Laboratorium 13 – Analiza i bazy danych

Test Driven Development(TDD)

Jakub Sacha, gr.1

1. Faza Red.

Na samym początku zaimplementowałem testy, które sprawdzają czy algorytm poprawnie tworzy oraz mnoży macierze.

```python
import pytest
import matrix

test_matrix_creation_from_dimensions_data = [
    (2, 3, 0, [[0, 0], [0, 0], [0, 0]]),
    (3, 3, 1, [[1, 1, 1], [1, 1, 1], [1, 1, 1]]),
    (4, 2, 3, [[3, 3, 3, 3], [3, 3, 3, 3]])
]


@pytest.mark.parametrize('m, n, value, expected_matrix_data', test_matrix_creation_from_dimensions_data)
def test_matrix_creation_from_dimensions(m, n, value, expected_matrix_data):
    test_matrix = matrix.Matrix((m, n), value)
    assert test_matrix.data == expected_matrix_data


test_matrix_creation_from_dimensions_data = [
    ([[0, 0], [0, 0], [0, 0]], [[0, 0], [0, 0], [0, 0]]),
    ([[1, 1, 1], [1, 1, 1], [1, 1, 1]], [[1, 1, 1], [1, 1, 1], [1, 1, 1]]),
    ([[3, 3, 3, 3], [3, 3, 3, 3]], [[3, 3, 3, 3], [3, 3, 3, 3]])
]


@pytest.mark.parametrize('input_matrix_data, expected_matrix_data', test_matrix_creation_from_dimensions_data)
def test_matrix_creation_from_data(input_matrix_data, expected_matrix_data):
    test_matrix = matrix.Matrix(input_matrix_data)
    assert test_matrix.data == expected_matrix_data


test_invalid_matrix_shape_data = [
    (-1, 5),
    (0, 0),
    (2, -5),
    (-2, -4)
]


@pytest.mark.parametrize('m, n', test_invalid_matrix_shape_data)
def test_invalid_matrix_shape(m, n):
    with pytest.raises(ValueError):
        matrix.Matrix((m, n))


def test_matrix_multiplication():
    d1 = [[1, 0, 2], [-1, 3, 1]]
    d3 = [[3, 1], [2, 1], [1, 0]]
    expected = [[5, 1], [4, 2]]
    actual = (matrix.Matrix(d1) * matrix.Matrix(d3)).data
    assert actual == expected


invalid_matrix_multiplication_data = [
    ([[1, 0, 2], [-1, 3, 1]], [[3, 1], [2, 1]]),
    ([[3, 1], [2, 1]], [[1, 0, 2], [-1, 3, 1]])
]


@pytest.mark.parametrize('d1, d2', invalid_matrix_multiplication_data)
def test_invalid_matrix_multiplication(d1, d2):
    with pytest.raises(ValueError):
        matrix.Matrix(d1) * matrix.Matrix(d2)
```

```
Tests failed: 13 of 13 tests – 0 ms

test.py::test_invalid_matrix_multiplication[d11-d21]

============================== 13 failed in 0.15s ==============================
```

2. Faza Green.

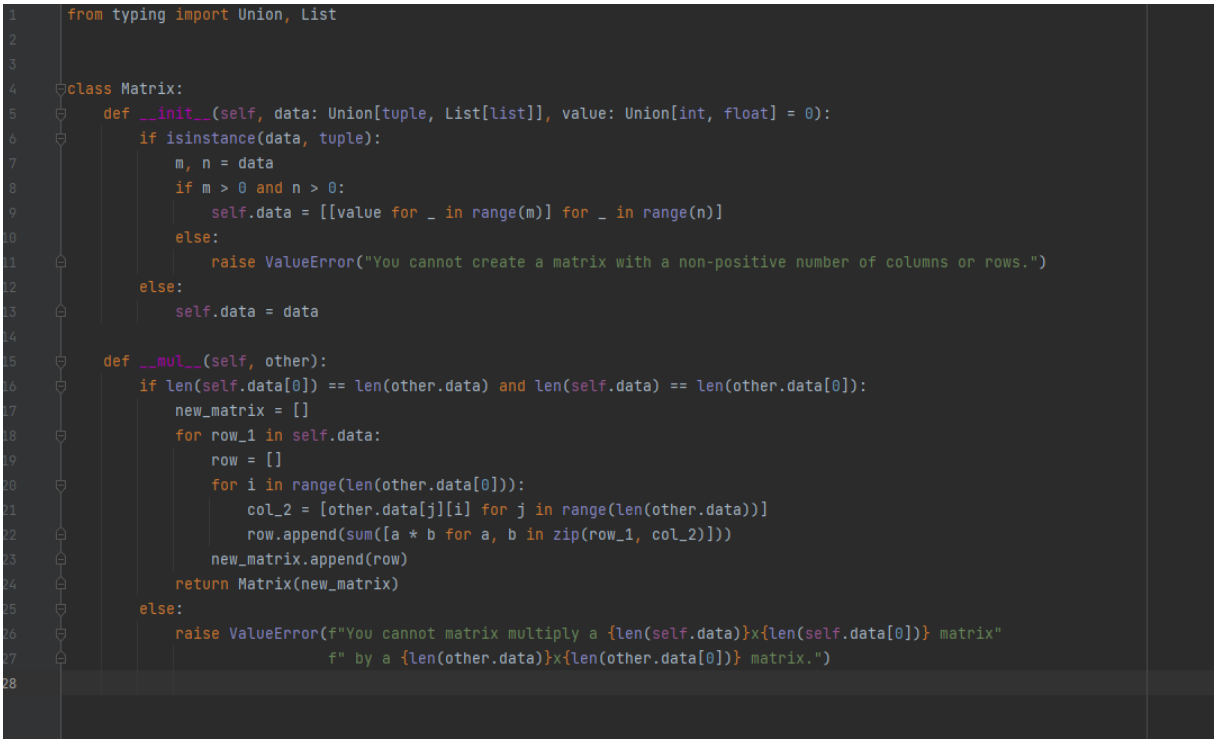
Kolejnym krokiem była implementacja kodu, który realizuje ww. zadania oraz sprawdzenie czy przechodzi on testy.

```python
class Matrix:
    def __init__(self, data, value = 0):
        if isinstance(data, tuple):
            m, n = data
            if m > 0 and n > 0:
                self.data = [[value for _ in range(m)] for _ in range(n)]
            else:
                raise ValueError("You cannot create a matrix with a non-positive number of columns or rows.")
        else:
            self.data = data

    def __mul__(self, other):
        if len(self.data[0]) == len(other.data) and len(self.data) == len(other.data[0]):
            new_matrix = []
            for row_1 in self.data:
                row = []
                for i in range(len(other.data[0])):
                    col_2 = [other.data[j][i] for j in range(len(other.data))]
                    row.append(sum([a * b for a, b in zip(row_1, col_2)]))
                new_matrix.append(row)
            return Matrix(new_matrix)
        else:
            raise ValueError(f"You cannot matrix multiply a {len(self.data)}x{len(self.data[0])} matrix"
                             f" by a {len(other.data)}x{len(other.data[0])} matrix.")
```

```
Tests passed: 13 of 13 tests – 0 ms

============================= test session starts ==============================
collecting ... collected 13 items

test.py::test_matrix_creation_from_dimensions[2-3-0-expected_matrix_data0] PASSED [  7%]
test.py::test_matrix_creation_from_dimensions[3-3-1-expected_matrix_data1] PASSED [ 15%]
test.py::test_matrix_creation_from_dimensions[4-2-3-expected_matrix_data2] PASSED [ 23%]
test.py::test_matrix_creation_from_data[input_matrix_data0-expected_matrix_data0] PASSED [ 30%]
test.py::test_matrix_creation_from_data[input_matrix_data1-expected_matrix_data1] PASSED [ 38%]
test.py::test_matrix_creation_from_data[input_matrix_data2-expected_matrix_data2] PASSED [ 46%]
test.py::test_invalid_matrix_shape[-1-5] PASSED                          [ 53%]
test.py::test_invalid_matrix_shape[0-0] PASSED                           [ 61%]
test.py::test_invalid_matrix_shape[2--5] PASSED                          [ 69%]
test.py::test_invalid_matrix_shape[-2--4] PASSED                         [ 76%]
test.py::test_matrix_multiplication PASSED                               [ 84%]
test.py::test_invalid_matrix_multiplication[d10-d20] PASSED              [ 92%]
test.py::test_invalid_matrix_multiplication[d11-d21] PASSED              [100%]

============================== 13 passed in 0.03s ==============================
```

3. Faza Refactor.

W tej części dodałem podpowiedzi typów do algorytmu.



```python
from typing import Union, List


class Matrix:
    def __init__(self, data: Union[tuple, List[list]], value: Union[int, float] = 0):
        if isinstance(data, tuple):
            m, n = data
            if m > 0 and n > 0:
                self.data = [[value for _ in range(m)] for _ in range(n)]
            else:
                raise ValueError("You cannot create a matrix with a non-positive number of columns or rows.")
        else:
            self.data = data

    def __mul__(self, other):
        if len(self.data[0]) == len(other.data) and len(self.data) == len(other.data[0]):
            new_matrix = []
            for row_1 in self.data:
                row = []
                for i in range(len(other.data[0])):
                    col_2 = [other.data[j][i] for j in range(len(other.data))]
                    row.append(sum([a * b for a, b in zip(row_1, col_2)]))
                new_matrix.append(row)
            return Matrix(new_matrix)
        else:
            raise ValueError(f"You cannot matrix multiply a {len(self.data)}x{len(self.data[0])} matrix"
                             f" by a {len(other.data)}x{len(other.data[0])} matrix.")

```