Homework 4

Problem 1: Desert and Green Land.

In order to create organized code logic, I created three functions to solve this problem. I created a function called search_matrix that traverses the matrix using the depth first search (DFS) algorithm, such that it observes and flags the visited islands with the number 69. The other helper function that I created was get_num_islands that works hand in hand with the DFS function such that it counts how many islands of 1's (or green lands) are at a given instance of a matrix. Finally, the calc_green_land takes as an input the dimensions of a matrix and the indices array that will be changed from desert lands (0's) to green lands (1's). It also takes as an input the argument `verbose`, which specifies whether to return the created matrix of green and desert lands as well as the p array, or just the p array, which denotes the number of green lands at any given instance. All these functions also take into account edge cases like incorrect inputs (an error will be thrown, or a 0 array will be returned depending on the input), a dimensionless matrix (a 0 array will be returned).

While the calc_green_land runs, it looks for the indices in the matrix that were flagged (i.e., equal to 69) and replaces them back with 1 to return the correct output.

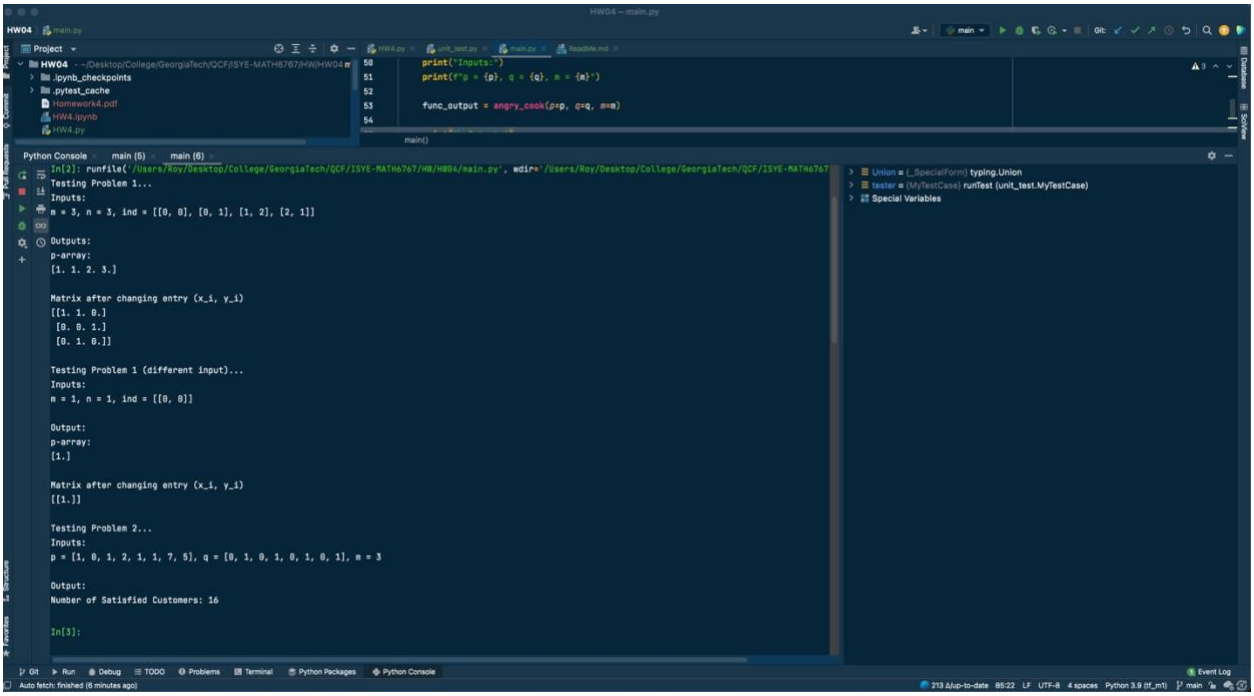
Problem 2: Angry Cook

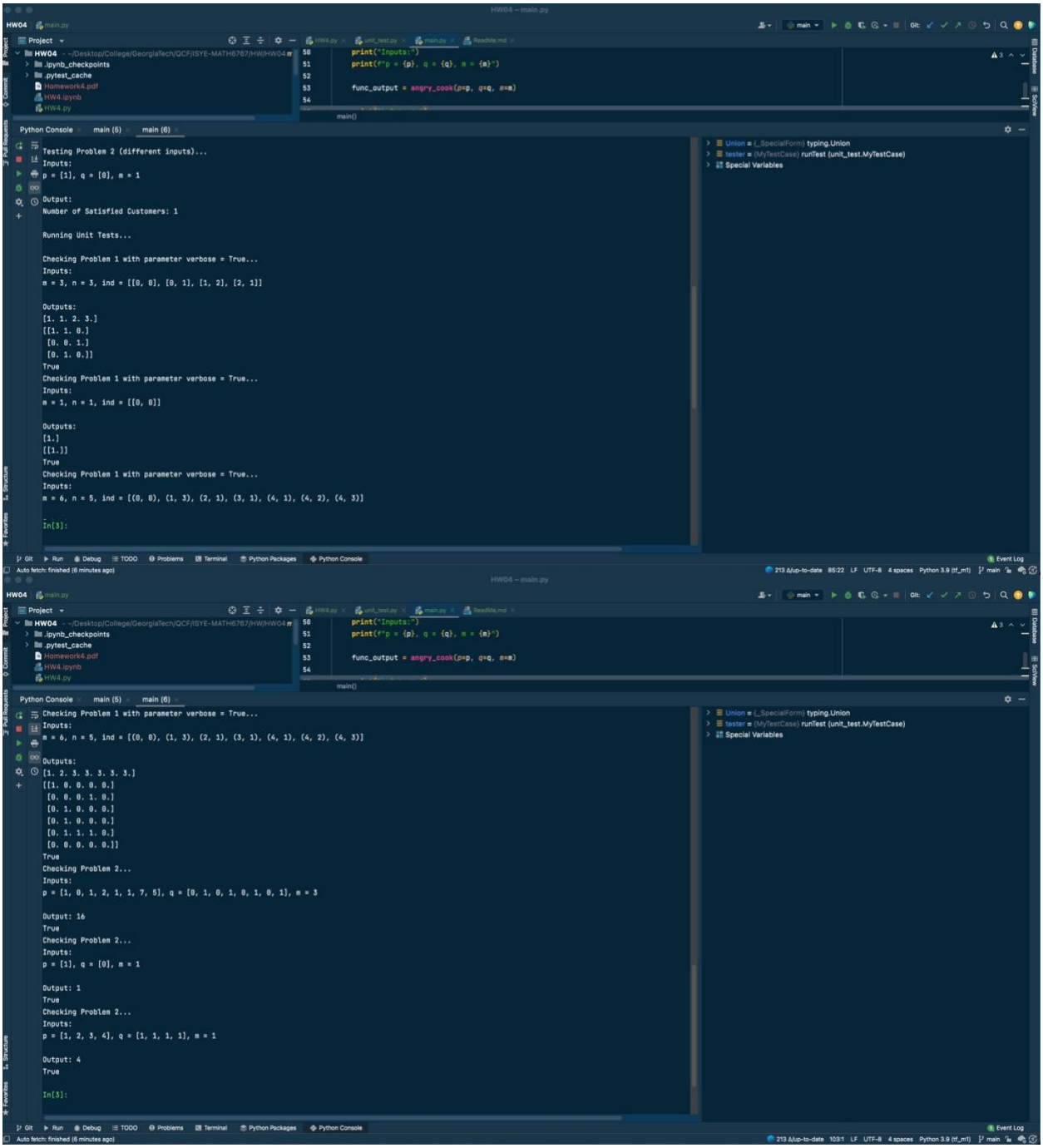
To implement this function, I first checked for edge cases like if the p or q arrays did not exist or were empty, or if the lengths did not match. I also transformed them to numpy arrays so that I can perform array operations with optimal speed. Firstly, I calculated the number of lost customers since 1 denotes angry and 0 does not, multiplying the p and q arrays together yields the number of customers that received horrible service. Then, I calculate the sliding window using the input m, where it denotes the saved customers from those lost ones and obtains the maximum in order to choose the optimal time to use m. Note that this sliding window approach could have been solved using for loops. However, the np.convolve method does exactly that and is a more efficient way mathematically and computationally. The `valid` argument was used since we want to return an array with the same length as that p and q, though it is not necessary since the padded values at the ends of the array will be less than the true maximum. Lastly, I flipped the 1's and 0's in the q array to calculate the number of customers who received good service and summed those values with the maximum convoluted value to obtain the necessary result.

Main.py

In the main.py file I run all the above functions on tests cases (see below for examples). I also created a unit_test.py file to run my unittests on. In both problems, I created my own test cases as well as the pdf known cases for the assignment.

Results:


HW04 – main.py
Project
HW04 ~/Desktop/College/GeorgiaTech/QCF/ISYE-MATH6767/HW/HW04
.ipynb_checkpoints
.pytest_cache
Homework4.pdf
HW4.ipynb
HW4.py
print("Inputs:")
print(f"p = {p}, q = {q}, m = {m}")
func_output = angry_cook(p=p, q=q, m=m)
main()
Python Console
main (5)
main (6)
In[2]: runfile('/Users/Roy/Desktop/College/GeorgiaTech/QCF/ISYE-MATH6767/HW/HW04/main.py', wdir='/Users/Roy/Desktop/College/GeorgiaTech/QCF/ISYE-MATH6767
Testing Problem 1...
Inputs:
m = 3, n = 3, ind = [[0, 0], [0, 1], [1, 2], [2, 1]]
Outputs:
p-array:
[1. 1. 2. 3.]
Matrix after changing entry (x_i, y_i)
[[1. 1. 0.]
[0. 0. 1.]
[0. 1. 0.]]
Testing Problem 1 (different input)...
Inputs:
m = 1, n = 1, ind = [[0, 0]]
Output:
p-array:
[1.]
Matrix after changing entry (x_i, y_i)
[[1.]]
Testing Problem 2...
Inputs:
p = [1, 0, 1, 2, 1, 1, 7, 5], q = [0, 1, 0, 1, 0, 1, 0, 1], m = 3
Output:
Number of Satisfied Customers: 16
In[3]:
Union = (_SpecialForm) typing.Union
tester = (MyTestCase) runTest (unit_test.MyTestCase)
Special Variables
Git
Run
Debug
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
Auto fetch: finished (6 minutes ago)
213 Δ/up-to-date 85:22 LF UTF-8 4 spaces Python 3.9 (tf_m1) main


HW04 – main.py
print("Inputs:")
print(f"p = {p}, q = {q}, m = {m}")
func_output = angry_cook(p=p, q=q, m=m)
main()
Python Console
main (5)
main (6)
Testing Problem 2 (different inputs)...
Inputs:
p = [1], q = [0], m = 1
Output:
Number of Satisfied Customers: 1
Running Unit Tests...
Checking Problem 1 with parameter verbose = True...
Inputs:
m = 3, n = 3, ind = [[0, 0], [0, 1], [1, 2], [2, 1]]
Outputs:
[1. 1. 2. 3.]
[[1. 1. 0.]
[0. 0. 1.]
[0. 1. 0.]]
True
Checking Problem 1 with parameter verbose = True...
Inputs:
m = 1, n = 1, ind = [[0, 0]]
Outputs:
[1.]
[[1.]]
True
Checking Problem 1 with parameter verbose = True...
Inputs:
m = 6, n = 5, ind = [(0, 0), (1, 3), (2, 1), (3, 1), (4, 1), (4, 2), (4, 3)]
In[3]:
Union = (_SpecialForm) typing.Union
tester = (MyTestCase) runTest (unit_test.MyTestCase)
Special Variables
Checking Problem 1 with parameter verbose = True...
Inputs:
m = 6, n = 5, ind = [(0, 0), (1, 3), (2, 1), (3, 1), (4, 1), (4, 2), (4, 3)]
Outputs:
[1. 2. 3. 3. 3. 3. 3.]
[[1. 0. 0. 0. 0.]
[0. 0. 0. 1. 0.]
[0. 1. 0. 0. 0.]
[0. 1. 0. 0. 0.]
[0. 1. 1. 1. 0.]
[0. 0. 0. 0. 0.]]
True
Checking Problem 2...
Inputs:
p = [1, 0, 1, 2, 1, 1, 7, 5], q = [0, 1, 0, 1, 0, 1, 0, 1], m = 3
Output: 16
True
Checking Problem 2...
Inputs:
p = [1], q = [0], m = 1
Output: 1
True
Checking Problem 2...
Inputs:
p = [1, 2, 3, 4], q = [1, 1, 1, 1], m = 1
Output: 4
True
In[3]: