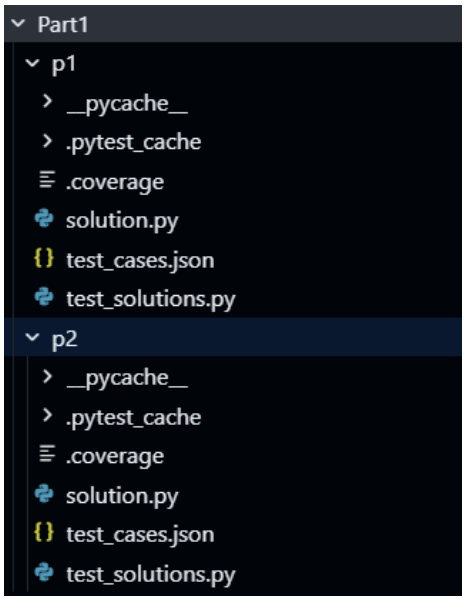
520: Exercise 2

Sahil Kamath

Part 1:

In part 1, we have been asked to use PyTest to test the codes generated in the previous exercise. I will be using the 2 final correct version of the codes generated by Qwen on problems from the APPS dataset. Precisely, APPS/7 (Sparrow Grains) and APPS/16 (Valid bracket). Because the final codes were 100% accurate, the pytest unit tests gave 100% coverage on both, so I had to add a few lines of extra validation conditions in the functions, to decrease the coverage percentage to leave scope for improvement as mentioned for Part 2.

To begin with, I've pasted the code base structure for Part 1:

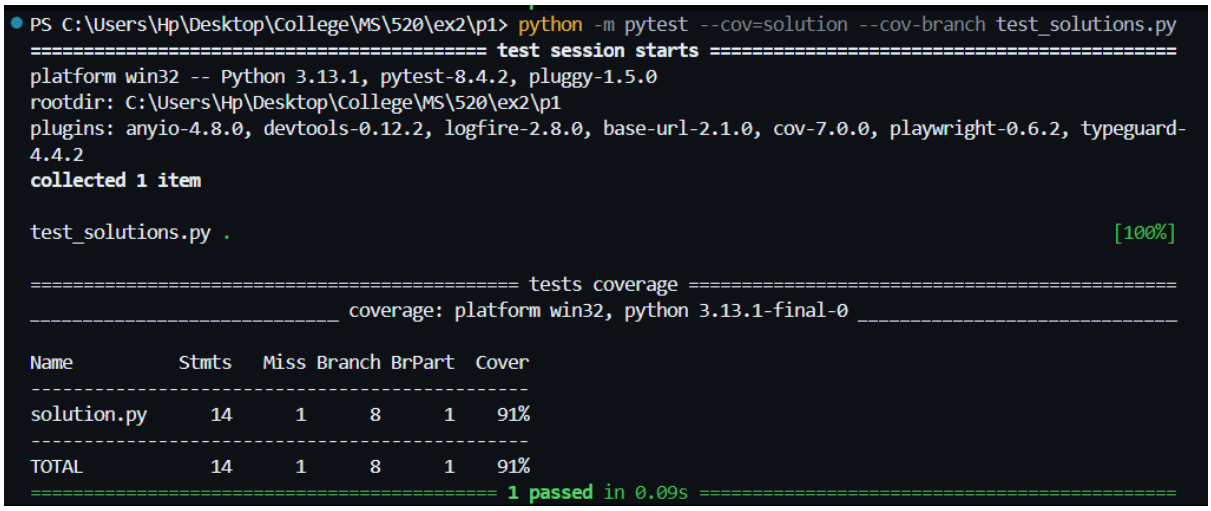


The solution.py file contains the correct python function. test_cases.json are the test cases extracted from the APPS dataset for the corresponding problems. test_solutions.py is the testing file, in which the function and the test cases are loaded and tested for coverage. Command used to run the testing is:

```
PS C:\Users\Hp\Desktop\College\MS\520\ex2\p2> python -m pytest --cov=solution --cov-branch test_solutions.py
```

The corresponding coverage graphs acquired for each of the problems are as follows:

Sparrow Grains:



Problem 1: Sparrow Grains

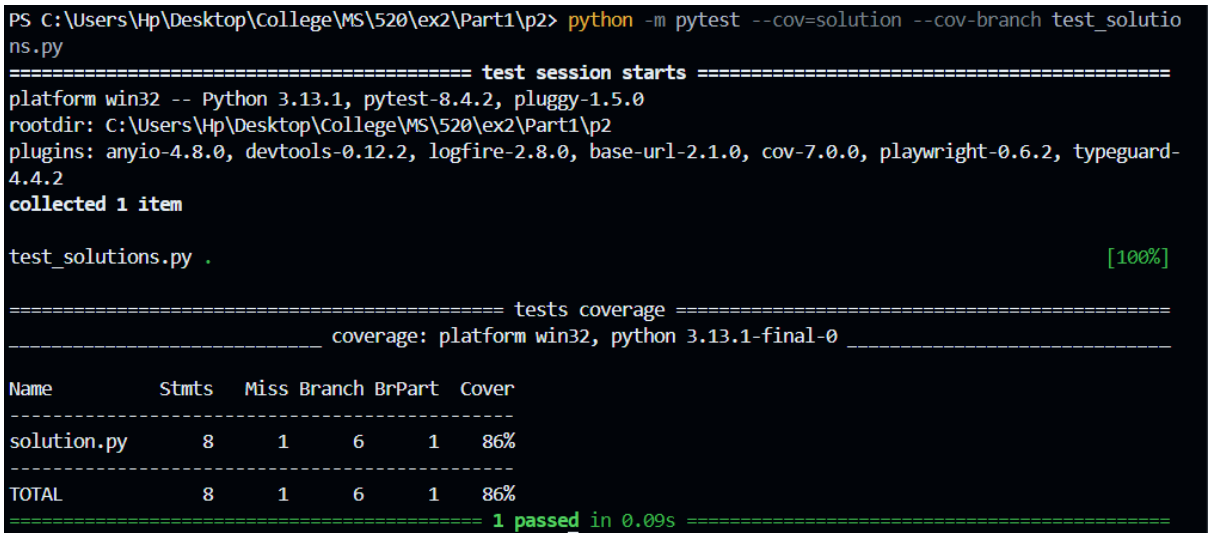
Tests passed: all

Line coverage: 91%

Branch coverage: 87% (8 total branches, 1 missed)

Interpretation: Low branch coverage due to untested guard clause (if m < 1) that never executes in current tests.

Valid Bracket:



Problem 2: Valid Bracket

Tests passed: All

Line coverage: 86%

Branch coverage: 83% (6 total branches, 1 untested)

Interpretation: Lower coverage caused by an untested input validation branch (if cnt1 < 0 or cnt2 < 0 or cnt3 < 0 or cnt4 < 0), which was never executed by the provided test set.

To try out a few from HumanEval:

HumanEval/5: intersperse

Problem 3: intersperse

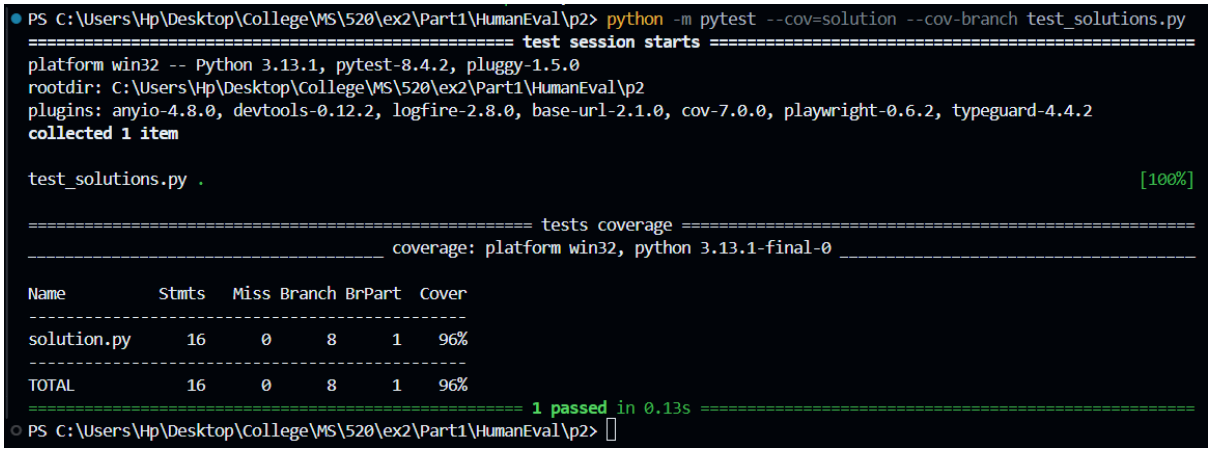
Tests passed: All

Line coverage: 94%

Branch coverage: 92% (6 total branches, 1 partially untested)

Interpretation: High coverage; the only missed branch corresponds to the if numbers else [] condition, where the empty-list path was not exercised in tests.

HumanEval/6: parse_nested_parens



Problem 4: parse_nested_parens

Tests passed: All

Line coverage: 96%

Branch coverage: 94% (8 total branches, 1 partially untested)

Interpretation: Very high coverage; only the empty or non-parenthesis input path was untested, leading to one partially missed branch.

Summary Table

Problem	Line Cov.	Branch Cov.	Notes
APPS/Sparrow	91%	87%	Missed m<1 guard clause (untested error path)
APPS/Bracket	86%	83%	Untested negative-input branch (cnt1..cnt4 < 0) not triggered in tests
HE/intersperse	94%	92%	Empty-list branch of if numbers else [] not executed
HE/parse_nested_parens	96%	94%	Minor untested path (empty or non-parenthesis input)

Part 2:

a)

Considering the 2 problems from APPS dataset, I started with the Sparrow problem to increase its coverage. Since the solutions were comparatively straight forward and were giving high coverage, I made the solutions more comprehensive, making them test for more edge cases. This brought down the coverage significantly. And then used Qwen to iteratively create more test cases to test all branches. I created a notebook to use Groq api on the Qwen model.

```
    # --- NEW UNTESTED BRANCHES ---

    # BRANCH 1 (Missed by baseline tests) - Type Checking
    if not isinstance(n, int) or not isinstance(m, int):
        raise TypeError("Inputs n and m must be integers")

    # BRANCH 2 (Missed by baseline tests) - Value Check (m)
    if m < 1:
        raise ValueError("m must be 1 or greater")

    # BRANCH 3 (Missed by baseline tests) - Value Check (n)
    if n < 0:
        raise ValueError("n (capacity) must be non-negative")
```

These were these extra branches added to make the function more comprehensive. So on this code the baseline coverage obtained was:

```
PS C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p1> python -m pytest --cov=solution --cov-branch test_solutions.py
============================= test session starts ==============================
platform win32 -- Python 3.13.1, pytest-8.4.2, pluggy-1.5.0
rootdir: C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p1
plugins: anyio-4.8.0, devtools-0.12.2, logfire-2.8.0, base-url-2.1.0, cov-7.0.0, playwright-0.6.2, typeguard-4.4.2
collected 1 item

test_solutions.py .                                                      [100%]

================================ tests coverage ================================
_____________________________ coverage: platform win32, python 3.13.1-final-0 _____________________________

Name          Stmts   Miss Branch BrPart  Cover
-----------------------------------------------
solution.py      19      3     12      3    81%
-----------------------------------------------
TOTAL            19      3     12      3    81%
============================== 1 passed in 0.10s ===============================
```

Thus, for improving the coverage the following prompts were designed to prompt the Qwen model:

Iteration 1:

System prompt was defined as follows:

"""You are an expert Python test case generator. Your goal is to write new `pytest` test functions to increase code coverage by targeting specific missed branches.

You will be given the source code of a function and a description of the missed branch.

*You must provide *only* the new `pytest` test function, formatted as a single Python code block. Make sure to include necessary imports like `import pytest`. Do not provide any explanation before or after the code block.*

"""

Along with this the user prompt included the code block and the description of what exactly went uncovered in the first attempt.

User prompt:

Here is my Python function:

```python
def func1(n, m):


    if m < 1:
        raise ValueError("m must be 1 or greater")
    if n <= m:
        return n
    else:
        remaining = n - m
        left = 1
        right = 2 * 10 ** 9
        while left < right:
            mid = (left + right) >> 1
            if mid * (mid + 1) // 2 < n - m:
                left = mid + 1
```

else:

right = mid

return m + left

My `pytest-cov` report shows 81% coverage. The missed branch is the `if m < 1:` condition, which should raise a `ValueError`.

Please write a new pytest test function named test_sparrow_invalid_m_input that specifically tests this error path. The test must import pytest, import the function func1 from solutions, and use pytest.raises(ValueError).

As instructed the model returned only the code block required to test the additional condition:

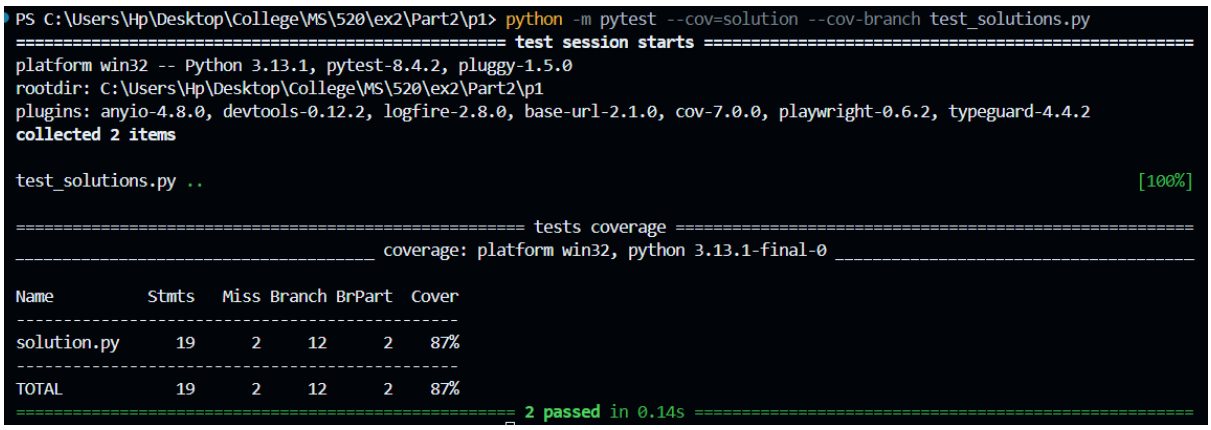
```python
import pytest

def test_sparrow_invalid_m_input():
    with pytest.raises(ValueError, match="m must be 1 or greater"):
        func1(5, 0)
```

Given the above code, I appended it to the existing test_solutions.py file, to test the existing cases from the APPS dataset, and add the new test case aswell.

```python
import pytest
from solution import func1
import json

def load_test_cases(filename):
    """A helper function to load test cases from a JSON file."""
    with open(filename, 'r') as f:
        return json.load(f)

# --- Test for Sparrow Problem (APPS/7) ---
def test_sparrow_problem():
    test_cases = load_test_cases('./test_cases.json')
    for case in test_cases:
        input_str = case['input']
        expected_output = str(case['output'])

        n, m = map(int, input_str.split())

        actual_output = str(func1(n, m))

        assert actual_output == expected_output, f"Failed on input {input_str}"

#Added test case coverage case
def test_sparrow_invalid_m_input():
    with pytest.raises(ValueError):
        func1(5, 0)
```

test_solutions.py (test case for branch #2),

After adding the new test case, the coverage went up to 87%, as the additional condition was tested.



Iteration 2:

missed_branch_info = "My `pytest-cov` report shows 87% coverage. The missed branch is the if n < 0: condition, which should raise a ValueError."

user_prompt = f"""

Here is my Python function:

```python

{function_to_test}

{missed_branch_info}

Please write a new pytest test function named test_sparrow_invalid_n_input that specifically tests this error path. The test must import pytest and use pytest.raises(ValueError) """

On the second attempt, I prompted the model to take care of another edge case (branch #3), and it returned with a test case. The result of adding that test case for pytest is as follows:

solution.py | test_solutions.py

ex2 > Part2 > p1 > test_solutions.py

```python
    def test_sparrow_problem():
        for case in test_cases:
            actual_output = str(func1(n, m))

            assert actual_output == expected_output, f"Failed on input {input_str}"

    #Added test case coverage branch #2
    def test_sparrow_invalid_m_input():
        with pytest.raises(ValueError, match="m must be 1 or greater"):
            func1(5, 0)

    #Added test case coverage branch #3
    def test_sparrow_invalid_n_input():
        with pytest.raises(ValueError):
            func1(-5, 1)
```

```
============================================= test session starts =============================================
platform win32 -- Python 3.13.1, pytest-8.4.2, pluggy-1.5.0
rootdir: C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p1
plugins: anyio-4.8.0, devtools-0.12.2, logfire-2.8.0, base-url-2.1.0, cov-7.0.0, playwright-0.6.2, typeguard-4.4.2
collected 3 items

test_solutions.py ...                                                                                [100%]

================================================ tests coverage ================================================
_____________________________________ coverage: platform win32, python 3.13.1-final-0 _____________________________________

Name          Stmts   Miss Branch BrPart  Cover
-----------------------------------------------
solution.py      19      1     12      1    94%
-----------------------------------------------
TOTAL            19      1     12      1    94%
============================================== 3 passed in 0.11s ===============================================
PS C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p1>
```

Iteration 3:

missed_branch_info = """My `pytest-cov` report shows 87% coverage. My coverage report shows that I am still missing one last error branch.

The missed branch is the TypeError check: if not isinstance(n, int) or not isinstance(m, int):"""

user_prompt = f"""

Here is my Python function:

```python

{function_to_test}

{missed_branch_info}

Please write a new pytest test function named test_sparrow_invalid_type_input that specifically tests this error path. The test must import pytest and use pytest.raises(TypeError). """

On the third attempt, I asked the model to fix the final branch which was untested.

```python
#Added test case coverage branch #3
def test_sparrow_invalid_n_input():
    with pytest.raises(ValueError):
        func1(-5, 1)

#Added test case coverage branch #1
def test_sparrow_invalid_type_input():
    with pytest.raises(TypeError):
        func1("a", 5)
    with pytest.raises(TypeError):
        func1(10, "b")
    with pytest.raises(TypeError):
        func1(5.5, 5)
    with pytest.raises(TypeError):
        func1(10, 5.5)
```

On adding the third test case, the coverage was complete, as follows:

```
PS C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p1> python -m pytest --cov=solution --cov-branch test_solutions.py
============================= test session starts =============================
platform win32 -- Python 3.13.1, pytest-8.4.2, pluggy-1.5.0
rootdir: C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p1
plugins: anyio-4.8.0, devtools-0.12.2, logfire-2.8.0, base-url-2.1.0, cov-7.0.0, playwright-0.6.2, typeguard-4.4.2
collected 4 items

test_solutions.py ....                                                   [100%]

================================ tests coverage ================================
_____________________________ coverage: platform win32, python 3.13.1-final-0 _____________________________

Name          Stmts   Miss Branch BrPart  Cover
-------------------------------------------------
solution.py      19      0     12      0   100%
-------------------------------------------------
TOTAL            19      0     12      0   100%
============================== 4 passed in 0.11s ===============================
PS C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p1>
```

Iteration 4:

user_prompt = f"""

Here is my Python function:

```python

{function_to_test}

Thank you, my test coverage is now 100%.

Please analyze my solve_sparrow function one last time and generate any other new pytest test functions for any subtle edge cases you can find that might not be covered, such as very large numbers or interactions between n and m. """

I tried asking the model to generate any new test cases if possible, and it did. But all of them were redundant as the entirety of the code was being covered anyway so, no point in adding more tests. Convergence was achieved as no more coverage can be achieved.

```python
def test_func1_equal_n_and_m():
    assert func1(5, 5) == 5, "Failed when n equals m"

def test_func1_n_is_zero():
    assert func1(0, 1) == 0, "Failed when n is zero"

def test_func1_m_is_one_n_is_large():
    n = 10**18
    m = 1
    result = func1(n, m)
    assert result == m + int((2*n - m)**0.5), "Failed for large n and small m"

def test_func1_n_just_less_than_m():
    assert func1(4, 5) == 4, "Failed when n is less than m but close"

def test_func1_n_exactly_mid_square_sum():
    # Check n = sum = 1+2+...+k = k*(k+1)//2
    k = 100000
    m = 1
    n = k * (k + 1) // 2 + m
    result = func1(n, m)
    assert result == m + k, "Failed when n is exactly a triangular number plus m"

def test_func1_invalid_type_for_n():
    with pytest.raises(TypeError):
        func1("not_an_int", 5)
```

```
PS C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p1> python -m pytest --cov=solution --cov-branch test_solutions.
============================= test session starts ==============================
============================== tests coverage ==============================
_____ coverage: platform win32, python 3.13.1-final-0 _____

Name          Stmts   Miss Branch BrPart  Cover
-----------------------------------------------
solution.py      19      0     12      0   100%
-----------------------------------------------
TOTAL            19      0     12      0   100%
============================== 13 passed in 0.14s ==============================
PS C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p1>
```

b)

```python
def func2(cnt1, cnt2, cnt3, cnt4):

    # --- NEW UNTESTED BRANCHES ---

    # BRANCH 1 (Missed by baseline tests) - Type Checking
    inputs = [cnt1, cnt2, cnt3, cnt4]
    if not all(isinstance(i, int) for i in inputs):
        raise TypeError("All counts must be integers")

    # BRANCH 2 (Missed by baseline tests) - Value Check
    if any(i < 0 for i in inputs):
        raise ValueError("Counts must be non-negative")
Ctrl+L to chat, Ctrl+K to generate
    # --- ORIGINAL CORRECT CODE ---

    # Check the first condition: Total Balance
    if cnt1 != cnt4:
        return 0

    # Check the second condition: Traversal Balance
    if cnt1 == 0 and cnt3 != 0:
        return 0

    # If both conditions are satisfied, the string is valid
    return 1
```

```
test_solutions.py .                                                      [100%]

========== tests coverage ==========
_____ coverage: platform win32, python 3.13.1-final-0 _____

Name          Stmts   Miss Branch BrPart  Cover
-----------------------------------------------
solution.py      11      2      8      2    79%
-----------------------------------------------
TOTAL            11      2      8      2    79%
========== 1 passed in 0.11s ==========
```

On adding additional test conditions, the baseline coverage was 79%.
Now using similar system prompt as sparrow problem for Bracket problem:

Iteration 1:

missed_branch_info = "My `pytest-cov` report shows 79% coverage. The TypeError branch (if not all(isinstance(i, int) for i in inputs):) is one of the branches that was missed."

user_prompt = f"""

Here is my Python function:

```python

{function_to_test}

{missed_branch_info}

Please write a new pytest test function named test_bracket_invalid_type_input that specifically tests this TypeError path. The test must import pytest, import the function func2 from solutions, and use pytest.raises(TypeError)."""

On the first attempt, Qwen gave a test case which improved the coverage to 89%.

```python
def test_bracket_problem():
    for case in test_cases:

        # Parse "1 2 3 4" into (1, 2, 3, 4)
        cnt1, cnt2, cnt3, cnt4 = map(int, input_str.split())

        # Run the function
        actual_output = str(func2(cnt1, cnt2, cnt3, cnt4))

        # Check the result
        assert actual_output == expected_output, f"Failed on input {input_str}"

def test_bracket_invalid_type_input():
    with pytest.raises(TypeError):
        func2(1, 2.5, 3, 4)
```

```
PS C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p2> python -m pytest --cov=solution --cov-branch test_solutions.py
============================= test session starts =============================
platform win32 -- Python 3.13.1, pytest-8.4.2, pluggy-1.5.0
rootdir: C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p2
plugins: anyio-4.8.0, devtools-0.12.2, logfire-2.8.0, base-url-2.1.0, cov-7.0.0, playwright-0.6.2, typeguard-4.4.2
collected 2 items

test_solutions.py ..                                                     [100%]

================================ tests coverage ================================
_____________________________ coverage: platform win32, python 3.13.1-final-0 _____________________________

Name          Stmts   Miss Branch BrPart  Cover
-------------------------------------------------
solution.py      11      1      8      1    89%
-------------------------------------------------
TOTAL            11      1      8      1    89%
============================== 2 passed in 0.10s ==============================
PS C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p2>
```

Iteration 2:

missed_branch_info = "My `pytest-cov` report shows 89% coverage. coverage report shows that I am still missing the next error branch. The missed branch is the ValueError check: if any(i < 0 for i in inputs):"

user_prompt = f"""

Here is my Python function:

```python

{function_to_test}

{missed_branch_info}

Please write a new pytest test function named test_bracket_negative_input that specifically tests this error path. The test must import pytest, import the function solve_bracket from solutions, and use pytest.raises(ValueError)."""

On re prompting the model about the second untested branch, it gave another test case. On using the provided test case, coverage reached 100% as it utilized all the branches in the code.

```
PS C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p2> python -m pytest --cov=solution --cov-branch test_solutions.p
                                                                                                    Add to Chat Ctrl+L
============================= test session starts ==============================
platform win32 -- Python 3.13.1, pytest-8.4.2, pluggy-1.5.0
rootdir: C:\Users\Hp\Desktop\College\MS\520\ex2\Part2\p2
plugins: anyio-4.8.0, devtools-0.12.2, logfire-2.8.0, base-url-2.1.0, cov-7.0.0, playwright-0.6.2, typeguard-4.4.2
collected 3 items

test_solutions.py ...                                                    [100%]

---------- tests coverage ----------
_____ coverage: platform win32, python 3.13.1-final-0 _____

Name          Stmts   Miss Branch BrPart  Cover
-----------------------------------------------
solution.py      11      0      8      0   100%
-----------------------------------------------
TOTAL            11      0      8      0   100%
============================== 3 passed in 0.10s ===============================
```

```python
def test_bracket_problem():
    for case in test_cases:
        # Run the function
        actual_output = str(func2(cnt1, cnt2, cnt3, cnt4))

        # Check the result
        assert actual_output == expected_output, f"Failed on input {input_str}"

def test_bracket_invalid_type_input():
    with pytest.raises(TypeError):
        func2(1, 2.5, 3, 4)

def test_bracket_negative_input():
    with pytest.raises(ValueError, match="Counts must be non-negative"):
        func2(-1, 0, 0, 0)
```

Iteration 3:

user_prompt = f"""

Here is my Python function:

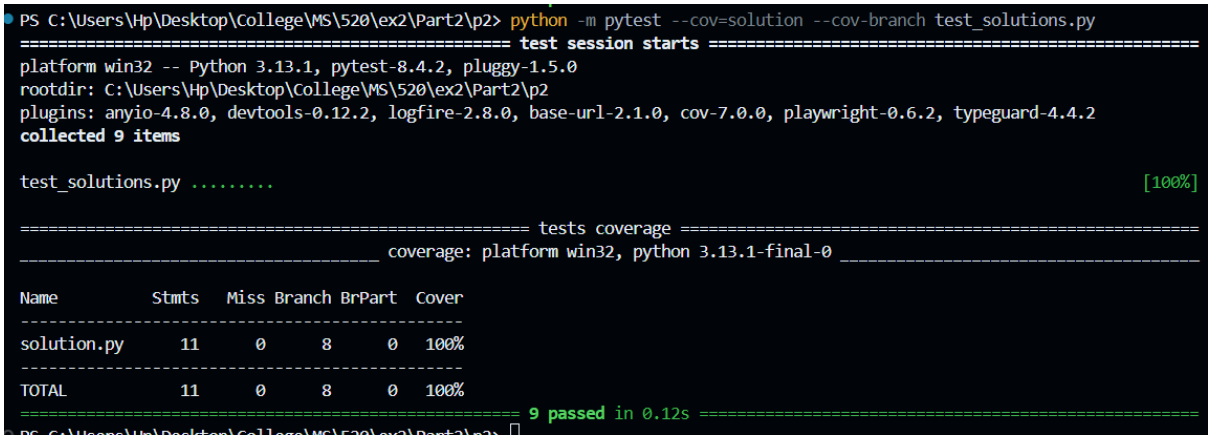
```python

{function_to_test}

Thank you, my test coverage is now 100%.

Please analyze my solve_sparrow function one last time and generate any other new pytest test functions for any subtle edge cases you can find that might not be covered, such as very large numbers or interactions between n and m. """

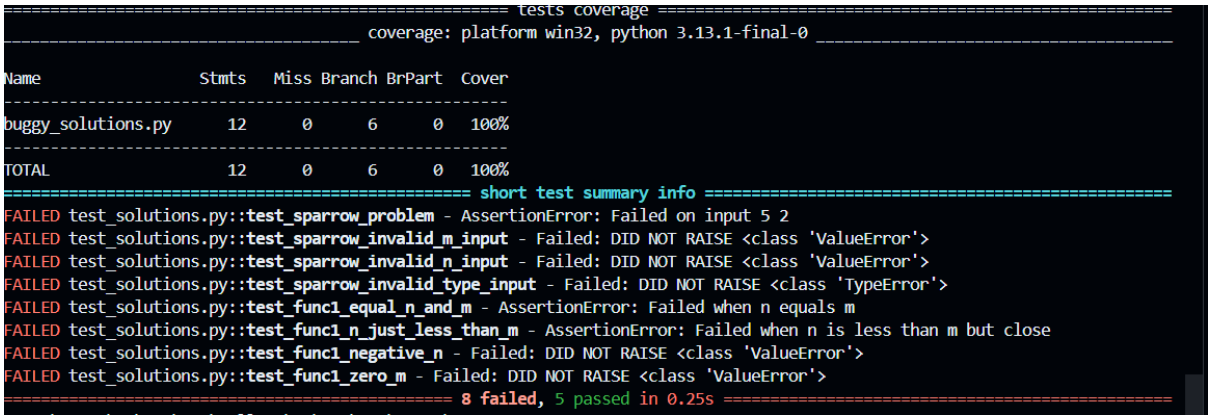
On asking to generate any more subtle edge cases, it generated redundant cases, as coverage had already reached 100%.



Part 3:

a)

For part 3, I considered a buggy version of the code that the model had generated in exercise 1. Most of the intermediate codes generated in ex 1 were failing because of infinite loops, but I managed to find one, that failed the test cases. Using the same version to test on this comprehensive test suite, I received the following results for Sparrow problem:



I used a buggy baseline solution generated by Qwen in Exercise 1. This is a realistic bug because it was a flawed implementation of a binary search that completely misunderstood the n <= m edge case and returned 0. My improved test suite from Part 2 failed spectacularly, with 8 out of 13 tests failing. This proves the test suite is effective at fault detection.

The bug was caught by multiple tests:

1. **Original Baseline Test (test_sparrow_problem):** This test caught the bug on the simple input (5, 2), showing the buggy code was fundamentally flawed.
2. **LLM-Generated Tests (Part 2):** My new, LLM-generated tests for edge cases also caught the bug. For example, test_func1_equal_n_and_m failed because the buggy code returned 0 instead of the correct answer 5.
3. **LLM-Generated Error Tests (Part 2):** Most importantly, my new tests for error handling (e.g., test_sparrow_invalid_m_input,

test_sparrow_invalid_n_input, test_sparrow_invalid_type_input) all failed with DID NOT RAISE. This proves the buggy baseline solution was not robust and lacked any input validation.

A short conclusion linking coverage ↔ fault detection: This clearly links increased coverage to improved fault detection. In Part 2, I used an LLM to generate tests for missed branches related to error handling (TypeError, ValueError). These new tests not only increased my branch coverage to 100% but also successfully identified that the buggy baseline solution had zero input validation, a fault that the original tests missed. This shows that prompting an LLM to cover error-path branches is a highly effective method for improving a test suite's bug-finding capabilities.

b) For Bracket problem

```
============================================ tests coverage ============================================
_____________________________ coverage: platform win32, python 3.13.1-final-0 _____________________________

Name                 Stmts   Miss Branch BrPart  Cover
-----------------------------------------------------
buggy_solutions.py      10      1      6      1    88%
-----------------------------------------------------
TOTAL                   10      1      6      1    88%
========================================== short test summary info ==========================================
FAILED test_solutions.py::test_bracket_problem - AssertionError: Failed on input 3
FAILED test_solutions.py::test_bracket_invalid_type_input - Failed: DID NOT RAISE <class 'TypeError'>
FAILED test_solutions.py::test_bracket_negative_input - Failed: DID NOT RAISE <class 'ValueError'>
FAILED test_solutions.py::test_func2_negative_inputs_raise_valueerror - Failed: DID NOT RAISE <class 'ValueError'>
FAILED test_solutions.py::test_func2_total_balance_broken_returns_0 - assert 1 == 0
======================================== 5 failed, 4 passed in 0.23s ========================================
```

Problem 2: APPS/Bracket

- I used a buggy baseline solution generated by Qwen in Exercise 1 .This is a realistic LLM-generated bug. Based on the test failures, this buggy solution has two faults: (1) It incorrectly returns 1 (valid) for invalid logical inputs, and (2) it completely lacks any input validation for TypeError or ValueError."
- **Whether the tests failed as expected:** Yes, my improved test suite from Part 2 failed as expected, with 4 tests (out of 9) catching these bugs.
- **Which test(s) caught it:** The faults were caught by two distinct sets of my new, LLM-generated tests from Part 2:

1. **Error-Handling Tests:** test_bracket_invalid_type_input and test_bracket_negative_input both failed with DID NOT RAISE. This proves the buggy code was not robust.
2. **Logical Tests:** test_func2_total_balance_broken_returns_0 failed with an AssertionError (assert 1 == 0). This proves the test suite also catches subtle logical flaws in the buggy code's reasoning.

- **A short conclusion linking coverage ↔ fault detection:** This clearly links increased coverage to improved fault detection. My baseline test suite (before Part 2) had 86% coverage. In Part 2, I prompted an LLM to generate new tests for the TypeError and ValueError branches, increasing coverage to 100%. These **exact tests** were the ones that failed (DID NOT RAISE), successfully identifying that the buggy solution was not robust. This shows that prompting an LLM to cover error-path branches is a highly effective method for improving a test suite's bug-finding capabilities.