

Task 4 Report

Task 1

Prompt strategy

I use 3 prompts in a chain of thoughts:

I: Please generate a failing unit test for the following function:

```
public boolean hasOneTrue(boolean[] array) {  
    if (array == null) {  
        return false;  
    }  
    int trueCount = -1;  
    for (int i = 0; i < array.length; i++) {  
        if (array[i]) {  
            if (trueCount < 1) {  
                trueCount++;  
            } else {  
                return false;  
            }  
        }  
    }  
    return trueCount > 0;  
}
```

Make sure that you understand each line of the code, your generated test is really failing and your assertion is correct.

AI: Here's a failing unit test for the `hasOneTrue` function:

```
@Test  
public void testHasOneTrue() {  
    boolean[] array = {true, false, true};  
    boolean result = hasOneTrue(array);  
    assertFalse(result);  
}
```

The expected result is `false` because there are two elements with a value of `true` in the array, but the function returns `true`.

I: Please generate four more failing tests for the same function. Make sure that you understand each line of the code, your generated tests are really failing and your assertions are correct.

AI: Here are four more failing unit tests for the `hasOneTrue` function:

```
@Test
public void testHasOneTrue_NullArray() {
    boolean[] array = null;
    boolean result = hasOneTrue(array);
    assertFalse(result);
}

@Test
public void testHasOneTrue_EmptyArray() {
    boolean[] array = {};
    boolean result = hasOneTrue(array);
    assertFalse(result);
}

@Test
public void testHasOneTrue_MultipleTrueValues() {
    boolean[] array = {true, false, true, true};
    boolean result = hasOneTrue(array);
    assertFalse(result);
}

@Test
public void testHasOneTrue_AllFalseValues() {
    boolean[] array = {false, false, false};
    boolean result = hasOneTrue(array);
    assertFalse(result);
}
```

In the first test, we pass a `null` array to the function, which should return `false`.

In the second test, we pass an empty array to the function, which should also return `false`.

In the third test, we pass an array with multiple true values, which should return `false`.

In the fourth test, we pass an array with all `false` values, which should also return `false`.

I: Please read through the function again and review your generated tests. Replace the wrong ones with new tests that are really failing and have correct assertions. Below is the function to test:

```
public boolean hasOneTrue(boolean[] array) {
    if (array == null) {
        return false;
    }
    int trueCount = -1;
    for (int i = 0; i < array.length; i++) {
        if (array[i]) {
            if (trueCount < 1) {
                trueCount++;
            }
        }
    }
    return trueCount == 1;
}
```

```

        } else {
            return false;
        }
    }
}
return trueCount > 0;
}

```

AI: Here are new failing unit tests for the `hasOneTrue` function:

```

@Test
public void testHasOneTrue_TwoTrueValues() {
    boolean[] array = {true, false, true};
    boolean result = hasOneTrue(array);
    assertFalse(result);
}

@Test
public void testHasOneTrue_AllFalseValues() {
    boolean[] array = {false, false, false};
    boolean result = hasOneTrue(array);
    assertFalse(result);
}

@Test
public void testHasOneTrue_OnlyTrueValue() {
    boolean[] array = {true};
    boolean result = hasOneTrue(array);
    assertTrue(result);
}

@Test
public void testHasOneTrue_MultipleTrueValues() {
    boolean[] array = {true, true, false, true};
    boolean result = hasOneTrue(array);
    assertFalse(result);
}

```

In the first test, we pass an array with two `true` values, which should return `false`.

In the second test, we pass an array with all `false` values, which should also return `false`.

In the third test, we pass an array with only one `true` value, which should return `true`.

In the fourth test, we pass an array with multiple `true` values, which should also return `false`.

Findings

1. HKUST GPT3.5 is effective for test generation if clear guidance is given. For example, after I emphasize on "failing" and "correct assertion", the generated tests become more usable.

2. GPT can forget previous information, so reminding it of the method under test can be an effective way to improve the response quality.
3. Asking GPT to review its generated tests can also enhance the correctness.
4. GPT still makes mistakes and have difficulty understand every possible branch of the code.
5. GPT generates 5 tests as demanded, all of which are compilable. Only `testHasOneTrue_AllFalseValues` is not failing while others can satisfy the requirements.

Advice

1. Developers should emphasize their requirements as hard as possible in their prompts.
2. Developers should remind LLMs of previous information in consequent prompts.
3. Developers should ask LLMs to review their generated tests.

Task 2

Prompt Strategy

To get the statement coverage of a given test on a given function, I only use one prompt. Another prompt is added for additional analysis.

I: What is the statement coverage of applying the following test `test052` on function `hasOneTrue`? Note that in the context of code coverage, a statement is typically defined as a single executable line of code. Code:

```
/**
 * Determines if the given boolean array has one element with a value of {@code true}.
 *
 * @param array the array of boolean values
 * @return true if there is exactly one true value in the array, false otherwise
 */
public boolean hasOneTrue(boolean[] array) {
    if (array == null) {
        return false;
    }
    int trueCount = -1;
    for (int i = 0; i < array.length; i++) {
        if (array[i]) {
            if (trueCount < 1) {
                trueCount++;
            } else {
                return false;
            }
        }
    }
    return trueCount > 0;
}

@Test
public void test052() throws Throwable {
```

```

        if (debug)
            System.out.format("%n%s%n", "Regression_0_Test0.test052");
        comp5111.assignment.cut.Subject subject0 = new
comp5111.assignment.cut.Subject();
        comp5111.assignment.cut.Subject.ParameterHandler parameterHandler1 =
subject0.new ParameterHandler();
        boolean[] booleanArray7 = new boolean[] { true, false, true, false, false };
        boolean boolean8 = parameterHandler1.hasOneTrue(booleanArray7);
        java.lang.Boolean boolean13 = parameterHandler1.toBooleanObject("", "", "",
""");
        org.junit.Assert.assertNotNull(booleanArray7);
        org.junit.Assert.assertEquals(java.util.Arrays.toString(booleanArray7), "[true, false, true, false, false]");
        org.junit.Assert.assertTrue("'" + boolean8 + "' != '" + true + "'", boolean8
== true);
        org.junit.Assert.assertEquals("'" + boolean13 + "' != '" + true + "'",
boolean13, true);
    }

```

AI: The test case `test052` executes 6 statements out of 10 statements in the `hasOneTrue` method. Therefore, the statement coverage is 60%.

I: Which statements are not covered?

AI: The following statements are not covered by the test case `test052`:

- Line 6: `if (array == null) {`
- Line 8: `int trueCount = -1;`
- Line 10: `for (int i = 0; i < array.length; i++) {`
- Line 12: `if (trueCount < 1) {`
- Line 14: `return false;`
- Line 16: `return trueCount > 0;`

Finding

1. In this specific case, ChatGPT cannot count the number of statements and calculate the coverage correctly.
2. Definition of "statement" in the context of testing is needed, otherwise GPT considers the entire "if" clause as a single statement.

Task 3

Prompt Strategy

I use 2 prompts in a chain of thoughts.

I: What is the branch coverage of applying the following test `test052` on function `hasOneTrue`? In the context of testing, a branch is one of the possible execution paths the code can take after a decision statement—e.g., an if statement—gets evaluated. This means that an if statement creates

two branches, one where the condition is satisfied and another otherwise. First extract the test input from test052, then list each branch and whether it is covered. Code:

```
/**
 * Determines if the given boolean array has one element with a value of {@code true}.
 *
 * @param array the array of boolean values
 * @return true if there is exactly one true value in the array, false otherwise
 */
public boolean hasOneTrue(boolean[] array) {
    if (array == null) {
        return false;
    }
    int trueCount = -1;
    for (int i = 0; i < array.length; i++) {
        if (array[i]) {
            if (trueCount < 1) {
                trueCount++;
            } else {
                return false;
            }
        }
    }
    return trueCount > 0;
}

@Test
public void test052() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "Regression_0_Test0.test052");
    comp5111.assignment.cut.Subject subject0 = new
comp5111.assignment.cut.Subject();
    comp5111.assignment.cut.Subject.ParameterHandler parameterHandler1 =
subject0.new ParameterHandler();
    boolean[] booleanArray7 = new boolean[] { true, false, true, false, false };
    boolean boolean8 = parameterHandler1.hasOneTrue(booleanArray7);
    java.lang.Boolean boolean13 = parameterHandler1.toBooleanObject("", "", "",
    "");
    org.junit.Assert.assertNotNull(booleanArray7);
    org.junit.Assert.assertEquals(java.util.Arrays.toString(booleanArray7), "[true, false, true, false, false]");
    org.junit.Assert.assertTrue("'" + boolean8 + "' != '" + true + "'", boolean8
    == true);
    org.junit.Assert.assertEquals("'" + boolean13 + "' != '" + true + "'",
    boolean13, true);
}
```

AI: The input of test052 is a boolean array with values [true, false, true, false, false].

The first branch is covered when `array == null`, which is not the case in this test, so it is not covered.

The second branch is covered when `array != null`. In this case, the loop is executed and there are two possible branches:

- The first branch is covered when `array[i]` is `false`, which happens twice in this test.
- The second branch is covered when `array[i]` is `true`. This branch has two sub-branches:
The first sub-branch is covered when `trueCount < 1`, which is true the first time true is found in the array.
The second sub-branch is covered when `trueCount >= 1`, which is not covered in this test because there is only one true value in the array.

Therefore, the branch coverage of this test on `hasOneTrue` is 3/5 or 60%.

I: How can we modify `test052` to improve the coverage? Please give an example and the coverage after the modification.

AI: To improve the coverage of `hasOneTrue`, we can modify `test052` to include a case where there is more than one true value in the array. Here's an example:

```
@Test
public void test052() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "Regression_0_Test0.test052");

    // Test case with one true value
    comp5111.assignment.cut.Subject subject0 = new comp5111.assignment.cut.Subject();
    comp5111.assignment.cut.Subject.ParameterHandler parameterHandler1 = subject0.new
ParameterHandler();
    boolean[] booleanArray1 = new boolean[] { true, false, true, false, false };
    boolean boolean2 = parameterHandler1.hasOneTrue(booleanArray1);
    assertTrue(boolean2);

    // Test case with multiple true values
    boolean[] booleanArray2 = new boolean[] { true, false, true, true, false };
    boolean boolean3 = parameterHandler1.hasOneTrue(booleanArray2);
    assertFalse(boolean3);

    java.lang.Boolean boolean4 = parameterHandler1.toBooleanObject("", "", "", "");
    assertNotNull(booleanArray1);
    assertEquals(java.util.Arrays.toString(booleanArray1), "[true, false, true, false,
false]");
    assertTrue("'" + boolean2 + "' != '" + true + "'", boolean2 == true);
    assertFalse("'" + boolean3 + "' != '" + false + "'", boolean3 == true);
    assertEquals("'" + boolean4 + "' != '" + true + "'", boolean4, true);
}
```

In this modified test, we added a second test case with an array that has two true values. This allows us to cover the second sub-branch of the true case in the loop.

With this modification, the branch coverage of `hasOneTrue` becomes 4/5 or 80%.

Finding

- Similar to statement coverage, here the specific definition of "branch" is also needed, or GPT considers an "if" statement as a branch.
- The analysis is mostly correct, but GPT strangely misunderstands the test input even though it has been printed out.
- The modified test improves the coverage if its analysis is correct, but this is not the case in reality.

Advice

- 1. Developers should include detailed definition of concepts in their context to ensure the LLM understand them correctly.
- 2. Developers should ensure that the LLM gives correct results on all test requirements, not only relying on a single number of coverage.