Weston Cox and Shane Norden

westonc@uark.edu and snorden@uark.edu

011002376 (WC) and 011008524 (SN)

Homework 1

Dr. Khoa Luu

Late Days: 0

For Homework 1, we were given some template code and asked to complete many different functions pertaining to topics like Classes and Inheritance, Matrix Multiplication, Powers of a Matrix, Iterative Functions, Fibonacci, Recursion, and Breadth First and Depth First searches.

Analysis

First, the Geometry class. Below is the output from the test_geometry() function.

```
Area of Triangle: 0.5000
Area of Rectangle: 4.0000
Area of Square: 4.0000
Area of Circle: 28.2743
Area of Shape: 1.0000
```

The test_geometry() function was given to us in the template code. Going through and hand calculating the tests by hand shows that this output is correct. Subsequently, our implementation is correct.

Second, is the Matrix Multiplication. Below is the output from the test_matrix_mul() function.

```
[Test Case 0]. Your implementation is correct!
[Test Case 1]. Your implementation is correct!
[Test Case 2]. Your implementation is correct!
[Test Case 3]. Your implementation is correct!
[Test Case 4]. Your implementation is correct!
[Test Case 5]. Your implementation is correct!
[Test Case 6]. Your implementation is correct!
[Test Case 7]. Your implementation is correct!
[Test Case 8]. Your implementation is correct!
[Test Case 9]. Your implementation is correct!
```

The output is pretty self explanatory. Our implementation of Matrix Multiplication is correct.

Third, is the Powers of Matrices. The test_pow() function is hard to understand and prove the implementation is correct, so we instantiated a numpy array below, and tested it.

```python
def test_pow_clearly():
    n = 2
    A = np.array([[1, -3], [2, 5]])
    print("Starting Array: \n", A)
    assert np.mean(np.abs(A.dot(A) - recursive_pow(A, n))) <= 1e-7, "Your implementation is wrong!"
    print("Recursive: A^{} =\n {}".format(n, recursive_pow(A, n)))
```

The output from this test is below.

```
Starting Array:
 [[ 1 -3]
 [ 2  5]]
Recursive: A^2 =
 [[ -5 -18]
 [ 12  19]]

Iterative: A^2 =
 [[ -5 -18]
 [ 12  19]]
```

```
Starting Array:
 [[ 4  0]
 [-3  7]]
Recursive: A^3 =
 [[  64    0]
 [-279  343]]

Iterative: A^3 =
 [[  64    0]
 [-279  343]]
```

Running the original array through a matrix calculator shows that the output is correct.

Fourth, are the Fibonacci functions. We had a lot of trouble figuring out where to start on these, so we heavily relied on the review from this point onwards. Below is the output from the test_fibonacci() function.

```
[Test Case 0]. Your implementation is correct!. fibo(2) = 2
[Test Case 1]. Your implementation is correct!. fibo(3) = 3
[Test Case 2]. Your implementation is correct!. fibo(4) = 5
[Test Case 3]. Your implementation is correct!. fibo(5) = 8
[Test Case 4]. Your implementation is correct!. fibo(6) = 13
[Test Case 5]. Your implementation is correct!. fibo(7) = 21
[Test Case 6]. Your implementation is correct!. fibo(8) = 34
[Test Case 7]. Your implementation is correct!. fibo(9) = 55
f(5, 2) = 8
f(5, 3) = 9
f(5, 4) = 7
f(6, 2) = 13
f(6, 3) = 17
f(6, 4) = 13
f(7, 2) = 21
f(7, 3) = 31
f(7, 4) = 25
f(8, 2) = 34
f(8, 3) = 57
f(8, 4) = 49
f(9, 2) = 55
f(9, 3) = 105
f(9, 4) = 94
f(10, 2) = 89
f(10, 3) = 193
f(10, 4) = 181
```

Like before, the output is pretty self explanatory.

Fifth and finally, we have the BFS, DFS, and find_minimum() implementations. Again, we relied heavily on the Homework 1 review slides to implement these. The output from the test_bfs_dfs_find_minimum() function is below. We had to change the order that the BFS, DFS, and find_minimum functions were called within the test function, as the find_minimum() function was not being called.

```
Find Minimum
Cost: 9
(0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (2, 4) -> (3, 4) -> (4, 4)
BFS
(0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (4, 3) -> (4, 4)
DFS
(0, 0) -> (0, 1) -> (0, 2) -> (1, 2) -> (2, 2) -> (2, 3) -> (3, 3) -> (4, 3) -> (4, 4)
```