IT314
Software Engineering

Lab 9

Name: Jainam Patel
Student Id: 202201514

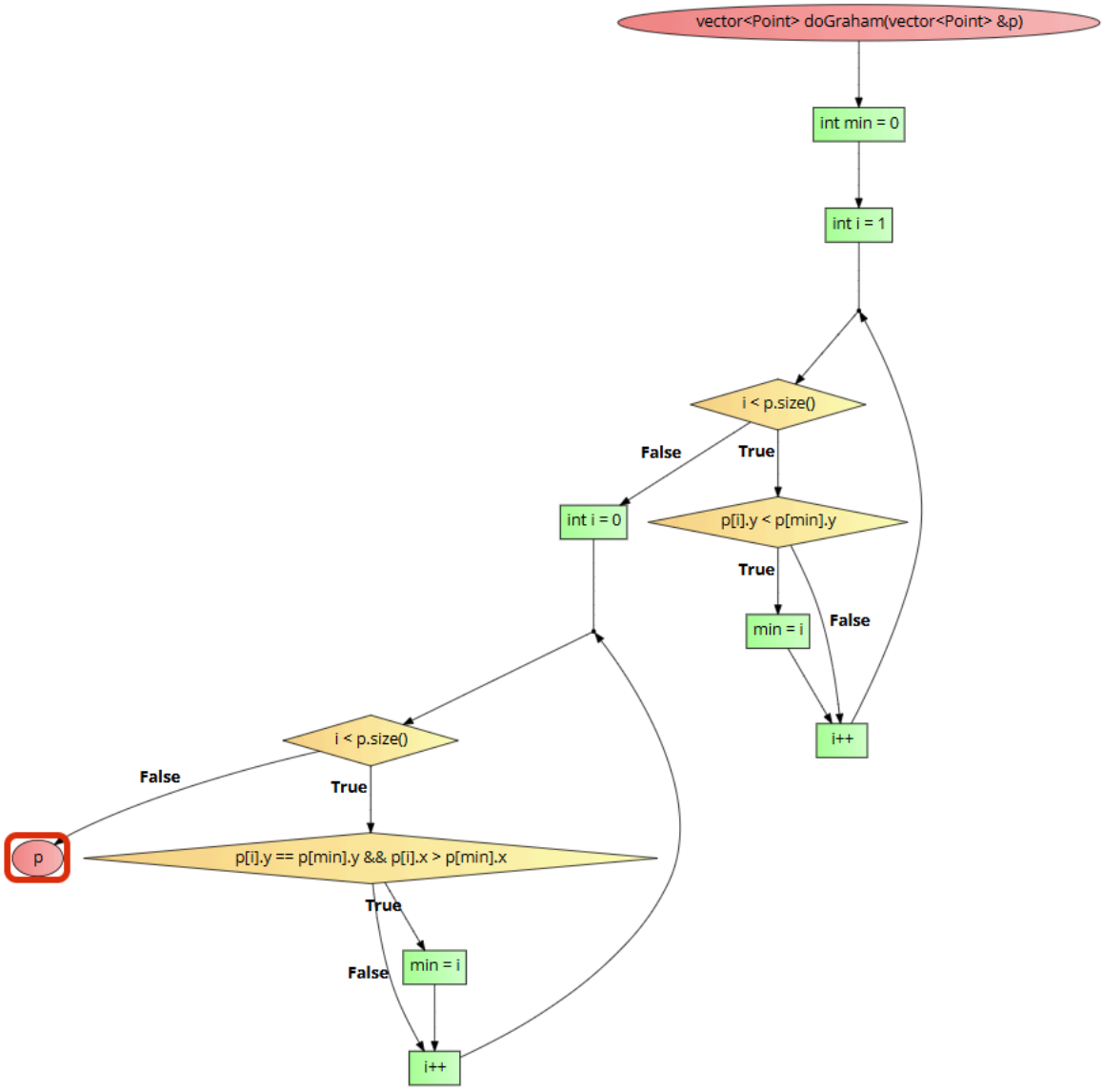
Q.1. The code below is part of a method in the ConvexHull class in the VMAP system. The following is a small fragment of a method in the ConvexHull class. For the purposes of this exercise, you do not need to know the intended function of the method. The parameter p is a Vector of Point objects, p.size()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

Control Flow Graph:



C++ Code of the Given Pseudocode:

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

class Point {
public:
    int x, y;

    Point(int x = 0, int y = 0) : x(x), y(y) {}
};

vector<Point> doGraham(vector<Point> &p) {
    int min = 0;

    for (int i = 1; i < p.size(); ++i) {
        if (p[i].y < p[min].y) {
            min = i;
        }
    }

    for (int i = 0; i < p.size(); ++i) {
        if (p[i].y == p[min].y && p[i].x > p[min].x) {
            min = i;
        }
    }

    return p;
}

int main() {
    int n;
    cout << "Enter the number of points: ";
    cin >> n;

    vector<Point> points;
    cout << "Enter the points (x y):\n";
    for (int i = 0; i < n; ++i) {
        int x, y;
        cin >> x >> y;
```

```
        points.push_back(Point(x, y));
    }

    points = doGraham(points);

    cout << "Processed points:\n";
    for (const auto &point : points) {
        cout << "(" << point.x << ", " << point.y << ")\n";
    }

    return 0;
}
```

---

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.

a. Statement Coverage

Test Cases for Statement Coverage:

1. **Test Case 1**: One point only. This ensures that basic execution occurs without entering loops multiple times.
    - **Input:** Points = (0,0)
    - **Expected Output:** (0,0)
2. **Test Case 2**: Multiple points with distinct y-coordinates.
    - **Input:** Points = (1,2),(3,4),(0,5)
    - **Expected Output:** (1,2)
3. **Test Case 3**: Multiple points with the same y-coordinate.
    - **Input:** Points = (1,2),(3,2),(0,2)

    - **Expected Output:** Since all points have the same y, we select the one with largest x, (3,1)

These cases ensure that all statements are executed.

b. Branch Coverage

Test Cases for Branch Coverage:

1. **Test Case 1**: Multiple points with distinct y-coordinates, ensuring the if condition in the first loop evaluates both True and False.
    - **Input:** Points = (2,3),(1,1),(3,4)
    - **Expected Outcome:** Minimum point is (1,1)
2. **Test Case 2**: Points with the same y-coordinate and different x-coordinates, to cover the second if condition in the second loop.
    - **Input:** Points = (2,1),(3,1),(1,5)
    - **Expected Outcome:** Minimum y-coordinate point is (2,1) but since both have the same y, we select the largest x, (3,1)
3. **Test Case 3**: One point only, ensuring both loops exit immediately.
    - **Input:** Points = (0,0)
    - **Expected Outcome:** Only point (0,0)

These cases test all branches by taking both True and False paths in each decision.

c. Basic Condition Coverage

This requires each condition in a decision to be tested both as True and False independently. In this code, the conditions are:

- p[i].y < p[min].y in the first loop.
- p[i].y == p[min].y and p[i].x > p[min].x in the second loop.

Test Cases for Basic Condition Coverage:

1. **Test Case 1**: All points have different y-coordinates, so p[i].y < p[min].y evaluates both True and False for different points.
    - **Input:** Points = (4,2),(1,3),(2,1)

    - **Expected Outcome:** Minimum point is (2,1)
2. **Test Case 2**: Multiple points with the same y-coordinate and varying x-coordinates to evaluate p[i].y == p[min].y and p[i].x > p[min].x conditions independently.
    - **Input:** Points = (1,2),(3,2),(2,2)
    - **Expected Outcome:** The minimum point based on y-coordinate is initially (1,2) but since (3,2) has the same y-coordinate and a larger x-coordinate, it is chosen.
3. **Test Case 3**: All points have the same y-coordinate and the same x-coordinate, so p[i].y == p[min].y evaluate True and p[i].x > p[min].x evaluate as False.
    - **Input:** Points = (1,1),(1,1),(1,1)
    - **Expected Outcome:** Any point can be selected since they are all the same.
4. **Test Case 4**: Points have varying x-coordinates with the same y-coordinate and ensure p[i].x > p[min].x is both True and False.
    - **Input:** Points = (1,1),(3,1),(2,1)
    - **Expected Outcome:** Since all have the same y-coordinate, the point with the largest x is selected: (3,1)

---

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.

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

class Point {
public:
    int x, y;

    Point(int x = 0, int y = 0) : x(x), y(y) {}
```

```
};

vector<Point> doGraham(vector<Point> &p) {
    int min = 0;

    // Deleted the first loop

    for (int i = 0; i < p.size(); ++i) {
        if (p[i].y == p[min].y && p[i].x > p[min].x) {
            min = i;
        }
    }

    return p;
}

int main() {
    int n;
    cout << "Enter the number of points: ";
    cin >> n;

    vector<Point> points;
    cout << "Enter the points (x y):\n";
    for (int i = 0; i < n; ++i) {
        int x, y;
        cin >> x >> y;
        points.push_back(Point(x, y));
    }

    points = doGraham(points);

    cout << "Processed points:\n";
    for (const auto &point : points) {
        cout << "(" << point.x << ", " << point.y << ")\n";
    }

    return 0;
}
```

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 for Path Coverage

Test Case 1: Zero Iterations for Both Loops

- **Description**: No points are provided, so both loops have zero iterations.
- **Input**: Points = []
- **Expected Output**: Since there are no points, doGraham should return an empty vector or handle this differently (depending on implementation).

Test Case 2: Zero Iterations for the First Loop, One Iteration for Second Loop

- **Description**: Only one point is provided, so only the second loop runs once.
- **Input**: Points = (0,0)
- **Expected Output**: Returns the same single point, (0,0)

Test Case 3: One Iteration for the First Loop, Two Iterations for the Second Loop

- **Description**: Two points are provided with different y-coordinates, so the first loop iterates once to find the minimum, and the second loop runs twice.
- **Input**: Points = (1,1),(2,3)
- **Expected Output**: The minimum y-coordinate point is (1,1).

Test Case 4: Two Iterations for the First Loop, Three Iterations for the Second Loop

- **Description**: Three points are provided so that the first loop iterates twice and the second loop iterates thrice.
- **Input**: Points = (2,3),(1,1),(3,1)
- **Expected Output**: The minimum y-coordinate point is initially (1,1), but since (3,1) has the same y and a larger x, (3,1) should be selected.

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 matches 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).

Control Flow Graph Factory Tool - **Yes**
Eclipse flow graph generator - **Yes**

---

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

Test Case	Description	Input	Coverage	Expected Output
1	Zero iterations for both loops	[]	Path Coverage (zero iterations)	[]
2	One iteration for second loop	(0,0)	Branch, Basic Condition, Path Coverage	(0,0)
3	One iteration for the first loop, two for the second	(1,1),(2,3)	Statement, Branch, Basic Condition, Path Coverage	(1,1)
4	One iteration for the first loop, two for the second	(1,1),(3,1)	Statement, Branch, Basic Condition, Path Coverage	(3,1)
5	Two iterations for the first loop, three for the second	(2,3),(1,1),(3,1)	Statement, Branch, Basic Condition, Path Coverage	(3,1)

---

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.

Deleting the Code

```
for (int i = 1; i < p.size(); ++i) {
    if (p[i].y < p[min].y) {
        min = i;
    }
}
```

Inserting the Code

```
for (int i = 0; i < p.size(); ++i) {
    if (p[i].y == p[min].y && p[i].x > p[min].x) {
        min = i;
    }

    if (true) {
        min = (min + 1) % p.size();
    }
}
```

Modification of the Code

```
for (int i = 1; i < p.size(); ++i) {
    if (p[i].y <= p[min].y) {
        min = i;
    }
}
```

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

Test Case	Input Points	Expected Output
1	(1, 1), (2, 2), (3, 0), (4, 4)	(3, 0)
2	(1, 2), (2, 2), (3, 2), (4, 1)	(4, 1)
3	(1, 2), (2, 2), (3, 2), (4, 2)	(4, 2)
4	(0, 5), (5, 5), (3, 4), (2, 1), (4, 2)	(2, 1)
5	(1, 1), (1, 1), (2, 2), (0, 0), (3, 3)	(0, 0)