IT – 314 | SE | LAB-9

(Harsh Gajjar – 202201140)

Mutation Testing

Q.1. 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;
            }
        }
```

For the given code fragment, you should carry out the following activities.

1. Convert the code comprising the beginning of the doGraham method into a control flow graph (CFG). You are free to write the code in any programming language.
2. Construct test sets for your flow graph that are adequate for the following criteria:
   a. Statement Coverage.
   b. Branch Coverage.
   c. Basic Condition Coverage.
3. 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. **You have to use the mutation testing tool.**
4. Create a test set that satisfies the path coverage criterion where every loop is explored at least zero, one or two times.

Lab Execution (how to perform the exercises): Use unit Testing framework, code coverage and mutation testing tools to perform the exercise.

1. After generating the control flow graph, check whether your CFG match with the CFG generated by ***Control Flow Graph Factory Tool*** and ***Eclipse flow graph generator***. (In your submission document, mention only "Yes" or "No" for each tool).

2. Devise minimum number of test cases required to cover the code using the aforementioned criteria.

3. This part of the exercise is very tricky and interesting. The test cases that you have derived in Step 2 are then used to identify the fault when you make some modifications in the code.

   Here, you need to insert/delete/modify a piece of code that will result in failure but it is not detected by your test set – derived in Step 2.

   Write/identify a mutation code for each of the three operation separately, i.e., by deleting the code, by inserting the code, by modifying the code.

4. Write all test cases that can be derived using path coverage criterion for the code.

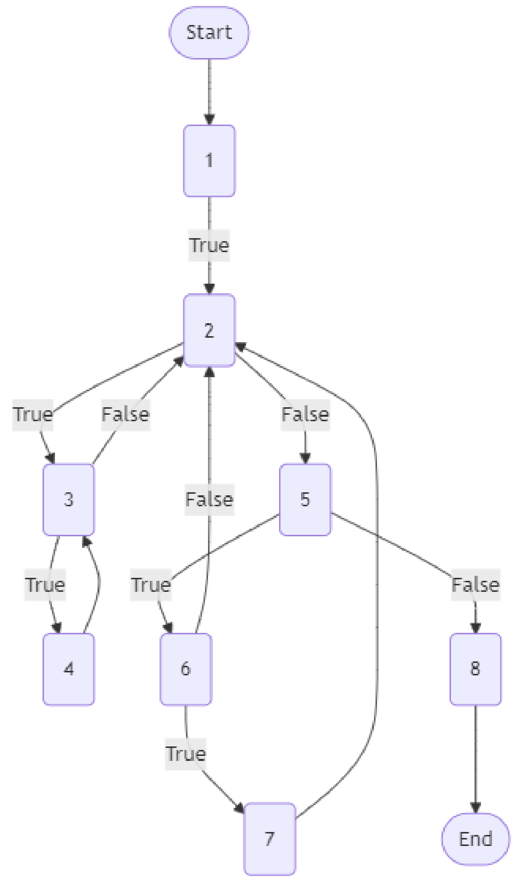
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class Vector:
    def __init__(self, points):
        self.points = points

    def size(self):
        return len(self.points)
    def get(self, i):
        return self.points[i]

 def doGraham(p):
    min_idx = 0
    for i in range(1, p.size()):
        if p.get(i).y < p.get(min_idx).y:
            min_idx = i
        elif p.get(i).y == p.get(min_idx).y and p.get(i).x >
p.get(min_idx).x:
            min_idx = i
    return min_idx
```

1. Control Flow Diagram



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

a. Statement Coverage.
b. Branch Coverage.
c. Basic Condition Coverage

There are two main parts to the doGraham function:

i. Initial Loop
   The point with the least y-value is found by iterating over the points in this loop. It retains the first point found if several points have the same y-value.
   - → Condition 1: i < p.size()– regulates the loop's end.
   - → Condition 2: ((Point) p.get(i)).y < ((Point) p.get(min)).y – if the current point's y-value is less than the existing minimum, y-updates min.

ii. The Second Loop
   In order to identify the point with the largest x-value but the same y-value as min, this loop iterates over the points once again.
   - → Condition 3: p.get(i) ((Point)).((Point) p.get(min) == y.y-checks to see if the current point and the minimum have the same value.
   - → Condition 4: ((Point) p.get(i)).x > ((Point) p.get(min)).x – if the current point's x is greater than the existing maximum x with the same y, x-updates min.

1. Coverage of Statements
   Every line of code needs to be run at least once in order to achieve statement coverage. Each component of the code can be exercised using a small number of test cases:
   - → Test Case 1: One Point
     - Input: [(0,0)]
     - Description:
       - This covers initialisation and both loops without making any changes; it initialises min=0 but won't update it because there is only one point.
     - Expected Output: 0
   - → Test Case 2: Two Points with Different Y-values
     - Input: [(0, 0), (1, 1)]
     - Description:
       - This covers the update to min in the first loop.

  - The second loop will execute without updating min as the y-values are different.
- Expected Output: 0

2. Branch Coverage

To achieve branch coverage, each branch in the code must be tested, ensuring both the true and false outcomes of each condition are covered.

→ Test Case 3: Multiple Points with Increasing Y-values
- Input: [(0, 1), (1, 2), (2, 3)]
- Description: 5
  - Ensures Condition 2 evaluates to both true and false.
  - The first loop finds the minimum y-value, but no updates occur in the second loop due to differing y-values.
- Expected Output: 0

→ Test Case 4: Points with the Same Y-value and Larger X-value
- Input: [(0, 0), (2, 0), (1, 1)]
- Description:
  - This covers both branches of Condition 4 in the second loop.
  - The second loop will update min to the point (2,0) as it has the same y as (0,0) but a larger x-value.
- Expected Output: 1

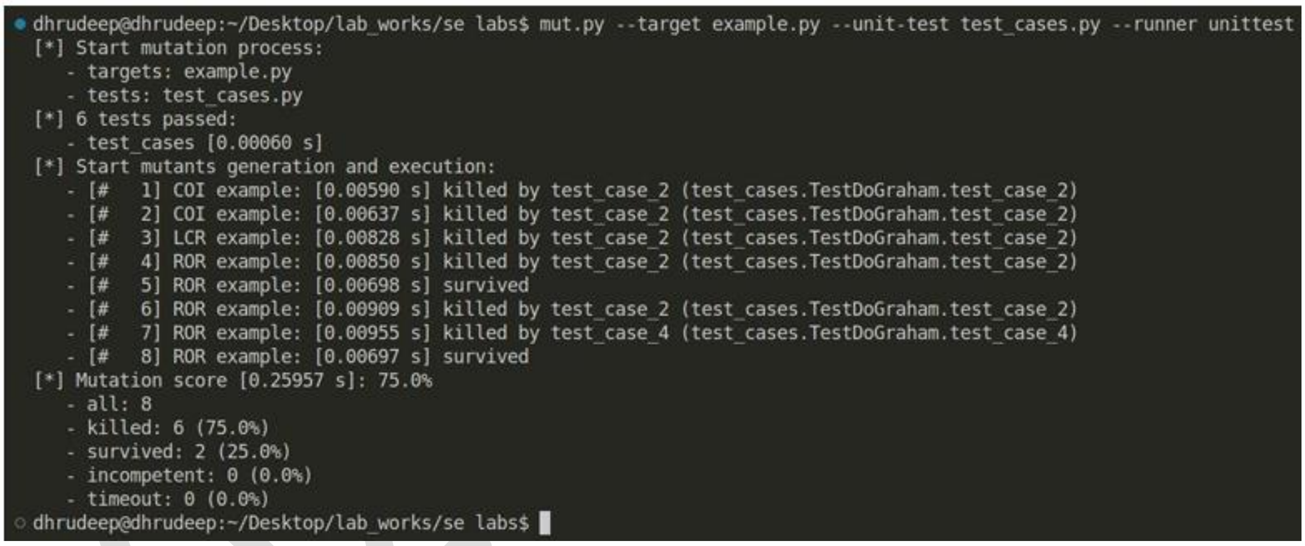
3. Basic Condition

Coverage To achieve basic condition coverage, each basic condition (part of a compound condition) must independently evaluate to true and false. This requires a more nuanced set of points to trigger all possible combinations of true and false for each condition.

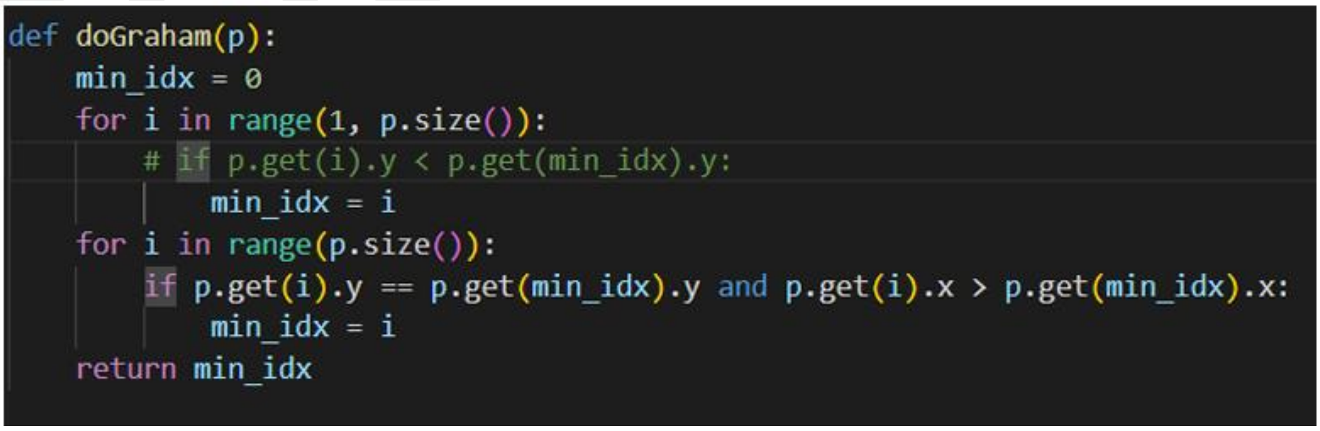
→ Test Case 5: Points with Equal Y-values but Different X-values
- Input: [(0, 0), (2, 0), (1, 0)]
- Description:
  - This tests Condition 4, ensuring that min is updated to the point with the largest x-value, which would be (2,0).
- Expected Output: 1

→ Test Case 6: Points with Different Y-values and Some with the Same Y
- Input: [(1, 1), (2, 1), (0, 0)]
- Description: 6
  - Ensures both Condition 2 (y comparison in the first loop) and Condition 3 (same y in the second loop) are evaluated to both true and false.
- Expected Output: 2

3. **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. You have to use the mutation testing tool.**

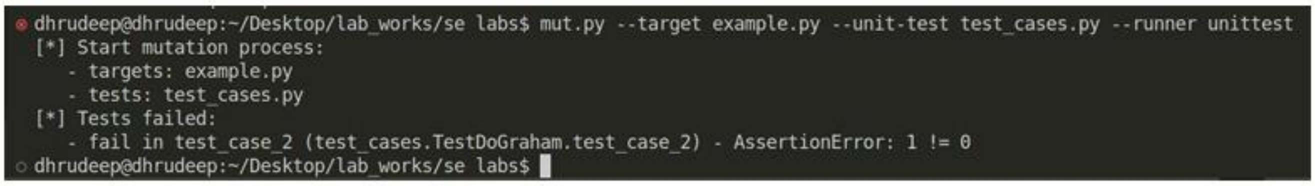


```
dhrudeep@dhrudeep:~/Desktop/lab_works/se labs$ mut.py --target example.py --unit-test test_cases.py --runner unittest
[*] Start mutation process:
   - targets: example.py
   - tests: test_cases.py
[*] 6 tests passed:
   - test_cases [0.00060 s]
[*] Start mutants generation and execution:
   - [#   1] COI example: [0.00590 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   2] COI example: [0.00637 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   3] LCR example: [0.00828 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   4] ROR example: [0.00850 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   5] ROR example: [0.00698 s] survived
   - [#   6] ROR example: [0.00909 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   7] ROR example: [0.00955 s] killed by test_case_4 (test_cases.TestDoGraham.test_case_4)
   - [#   8] ROR example: [0.00697 s] survived
[*] Mutation score [0.25957 s]: 75.0%
   - all: 8
   - killed: 6 (75.0%)
   - survived: 2 (25.0%)
   - incompetent: 0 (0.0%)
   - timeout: 0 (0.0%)
dhrudeep@dhrudeep:~/Desktop/lab_works/se labs$
```



```python
def doGraham(p):
    min_idx = 0
    for i in range(1, p.size()):
        # if p.get(i).y < p.get(min_idx).y:
            min_idx = i
    for i in range(p.size()):
        if p.get(i).y == p.get(min_idx).y and p.get(i).x > p.get(min_idx).x:
            min_idx = i
    return min_idx
```

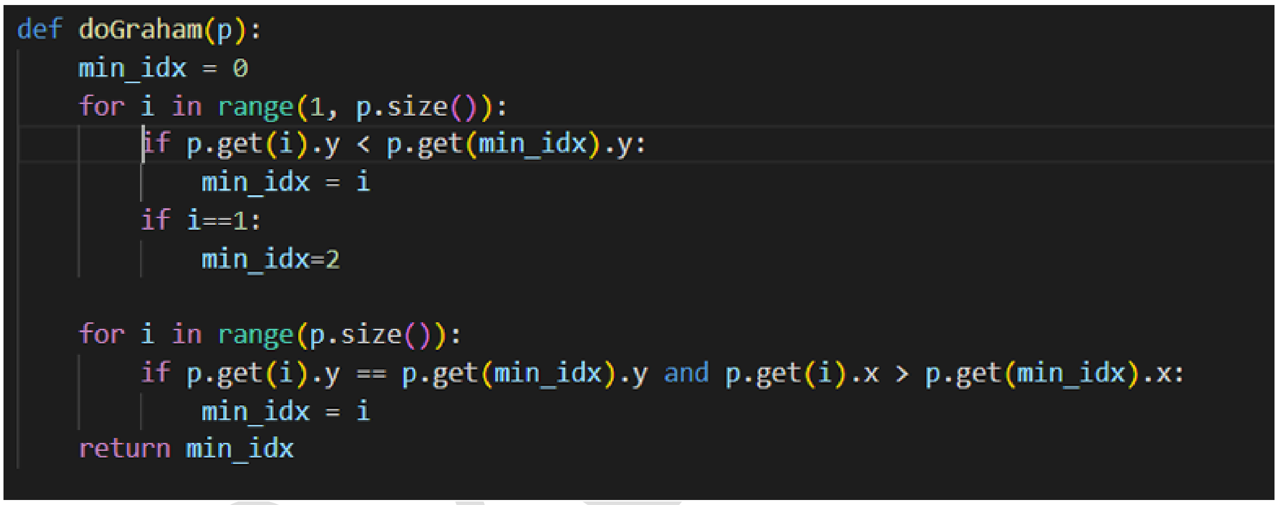
Output of the Mutation testing:



```
dhrudeep@dhrudeep:~/Desktop/lab_works/se labs$ mut.py --target example.py --unit-test test_cases.py --runner unittest
 [*] Start mutation process:
    - targets: example.py
    - tests: test_cases.py
 [*] Tests failed:
    - fail in test_case_2 (test_cases.TestDoGraham.test_case_2) - AssertionError: 1 != 0
dhrudeep@dhrudeep:~/Desktop/lab_works/se labs$
```

2. Mutation by Inserting Code:

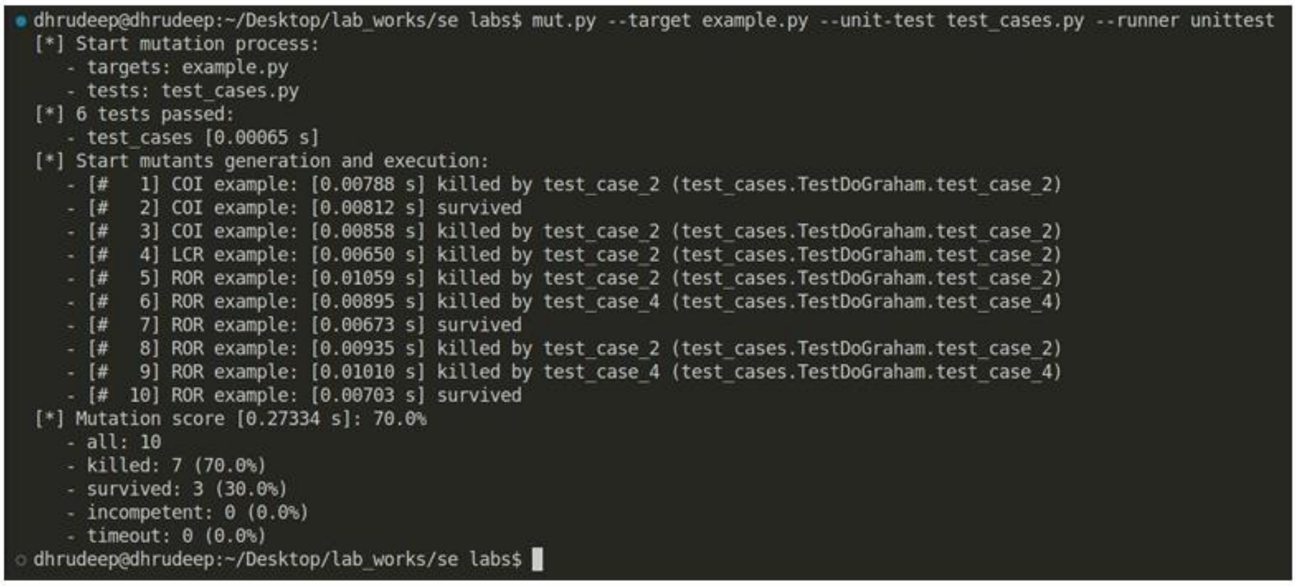
Added an if Condition:



```
def doGraham(p):
    min_idx = 0
    for i in range(1, p.size()):
        if p.get(i).y < p.get(min_idx).y:
            min_idx = i
        if i==1:
            min_idx=2

    for i in range(p.size()):
        if p.get(i).y == p.get(min_idx).y and p.get(i).x > p.get(min_idx).x:
            min_idx = i
    return min_idx
```

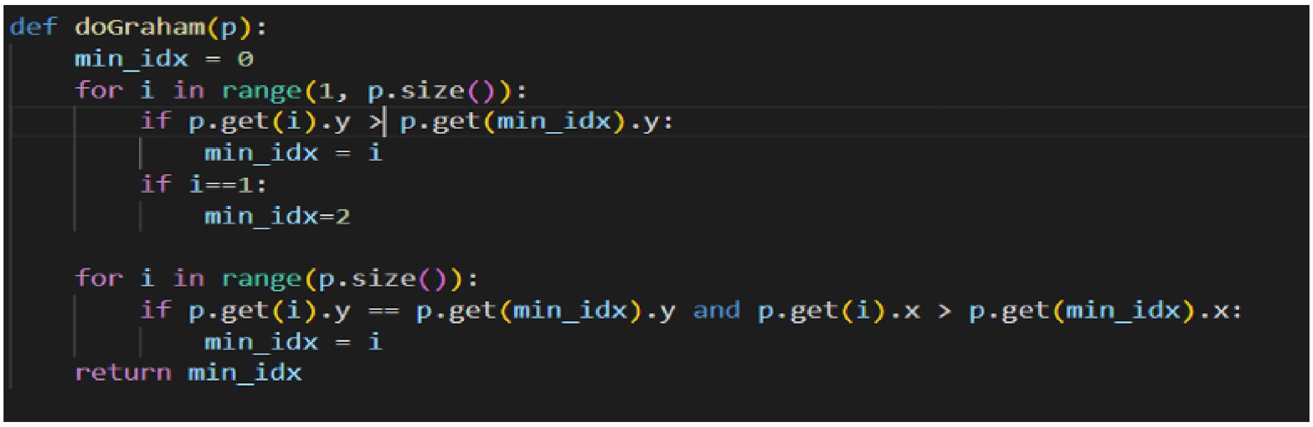
Output of the Mutation testing:



```
dhrudeep@dhrudeep:~/Desktop/lab_works/se labs$ mut.py --target example.py --unit-test test_cases.py --runner unittest
 [*] Start mutation process:
    - targets: example.py
    - tests: test_cases.py
 [*] 6 tests passed:
    - test_cases [0.00065 s]
 [*] Start mutants generation and execution:
    - [#   1] COI example: [0.00788 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
    - [#   2] COI example: [0.00812 s] survived
    - [#   3] COI example: [0.00858 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
    - [#   4] LCR example: [0.00650 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
    - [#   5] ROR example: [0.01059 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
    - [#   6] ROR example: [0.00895 s] killed by test_case_4 (test_cases.TestDoGraham.test_case_4)
    - [#   7] ROR example: [0.00673 s] survived
    - [#   8] ROR example: [0.00935 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
    - [#   9] ROR example: [0.01010 s] killed by test_case_4 (test_cases.TestDoGraham.test_case_4)
    - [#  10] ROR example: [0.00703 s] survived
 [*] Mutation score [0.27334 s]: 70.0%
    - all: 10
    - killed: 7 (70.0%)
    - survived: 3 (30.0%)
    - incompetent: 0 (0.0%)
    - timeout: 0 (0.0%)
dhrudeep@dhrudeep:~/Desktop/lab_works/se labs$
```

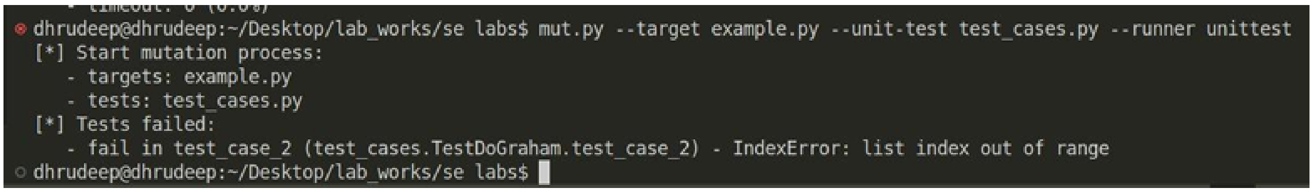
3. Mutation by Modifying Code:

Changing the inequality < to >:


```python
def doGraham(p):
    min_idx = 0
    for i in range(1, p.size()):
        if p.get(i).y > p.get(min_idx).y:
            min_idx = i
        if i==1:
            min_idx=2

    for i in range(p.size()):
        if p.get(i).y == p.get(min_idx).y and p.get(i).x > p.get(min_idx).x:
            min_idx = i
    return min_idx
```

Output of the Mutation testing:


```
dhrudeep@dhrudeep:~/Desktop/lab_works/se labs$ mut.py --target example.py --unit-test test_cases.py --runner unittest
 [*] Start mutation process:
   - targets: example.py
   - tests: test_cases.py
 [*] Tests failed:
   - fail in test_case_2 (test_cases.TestDoGraham.test_case_2) - IndexError: list index out of range
dhrudeep@dhrudeep:~/Desktop/lab_works/se labs$
```

4. Create a test set that satisfies the path coverage criterion where every loop is explored at least zero, one or two times.

1. Test Case: Zero Iterations of the First Loop

● Description: This test verifies the function's behavior when all points have the same coordinates, ensuring the first loop doesn't execute.

● Input:

○ Points: (2,2), (2,2), (2,2)

● Expected Output: 0 (The index of the only point)

2. Test Case: One Iteration of the First Loop

● Description: This test checks the case where the first point has a higher y-value than the second, causing the first loop to run once.

● Input:

○ Points: (2,3), (1,1), (2,3), (1,1), (2,3), (1,1)

● Expected Output: 1 (The index of the point with the minimum y-value)

3. Test Case: Two Iterations of the First Loop

● Description: This test evaluates how the function handles three points with decreasing y-values, ensuring the first loop iterates twice.

● Input:

○ Points: (3,3), (2,2), (1,1), (3,3), (2,2), (1,1), (3,3), (2,2), (1,1)

● Expected Output: 2 (The index of the point with the minimum y-value)

4. Test Case: Zero Iterations of the Second Loop

● Description: This test confirms that the second loop does not run when all points are identical.

● Input:

○ Points: (5,5), (5,5), (5,5)

10

● Expected Output:0 (The index of the only point)

5. Test Case: One Iteration of the Second Loop

● Description: This test examines the function's behavior when two points share the same y-value, ensuring the second loop runs once.

● Input:

○ Points: (4,0), (7,0), (4,0), (7,0), (4,0), (7,0)

● Expected Output: 1 (The index of the point with the larger x-value)

6. Test Case: Two Iterations of the Second Loop

● Description: This test checks the function's behavior with three points having the same y-value, ensuring the second loop runs twice and selects the point with the largest x-value.

● Input:

○ Points: (1,0), (3,0), (5,0), (1,0), (3,0), (5,0), (1,0), (3,0), (5,0)

● Expected Output: 2 (The index of the point with the largest x-value