Avery Nelson --- 0 late days used

010920903

amn018@uark.edu

Artificial Intelligence Assignment 1 Report

Description of Implementation

The goal of this assignment is to implement a base class and the constructors for 5 classes. This required that the 5 classes inherit the base class, one of which inherits it through another one of the 5 classes. This required creating a function meant to count the number of instances of the base class, then implementing constructors and area functions of each of the 5 shape classes. This was fairly straightforward and mostly just required implementing code to calculate the basic area functions for a triangle, rectangle, square, circle, and polygon respectively.

The next requirement was to implement funct6ions for matrix calculations. The first step was meant to multiply the two given matrices without using np.dot, .dot, or & operator of NumPy library in the implementation. I ran some simple checks then used nested loops. While not particularly efficient, this program required relatively little out of the nested loops and as such the time was not a significant issue. Then the assignment required that a recursive function and an iterative function both be used to perform the powers of a matrix. Neither was particularly hard to implement or test, and differed in approach, not the outcome, proving that they worked correctly.

The next task of the assignment was calculating the n^{th} Fibonacci number using powers of the matrix *A*. This step had built in testing, so it was verified easily. It required calling the recursive power function of the matrix A and was confirmed by the assertions in the testing functions. Then, we needed to implement a function that calculates the f_n of a recursive function below

Given two number n and k, the recursive function is defined as follows

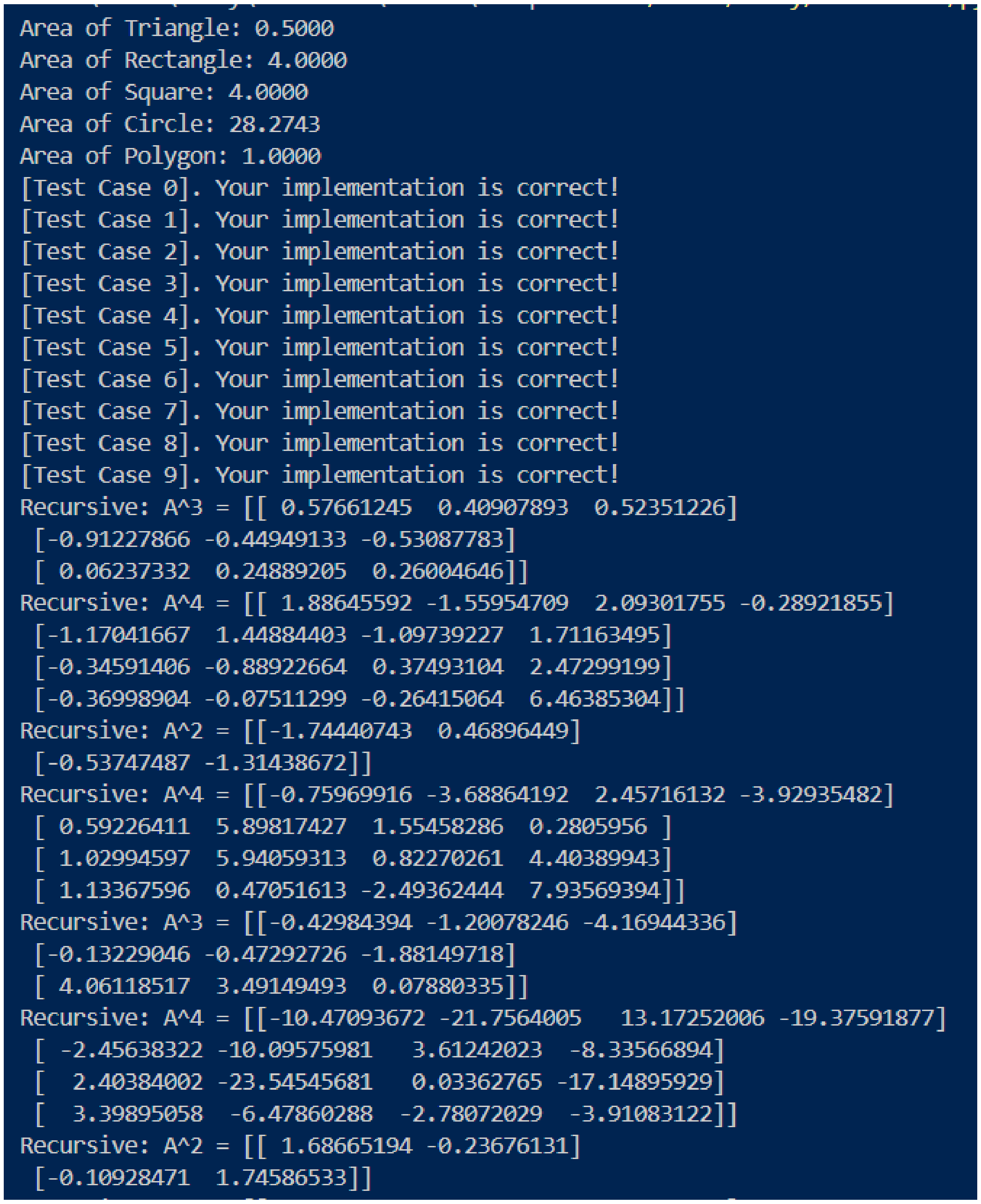
$$\begin{array}{ll} f_n = 1 & if\ n < k \\ f_n = f_{n-1} + f_{n-2} + \ldots + f_{n-k} & if\ n \geq k \end{array}$$

using power of matrix. I had more issues with this problem because the testing did not announce if the results were valid, so upon checking I realized that my initial function was not correct. My correct implementation required cycling through the range and appending results (up to the required number) then summing the requested number of preceding values.

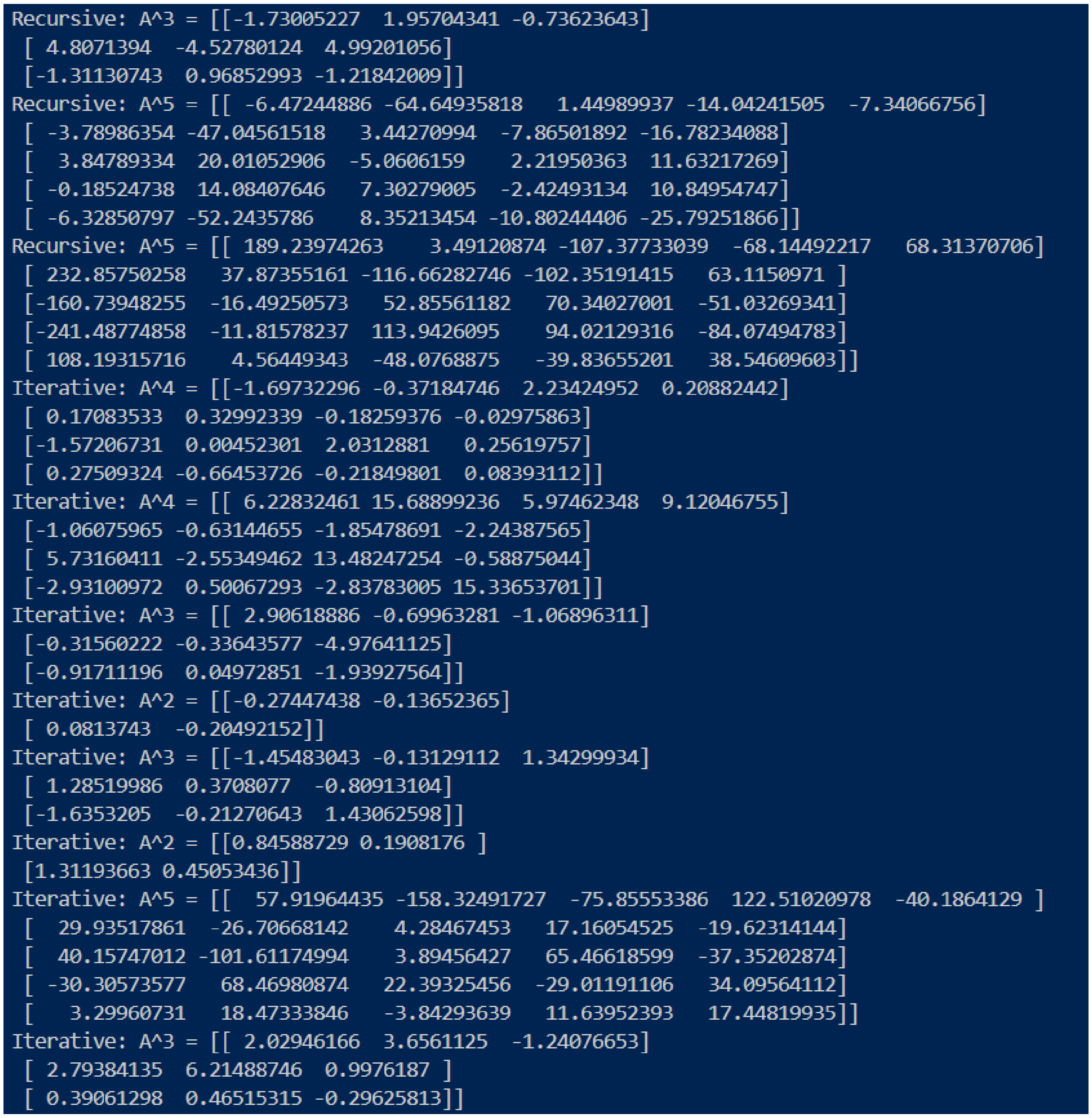
The final requirement was to implement a depth-first search and a breadth-first search based on a provided *mxn* matrix with a zero-based index. With a value of 0, a cell cannot be accessed. The goal is to find a path that reaches the bottom right corner of the matrix. Both of these are standard algorithms that require tracking the cells that were visited and determining which valid cells are available to move to next while attempting to reach the goal node. The final goal (and advanced problem) was to find the minimum cost path through the matrix, much the same way the last problem was solved, but with the goal of minimizing the total cost of the path. This is solved by determining the cost of valid paths and choosing the one with the minimum cost.

Results

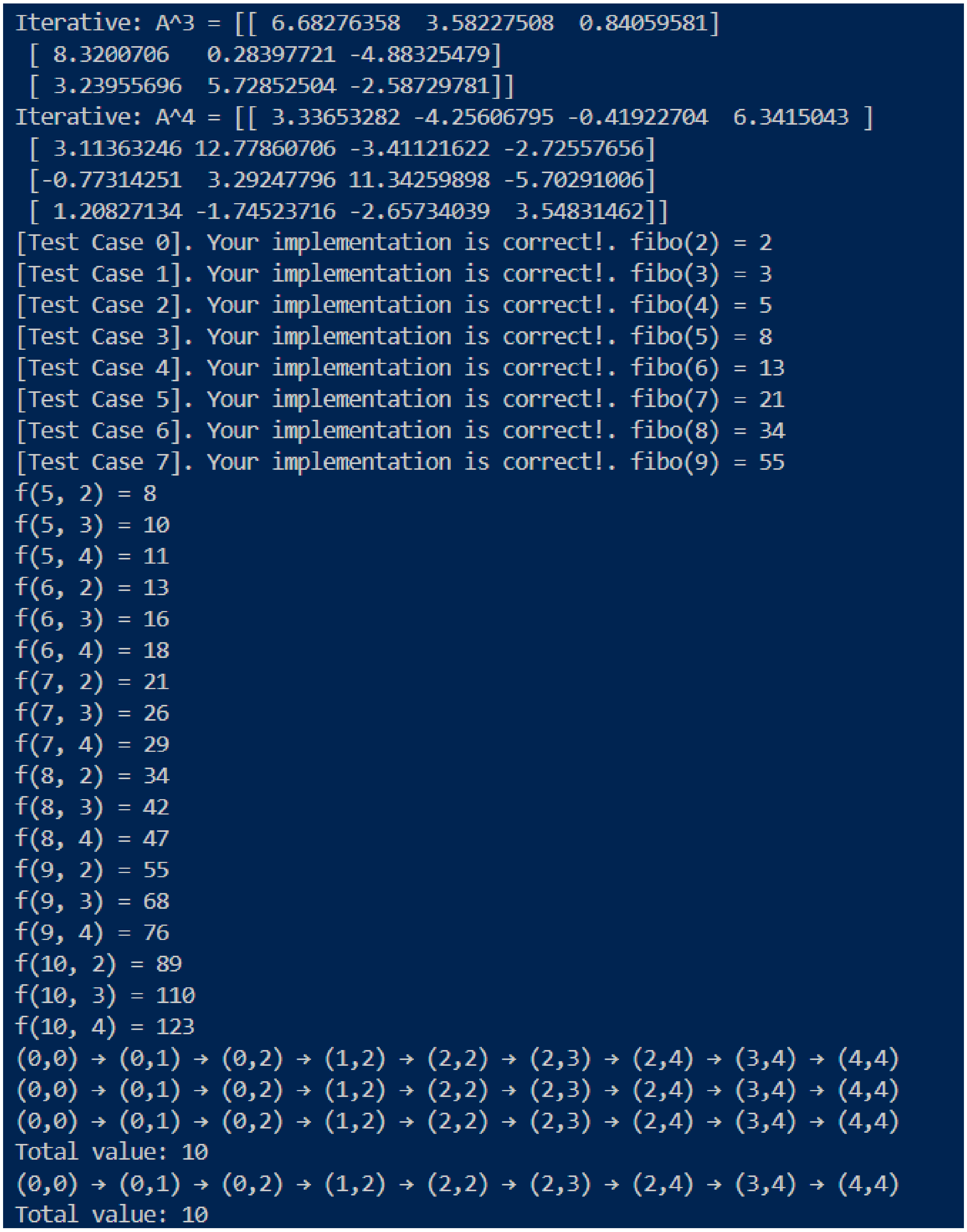
Overall, my implementations of each of the problems appears to be successful. Areas I could confirm manually, which the tests of both matrix multiplication and the Fibonacci confirmed their accuracy for me. I ran a test case of the recursive and iterative powers because the randomized number made it hard to confirm that both were returning the same result and thus working in the same way. I also confirmed the f(n, k) outputs and searches manually since confirmation was not built-in. f(n,k) aligns with what the manually calculated sum of the selected previous numbers should be. The results of the DFS and BFS implementations both also give valid paths spanning from the start node to the goal node. The minimum then calculates the path with the lowest cost of the available valid paths. Thus, the project successfully completes the required tasks given the test cases and parameters provided.



```
Area of Triangle: 0.5000
Area of Rectangle: 4.0000
Area of Square: 4.0000
Area of Circle: 28.2743
Area of Polygon: 1.0000
[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!
Recursive: A^3 = [[ 0.57661245  0.40907893  0.52351226]
 [-0.91227866 -0.44949133 -0.53087783]
 [ 0.06237332  0.24889205  0.26004646]]
Recursive: A^4 = [[ 1.88645592 -1.55954709  2.09301755 -0.28921855]
 [-1.17041667  1.44884403 -1.09739227  1.71163495]
 [-0.34591406 -0.88922664  0.37493104  2.47299199]
 [-0.36998904 -0.07511299 -0.26415064  6.46385304]]
Recursive: A^2 = [[-1.74440743  0.46896449]
 [-0.53747487 -1.31438672]]
Recursive: A^4 = [[-0.75969916 -3.68864192  2.45716132 -3.92935482]
 [ 0.59226411  5.89817427  1.55458286  0.2805956 ]
 [ 1.02994597  5.94059313  0.82270261  4.40389943]
 [ 1.13367596  0.47051613 -2.49362444  7.93569394]]
Recursive: A^3 = [[-0.42984394 -1.20078246 -4.16944336]
 [-0.13229046 -0.47292726 -1.88149718]
 [ 4.06118517  3.49149493  0.07880335]]
Recursive: A^4 = [[-10.47093672 -21.7564005   13.17252006 -19.37591877]
 [ -2.45638322 -10.09575981   3.61242023  -8.33566894]
 [  2.40384002 -23.54545681   0.03362765 -17.14895929]
 [  3.39895058  -6.47860288  -2.78072029  -3.91083122]]
Recursive: A^2 = [[ 1.68665194 -0.23676131]
 [-0.10928471  1.74586533]]
```



```
Recursive: A^3 = [[-1.73005227  1.95704341 -0.73623643]
 [ 4.8071394  -4.52780124  4.99201056]
 [-1.31130743  0.96852993 -1.21842009]]
Recursive: A^5 = [[ -6.47244886 -64.64935818   1.44989937 -14.04241505  -7.34066756]
 [ -3.78986354 -47.04561518   3.44270994  -7.86501892 -16.78234088]
 [  3.84789334  20.01052906  -5.0606159    2.21950363  11.63217269]
 [ -0.18524738  14.08407646   7.30279005  -2.42493134  10.84954747]
 [ -6.32850797 -52.2435786    8.35213454 -10.80244406 -25.79251866]]
Recursive: A^5 = [[ 189.23974263    3.49120874 -107.37733039  -68.14492217   68.31370706]
 [ 232.85750258   37.87355161 -116.66282746 -102.35191415   63.1150971 ]
 [-160.73948255  -16.49250573   52.85561182   70.34027001  -51.03269341]
 [-241.48774858  -11.81578237  113.9426095    94.02129316  -84.07494783]
 [ 108.19315716    4.56449343  -48.0768875   -39.83655201   38.54609603]]
Iterative: A^4 = [[-1.69732296 -0.37184746  2.23424952  0.20882442]
 [ 0.17083533  0.32992339 -0.18259376 -0.02975863]
 [-1.57206731  0.00452301  2.0312881   0.25619757]
 [ 0.27509324 -0.66453726 -0.21849801  0.08393112]]
Iterative: A^4 = [[ 6.22832461 15.68899236  5.97462348  9.12046755]
 [-1.06075965 -0.63144655 -1.85478691 -2.24387565]
 [ 5.73160411 -2.55349462 13.48247254 -0.58875044]
 [-2.93100972  0.50067293 -2.83783005 15.33653701]]
Iterative: A^3 = [[ 2.90618886 -0.69963281 -1.06896311]
 [-0.31560222 -0.33643577 -4.97641125]
 [-0.91711196  0.04972851 -1.93927564]]
Iterative: A^2 = [[-0.27447438 -0.13652365]
 [ 0.0813743  -0.20492152]]
Iterative: A^3 = [[-1.45483043 -0.13129112  1.34299934]
 [ 1.28519986  0.3708077  -0.80913104]
 [-1.6353205  -0.21270643  1.43062598]]
Iterative: A^2 = [[0.84588729 0.1908176 ]
 [1.31193663 0.45053436]]
Iterative: A^5 = [[  57.91964435 -158.32491727  -75.85553386  122.51020978  -40.1864129 ]
 [  29.93517861  -26.70668142    4.28467453   17.16054525  -19.62314144]
 [  40.15747012 -101.61174994    3.89456427   65.46618599  -37.35202874]
 [ -30.30573577   68.46980874   22.39325456  -29.01191106   34.09564112]
 [   3.29960731   18.47333846   -3.84293639   11.63952393   17.44819935]]
Iterative: A^3 = [[ 2.02946166  3.6561125  -1.24076653]
 [ 2.79384135  6.21488746  0.9976187 ]
 [ 0.39061298  0.46515315 -0.29625813]]
```



```
Iterative: A^3 = [[ 6.68276358  3.58227508  0.84059581]
 [ 8.3200706   0.28397721 -4.88325479]
 [ 3.23955696  5.72852504 -2.58729781]]
Iterative: A^4 = [[ 3.33653282 -4.25606795 -0.41922704  6.3415043 ]
 [ 3.11363246 12.77860706 -3.41121622 -2.72557656]
 [-0.77314251  3.29247796 11.34259898 -5.70291006]
 [ 1.20827134 -1.74523716 -2.65734039  3.54831462]]
[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) = 10
f(5, 4) = 11
f(6, 2) = 13
f(6, 3) = 16
f(6, 4) = 18
f(7, 2) = 21
f(7, 3) = 26
f(7, 4) = 29
f(8, 2) = 34
f(8, 3) = 42
f(8, 4) = 47
f(9, 2) = 55
f(9, 3) = 68
f(9, 4) = 76
f(10, 2) = 89
f(10, 3) = 110
f(10, 4) = 123
(0,0) → (0,1) → (0,2) → (1,2) → (2,2) → (2,3) → (2,4) → (3,4) → (4,4)
(0,0) → (0,1) → (0,2) → (1,2) → (2,2) → (2,3) → (2,4) → (3,4) → (4,4)
(0,0) → (0,1) → (0,2) → (1,2) → (2,2) → (2,3) → (2,4) → (3,4) → (4,4)
Total value: 10
(0,0) → (0,1) → (0,2) → (1,2) → (2,2) → (2,3) → (2,4) → (3,4) → (4,4)
Total value: 10
```