IT-314: Software Engineering

Lab-09: Mutation Testing

Hitanshu variya – 202201510

BTech, ICT + CS

Section - B

Question 01) The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise, you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size() is the size of the vector p, (p.get(i)).x is the x component of the ith point appearing in p, similarly for (p.get(i)).y. This exercise is concerned with structural testing of code, so the focus is on creating test sets that satisfy some particular coverage criteria.

```
Vector doGraham (Vector p) {
    int i, j,min,M;
    Point t;
    min = 0;

    // search for minimum:
    for(i=1; i < p.size(); ++i) {
          if(((Point) p.get(i)).y < ((Point) p.get(min)).y) {
           min = i;
             }
       }

    // continue along the values with same y component
    for(i=0; i<p.size(); ++i) {
             if(( ((Point) p.get(i)).y == ((Point) p.get(min)) .y) &&
                  (((Point) p.get(i)).x >((Point) p.get(min)).x )) {
           min = i;
        }
    }
}
```

Solution:

1. Building out an executable Python code:
Solution:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def doGraham(points):
    min_index = 0

    for i in range(1, len(points)):
        if(points[i].y < points[min_index].y):
            min_index = i

    for i in range(0, len(points)):
        if(points[i].y == points[min_index].y and points[i].x > points[min_index].x):
            min_index = i

    return points[min_index]
```

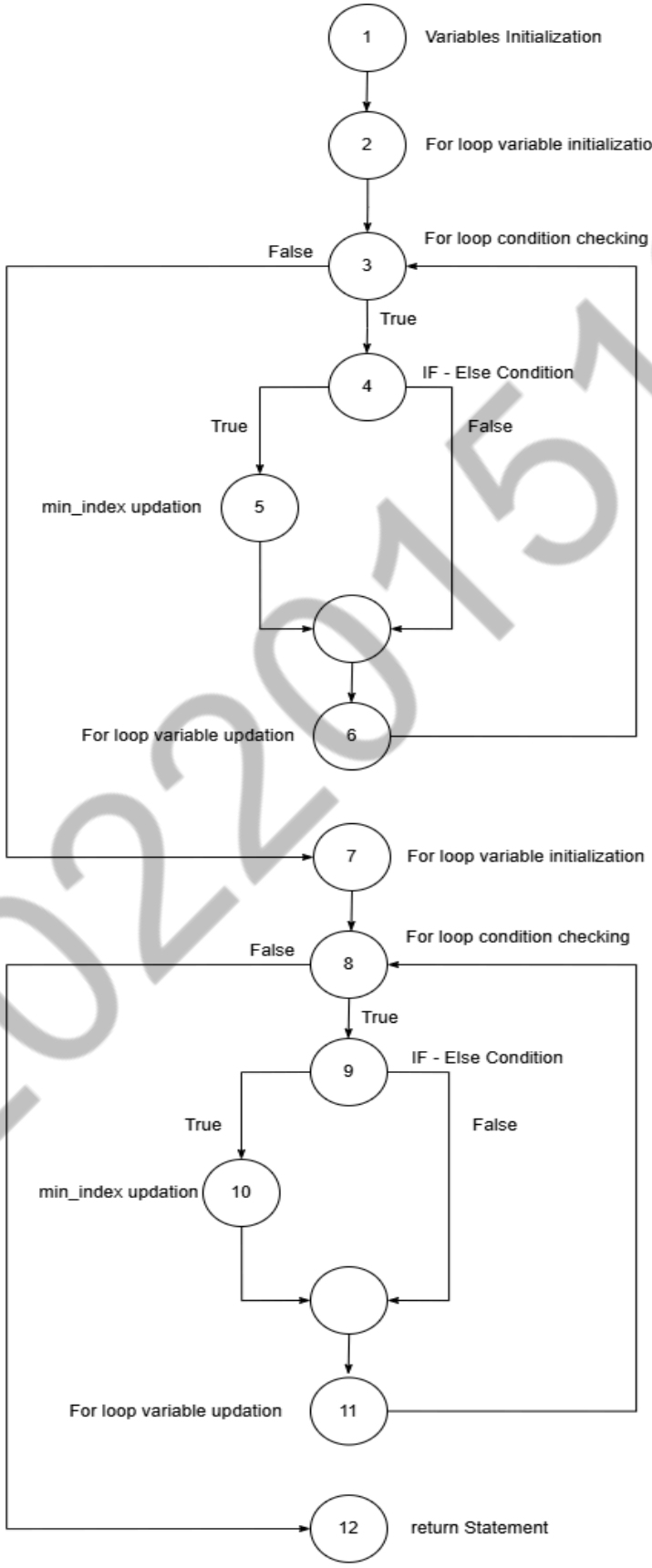
Note: Here, in the original code snippet, there is no return statement for the type Vector, hence, I am considering (for simplicity) that the points at the min_index will be returned instead of the points vector, as there is no updation in the vector points or creation of a new vector.

➔ Also, The class Point was introduced for the sake of making the code executable. This class holds the ordered point pairs (x_coordinate, y_coordinate).

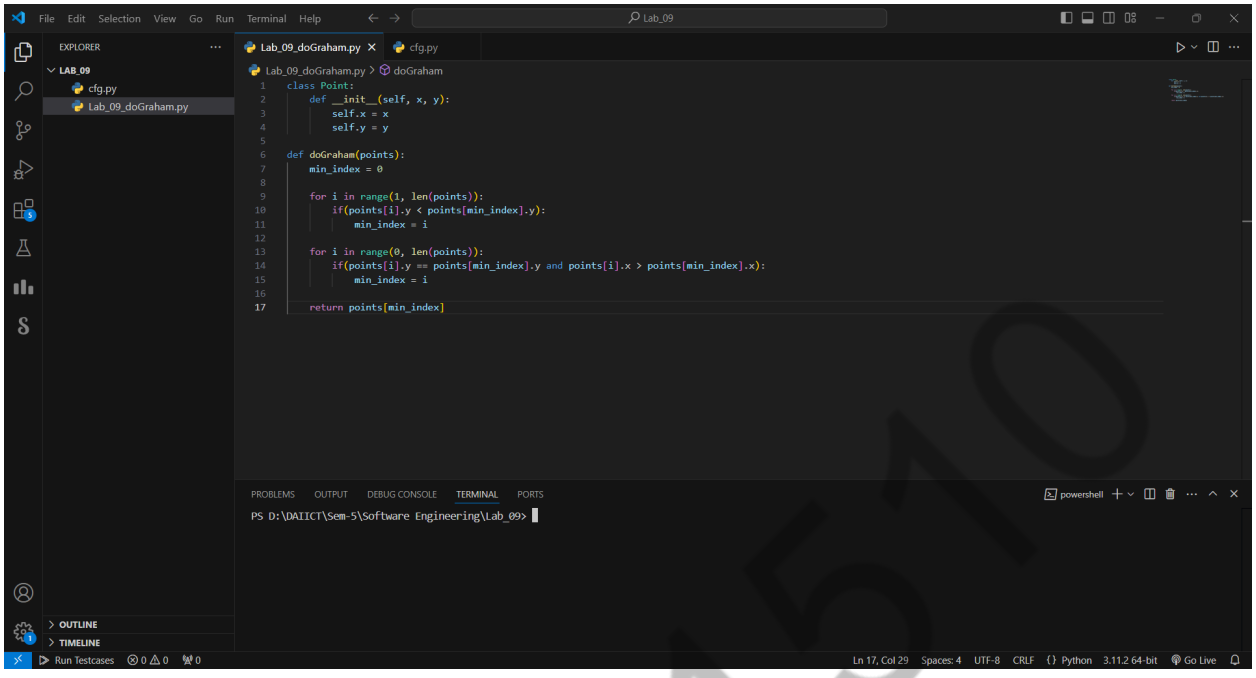
❖ Handling the Flow points / nodes in the code snippet:

```
        def doGraham(points):
1|          min_index = 0
2|          for i in range(1, len(points)):
4|              if(points[i].y < points[min_index].y):
5|                  min_index = i
7|          for i in range(0, len(points)):
9|              if(points[i].y == points[min_index].y and points[i].x > points[min_index].x):
10|                 min_index = i
12|         return points[min_index]
```

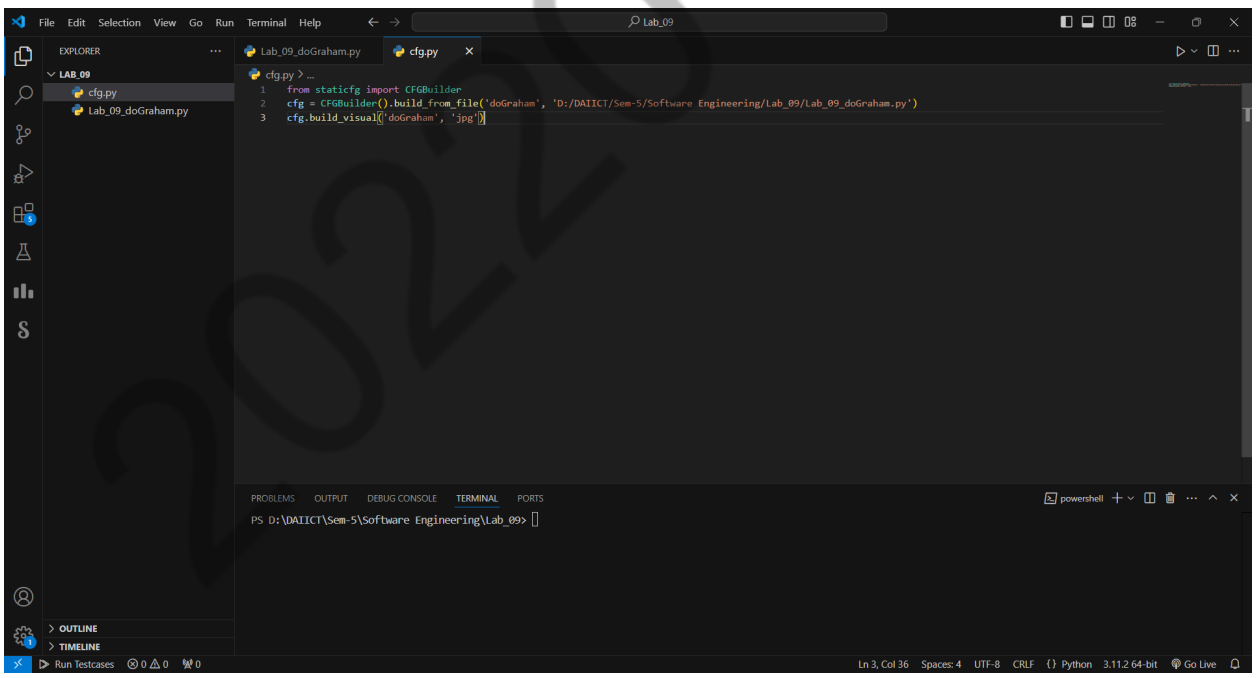
❖ Converting the python code into the Control Flow graph (CFG):



➔ This is the Python code script in VS Code.

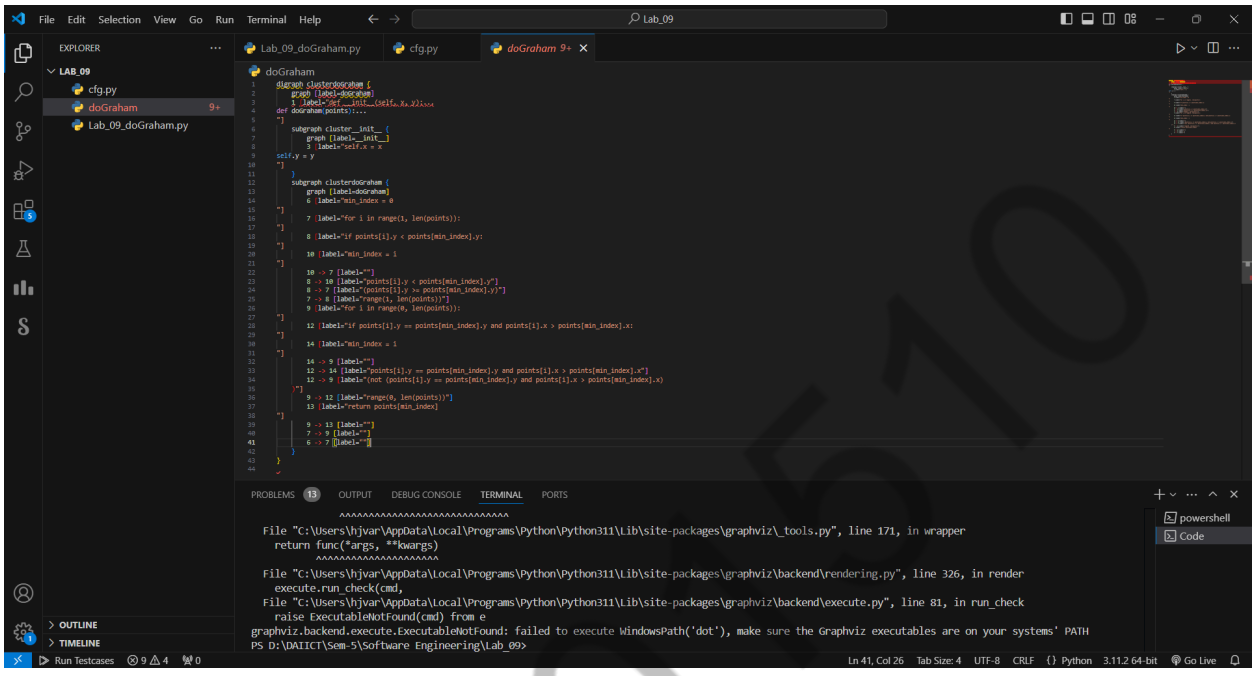


➔ This is the CFG script for converting the python code into the Control flow graph using staticfg (CFGBuilder) library in python.

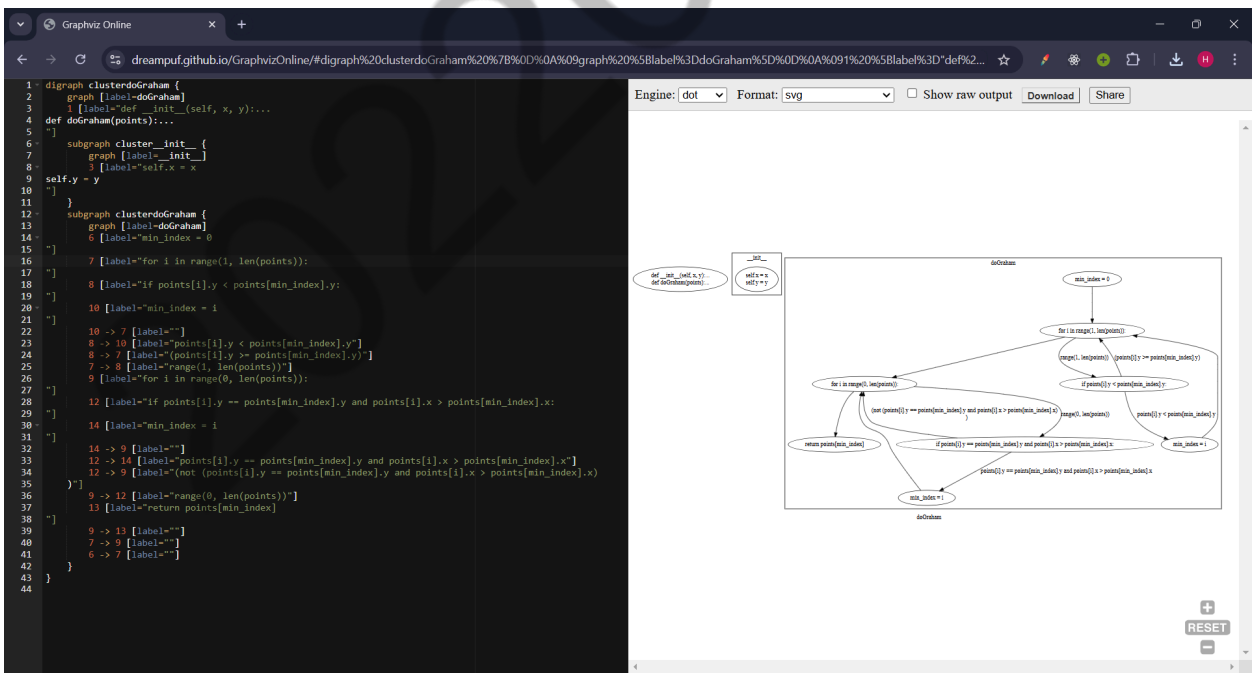


➔ Upon running the code, this will generate a doGraham.py file written in “DOT Language” used by Graphviz tool to plot the control flow graph.

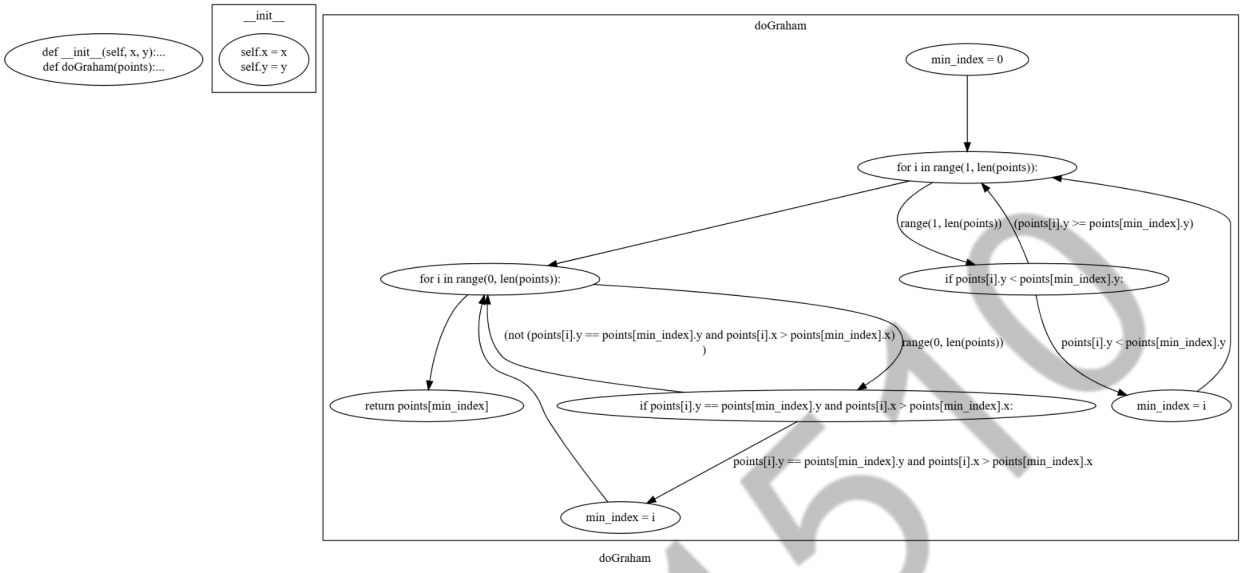
➔ But, since the graphviz is not installed in the PC directory, I am using the online graphviz compiler to compile the DOT Code.



➔ Upon compiling the code, we will get the corresponding python code control flow Graph.



➔ This is the Control Flow graph for the doGraham Class provided in the code snippet using the Control Flow Graph Factory Tool (Graphviz).



2. After generating the control flow graph, check whether your CFG match with the CFG generated by Control Flow Graph Factory Tool.

Solution:

➔ Yes, my CFG and the CFG generated by the Control Flow Graph Factory Tool (Graphviz) are the same and matching.

Notes:

➔ Here, the number of nodes are different in both the CFG generated by me and the one generated by the tool, but the logic of the CFG is the same.

➔ Here, I have used more node to represent the for loop states (Initialization, Condition, Updation) and also, there I have used the auxiliary nodes at the end of If-else conditions to merge the flow.

➔ Hence, due to these factors, my number of nodes are high compared to the CFG generated by the tool.

3. **Devise minimum number of test cases required to cover the code using the aforementioned Criteria.**

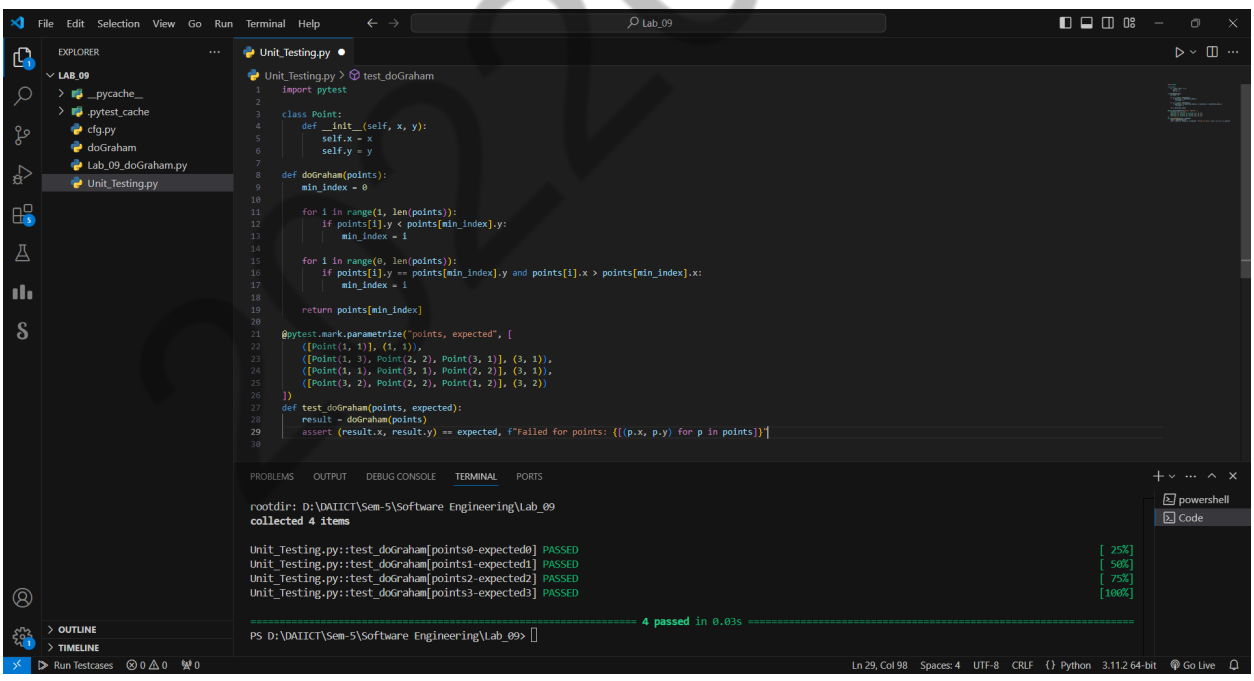
Solution:

Test Case	Input Points	Expected Output	Reason
1	[(1, 1)]	(1, 1)	Tests the base case with only one point.
2	[(1, 3), (2, 2), (3, 1)]	(3, 1)	Verifies selection of the point with the smallest y value.
3	[(1, 1), (3, 1), (2, 2)]	(3, 1)	Covers the condition where multiple points have the same y, selecting the one with largest x.
4	[(3, 2), (2, 2), (1, 2)]	(3, 2)	Ensures correct selection among points with the same y by choosing the one with the largest x.

➔ Based on the above Test cases, I have performed the unit - testing in order to find any bugs in the code.

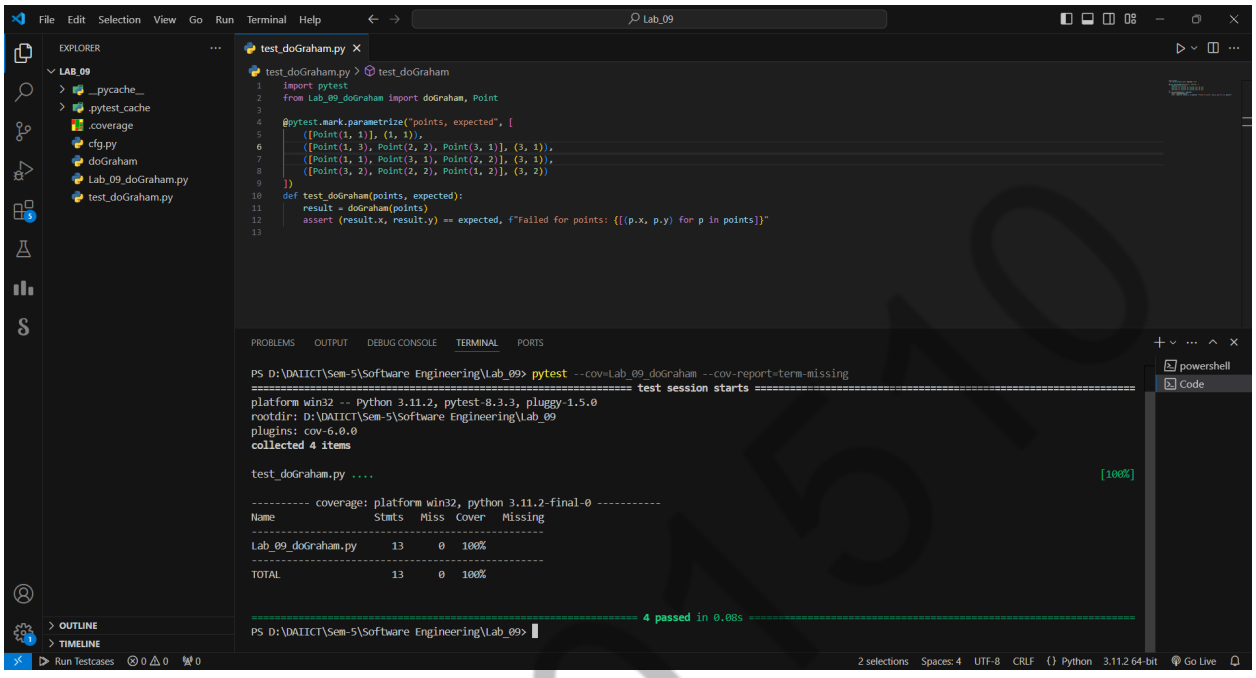
➔ Unit Testing is executed using the “pytest” library in Python.

➔ Here, all the test cases were passed and the final status of the code is “OK”.



➔ To look for the Code Coverage using these sets of Test cases, I am using the “pytest-cov” library.

➔ Here, the library analyzed the test cases and responded with **100% code coverage**.



4. Construct test sets for your flow graph that are adequate for the following criteria:

a. Statement Coverage.

b. Branch Coverage.

c. Basic Condition Coverage.

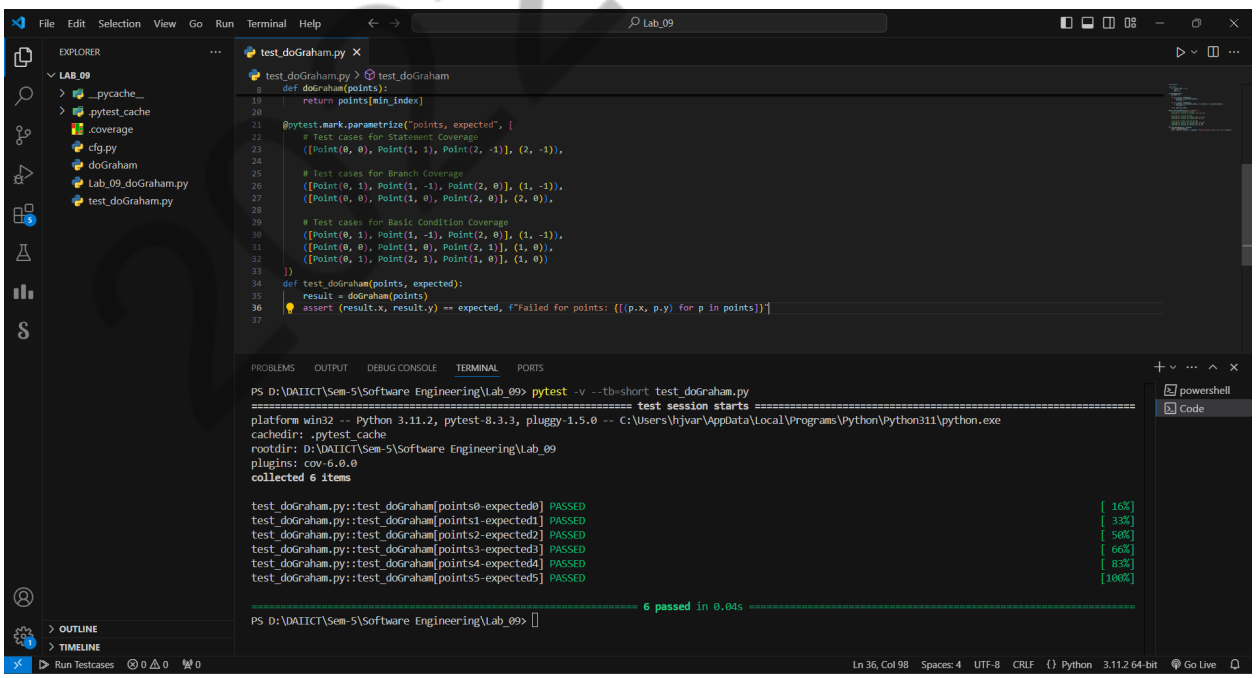
Solution:

Coverage Criterion	Test Case	Points	Expected Output
Statement Coverage	TC1	[(0, 0), (1, 1), (2, -1)]	(2, -1)
Branch Coverage	TC1	[(0, 1), (1, -1), (2, 0)]	(1, -1)
	TC2	[(0, 0), (1, 0), (2, 0)]	(2, 0)
Basic Condition Coverage	TC1	[(0, 1), (1, -1), (2, 0)]	(1, -1)
	TC2	[(0, 0), (1, 0), (2, 1)]	(1, 0)
	TC3	[(0, 1), (2, 1), (1, 0)]	(1, 0)

➔ Based on the above Test cases, I have performed the unit - testing in order to find any bugs in the code.

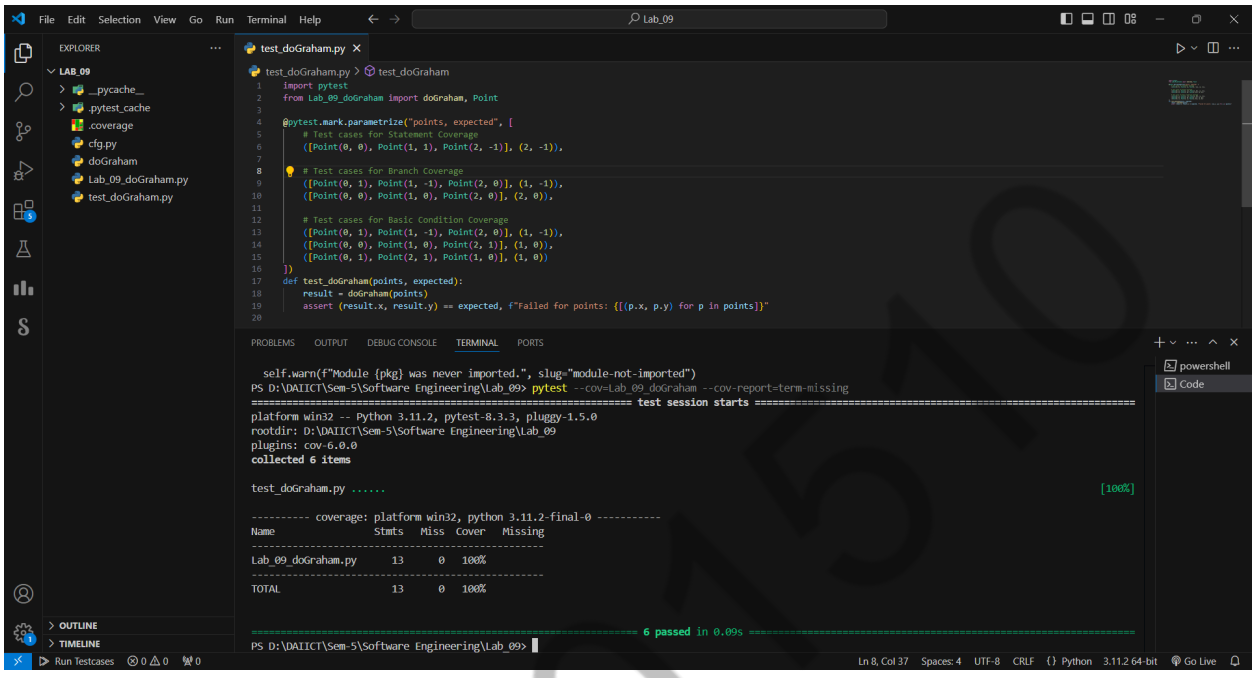
➔ Unit Testing is executed using the “pytest” library in Python.

➔ Here, all the test cases were passed and the final status of the code is “OK”.



➔ To look for the Code Coverage using these sets of Test cases, I am using the “pytest-cov” library.

➔ Here, the library analyzed the test cases and responded with **100% code coverage**.

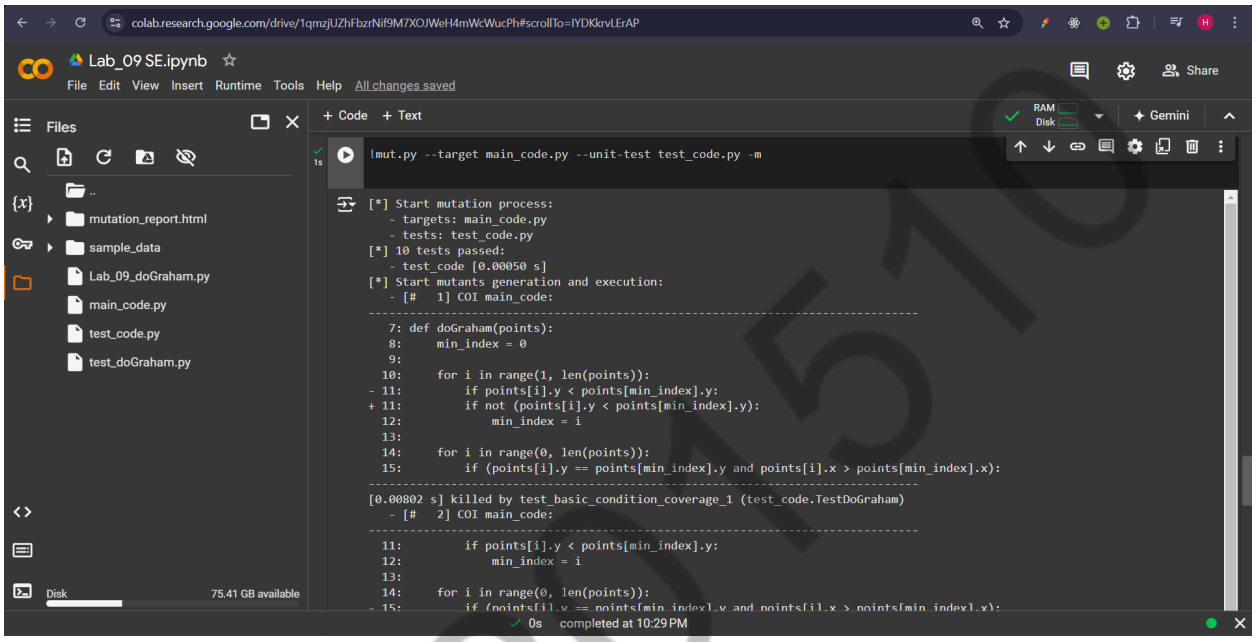


5. **For the test set you have just checked can you find a mutation of the code (i.e. the deletion, Change or insertion of some code) that will result in failure but is not detected by your test set.**

Solution:

➔ Here, I am using the “mutpy” library for mutation testing.

➔ Following is the result for the mutation testing,



➔ This is the result obtained using the html report generated by mutpy,

MutPy mutation report

04.11.2024 16:55

Target

- main_code.py

Tests [6]

- test_code [0.0 s]

Result summary

- Score - 75.0%
- Time - 0.2 s

Mutants [8]

- killed - 6
- survived - 2
- incompetent - 0
- timeout - 0

75.0% | 25.0%

#	Module	Operator	Tests run	Duration	Result
1		COI [11]	1	0.008 s	killed
2		COI [15]	1	0.008 s	killed
3		LCR [15]	1	0.008 s	killed
4		ROR [11]	1	0.008 s	killed
5		ROR [11]	6	0.006 s	survived
6		ROR [15]	1	0.008 s	killed
7		ROR [15]	2	0.009 s	killed
8		ROR [15]	6	0.007 s	survived

➔ Here, there are two mutants which survived,

Mutation #5

Details

- module -
- survived
- duration - 0.006 s
- tests run - 6

Mutations

- ROR - line 11

Mutant

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def doGraham(points):
    min_index = 0

    for i in range(1, len(points)):
        if points[i].y <= points[min_index].y:
            min_index = i

    for i in range(0, len(points)):
        if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
            min_index = i

    return points[min_index]
```

Mutation #8

Details

- module -
- survived
- duration - 0.007 s
- tests run - 6

Mutations

- ROR - line 15

Mutant

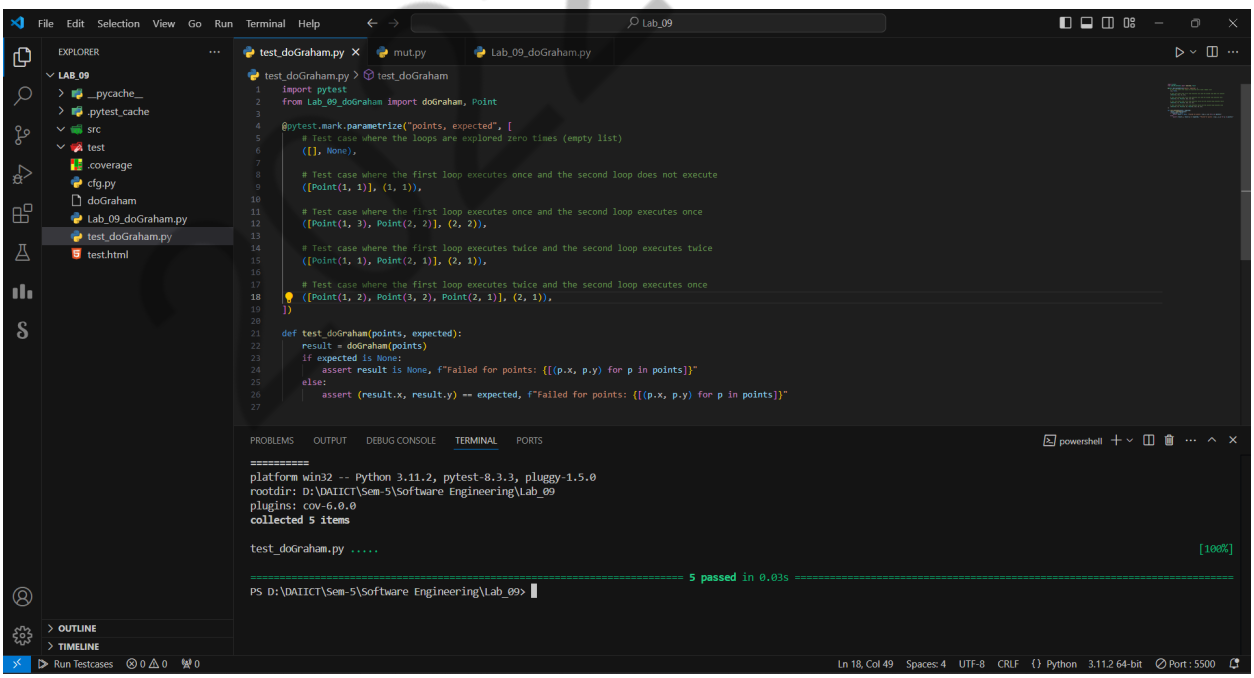
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def doGraham(points):
    min_index = 0

    for i in range(1, len(points)):
        if points[i].y < points[min_index].y:
            min_index = i

    for i in range(0, len(points)):
        if (points[i].y == points[min_index].y and points[i].x >= points[min_index].x):
            min_index = i

    return points[min_index]
```

6. **Create a test set that satisfies the path coverage criterion where every loop is explored at Least Zero, one or two times.**

Solution:

Test Case	Points	Expected Output	Reason
TC1	[ ]	None	Test case where the loops are explored zero times
TC2	[Point(1, 1)]	(1, 1)	Test case where the first loop executes once and the second loop does not execute
TC3	[Point(1, 3), Point(2, 2)]	(2, 2)	Test case where the first loop executes once and the second loop executes once
TC4	[Point(1, 1), Point(2, 1)]	(2, 1)	Test case where the first loop executes twice and the second loop executes twice
TC5	[Point(1, 2), Point(3, 2), Point(2, 1)]	(2, 1)	Test case where the first loop executes twice and the second loop executes once

➔ Here, all the test cases were passed and the final status of the code is “OK”.



➔ To look for the Code Coverage using these sets of Test cases, I am using the “pytest-cov” library.

➔ Here, the library analyzed the test cases and responded with **100% code coverage**.