Dhirubhai Ambani
Institute of Information and Communication Technology

Lab 9

Subject: Software Engineering

Subject code: IT-314

Student Name: Pari Chauhan

Student Id: 202201189

[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.

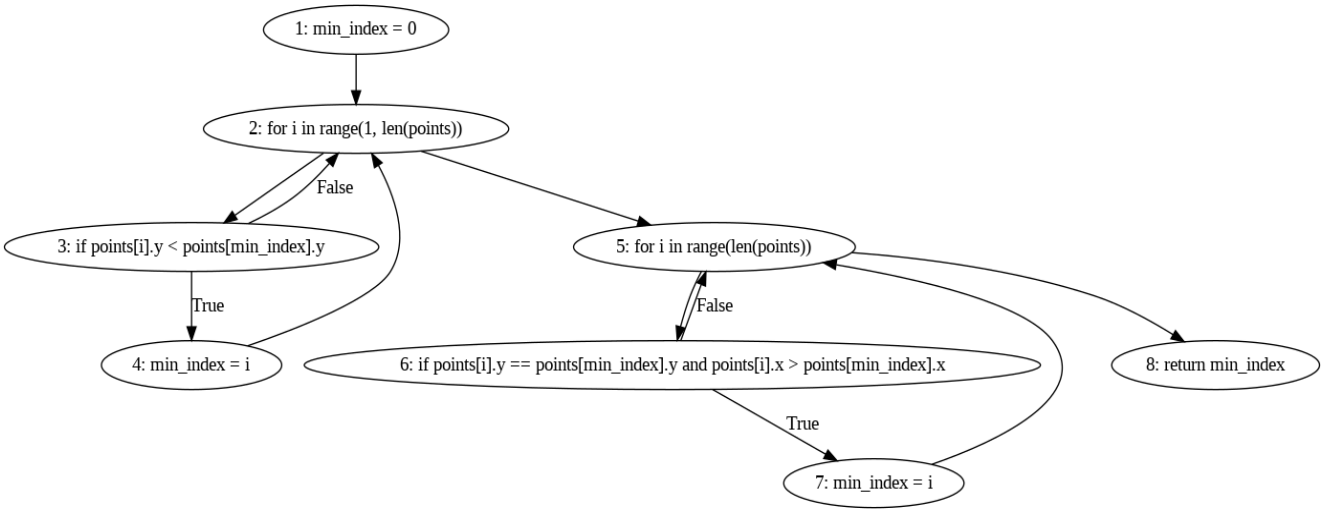
For the given code fragment, you should carry out the following activities.
1. Convert the code comprising the beginning of the doGraham method into a control flow graph (CFG). You are free to write the code in any programming language.

Code:

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

def do_graham(points): min_index = 0

  for i in range(1, len(points)):
        if points[i].y &lt;
          points[min_index].y:
          min_index = i
```

```
    for i in range(len(points)):
      if points[i].y == points[min_index].y and points[i].x &gt;
          points[min_index].x: min_index = i

    return min_index
```

Control Flow Diagram:



[Q.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.

1. Statement Coverage

Statement coverage ensures each statement of code executes at least once. For 100 % coverage.

Test cases:

T1 : Input vector with origin points ([(0, 0)]

T2 : Input vector with only 1 point (e.g., [(2, 3)]

T3 : Input vector with 2 points, one lower y value (e.g, [(1, 2), (2, 0)])

T4 : Input vector with points having the same y value but different x values (e.g, [(1, 2), (3, 2), (4, 2), (5, 2)]

2. Branch coverage

Branch coverage ensures that every possible branch (true/false) of each decision point (if condition) is executed at least once.

Test cases:

T1 : Input vector with only one point (e.g., [(0, 0)])

The first loop exits immediately, covering the loop without changes

T2 : Input vector with only 1 point (e.g., [(2, 3)]

The first loop exits immediately, covering the loop without changes

T3 : Input vector with 2 points, one lower y value (e.g, [(1, 2), (2, 0)])

The minimum is updated to the second point.

T4 : Input vector with points having the same y value but different x values (e.g, [(1, 2), (3, 2), (4, 2), (5, 2)]

True branch for 2nd for loop's if condition.

T5 : Input vector with all points having the same y value (e.g., [(1, 1), (1, 1), (1, 1)])

False branch : 2nd for loop executes without changing min.

3. Basic condition coverage

Basic Condition Coverage requires that each individual condition within a decision (e.g., each sub-condition in an if statement) be evaluated both to true and false.

In the first loop:

Condition 1: ((Point) p.get(i)).y < ((Point) p.get(min)).y

In the second loop:

Condition 2: ((Point) p.get(i)).y == ((Point) p.get(min)).y

Condition 3: ((Point) p.get(i)).x > ((Point) p.get(min)).x

Test cases:

T1 : Input vector with only one point (e.g., [(0, 0)])

Covers condition 1, y as false since no comparison can be made.

T2 : Input vector with only 1 point (e.g., [(2, 3)]

Covers condition 1 as true.

T3 : Input vector with 2 points, one lower y value (e.g, [(1, 2), (2, 0)])

Converse condition 1 as true and 2nd as false.

T4 : Input vector with points having the same y value but different x values (e.g, [(1, 2), (3, 2), (4, 2), (5, 2)]

Converse all conditions as true.

T5 : Input vector with all points having the same y value (e.g., [(1, 1), (1, 1), (1, 1)])

Covers all conditions as false.

Summary of test cases :

Test case	Input data	Coverage
1	[(0, 0)]	Statement coverage
2	[(2, 3)	
3	[(1, 2), (2, 0)	
4	[(1, 2), (3, 2), (4, 2), (5, 2)]	
1	[(0, 0)]	Branch coverage
2	[(2, 3)	
3	[(1, 2), (2, 0)	
4	[(1, 2), (3, 2), (4, 2), (5, 2)]	
5	[(1, 1), (1, 1), (1, 1)]	
1	[(0, 0)]	Basic condition coverage
2	[(2, 3)	
3	[(1, 2), (2, 0)	
4	[(1, 2), (3, 2), (4, 2), (5, 2)]	
5	[(1, 1), (1, 1), (1, 1)]	

[Q.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.

Mutation Testing :

Mutation testing involves modifying (mutating) the code in small ways to create "mutants" and checking if the existing test suite detects the changes. If a mutant is created but the test suite doesn't detect it (i.e., it still passes as if no mutation occurred), then the test suite is considered inadequate for detecting that particular type of fault.

```
[*] Start mutation process:
   - targets: point
   - tests: test_points
[*] 4 tests passed:
   - test_points [0.36220 s]
[*] Start mutants generation and execution:
   - [#   1] COI point:
----------------------------------------------------------------------------
   6:
   7: def find_min_point(points):
   8:     min_index = 0
   9:     for i in range(1, len(points)):
- 10:         if points[i].y < points[min_index].y:
+ 10:         if not (points[i].y < points[min_index].y):
  11:             min_index = i
  12:     for i in range(len(points)):
  13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
  14:             min_index = i
----------------------------------------------------------------------------
[0.23355 s] killed by test_points.py::TestFindMinPoint::test_multiple_points_with_ties
   - [#   2] COI point:
----------------------------------------------------------------------------
---9:-----for-i-in-range(1,-len(points)):----------------------------------------
[0.23355 s] killed by test_points.py::TestFindMinPoint::test_multiple_points_with_ties
   - [#   2] COI point:
----------------------------------------------------------------------------
   9:     for i in range(1, len(points)):
  10:         if points[i].y < points[min_index].y:
  11:             min_index = i
  12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if not ((points[i].y == points[min_index].y and points[i].x > points[min_index].x))
  14:             min_index = i
  15:     return points[min_index]
----------------------------------------------------------------------------
[0.27441 s] killed by test_points.py::TestFindMinPoint::test_multiple_points_with_same_y
   - [#   3] LCR point:
----------------------------------------------------------------------------
   9:     for i in range(1, len(points)):
  10:         if points[i].y < points[min_index].y:
  11:             min_index = i
  12:     for i in range(len(points)):
- 13:         if (points[i].y == points[min_index].y and points[i].x > points[min_index].x):
+ 13:         if (points[i].y == points[min_index].y or points[i].x > points[min_index].x):
  14:             min_index = i
  15:     return points[min_index]
```

1. Deletion mutation

- **Mutation :** Remove the line min=0 at the start of the code.

- **Expected effects :**
    - If min is not initialized to 0, it could hold a random value. This would affect the correctness of the min selection in both loops. This could lead to any garbage value.
    - **Mutation outcome** : This mutation could result in an incorrect starting index for min, leading to erroneous min point selection.

2. Change mutation

- **Mutation** : modify the condition in the first if statement to :

  if ((Point) p.get(i)).y <= ((Point) p.get(min)).y

- **Expected effects :**
    - Changing < to <= would make the method select points with the same y value (instead of strictly lower y), possibly affecting the result by not properly finding the lowest y point.
    - **Mutation outcome :** If the points have the same y values then the code could return an x value lower than expected.

3. Insertion mutation

- **Mutation** : Insert an extra min = i; statement at the end of the second for loop.

- **Expected effects**
    - It would update the min value to the index of the last point in the vector p, which is incorrect. This happens unconditionally at the end of the second loop.
    - **Mutation outcome :** The method will fail to return the correct point in cases where the actual "minimum" point occurs earlier in the list or if the points have equal y values but the last point has a smaller x value. It would select the last point as minimum.

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

Test Cases:

T1 : Zero iterations

Input : An empty vector p.
Description: This case ensures that no iterations of either loop occur.
Expected Output: The function should return an empty result or a specific value indicating no points.

T2 : First loop executes 0 times; Second loop executes 0 times
Input: p = [(1, 2)] (Only one point, the first loop does not run)

Description :

- First loop executes 0 times because p.size() is 1.
- Second loop executes 0 times because there are no points with the same y value as min (since there's only one point).

Expected Output: The min value should remain 0, pointing to the first (and only) point (1, 2).

T3 : First loop executes 1 time; Second loop executes 0 times
Input: p = [(1, 2), (3, 4)] (Two points, the first loop runs once, and the second loop doesn't execute)

Description :

- First loop executes once, and since the second point has a larger y value, min will not be updated (remains 0).
- Second loop does not execute because there is no point with the same y value as min (i.e., the y value for p[0] is 2, and no other point has y == 2).

Expected Output: The min value should remain 0, pointing to the first point (1, 2).

T4 : Executes both the loops 1 time.
Input: p = [(1, 2), (2, 2)] (Two points, the first loop runs once, and the second loop runs once)

Description :

- First loop runs once, and min will remain 0 because both points have the same y value.
- Second loop executes once because the second point has the same y value as min, and it has a larger x value than the first point. This updates min to 1.

Expected outcome : The min value should be updated to 1, pointing to the second point (2, 2).

T5 : First loop executes 2 times; Second loop executes 0 times

Input: p = [(1, 2), (3, 2), (2, 5)] (Three points, the first loop runs twice, and the second loop doesn't execute)

Description :

- First loop runs twice, and min will remain 0 because no point has a smaller y value than (1, 2).
- Second loop does not execute because there are no points with y == 2 and x > 1. So, no point with the same y value and larger x is found.

Expected outcome: The min value should remain 0, pointing to the first point (1, 2).

T6 : First loop executes 2 times; Second loop executes 1 time

Input: p = [(1, 2), (3, 2), (2, 5)] (Three points, the first loop runs twice, and the second loop runs once)

Description :

- First loop runs twice, and min will remain 0 because no point has a smaller y value than (1, 2).
- Second loop runs once because the second point (3, 2) has the same y value as min and has a larger x value than (1, 2). So, min is updated to 1.

Expected outcome: The min value should be updated to 1, pointing to the second point (3, 2).

T7 : First loop executes 2 times; Second loop executes 2 times

Input: p = [(1, 2), (3, 2), (2, 5)] (Three points, the first loop runs twice, and the second loop runs twice).

Description :

- First loop runs twice, and min will remain 0 because no point has a smaller y value than (1, 2).
- Second loop runs twice because both the second and third points have y == 2. The second point has a larger x value, so min will be updated to 1 when comparing (3, 2) with (1, 2). The third point will also be compared, but it will not affect min since it has a smaller x value.

Expected outcome: The min value should be updated to 1, pointing to the second point (3, 2).

Summary of Test Set for Path Coverage :

	Input data	Expected outcome	Description
T1	[]	Empty	Empty vector (zero iterations for both loops).
T2	[(1, 2)]	min = 0	First loop executes 0 times, second loop executes 0 times.
T3	p = [(1, 2), (3, 4)]	min = 0	First loop executes 1 time, second loop executes 0 times.
T4	p = [(1, 2), (2, 2)]	min = 1	First loop executes 1 time, second loop executes 1 time.
T5	p = [(1, 2), (3, 2), (2, 5)]	min = 0	First loop executes 2 times, second loop executes 0 times.
T6	p = [(1, 2), (3, 2), (2, 5)]	min = 1	First loop executes 2 times, second loop executes 1 time.
T7	p = [(1, 2), (3, 2), (2, 5)]	min = 1	First loop executes 2 times, second loop executes 2 times.

Mutation Testing using mut.py Tool:

```
[0.12519 s] survived
[*] Mutation score [1.53947 s]: 75.0%
   - all: 8
   - killed: 6 (75.0%)
   - survived: 2 (25.0%)
   - incompetent: 0 (0.0%)
   - timeout: 0 (0.0%)
```

Lab Execution

[Q.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).

Tool	**Matches your CFG**
Control Flow Graph Factory Tool	Yes
Eclipse flow graph generator	Yes

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

	Input data	**Expected outcome**	**Description**
T1	[]	Empty	Empty vector (zero iterations for both loops).
T2	[(1, 2)]	min = 0	First loop executes 0 times, second loop executes 0 times.
T3	p = [(1, 2), (3, 4)]	min = 0	First loop executes 1 time, second loop executes 0 times.
T4	p = [(1, 2), (2, 2)]	min = 1	First loop executes 1 time, second loop executes 1 time.
T5	p = [(1, 2), (3, 2), (2, 5)]	min = 0	First loop executes 2 times, second loop executes 0 times.
T6	p = [(1, 2), (3, 2), (2, 5)]	min = 1	First loop executes 2 times, second loop executes 1 time.
T7	p = [(1, 2), (3, 2), (2, 5)]	min = 1	First loop executes 2 times, second loop executes 2 times.

[Q.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 operations separately, i.e., by deleting the code, by inserting the code, by modifying the code.

Mutation type	Description	Impact
Deletion	Removed the comparison condition in the first loop.	The first loop does not update min correctly, leading to an incorrect result.
Insertion	Inserted min = i; inside the second loop.	Forces the min value to be set to the last index in every iteration of the second loop, causing an incorrect result.
Modification	Changed the comparison condition in the first loop to compare x values instead of y values.	The program incorrectly uses x for finding the minimum, leading to an incorrect result.

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

Test Cases for Path Coverage:

	Input data	Expected outcome	Description
T1	[]	Empty	Empty vector(zero iterations for both loops).
T2	[(1, 2)]	min = 0	First loop executes 0 times; Second loop executes 0 times
T3	[(1, 2), (3, 4)]	min = 0	First loop executes 1 time; Second loop executes 0 times.
T4	[(1, 2), (2, 2)]	min = 1	First loop executes 1 time; Second loop executes 1 time.
T5	[(1, 2), (3, 2), (2, 5)]	min = 0	First loop executes 2 times; Second loop executes 0 times.

T6	[(1, 2), (3, 2), (2, 5)]	min = 1	First loop executes 2 times; Second loop executes 1 time.
T7	[(1, 2), (3, 2), (2, 2)]	min = 1	First loop executes 2 times; Second loop executes 2 times.
T8	[(1, 2), (3, 2), (2, 5)]	min = 1	First loop executes 2 times; Second loop executes 1 time.
T9	[(1, 2), (3, 2), (2, 2)]	min = 1	First loop executes 1 time; Second loop executes 2 times.
T10	[(1, 1), (1, 1), (2, 1), (3, 3)]	min = 3	Duplicate points with one being the max x; tests handling of duplicates.