IT-314
Software Engineering

Mutation Testing

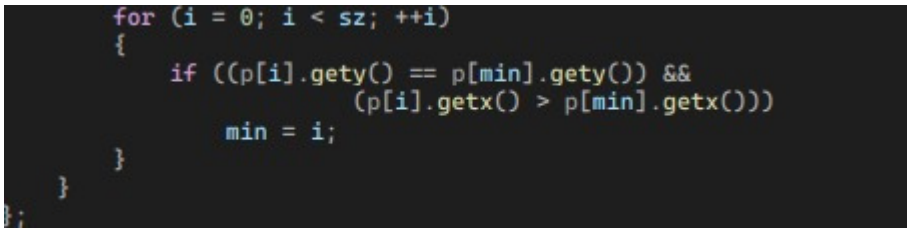
Jal Dani
202201315

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.

1. Convertthecodecomprisingthebeginningofthedo Grahammethodintoacontrol flow graph (CFG). You are free to write the code in any programming language.

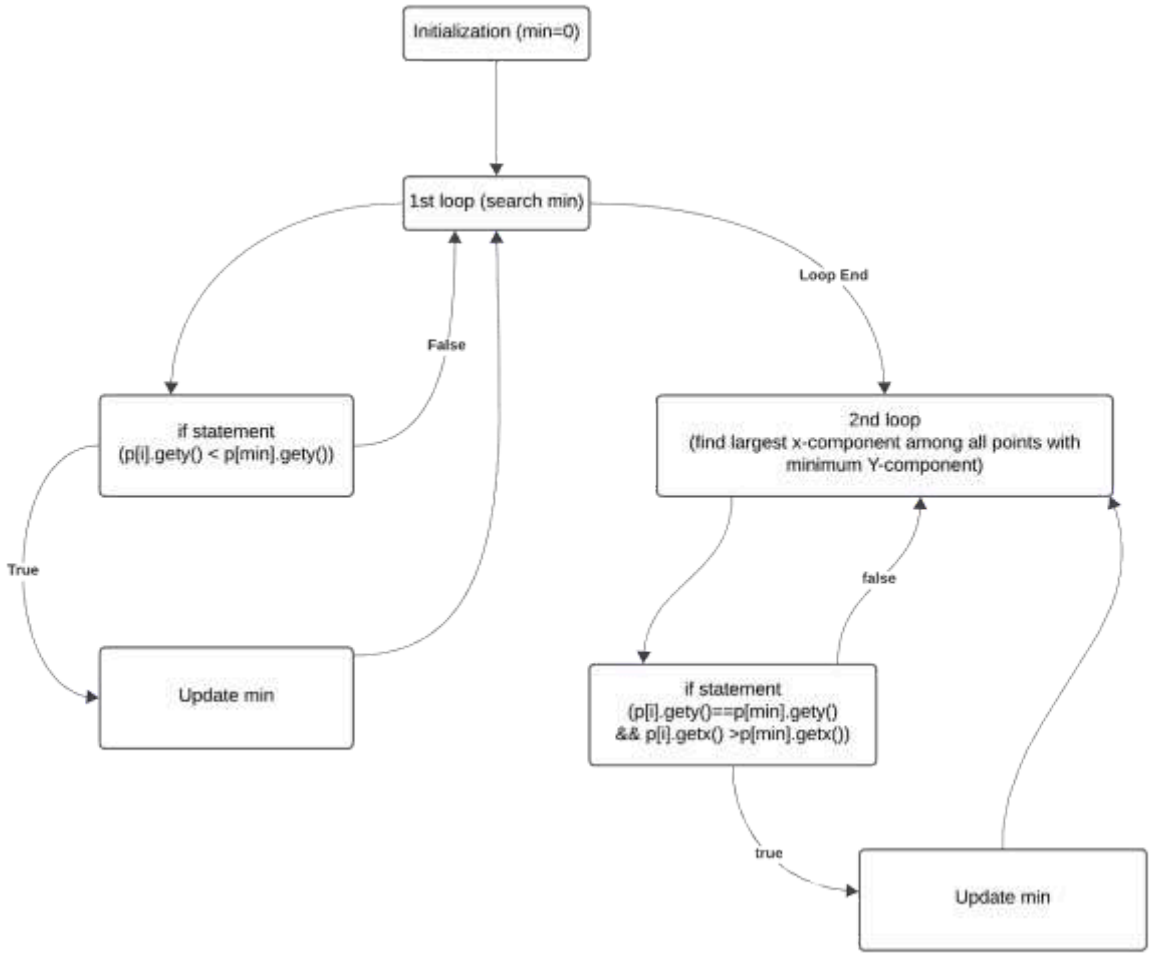
code in C++

```
#include <bits/stdc++.h>
using namespace std;

class Point{
    int x, y;

public:
    Point(int x, int y)
    {
        this->x = x;
        this->y = y;
    }
    int gety()
    {
        return y;
    }
    int getx()
    {
        return x;
    }
};

class ConvexHull{
public:
    void doGrahm(vector<Point> p)
    {
        int i = 1, sz = p.size();
        int min = 0;

        // search for minimum
        for (; i < sz; ++i)
        {
            if (p[i].gety() < p[min].gety())
                min = i;
        }
```



```
        for (i = 0; i < sz; ++i)
        {
            if ((p[i].gety() == p[min].gety()) &&
                        (p[i].getx() > p[min].getx()))
                min = i;
        }
    }
};
```

CFG



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

a) StatementCoverage.
b) BranchCoverage.
c) BasicConditionCoverage.

Sol – a) For Statement coverage every line of the code has to be executed at least once, ensuring all parts of CGF are covered by our test cases.

Test Set for Statement Coverage

• TC 1: An input vector with a single point (e.g., [(0, 0)]).

• TC 2: An input vector with two points, where one has a lower y-value (e.g., [(1, 1), (2, 0)]).

• TC3:Aninputvectorwithpointsthatsharethesamey-valuebuthavedifferent xvalues (e.g., [(1, 1), (2, 1), (3, 1)]).

Sol – b) To achieve Branch Coverage, each possible branch from a decision point must be taken at least once, meaning both the true and false outcomes of each condition are tested.

Test Set for Branch Coverage

-> TC 1: An input vector with a single point (e.g., [(0, 0)])

● True branch: The first loop exits immediately, covering the loop without making changes.

-> TC 2: An input vector with two points, where one has a lower y-value (e.g., [(1, 1),

(2, 0)])

● True branch: The minimum is updated to the second point.

-> TC 3: An input vector with points that have the same y-value but different x-values (e.g.,

[(1, 1), (3, 1), (2, 1)])

● True branch for the second condition.

-> TC 4: An input vector where all points have the same y-value (e.g., [(1, 1), (1, 1),

(1, 1)])

● False branch: Ensures the second loop runs without changing the minimum.

Sol – c) Basic Condition Coverage requires that each condition in the decision points is evaluated to both true and false at least once.

Test Set for Basic Condition Coverage

● Test Case 1: Input vector with only one point (e.g., [(0, 0)])

  ○ Coversthefirstconditionp.get(i).y<p.get(min).yasfalsesince nocomparisons can be made.

● TestCase2:Inputvectorwithtwopointswiththesecondpointhavingaloweryvalue (e.g., [(1, 1), (2, 0)])

◯ Covers the first condition true and the second condition false.

● TestCase3:Inputvectorwithpointsthathavethesameybutdifferentxvalues(e.g., [(1,1), (3, 1), (2,1)])

◯ Covers the second condition as true.

● TestCase4:Inputvectorwithpointsthathavethesameyandxvalues(e.g.,[(1, 1),(1, 1), (1, 1)])

◯ Covers both conditions as false.

Mutation Testing

1. Deletion Mutation

●Mutation :Removethelinemin=0;atthestartofthemethod.

● Expected Effect:

◯ Ifminisnotinitializedto0,itcouldholdarandomvalue.Thiswould affect the correctness of the min selection in both loops.

Mutation Outcome: This mutation could result in an incorrect starting index for min, leading to a faulty minimum point selection.

2. Change Mutation

Mutation: Modify the condition in the first if statement from < to <=:if ((Point) p.get(i)).y <= ((Point) p.get(min)).y

● Expected Effect:

◯ 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.

● MutationOutcome:Thecodecouldreturnapointwithanxvaluelowerthan expected if points have the same y value.

3. Insertion Mutation

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

● Expected Effect:

- ○ Thiswouldupdatemintothelastindexofp,whichisincorrect because min should only reflect the index of the minimum point.
- ○ Mutation Outcome: The method could incorrectly select the last point as the minimum, especially if the test cases don't specifically check the final value of min.

Test Cases for Path Coverage

To satisfy the path coverage criterion and ensure every loop is explored zero, one, or two times, we will create the following test cases:

Test Case 1: Zero Iterations Input:

An empty vector p.

Description: This case ensures that no iterations of either loop occur.

Expected Output: The function should handle this case gracefully (ideally return an empty result or a specific value indicating no points).

Test Case 2: One Iteration (First Loop)

Input: A vector with one point p (e.g., [(3, 4)]).

Description: This case ensures that the first loop runs exactly once (the minimum pointis the only point).

Expected Output: The function should return the only point in p.

Test Case 3: One Iteration (Second Loop)

Input: A vector with two points that have the same y-coordinate but different x-coordinates (e.g.,[(1, 2), (3, 2)]).

Description: This case ensures that the first loop finds the minimum point, and the second loop runs exactly once to compare the x-coordinates.

Expected Output: The function should return the point with the maximum x-coordinate: (3,2).

Test Case 4: Two Iterations (First Loop)

Input: A vector with multiple points, ensuring at least two with the same y-coordinate (e.g.,[(3, 1), (2, 2), (5, 1)]).

Description: This case ensures that the first loop finds the minimum y-coordinate (first iteration for (3,1)) and continues to the second loop.

Expected Output: Should return (5, 1) as it has the maximum x-coordinate among points with the same y.

Test Case 5: Two Iterations (Second Loop)

Input: A vector with points such that more than one point has the same minimum y-coordinate (e.g., [(1,1), (4, 1), (3, 2)]).

Description: This case ensures the first loop finds (1, 1), and the second loop runs twice to check other points with y = 1.

Expected Output: Should return (4, 1) since it has the maximum x-coordinate.

Test Case	InputVector	Description	Expected Output
TC1	[]	EmptyVector	Handle carefully
TC2	[(3,4)]	One point	[(3,4)]
TC3	[(1,2), (3,2)]	Two points with same Y coordinate	[(3,2)]
TC4	[(3,1), (2,2), (5,1)]	Multiple points	[(5,1)]
TC5	[(1,1), (4,1), (3,2)]	Multiple points	[(4,1)]

Lab Execution (how to perform the exercises):

1. Aftergeneratingthecontrolflowgraph,checkwhetheryourCFGmatchwith 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 CFG?
CFG Factory Tool	Yes
Eclipse flow graph generator	Yes

2. Deviseminimumnumberoftestcasesrequiredtocoverthecode using the aforementioned criteria.

Test Case	InputVector	Description	Expected Output
TC1	[]	EmptyVector	Handle carefully
TC2	[(3,4)]	One point (Single iteration of first loop)	[(3,4)]
TC3	[(1,2), (3,2)]	Two points with same Y coordinate (one iteration of 2nd loop)	[(3,2)]
TC4	[(3,1), (2,2), (5,1)]	Multiple points (1st loop runs twice with multiple outputs)	[(5,1)]
TC5	[(1,1), (4,1), (3,2)]	Multiple points (2nd loop runs twice (y=1))	[(4,1)]

3. Thispartoftheexerciseisverytrickyandinteresting.Thetestcasesthatyou 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 on TCs
Deletion	Delete the line that updates min for the minimum y-component	TCs like [(1,1), (2,0)] pass despite incorrect processing
Insertion	~~Insert an early return if the size of vector~~ pis1	[(3, 4)] pass without processing correctly
Modification	Change the comparison operation < to <=	TCs like [(1,1), (1,1), (1,2)] might pass while still failing in logic

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

Test Case	InputVector	Description	Expected Output
TC1	[]	EmptyVector	Handle carefully
TC2	[(3,4)]	One point (Single iteration of first loop)	[(3,4)]
TC3	[(1,2), (3,2)]	Two points with same Y coordinate (one iteration of 2nd loop)	[(3,2)]
TC4	[(3,1), (2,2), (5,1)]	Multiple points (1st loop runs twice with multiple outputs)	[(5,1)]
TC5	[(1,1), (4,1), (3,2)]	Multiple points (2nd loop runs twice (y=1))	[(4,1)]
TC6	[(2,2),(2,3),(2,1)]	Multiple points with same x-coordinate; check min y	[(2,1)]
TC7	[(0, 0), (1, 1), (1, 0), (0, 1)]	Multiple points in a rectangle; checks multiple comparisons.	[(1,0)]
TC8	[(3, 1), (2, 1), (1, 2)]	Multiple points with some ties; checks the max x among min y points.	[(3, 1)]
TC9	[(4, 4), (4, 3), (4, 5), (5, 4)]	Points with the same x-coordinate; checks formax y.	[(5,4)]