram.

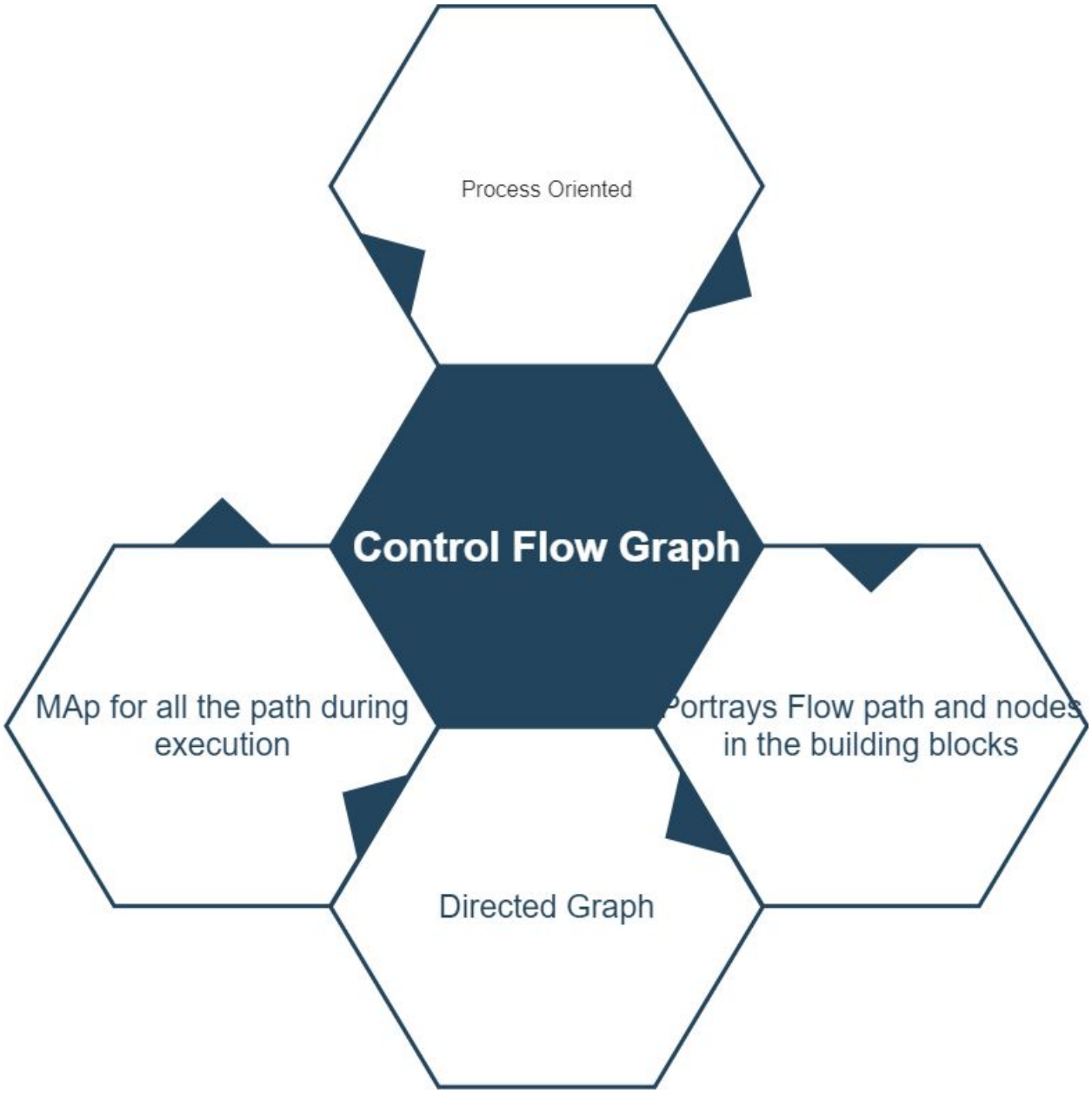

Diagram 1:Characteristics of Control Flow Graph

Any Control Flow Graph consists mainly of two dedicated blocks Entry Block and Exit Block.

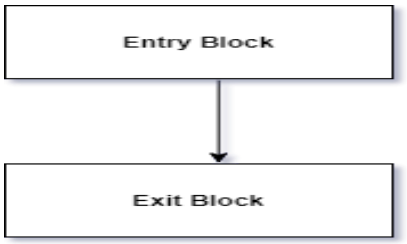

Diagram 2- CFG Entry and Exit Blocks

Besides this Control flow graph consists of the start node, end node, and flows between the nodes. In this project, the control flow graph has been generated before writing the test cases to understand the flow in the programs.

Technological Stack

- Source Code Description : A game like app for multiple utility functions in programming.
- Language of Programming : JAVA
- Editor with Dependencies : Eclipse Neon 2020-09,JDK-1.8
- Build Automation : Maven

To convert any existing java project to maven project follow these steps:

1. Right Click on the Project.
2. Go to Configure
3. Click on Convert to Maven Project

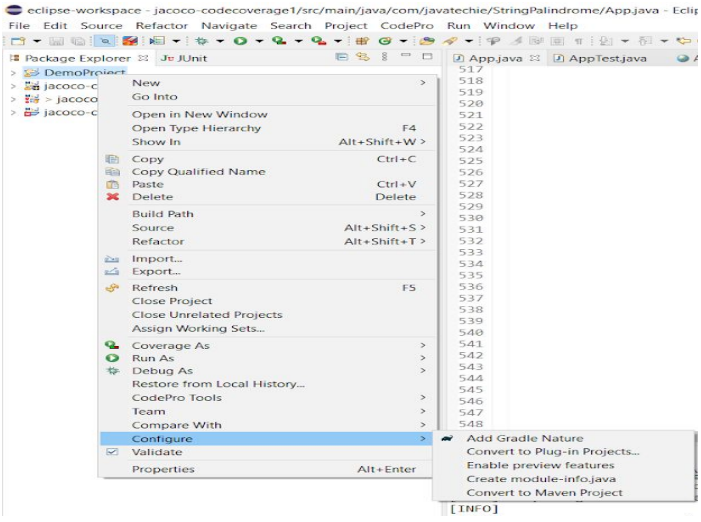

Diagram 3 : Convert into Maven Project

After Converting Pom.xml file will be generated which contains all the project information and configuration details/dependencies maven will use to build the project.It contains all the defaults for this project.

- Testing and Framework : Unit Testing framework Junit.

  Junit is integrated to the project through writing dependency in pom.xml file.

```
<dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
            <scope>test</scope>
        </dependency>
</dependencies>
```

- Code Coverage and Format : Jacoco library exported to HTML,XML,CSV format.

  Jacoco is also integrated through writing dependency in pom.xml file just like Junit. It is a maven plug-in which is executing maven tasks prepare-agent and prepare-package before execution and report generation after the unit test.

```
<build>
        <plugins>
            <plugin>
                <groupId>org.jacoco</groupId>
                <artifactId>jacoco-maven-plugin</artifactId>
                <version>0.8.3</version>
                <executions>
                    <execution>
                        <id>prepare-agent</id>
                        <goals>
                            <goal>prepare-agent</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>report</id>
                        <phase>prepare-package</phase>
                        <goals>
                            <goal>report</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>post-unit-test</id>
                        <phase>test</phase>
                        <goals>
                            <goal>report</goal>
                        </goals>
```

```
                                                    <configuration>
                                                        <!-- Sets the path to the file which
contains the execution data. -->

                                                        <dataFile>target/jacoco.exec</dataFile>
                                                        <!-- Sets the output directory for the
code coverage report. -->

<outputDirectory>target/my-reports</outputDirectory>
                                                    </configuration>
                                                </execution>
                                            </executions>
                                            <configuration>
                                                <systemPropertyVariables>

<jacoco-agent.destfile>target/jacoco.exec</jacoco-agent.destfile>
                                                </systemPropertyVariables>
                                            </configuration>
                                        </plugin>
                                    </plugins>
                                </build>
```

- CFG Generator and format : Eclipse CFG Generator exported to png,jpeg.

  It is a CFG generator tool which is integrated through Eclipse.Follow these steps:
  1. Go to **Help**
  2. Go to **install new software**
  3. In the **Work with Text** Field give "http://eclipsefcg.sourceforge.net/installation.html" and press Enter, then follow wizard instructions to install and click Finish.

- Null test cases automation : Codepro

  It is a tool to analyze code dependencies,audit the complete code and generate Null test cases for the class.It is installed just like Eclipse CFG Generator.Just in Work with Text Field give "https://dl.google.com/eclipse/inst/codepro/latest/3.7" and follow the wizard.

Control Flow Graph

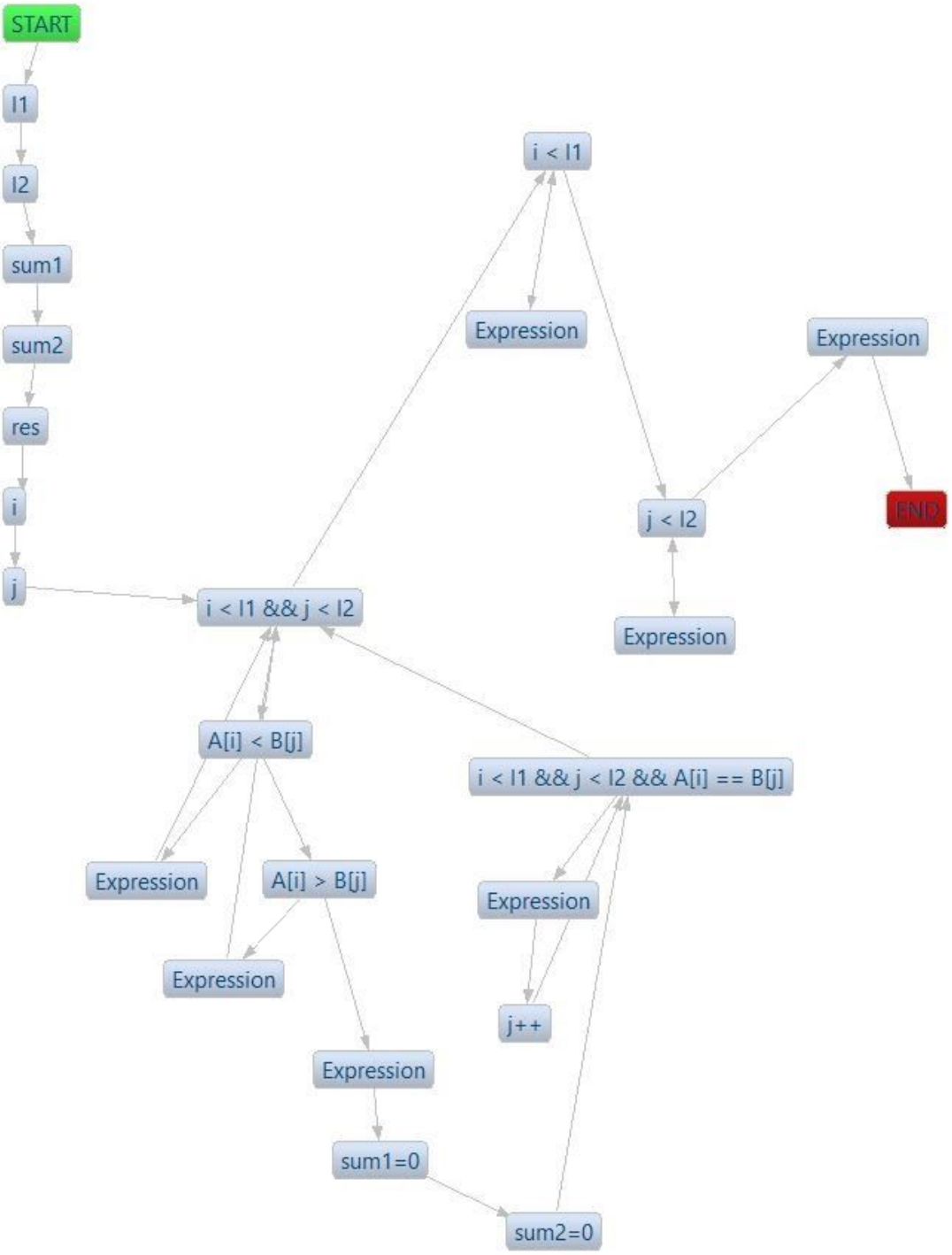

Diagram 4: Control flow graph generated from Eclipse CFG for Function1(maxpath_sum)

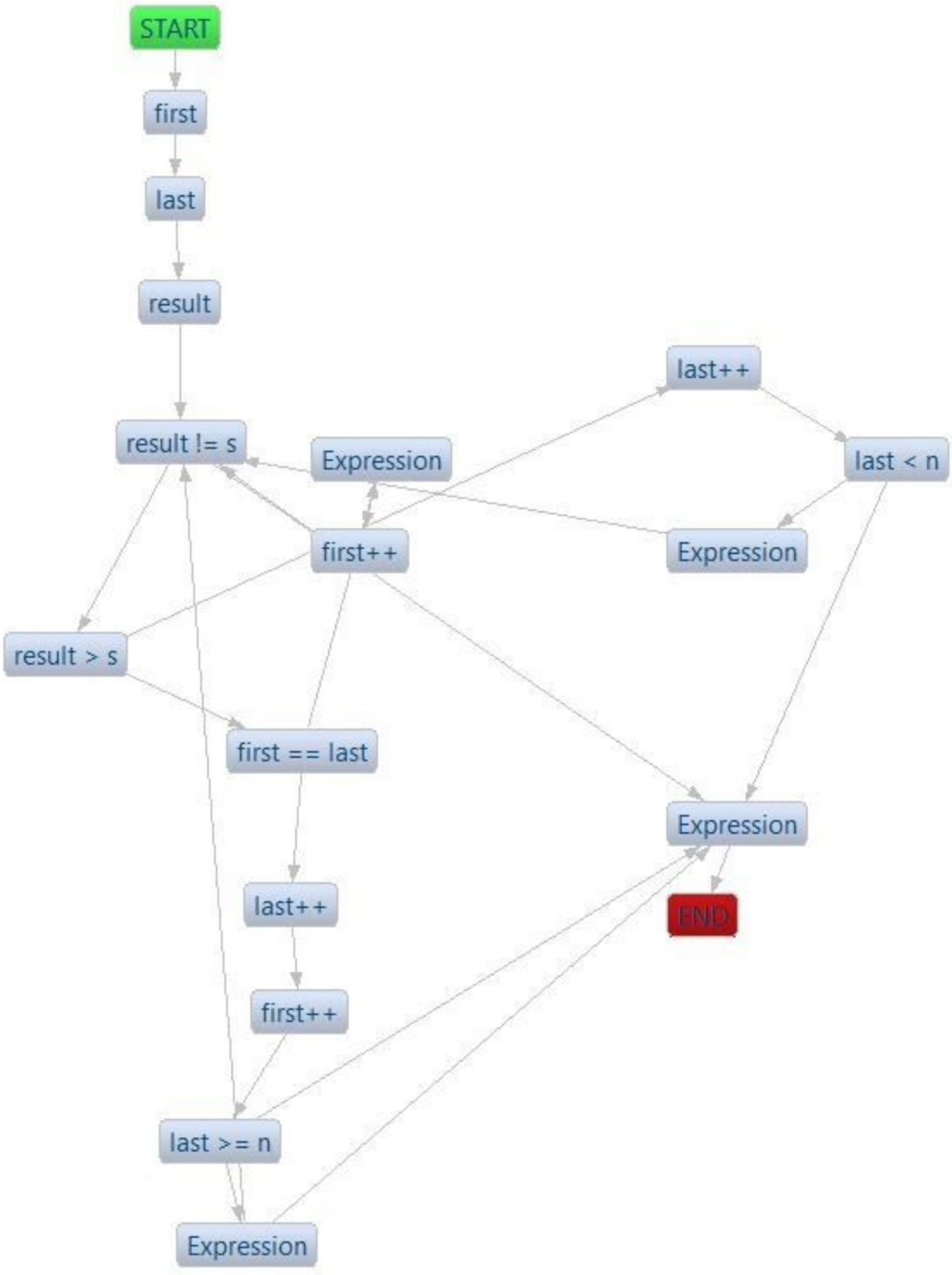

Diagram 5: Control flow graph generated from Eclipse CFG for Function2(subarray_sum)

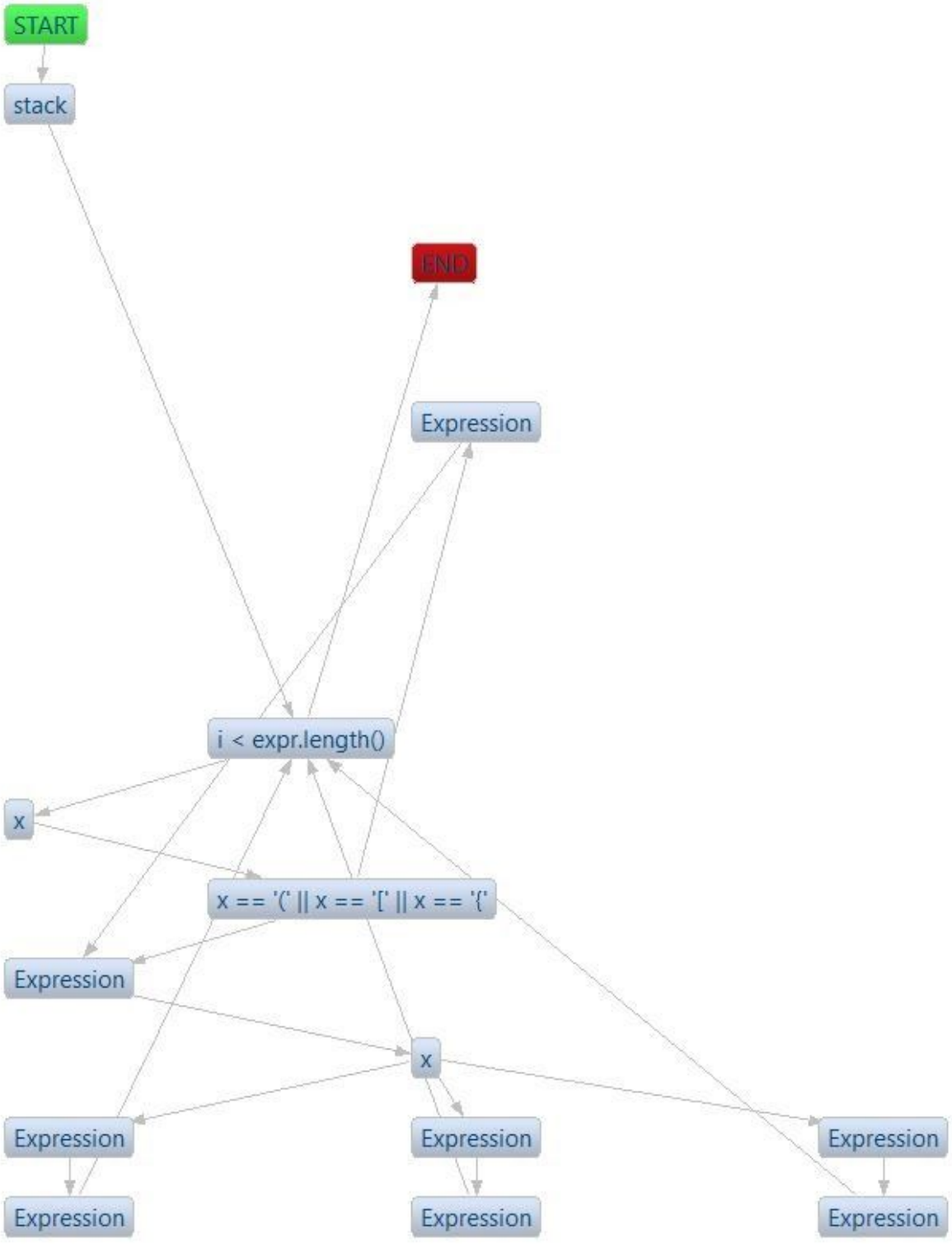

Diagram 6: Control flow graph generated from Eclipse CFG for Function3(isBalParen)

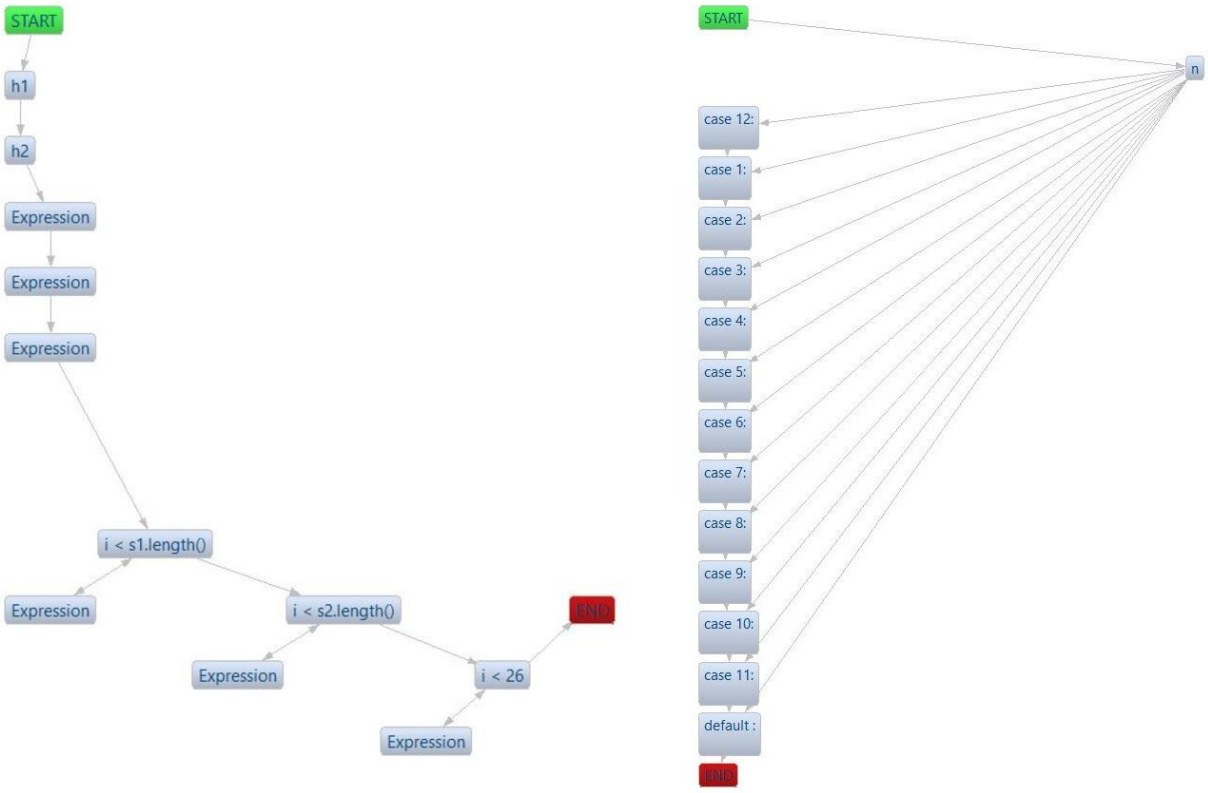

Diagram 7: Control flow graph generated from Eclipse CFG for Function4&5(anagram&isSeason)

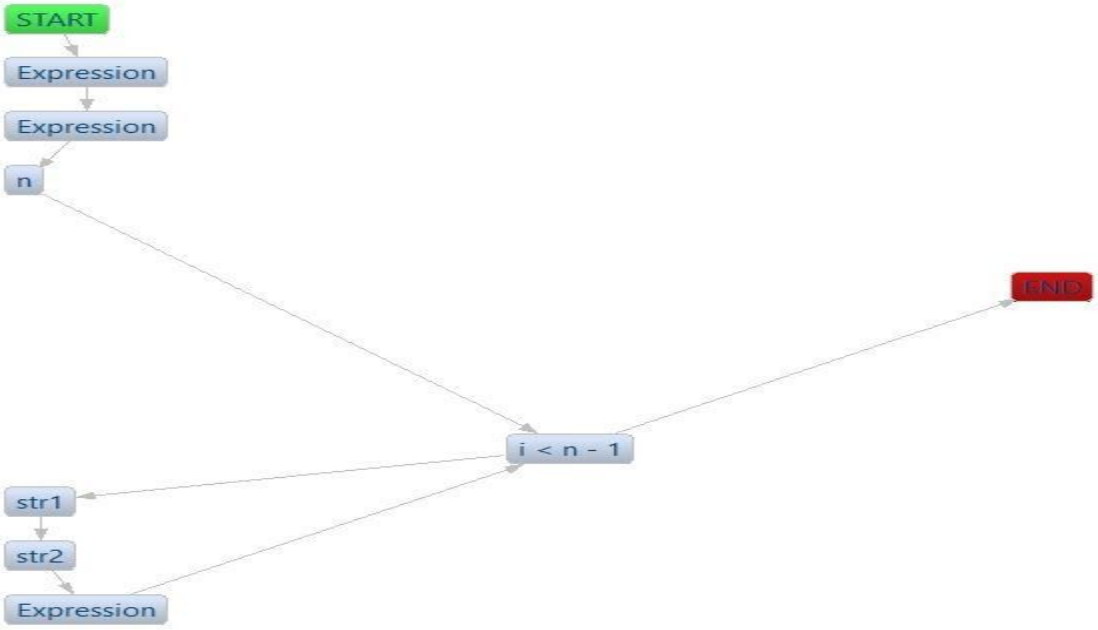

Diagram 8 : Control flow graph generated from Eclipse CFG for Function6(isrotPalind)

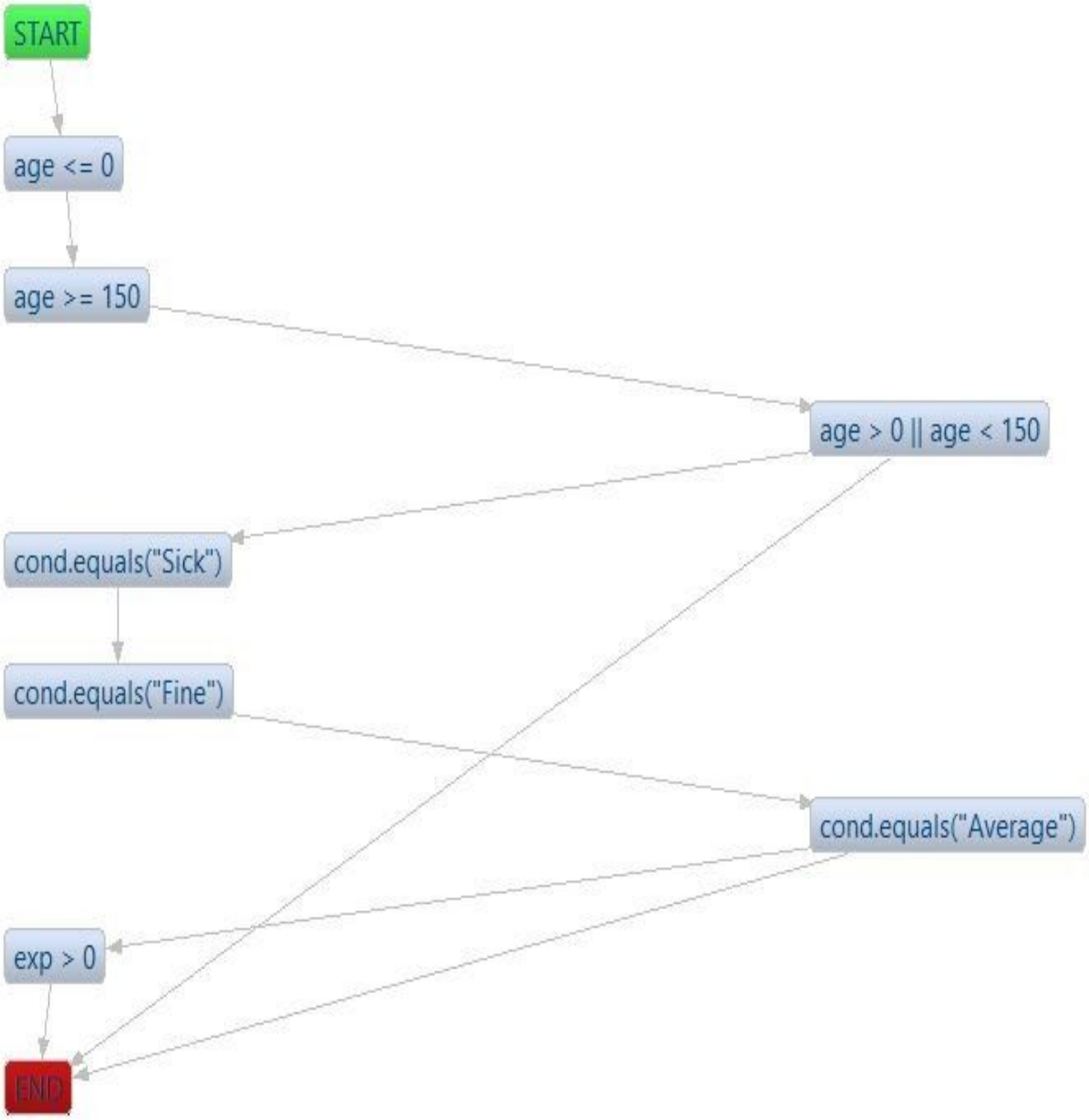

Diagram 9 : Control Flow Graph generated from EclipseCFG for Function7(isFit)

Test Case Generation

The test cases for each function is generated after going through these Control Flow Graphs and making sure to maintain high percentage for node coverage and branch coverage.
The test source code was written under same phase with main source code with phase tag under same package name.The Test cases are written with tags and assert statements of Junit. Few Examples are mentioned below.

```
@Test
        public void isSeason(){
           System.out.println("test case isRotated");
           assertEquals("Winter",App.isSeason(1));
           assertEquals("Winter",App.isSeason(2));
           assertEquals("Winter",App.isSeason(12));
           assertEquals("Spring",App.isSeason(3));
           assertEquals("Spring",App.isSeason(4));
           assertEquals("Spring",App.isSeason(5));
           assertEquals("Spring",App.isSeason(3));
           assertEquals("Summer",App.isSeason(6));
           assertEquals("Summer",App.isSeason(7));
           assertEquals("Summer",App.isSeason(8));
           assertEquals("Autumn",App.isSeason(9));
           assertEquals("Autumn",App.isSeason(10));
           assertEquals("Autumn",App.isSeason(11));
           assertEquals("Invalid",App.isSeason(13));
           assertEquals("Invalid",App.isSeason(0));

         }
//below 4 tests are for same function
@Test
        public void LCS1()
        {
                String str1 = "abaaa",str2="baabaca";
                List<String> ans = Arrays.asList("aaaa","abaa","baaa");
                assertEquals(ans, app.LCS(str1,str2));
        }
        @Test
        public void LCS2()
        {
                String str1 = "abcbdab",str2="bdcaba";
                List<String> ans = Arrays.asList("bcab","bcba","bdab");
                assertEquals(ans, app.LCS(str1,str2));
        }
        @Test
        public void LCS3()
        {
                String str1 = "abcbdab",str2="pqxprst";
                List<String> ans = Arrays.asList();
```

```
        assertEquals(ans, app.LCS(str1,str2));
    }
    @Test
    public void LCS4()
    {
        String str1 = "AGTGATG",str2="GTTAG";
        List<String> ans = Arrays.asList("GTAG","GTTG");
        assertEquals(ans, app.LCS(str1,str2));
    }
```

The Null test cases are generated by CodePro tools by **right clicking on project then go to codepro tools->Generate Test Cases.**

The linkage for the project can be found by CodePro tools by **right clicking on project then go to codepro tools->Analyze dependencies.**

The source code is completely audited for correctness of the project by CodePro tools by **right clicking on project then go to codepro tools->Audit Code.**

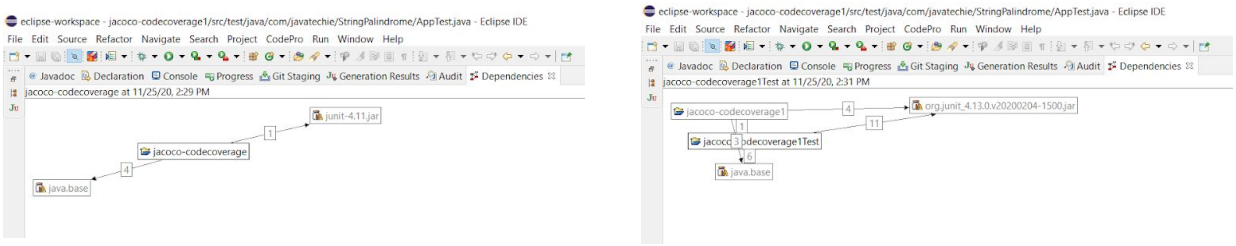

Diagram 10 : Dependency graph for Java project

Code Coverage

Jacoco generates code coverage report in three formats HTML,CSV and XML.It can be generated by **right clicking on project -> Run as -> Maven Test.**It can be viewed in any browser even in eclipse built in.Code coverage report reflect instruction coverage,branch coverage,complexity,etc.

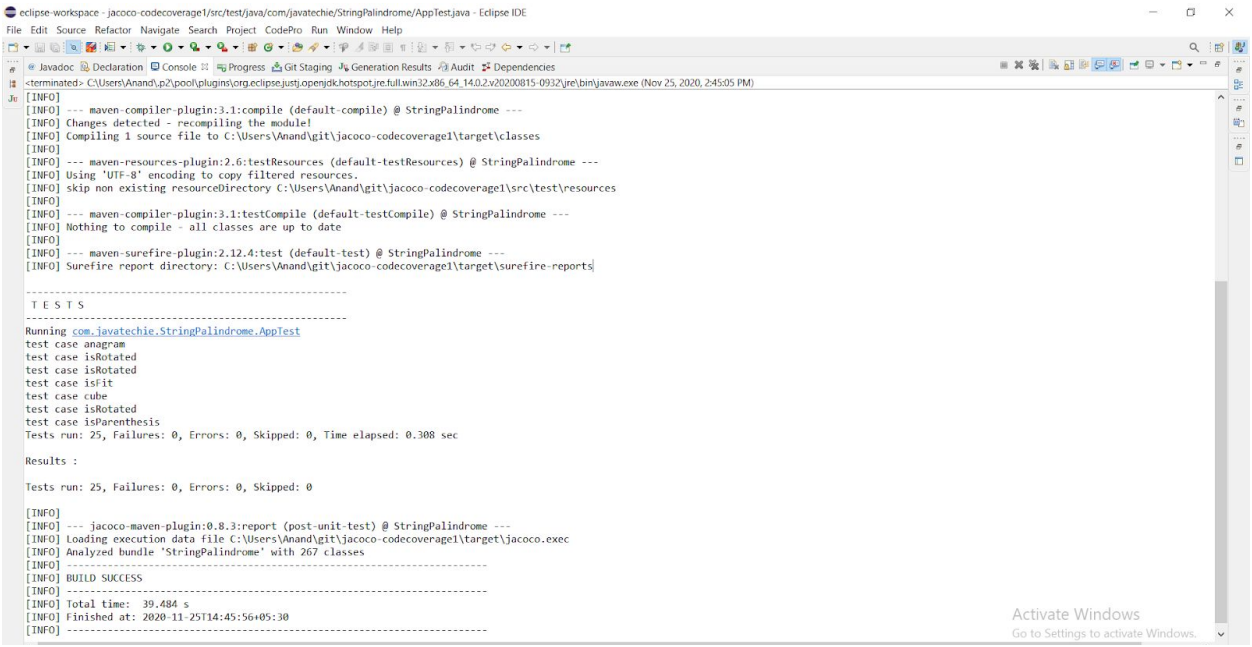

Diagram 11 : Latest output of Maven Test

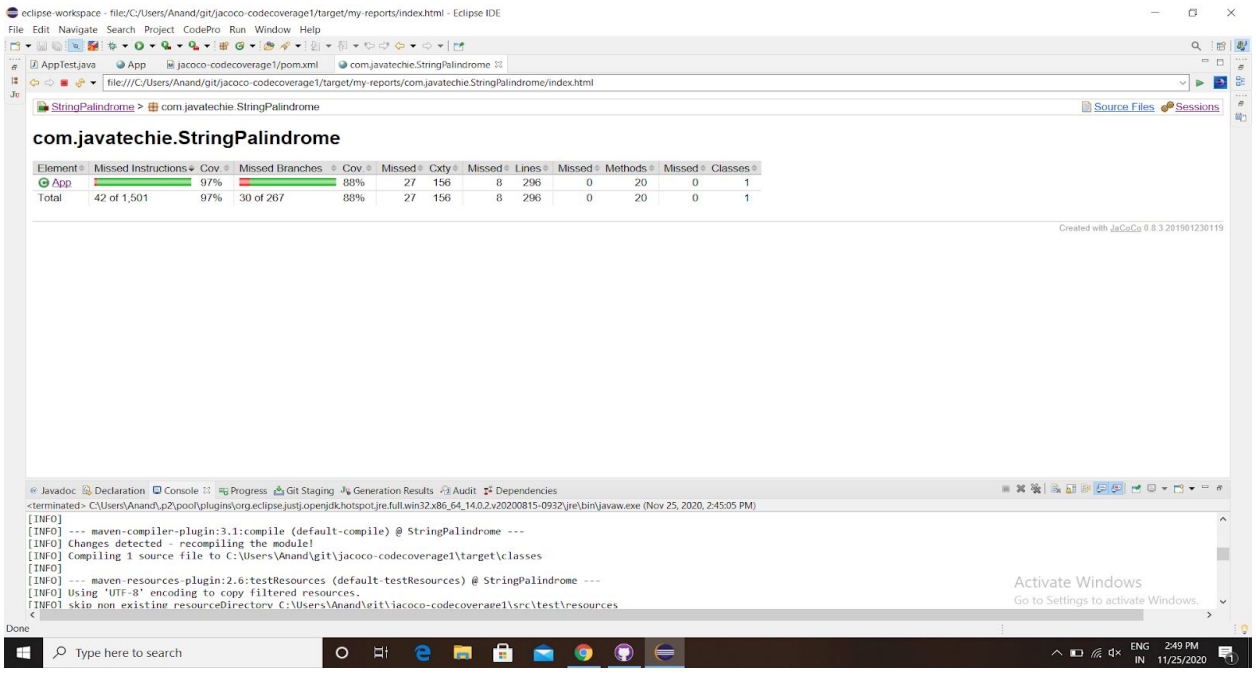

Diagram 12: Overall Code Coverage Report

The overall instruction coverage is 97% and overall branch coverage is 88%

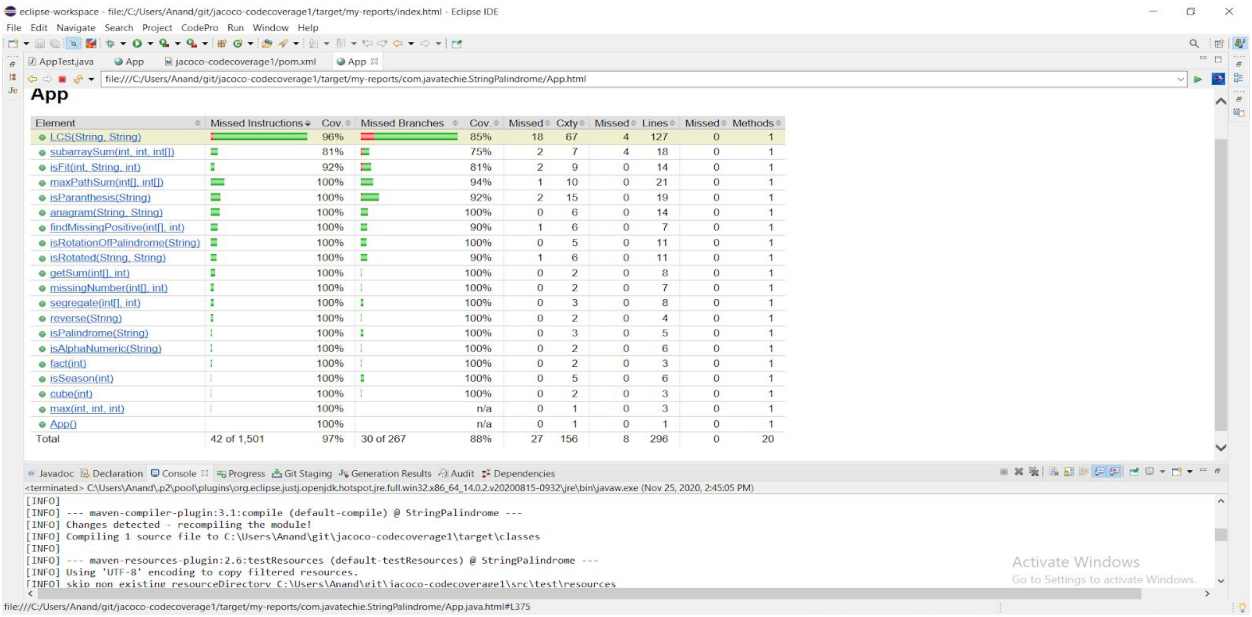

Diagram 13 : Detailed code coverage

The highest and lowest instruction coverage are 100% and 81% respectively.
The highest and lowest branch coverage are 100% and 75% respectively.

References

- https://junit.org/junit5/
- https://www.jacoco.org/jacoco/trunk/doc/
- http://eclipsefcg.sourceforge.net/
- https://github.com/Java-Techie-jt

Acknowledgements

We would like to express our deepest gratitude to Associate Professor **Meenakshi D' Souza IIIT Bangalore**,for providing us with the opportunity to work on the project for Control Flow Graph based testing and providing ample knowledge in the subject matter.

After taking this course and project, our knowledge in the field of system testing is deeply enhanced and we have gained so many theoretical and experimental insights.

Thanking you,
Amitesh Anand MT2019117
Sujindar Selvaraj MT2019143

List of Figures and link for the project

Link of the project :"https://github.com/Amitesh591/jacoco-codecoverage"

Work Distribution

We have worked through version control system Git and on Github repository.
We have both have developed the source code and added functionalities and pushed the code to merge on Github repository mentioned in the link above.For the final report generation we made google doc and written together.

Amitesh Anand(MT2019117): created template ,added functionalities,integrated various tools,generated CFG,code coverage report and drafted final report.

Sujindar Selvaraj(MT2019143): created template ,added functionalities,integrated various tools,generated CFG,code coverage report and drafted final report.