Lab : 9

Mutation Testing

Jaimin Prajapati - 202201228

8th November, 2024

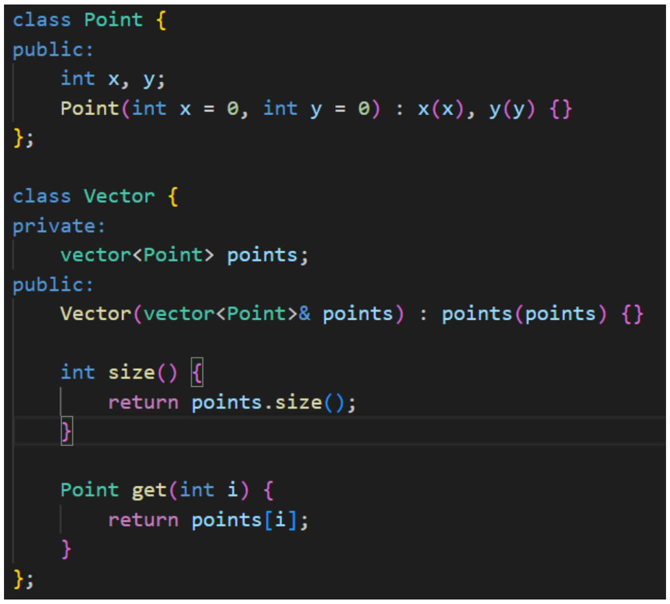
Q.1. The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise, you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size() 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;
            }
        }
```

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.

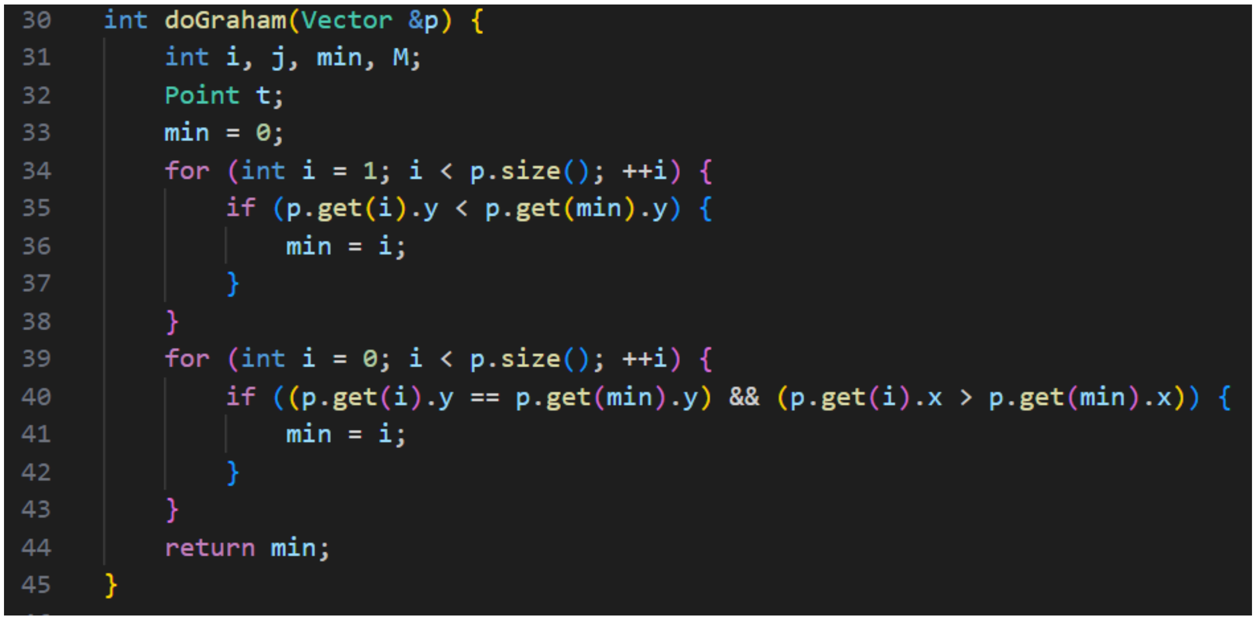
Answer :

Required class :



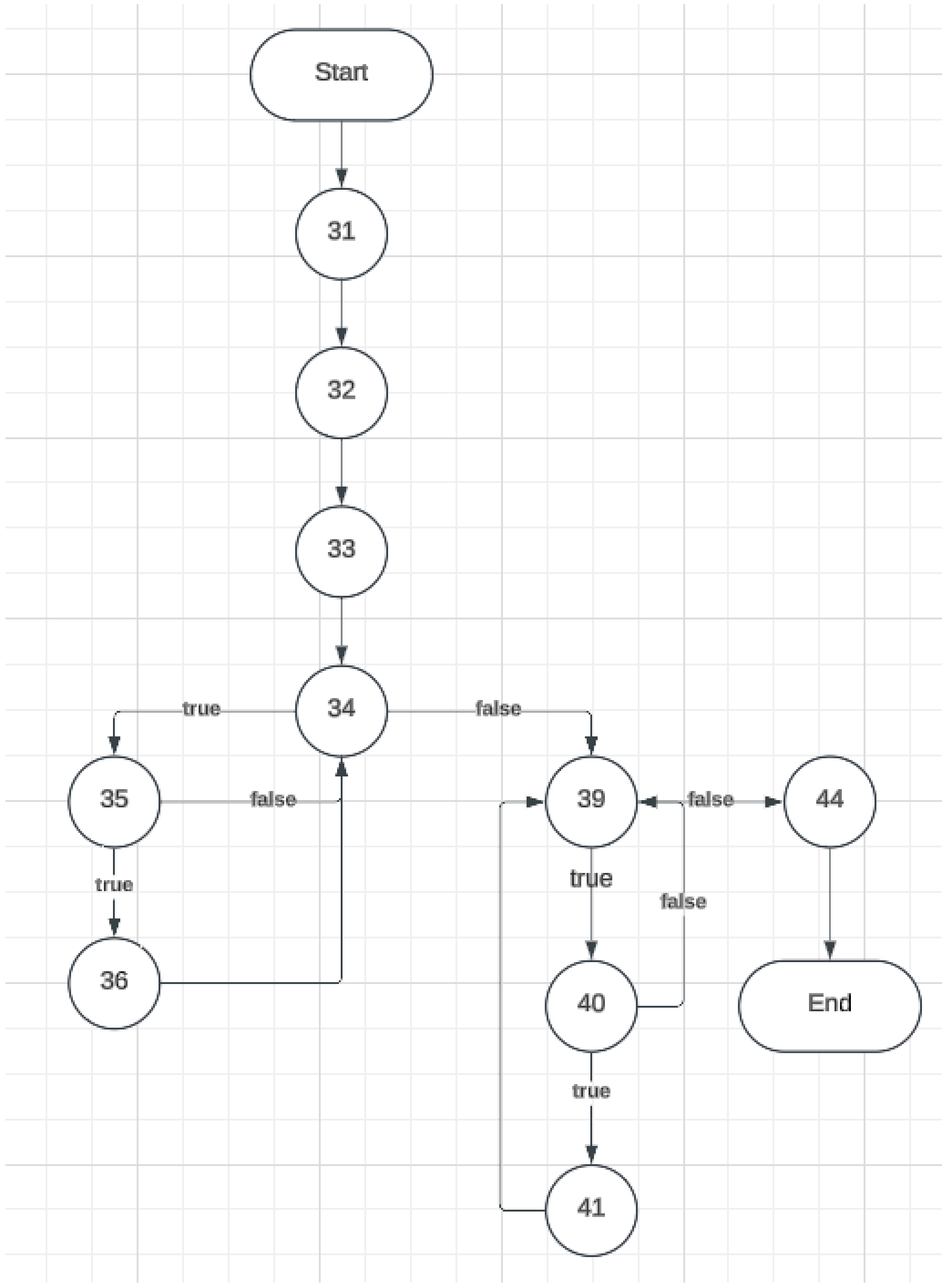
```cpp
class Point {
public:
    int x, y;
    Point(int x = 0, int y = 0) : x(x), y(y) {}
};

class Vector {
private:
    vector<Point> points;
public:
    Vector(vector<Point>& points) : points(points) {}

    int size() {
        return points.size();
    }

    Point get(int i) {
        return points[i];
    }
};
```

Code in C++ :



```cpp
int doGraham(Vector &p) {
    int i, j, min, M;
    Point t;
    min = 0;
    for (int i = 1; i < p.size(); ++i) {
        if (p.get(i).y < p.get(min).y) {
            min = i;
        }
    }
    for (int i = 0; i < p.size(); ++i) {
        if ((p.get(i).y == p.get(min).y) && (p.get(i).x > p.get(min).x)) {
            min = i;
        }
    }
    return min;
}
```

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.

Answer :

In the `doGraham` method, there are two main sections:

1. **First Loop:** This loop iterates through the points to identify the one with the smallest y-coordinate. If multiple points share the same y-coordinate, the first one encountered is selected.
    - **Condition 1:** `i < p.size()` – This controls the loop's termination.
    - **Condition 2:** `((Point) p.get(i)).y < ((Point) p.get(min)).y` – This updates the `min` index if the current point's y-value is smaller than the current minimum.
2. **Second Loop:** This loop iterates through the points again to find the point with the same y-coordinate as the minimum but with the largest x-coordinate.
    - **Condition 3:** `((Point) p.get(i)).y == ((Point) p.get(min)).y` – This checks if the current point shares the same y-coordinate as the one with the minimum y.
    - **Condition 4:** `((Point) p.get(i)).x > ((Point) p.get(min)).x` – This updates the `min` index if the current point's x-coordinate is larger than the current maximum x with the same y-coordinate.

Coverage Criteria:

- **Statement Coverage:** Ensures that each line of code is executed at least once.
- **Branch Coverage:** Ensures that each possible branch (true/false path) is taken at least **once.**
- **Basic Condition Coverage:** Ensures that each individual condition (part of a compound condition) is evaluated for both true and false outcomes independently.

1. Statement Coverage

To achieve statement coverage, every line of code must be executed at least once. This can be done by creating minimal test cases that cover each part of the code:

- **Test Case 1**: A single point (0, 0)
    - This will initialize `min = 0` but won't update it, as there is only one point.
    - It covers the initialization and both loops without causing any updates.

- **Test Case 2**: Two points with different y-values, e.g., [(0, 0), (1, 1)]
    - This test case will trigger the update of `min` in the first loop.
    - The second loop will execute without updating `min`, as the y-values are different.

2. Branch Coverage

To achieve branch coverage, each branch in the code must be tested, ensuring that both the true and false outcomes of each condition are covered. This requires more diverse test cases:

- **Test Case 3**: Multiple points with increasing y-values, e.g., [(0, 1), (1, 2), (2, 3)]
    - This ensures that Condition 2 is evaluated to both true and false. The first loop will find the minimum y-value, but no updates will occur in the second loop since the y-values are different.
- **Test Case 4**: Points where one point has a larger x-value but the same y as the minimum, e.g., [(0, 0), (2, 0), (1, 1)]
    - This will test both branches of Condition 4 in the second loop.
    - The second loop will update `min` to the point (2, 0) because it has the same y as (0, 0) but a larger x-value.

3. Basic Condition Coverage

To achieve basic condition coverage, each individual condition (the components of compound conditions) must be evaluated both to true and false. This requires a more detailed set of test cases to cover all possible combinations of true and false for each condition:

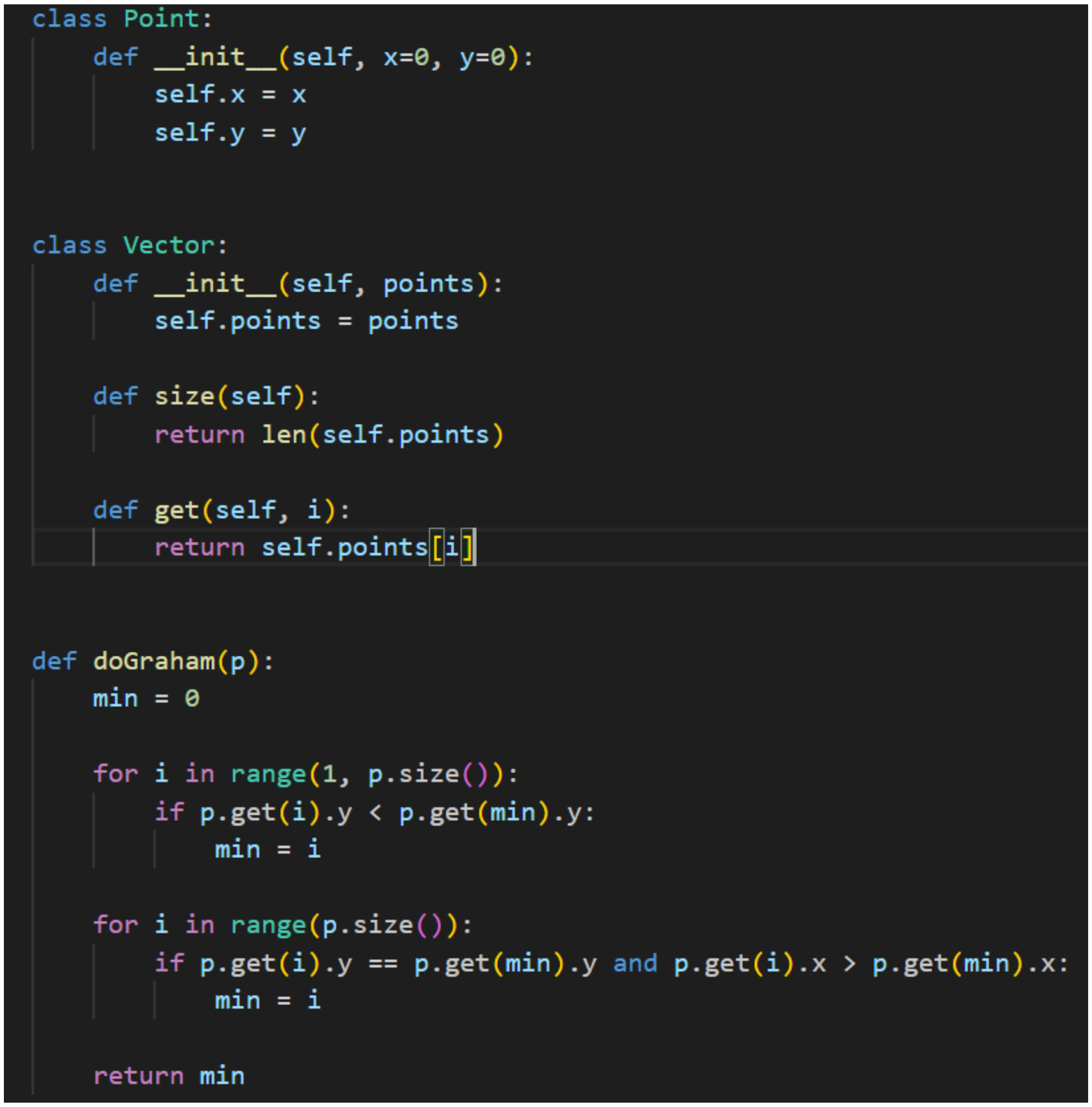
- **Test Case 5**: Points with equal y-values but different x-values, e.g., [(0, 0), (2, 0), (1, 0)]
    - This will test Condition 4, ensuring that `min` is updated to the point with the largest x-value, which in this case is (2, 0).
- **Test Case 6**: Points with different y-values, with some points sharing the same y, e.g., [(1, 1), (2, 1), (0, 0)]
    - This ensures that both Condition 2 (the y-value comparison in the first loop) and Condition 3 (the check for matching y-values in the second loop) are evaluated to both true and false.

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.

Answer :

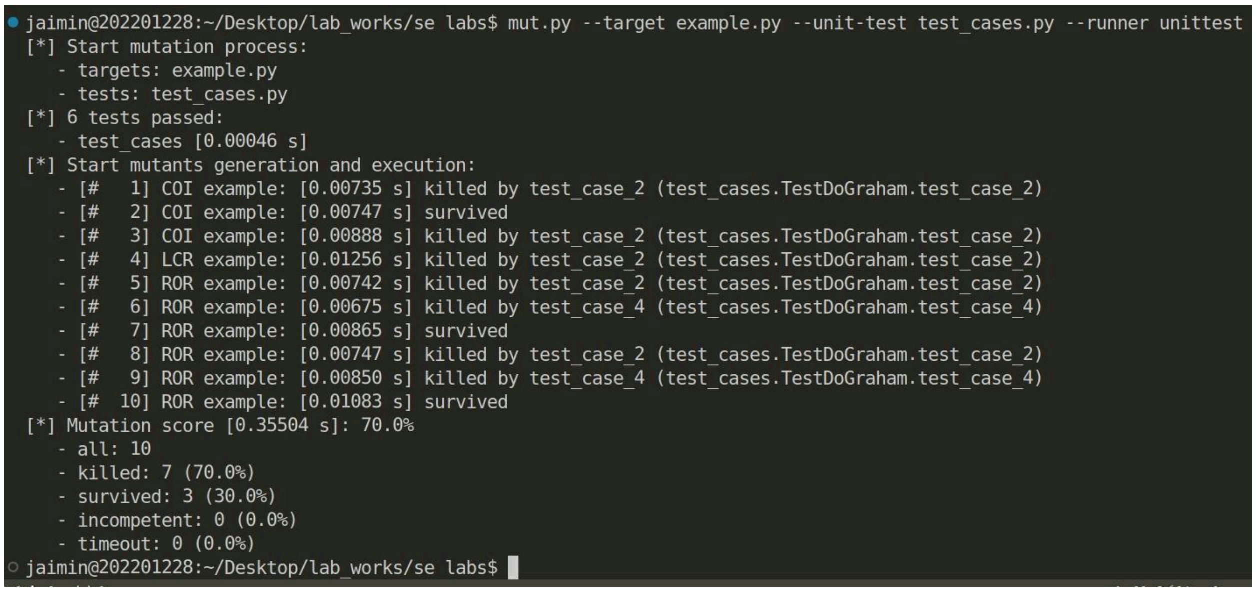
Convert the code from c++ to python to perform mutation testing using the python tool named mutpy.

Code :



```python
class Point:
    def __init__(self, x=0, y=0):
        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 = 0

    for i in range(1, p.size()):
        if p.get(i).y < p.get(min).y:
            min = i

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

    return min
```

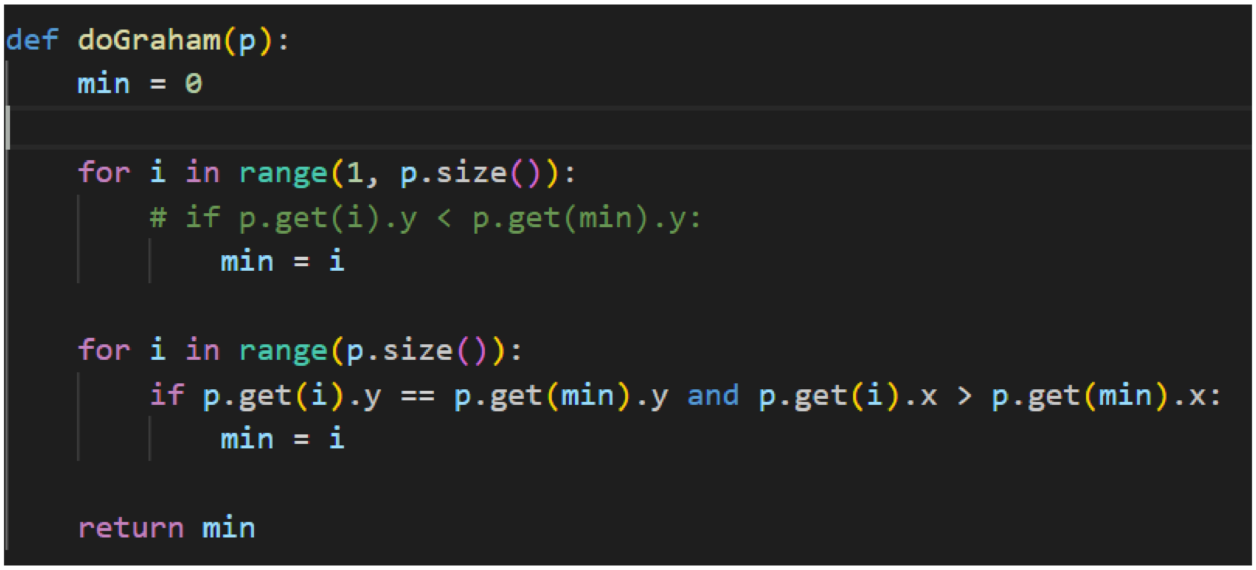
Mutation testing with mut.py in python :



```
jaimin@202201228:~/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 [0.00046 s]
[*] Start mutants generation and execution:
   - [#   1] COI example: [0.00735 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   2] COI example: [0.00747 s] survived
   - [#   3] COI example: [0.00888 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   4] LCR example: [0.01256 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   5] ROR example: [0.00742 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   6] ROR example: [0.00675 s] killed by test_case_4 (test_cases.TestDoGraham.test_case_4)
   - [#   7] ROR example: [0.00865 s] survived
   - [#   8] ROR example: [0.00747 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   9] ROR example: [0.00850 s] killed by test_case_4 (test_cases.TestDoGraham.test_case_4)
   - [#  10] ROR example: [0.01083 s] survived
[*] Mutation score [0.35504 s]: 70.0%
   - all: 10
   - killed: 7 (70.0%)
   - survived: 3 (30.0%)
   - incompetent: 0 (0.0%)
   - timeout: 0 (0.0%)
jaimin@202201228:~/Desktop/lab_works/se labs$
```

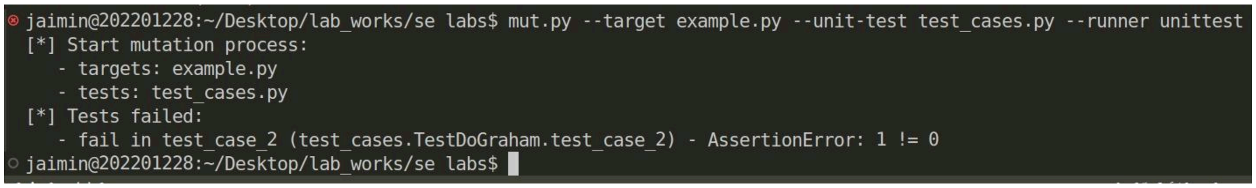
1. Mutation by Deleting Code:

Commented the if condition:



```python
def doGraham(p):
    min = 0

    for i in range(1, p.size()):
        # if p.get(i).y < p.get(min).y:
            min = i

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

    return min
```

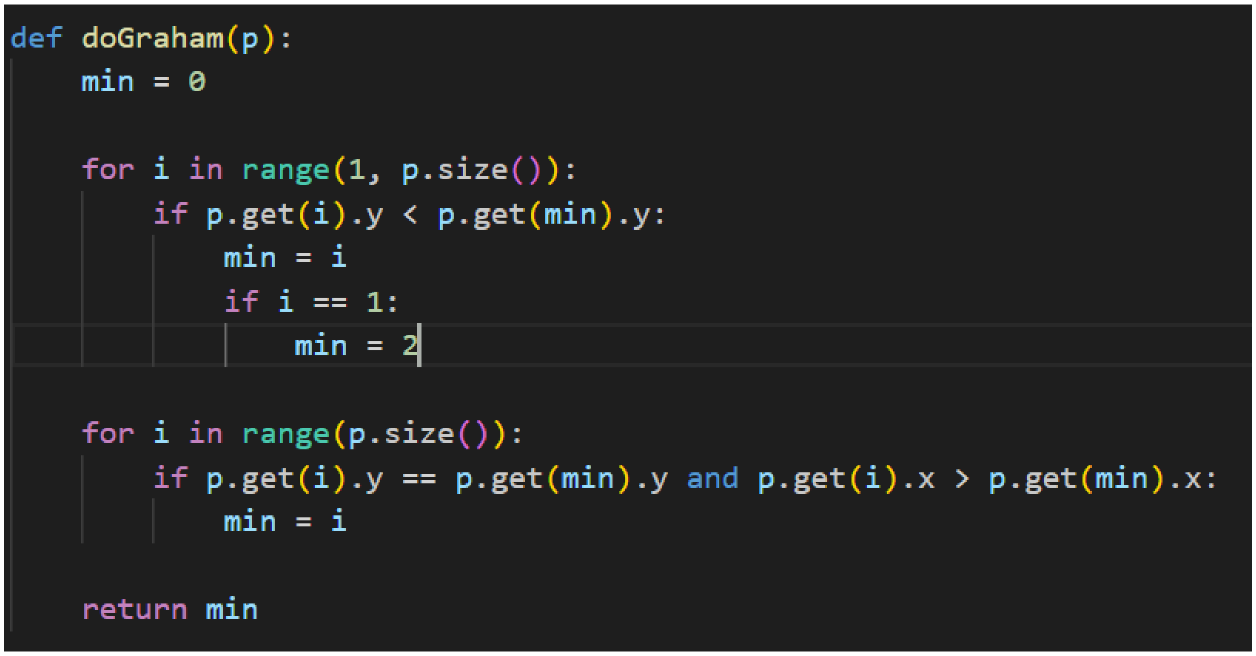
Output of the Mutation testing :



```
jaimin@202201228:~/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
jaimin@202201228:~/Desktop/lab_works/se labs$
```

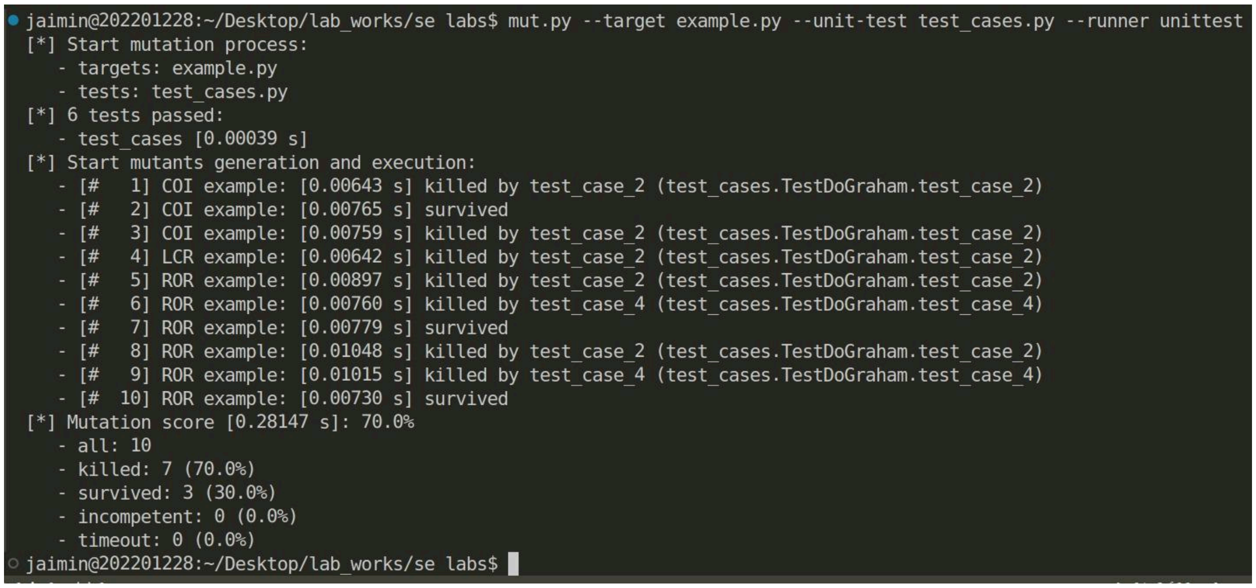
2. Mutation by Inserting Code:

Added an if Condition:



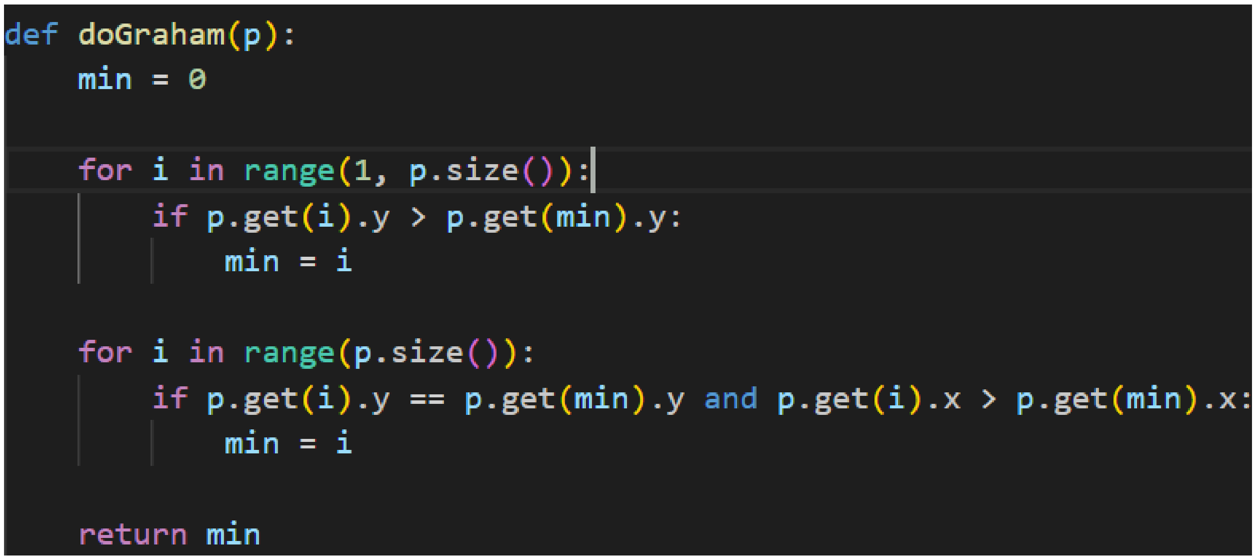
```
def doGraham(p):
    min = 0

    for i in range(1, p.size()):
        if p.get(i).y < p.get(min).y:
            min = i
            if i == 1:
                min = 2

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

    return min
```

Output of Mutation testing:

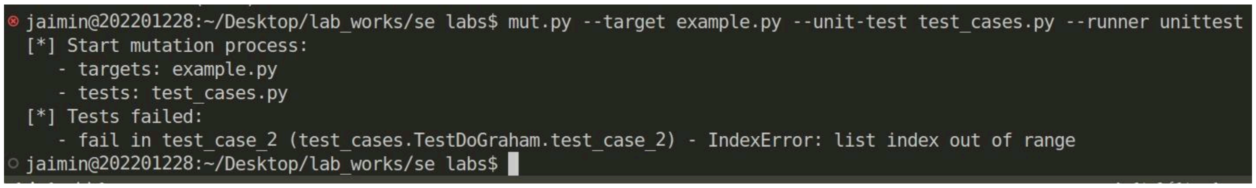

```
jaimin@202201228:~/Desktop/lab_works/se labs$ mut.py --target example.py --unit-test test_cases.py --runner unittest
[*] Start mutation process:
   - targets: example.py
   - tests: test_cases.py
[*] 6 tests passed:
   - test_cases [0.00039 s]
[*] Start mutants generation and execution:
   - [#   1] COI example: [0.00643 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   2] COI example: [0.00765 s] survived
   - [#   3] COI example: [0.00759 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   4] LCR example: [0.00642 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   5] ROR example: [0.00897 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   6] ROR example: [0.00760 s] killed by test_case_4 (test_cases.TestDoGraham.test_case_4)
   - [#   7] ROR example: [0.00779 s] survived
   - [#   8] ROR example: [0.01048 s] killed by test_case_2 (test_cases.TestDoGraham.test_case_2)
   - [#   9] ROR example: [0.01015 s] killed by test_case_4 (test_cases.TestDoGraham.test_case_4)
   - [#  10] ROR example: [0.00730 s] survived
[*] Mutation score [0.28147 s]: 70.0%
   - all: 10
   - killed: 7 (70.0%)
   - survived: 3 (30.0%)
   - incompetent: 0 (0.0%)
   - timeout: 0 (0.0%)
jaimin@202201228:~/Desktop/lab_works/se labs$
```

3. Mutation by Modifying Code:

Changing the inequality < to >:


```python
def doGraham(p):
    min = 0

    for i in range(1, p.size()):
        if p.get(i).y > p.get(min).y:
            min = i

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

    return min
```

Output of Mutation testing:


```
jaimin@202201228:~/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
jaimin@202201228:~/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.

Answer :

1. **Test Case: First Loop Zero Iterations**
   Description: This test case checks the function's behavior when given a single point, confirming that the first loop does not run.
   Input:
- Points: (0,0)(0,0)(0,0)

  Expected Output:
- 0 (the index of the single point)

2. **Test Case: Single Iteration of the First Loop**
   Description: This test examines a scenario with two points where the first point has a smaller y-coordinate than the second, resulting in a single iteration of the first loop.
   Input:
- Points: (0,0), (1,1), (0,0), (1,1), (0,0), (1,1)

  Expected Output:
- 0 (the index of the point with the minimum y-coordinate)

3. **Test Case: Two Iterations of the First Loop**
   Description: This test evaluates the function's handling of three points with progressively increasing y-coordinates, confirming that the first loop runs twice.
   Input:
- Points: (0,1), (1,2), (2,3), (0,1), (1,2), (2,3), (0,1), (1,2), (2,3)

  Expected Output:
- 0 (the index of the point with the lowest y-coordinate)

4. **Test Case: No Iterations of the Second Loop**
   Description: This test confirms that the second loop does not run when there is only a single point provided.
   Input:
- Points: (0,0), (0,0), (0,0)

  Expected Output:
- 0 (the index of the sole point)

5. **Test Case: Single Iteration of the Second Loop**
   Description: This test verifies the function's behavior with two points that have the same y-coordinate, ensuring the second loop executes once.
   Input:
- Points: (0,0), (1,0), (0,0), (1,0), (0,0), (1,0)

  Expected Output:
- 1 (the index of the point with the higher x-coordinate)

6. **Test Case: Two Iterations of the Second Loop**
   Description: This test checks the function's behavior with three points, all sharing the same y-coordinate, to confirm that the second loop runs twice and correctly identifies the last point.
   Input:
- Points: (0,0), (1,0), (2,0), (0,0), (1,0), (2,0), (0,0), (1,0), (2,0)

  Expected Output:
- 2 (the index of the point with the highest x-coordinate)