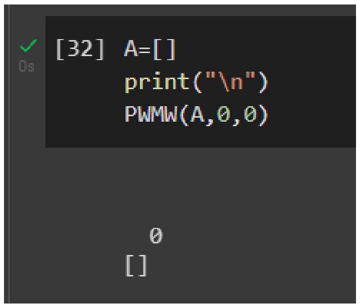
Benchmark Suite

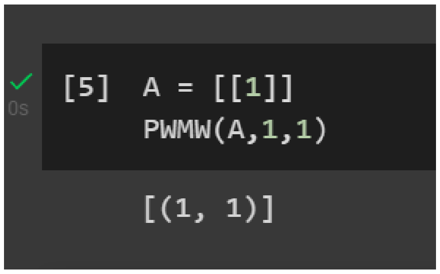
Black Box Testing

- No cells.



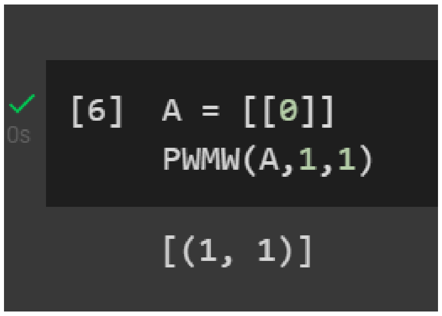
```
[32] A=[]
     print("\n")
     PWMW(A,0,0)

      0
     []
```

- There is only one cell.
  * with weed



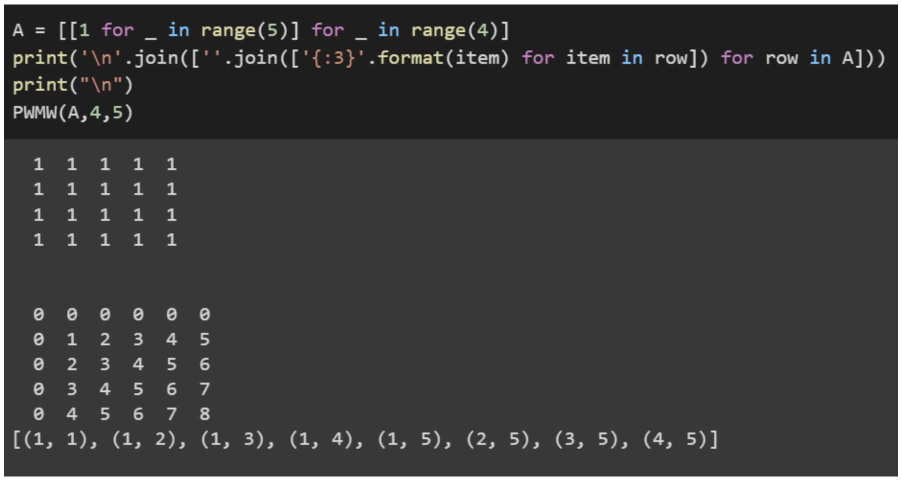
```
[5]  A = [[1]]
     PWMW(A,1,1)

     [(1, 1)]
```

  * without weed



```
[6]  A = [[0]]
     PWMW(A,1,1)

     [(1, 1)]
```

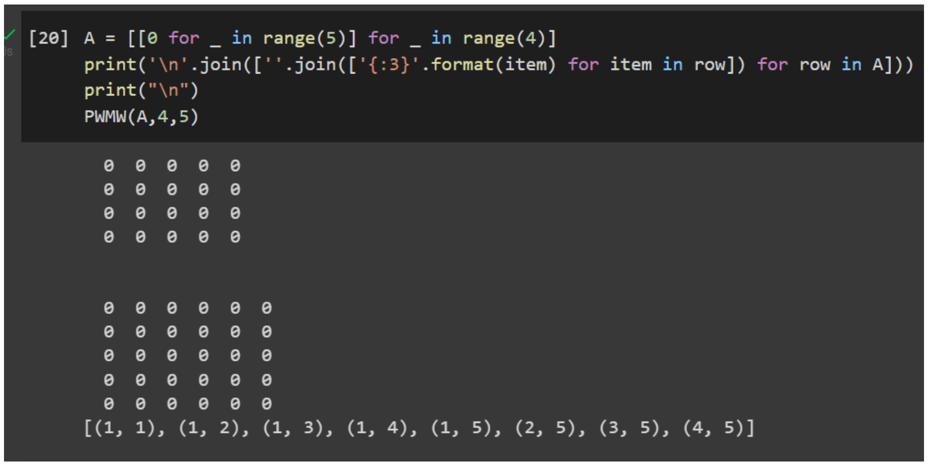
- All the cells have weed.



```
A = [[1 for _ in range(5)] for _ in range(4)]
print('\n'.join([''.join(['{:3}'.format(item) for item in row]) for row in A]))
print("\n")
PWMW(A,4,5)
```

```
  1  1  1  1  1
  1  1  1  1  1
  1  1  1  1  1
  1  1  1  1  1


  0  0  0  0  0  0
  0  1  2  3  4  5
  0  2  3  4  5  6
  0  3  4  5  6  7
  0  4  5  6  7  8
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 5), (3, 5), (4, 5)]
```

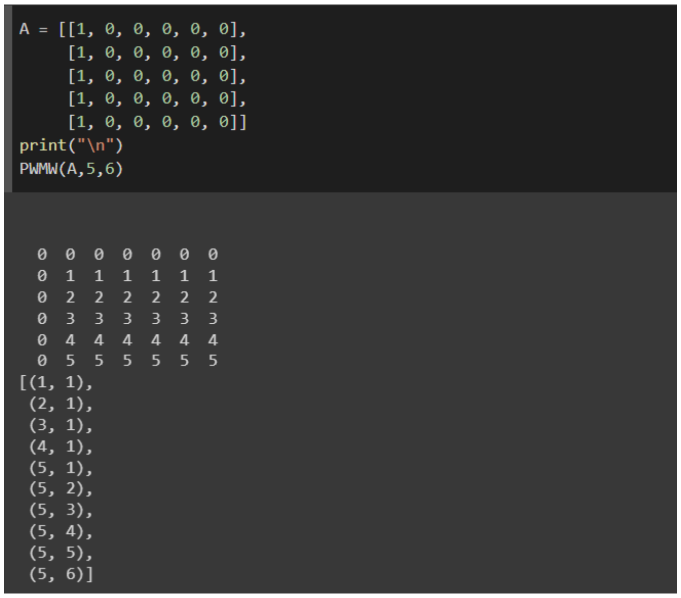
- None of the cells have weed.



```
[20] A = [[0 for _ in range(5)] for _ in range(4)]
     print('\n'.join([''.join(['{:3}'.format(item) for item in row]) for row in A]))
     print("\n")
     PWMW(A,4,5)
```

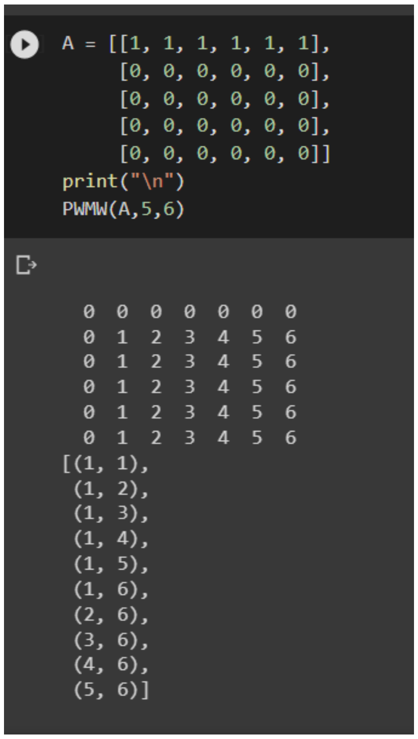
```
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0
  0  0  0  0  0


  0  0  0  0  0  0
  0  0  0  0  0  0
  0  0  0  0  0  0
  0  0  0  0  0  0
  0  0  0  0  0  0
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 5), (3, 5), (4, 5)]
```

- All the weeds are aligned on an edge.



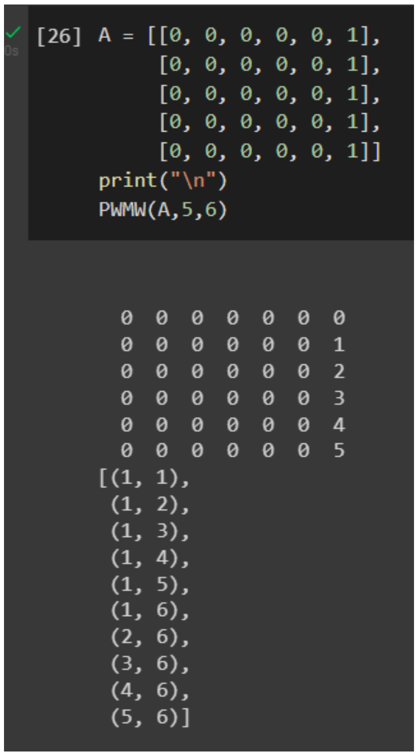
```
A = [[1, 0, 0, 0, 0, 0],
     [1, 0, 0, 0, 0, 0],
     [1, 0, 0, 0, 0, 0],
     [1, 0, 0, 0, 0, 0],
     [1, 0, 0, 0, 0, 0]]
print("\n")
PWMW(A,5,6)
```

```


  0  0  0  0  0  0  0
  0  1  1  1  1  1  1
  0  2  2  2  2  2  2
  0  3  3  3  3  3  3
  0  4  4  4  4  4  4
  0  5  5  5  5  5  5
[(1, 1),
 (2, 1),
 (3, 1),
 (4, 1),
 (5, 1),
 (5, 2),
 (5, 3),
 (5, 4),
 (5, 5),
 (5, 6)]
```



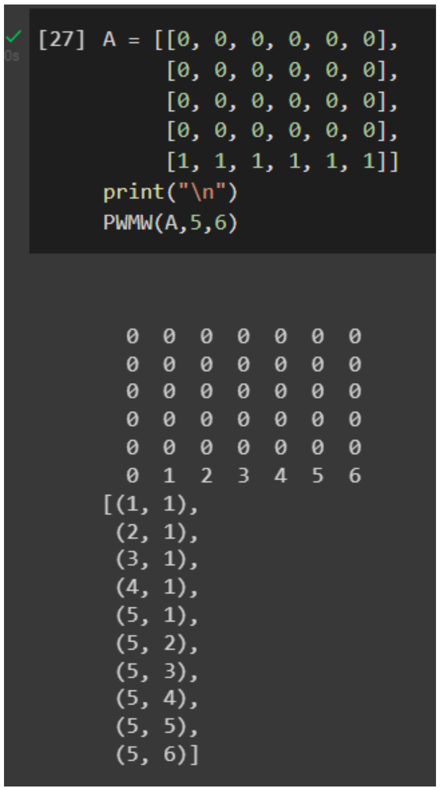
```
A = [[1, 1, 1, 1, 1, 1],
     [0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0]]
print("\n")
PWMW(A,5,6)
```

```


  0  0  0  0  0  0  0
  0  1  2  3  4  5  6
  0  1  2  3  4  5  6
  0  1  2  3  4  5  6
  0  1  2  3  4  5  6
  0  1  2  3  4  5  6
[(1, 1),
 (1, 2),
 (1, 3),
 (1, 4),
 (1, 5),
 (1, 6),
 (2, 6),
 (3, 6),
 (4, 6),
 (5, 6)]
```



```
[26] A = [[0, 0, 0, 0, 0, 1],
          [0, 0, 0, 0, 0, 1],
          [0, 0, 0, 0, 0, 1],
          [0, 0, 0, 0, 0, 1],
          [0, 0, 0, 0, 0, 1]]
     print("\n")
     PWMW(A,5,6)
```

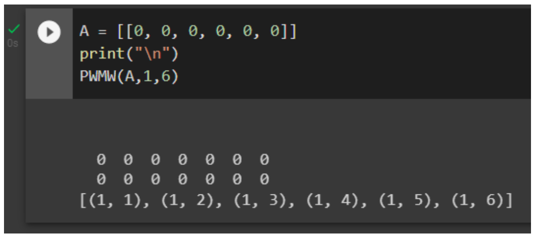
```


  0  0  0  0  0  0  0
  0  0  0  0  0  0  1
  0  0  0  0  0  0  2
  0  0  0  0  0  0  3
  0  0  0  0  0  0  4
  0  0  0  0  0  0  5
[(1, 1),
 (1, 2),
 (1, 3),
 (1, 4),
 (1, 5),
 (1, 6),
 (2, 6),
 (3, 6),
 (4, 6),
 (5, 6)]
```



```
[27] A = [[0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0],
          [1, 1, 1, 1, 1, 1]]
     print("\n")
     PWMW(A,5,6)
```

```
       0  0  0  0  0  0  0
       0  0  0  0  0  0  0
       0  0  0  0  0  0  0
       0  0  0  0  0  0  0
       0  0  0  0  0  0  0
       0  1  2  3  4  5  6
     [(1, 1),
      (2, 1),
      (3, 1),
      (4, 1),
      (5, 1),
      (5, 2),
      (5, 3),
      (5, 4),
      (5, 5),
      (5, 6)]
```

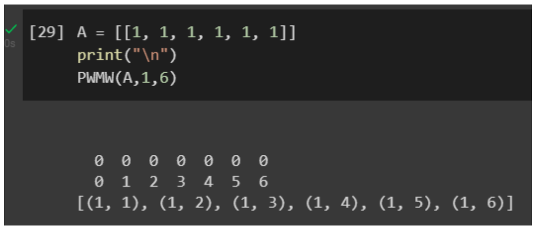
- Matrix is 1 x m.
  * no weeds



```
A = [[0, 0, 0, 0, 0, 0]]
print("\n")
PWMW(A,1,6)
```

```
  0  0  0  0  0  0  0
  0  0  0  0  0  0  0
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)]
```

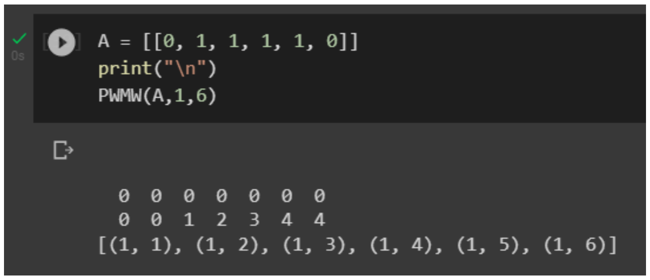
  *all of them have weed.



```
[29] A = [[1, 1, 1, 1, 1, 1]]
     print("\n")
     PWMW(A,1,6)
```

```
  0  0  0  0  0  0  0
  0  1  2  3  4  5  6
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)]
```

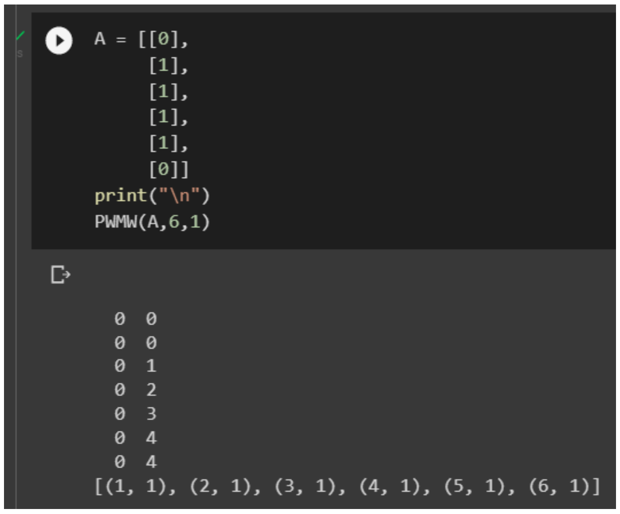
  *some weeds



```
A = [[0, 1, 1, 1, 1, 0]]
print("\n")
PWMW(A,1,6)
```

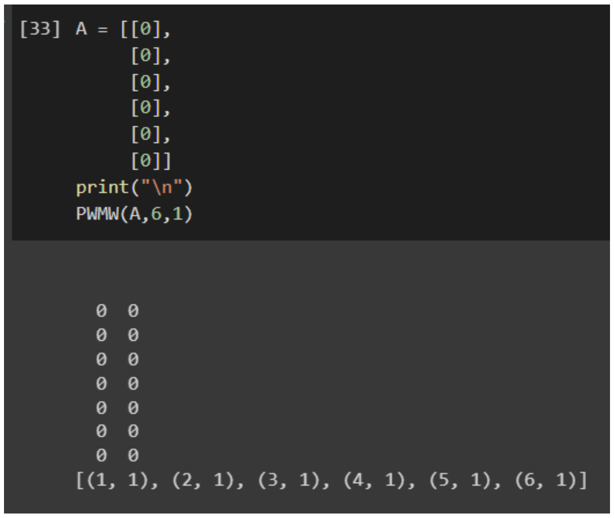
```
  0  0  0  0  0  0  0
  0  0  1  2  3  4  4
[(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6)]
```

- Matrix is n x 1.



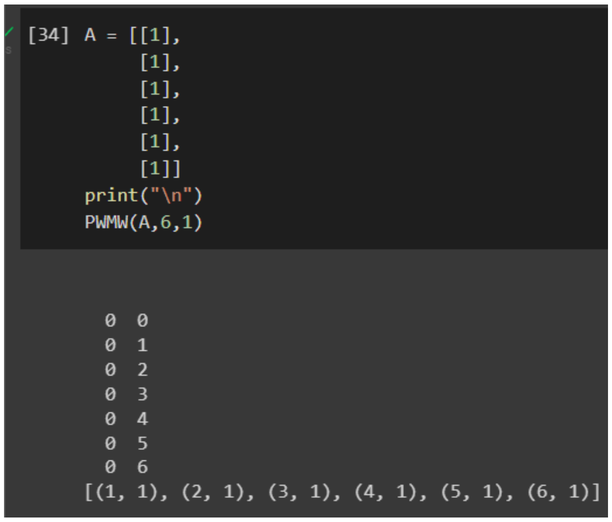
```
A = [[0],
     [1],
     [1],
     [1],
     [1],
     [0]]
print("\n")
PWMW(A,6,1)
```

```
 0  0
 0  0
 0  1
 0  2
 0  3
 0  4
 0  4
[(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1)]
```



```
[33] A = [[0],
          [0],
          [0],
          [0],
          [0],
          [0]]
     print("\n")
     PWMW(A,6,1)
```

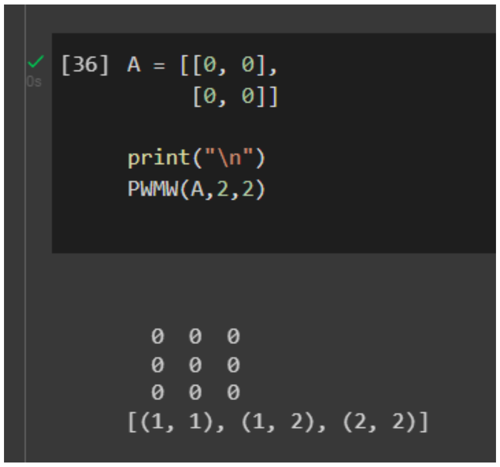
```
 0  0
 0  0
 0  0
 0  0
 0  0
 0  0
 0  0
[(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1)]
```



```
[34] A = [[1],
          [1],
          [1],
          [1],
          [1],
          [1]]
     print("\n")
     PWMW(A,6,1)
```

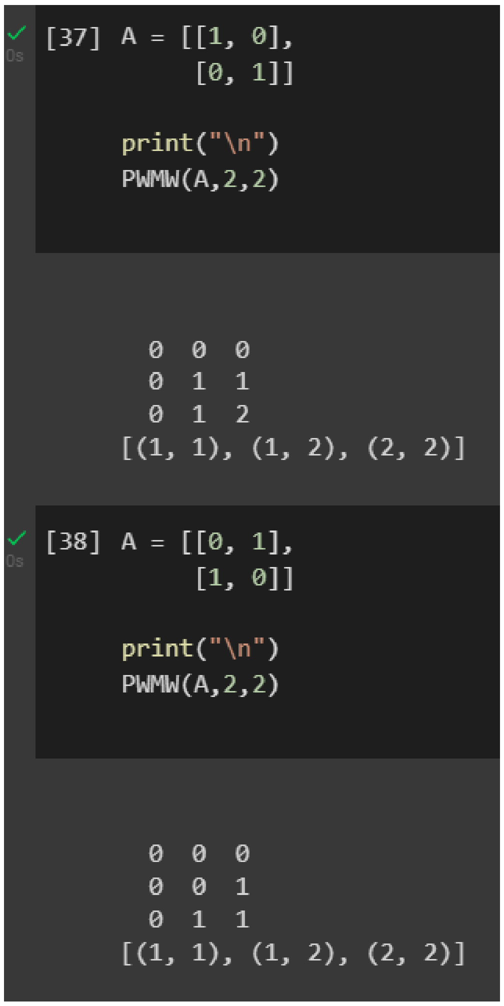
```
 0  0
 0  1
 0  2
 0  3
 0  4
 0  5
 0  6
[(1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (6, 1)]
```

- Matrix is 2 x 2.



```
[36] A = [[0, 0],
          [0, 0]]

     print("\n")
     PWMW(A,2,2)
```

```
 0  0  0
 0  0  0
 0  0  0
[(1, 1), (1, 2), (2, 2)]
```



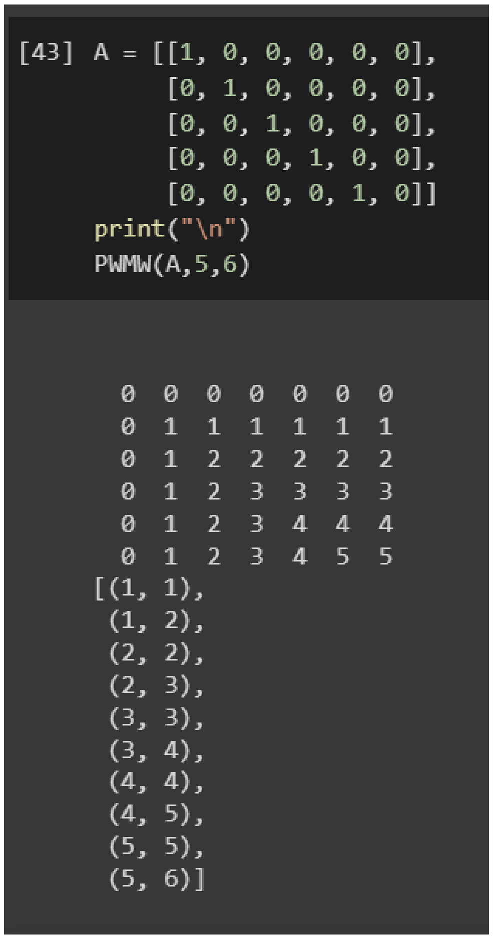
```
[37] A = [[1, 0],
          [0, 1]]

     print("\n")
     PWMW(A,2,2)
```

```
 0  0  0
 0  1  1
 0  1  2
[(1, 1), (1, 2), (2, 2)]
```

```
[38] A = [[0, 1],
          [1, 0]]

     print("\n")
     PWMW(A,2,2)
```

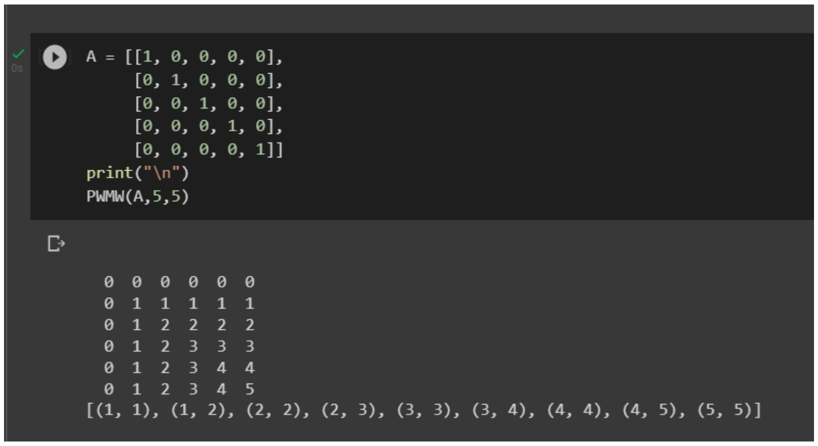
```
 0  0  0
 0  0  1
 0  1  1
[(1, 1), (1, 2), (2, 2)]
```

- Matrix has weeds diagonally.



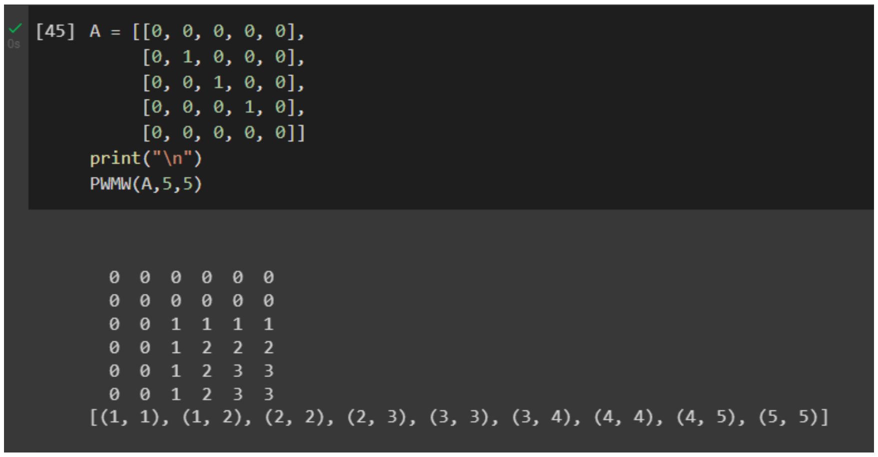
```
[43] A = [[1, 0, 0, 0, 0, 0],
          [0, 1, 0, 0, 0, 0],
          [0, 0, 1, 0, 0, 0],
          [0, 0, 0, 1, 0, 0],
          [0, 0, 0, 0, 1, 0]]
     print("\n")
     PWMW(A,5,6)
```

```
   0  0  0  0  0  0  0
   0  1  1  1  1  1  1
   0  1  2  2  2  2  2
   0  1  2  3  3  3  3
   0  1  2  3  4  4  4
   0  1  2  3  4  5  5
[(1, 1),
 (1, 2),
 (2, 2),
 (2, 3),
 (3, 3),
 (3, 4),
 (4, 4),
 (4, 5),
 (5, 5),
 (5, 6)]
```

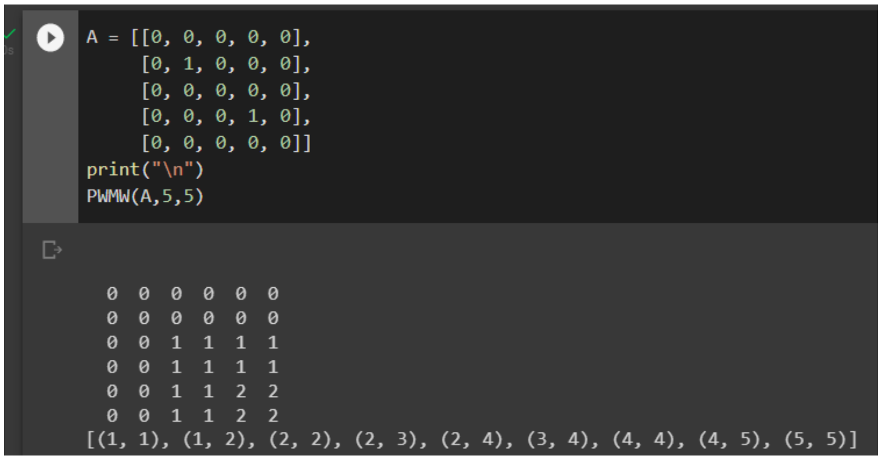


```
A = [[1, 0, 0, 0, 0],
     [0, 1, 0, 0, 0],
     [0, 0, 1, 0, 0],
     [0, 0, 0, 1, 0],
     [0, 0, 0, 0, 1]]
print("\n")
PWMW(A,5,5)
```

```
   0  0  0  0  0  0
   0  1  1  1  1  1
   0  1  2  2  2  2
   0  1  2  3  3  3
   0  1  2  3  4  4
   0  1  2  3  4  5
[(1, 1), (1, 2), (2, 2), (2, 3), (3, 3), (3, 4), (4, 4), (4, 5), (5, 5)]
```

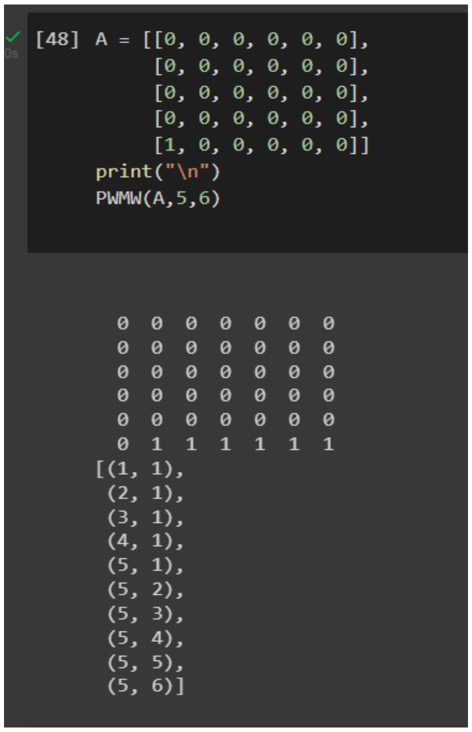


```
[45] A = [[0, 0, 0, 0, 0],
          [0, 1, 0, 0, 0],
          [0, 0, 1, 0, 0],
          [0, 0, 0, 1, 0],
          [0, 0, 0, 0, 0]]
     print("\n")
     PWMW(A,5,5)


      0  0  0  0  0  0
      0  0  0  0  0  0
      0  0  1  1  1  1
      0  0  1  2  2  2
      0  0  1  2  3  3
      0  0  1  2  3  3
     [(1, 1), (1, 2), (2, 2), (2, 3), (3, 3), (3, 4), (4, 4), (4, 5), (5, 5)]
```



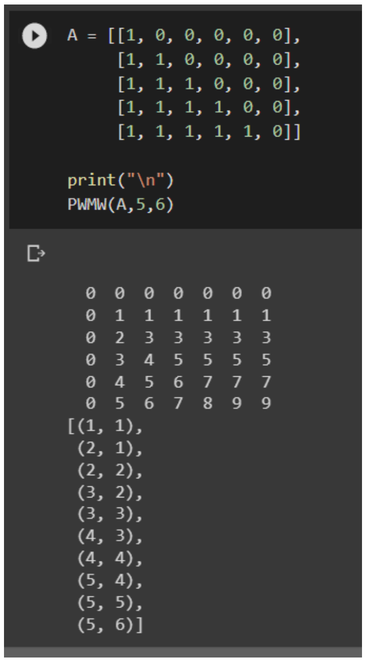
```
A = [[0, 0, 0, 0, 0],
     [0, 1, 0, 0, 0],
     [0, 0, 0, 0, 0],
     [0, 0, 0, 1, 0],
     [0, 0, 0, 0, 0]]
print("\n")
PWMW(A,5,5)


 0  0  0  0  0  0
 0  0  0  0  0  0
 0  0  1  1  1  1
 0  0  1  1  1  1
 0  0  1  1  2  2
 0  0  1  1  2  2
[(1, 1), (1, 2), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4), (4, 5), (5, 5)]
```

- The Matrix has only one weed on the left bottom corner.



```
[48] A = [[0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0],
          [1, 0, 0, 0, 0, 0]]
     print("\n")
     PWMW(A,5,6)


      0  0  0  0  0  0  0
      0  0  0  0  0  0  0
      0  0  0  0  0  0  0
      0  0  0  0  0  0  0
      0  0  0  0  0  0  0
      0  1  1  1  1  1  1
     [(1, 1),
      (2, 1),
      (3, 1),
      (4, 1),
      (5, 1),
      (5, 2),
      (5, 3),
      (5, 4),
      (5, 5),
      (5, 6)]
```

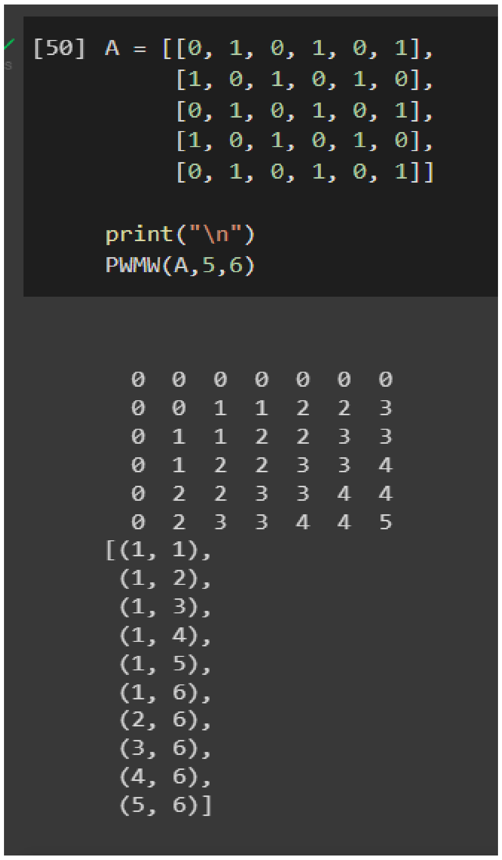
- Only one side of the diagonal has weeds.



```
A = [[1, 0, 0, 0, 0, 0],
     [1, 1, 0, 0, 0, 0],
     [1, 1, 1, 0, 0, 0],
     [1, 1, 1, 1, 0, 0],
     [1, 1, 1, 1, 1, 0]]

print("\n")
PWMW(A,5,6)
```

```
  0  0  0  0  0  0  0
  0  1  1  1  1  1  1
  0  2  3  3  3  3  3
  0  3  4  5  5  5  5
  0  4  5  6  7  7  7
  0  5  6  7  8  9  9
[(1, 1),
 (2, 1),
 (2, 2),
 (3, 2),
 (3, 3),
 (4, 3),
 (4, 4),
 (5, 4),
 (5, 5),
 (5, 6)]
```

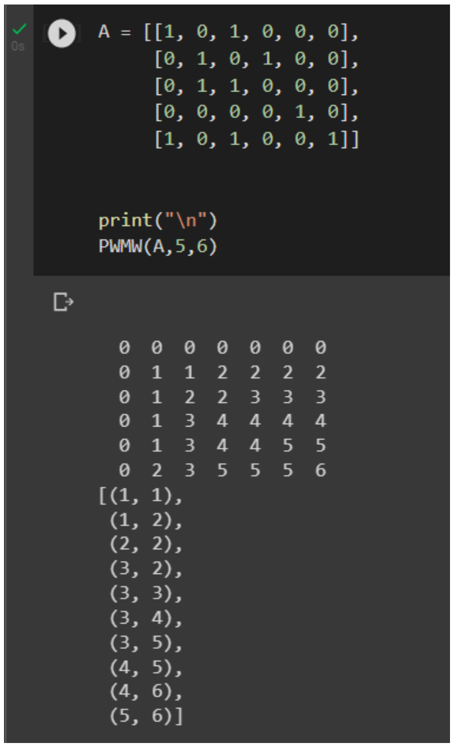
- One cell has weed, one hasn't, one has, one hasn't…



```
[50] A = [[0, 1, 0, 1, 0, 1],
          [1, 0, 1, 0, 1, 0],
          [0, 1, 0, 1, 0, 1],
          [1, 0, 1, 0, 1, 0],
          [0, 1, 0, 1, 0, 1]]

     print("\n")
     PWMW(A,5,6)
```

```
  0  0  0  0  0  0  0
  0  0  1  1  2  2  3
  0  1  1  2  2  3  3
  0  1  2  2  3  3  4
  0  2  2  3  3  4  4
  0  2  3  3  4  4  5
[(1, 1),
 (1, 2),
 (1, 3),
 (1, 4),
 (1, 5),
 (1, 6),
 (2, 6),
 (3, 6),
 (4, 6),
 (5, 6)]
```

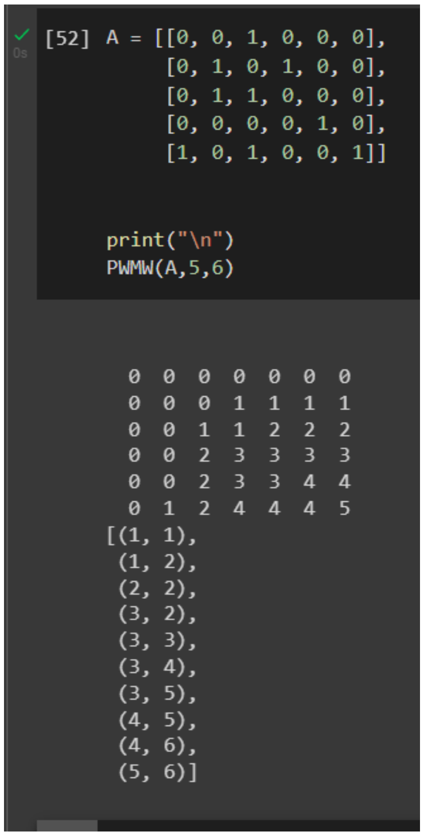
- Case in the homework document.



```
A = [[1, 0, 1, 0, 0, 0],
     [0, 1, 0, 1, 0, 0],
     [0, 1, 1, 0, 0, 0],
     [0, 0, 0, 0, 1, 0],
     [1, 0, 1, 0, 0, 1]]


print("\n")
PWMW(A,5,6)
```

```
  0  0  0  0  0  0  0
  0  1  1  2  2  2  2
  0  1  2  2  3  3  3
  0  1  3  4  4  4  4
  0  1  3  4  4  5  5
  0  2  3  5  5  5  6
[(1, 1),
 (1, 2),
 (2, 2),
 (3, 2),
 (3, 3),
 (3, 4),
 (3, 5),
 (4, 5),
 (4, 6),
 (5, 6)]
```

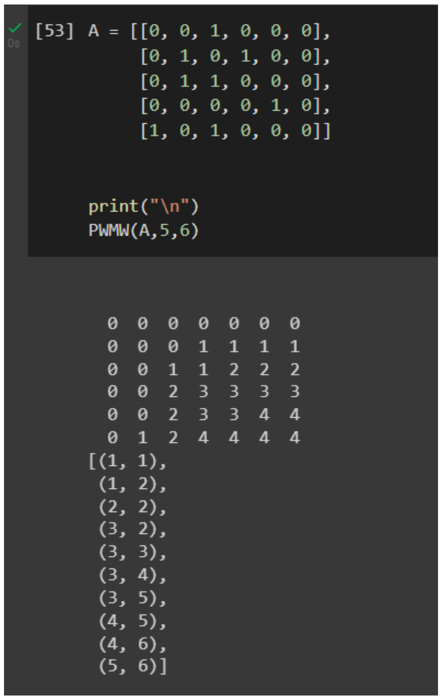
- Case in the homework document, but (1,1) is empty.



```
[52] A = [[0, 0, 1, 0, 0, 0],
          [0, 1, 0, 1, 0, 0],
          [0, 1, 1, 0, 0, 0],
          [0, 0, 0, 0, 1, 0],
          [1, 0, 1, 0, 0, 1]]


     print("\n")
     PWMW(A,5,6)
```

```
  0  0  0  0  0  0  0
  0  0  0  1  1  1  1
  0  0  1  1  2  2  2
  0  0  2  3  3  3  3
  0  0  2  3  3  4  4
  0  1  2  4  4  4  5
[(1, 1),
 (1, 2),
 (2, 2),
 (3, 2),
 (3, 3),
 (3, 4),
 (3, 5),
 (4, 5),
 (4, 6),
 (5, 6)]
```

- Case in the homework document, but (1,1) and (5,6) is empty.



```
[53] A = [[0, 0, 1, 0, 0, 0],
          [0, 1, 0, 1, 0, 0],
          [0, 1, 1, 0, 0, 0],
          [0, 0, 0, 0, 1, 0],
          [1, 0, 1, 0, 0, 0]]


     print("\n")
     PWMW(A,5,6)
```

```
   0  0  0  0  0  0  0
   0  0  0  1  1  1  1
   0  0  1  1  2  2  2
   0  0  2  3  3  3  3
   0  0  2  3  3  4  4
   0  1  2  4  4  4  4
[(1, 1),
 (1, 2),
 (2, 2),
 (3, 2),
 (3, 3),
 (3, 4),
 (3, 5),
 (4, 5),
 (4, 6),
 (5, 6)]
```

I tested all the results. I got what I expected. My algorithm passes the black box testing.

White Box Testing

To show that I identified the decision points and the cases I can encounter while solving the problem.
The cases I can encounter while solving the problem:

- During filling out the matrix S with the subproblems to the max weed that can be reached for a cell
    1. The upper cell is bigger than the left cell in matrix S.
    2. The left cell is bigger than the upper cell in the matrix S.
    3. The left cell and the upper cell are equal to each other in the matrix S.
- Finding the path by backtracking the matrix S
    1. The left cell is bigger than the upper cell in the matrix S.
    2. The upper cell is bigger than the left cell in the matrix S.
    3. They are equal to each other in the matrix S.
       i. The upper cell is navigable.
       ii. The upper cell is not navigable.

Note:
S[i][j] is a navigable cell if it is a cell among the green ones.

0	0	0	0	0	0	0
0	1	1	2	2	2	2
0	1	2	2	3	3	3
0	1	3	4	4	4	4
0	1	3	4	4	5	5
0	2	3	5	5	5	6

To test those cases, I added some print comments into my code.

The original code is here:

```python
def PWMW(A,numOfRows,numOfCols):
  S = [[0 for x in range(numOfCols+1)] for x in range(numOfRows+1)]

  for d in range(1,numOfRows+1):
    for e in range(1,numOfCols+1):
      if S[d-1][e] > S[d][e-1]:
        S[d][e] = S[d-1][e] + A[d-1][e-1]
      else:
        S[d][e] = S[d][e-1] + A[d-1][e-1];

  numOfWeeds = S[numOfRows][numOfCols]

  path = []

  i = numOfRows
  j = numOfCols

  while (i != 0 and j != 0):

      if S[i][j-1] > S[i-1][j]: # left cell > upper cell
        path.append((i,j)) # add current cell to the path
        j -= 1 # go left

      else: # left cell = upper cell OR upper cell > left cell
        path.append((i,j)) # add current cell to the path
        if i - 1 > 0:   # if the upper cell is a navigable cell.
          i -= 1 # go up
        else: # go left if the upper cell is not navigable
          j -= 1 # go left

  path = path[::-1]
  return path
```

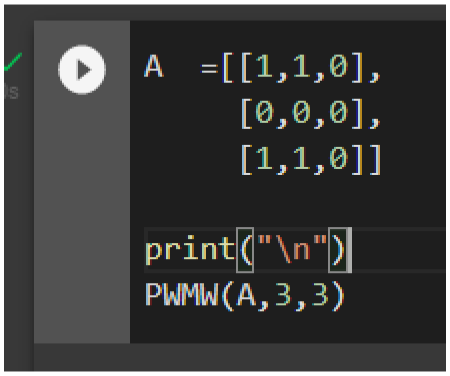
The code with the print statements is here:

```
def PWMW(A,numOfRows,numOfCols):
  S = [[0 for x in range(numOfCols+1)] for x in range(numOfRows+1)]

  for d in range(1,numOfRows+1):
    for e in range(1,numOfCols+1):
      if S[d-1][e] > S[d][e-1]:
        print("Test case 1: the upper cell is bigger than the left cell in
matrix S")
        S[d][e] = S[d-1][e] + A[d-1][e-1]
      else:
        S[d][e] = S[d][e-1] + A[d-1][e-1]
        if S[d-1][e] < S[d][e-1]:
          print("Test case 2: the left cell is bigger than the upper cell
in the matrix S")
        else:
          print("Test case 3: the left cell and the upper cell are equal
to each other in the matrix S")

  numOfWeeds = S[numOfRows][numOfCols]
  print('\n'.join([''.join(['{:3}'.format(item) for item in row]) for row
in S]))
  numOfWeeds = S[numOfRows][numOfCols]


  path = []

  i = numOfRows
  j = numOfCols
  while (i != 0 and j != 0):

      if S[i][j-1] > S[i-1][j]: # left cell > upper cell
        path.append((i,j)) # add current cell to the path
        j -= 1 # go left
        print("Test case 4: The left cell is bigger than the upper cell")

      else: # left cell = upper cell OR upper cell > left cell
        path.append((i,j)) # add current cell to the path
        print("Test case 5: The upper cell is bigger than the left cell OR
they are equal to each other")
        if S[i-1][j] > S[i][j-1]: # if the upper cell is bigger than the
left cell
          print("Test case 6: The upper cell is bigger than the left
cell")
          i -= 1 # go up
        else:
          print("Test case 7: The upper cell and the left cell are equal
to each other")
          if i - 1 > 0: # if the upper cell is navigable
            print("Test case 8: The upper cell and the left cell are equal
to each other and the upper cell is navigable")
            i -= 1 # go up
          else:
            print("Test case 9: The upper cell and the left cell are equal
to each other but the upper cell is NOT navigable")
            j -= 1 # go left
  path = path[::-1]
  return path
```

As you can see, in the while loop, I added an if-else block into the else block. I created this if - else block so that I can see if the upper cell is bigger than the left cell or it is equal to the left cell. (to cover the test cases I wrote above).
After understanding if the upper cell is bigger than the left cell or it is equal to the left,
I added another if else block under the case that they are equal to each other.
By doing this, I test if the upper cell is navigable or not. If it is navigable, I go up, otherwise, I go left.

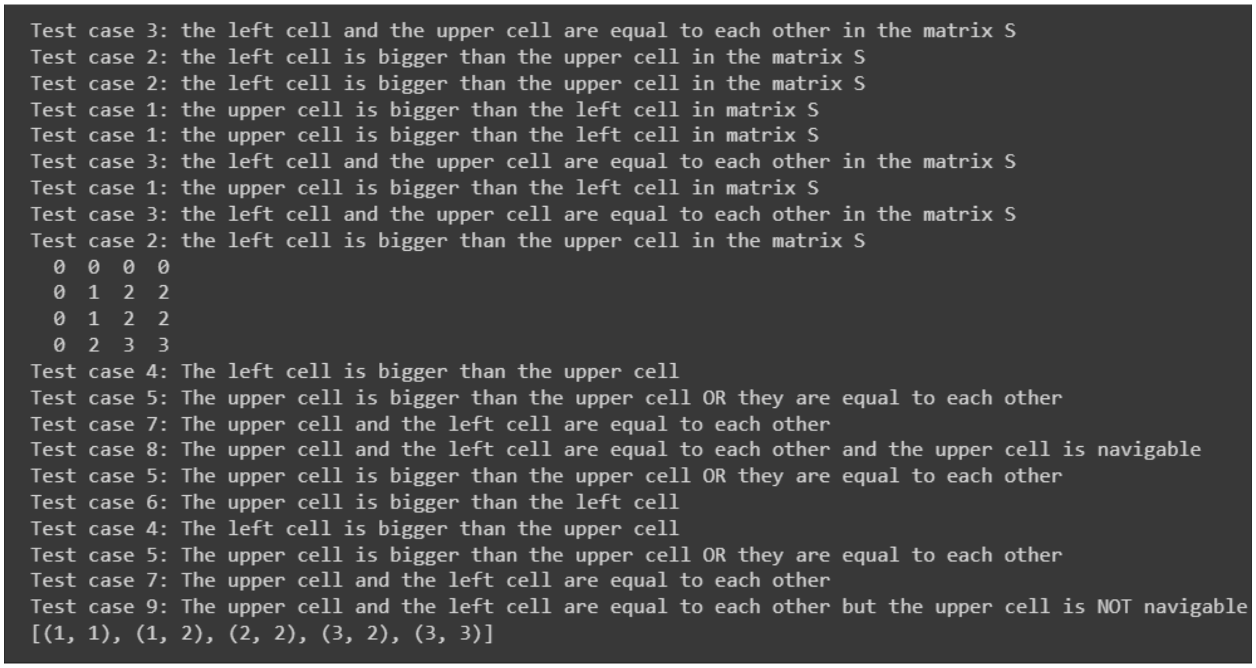
I will run this version of my algorithm and see if all the cases I met.

I prepared a case where I should check all of the test cases.



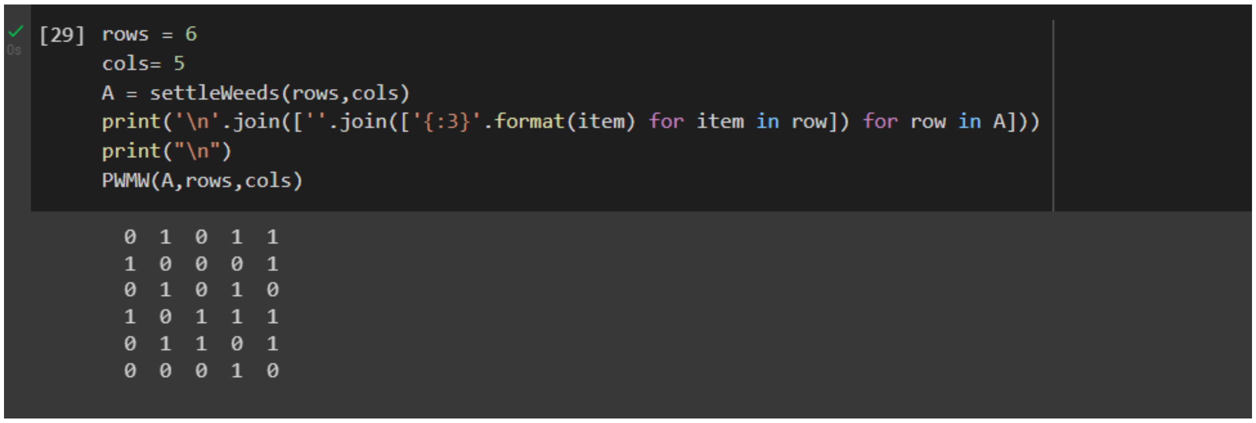
```
A  =[[1,1,0],
     [0,0,0],
     [1,1,0]]

print("\n")
PWMW(A,3,3)
```

Here is the output:



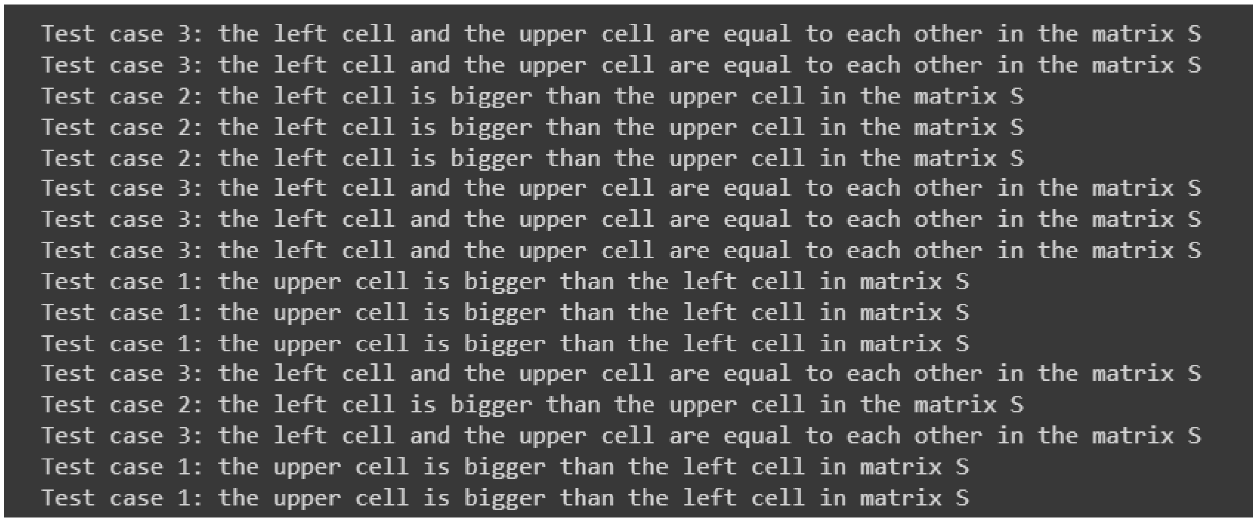
```
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 2: the left cell is bigger than the upper cell in the matrix S
Test case 2: the left cell is bigger than the upper cell in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 2: the left cell is bigger than the upper cell in the matrix S
  0  0  0  0
  0  1  2  2
  0  1  2  2
  0  2  3  3
Test case 4: The left cell is bigger than the upper cell
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 8: The upper cell and the left cell are equal to each other and the upper cell is navigable
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 6: The upper cell is bigger than the left cell
Test case 4: The left cell is bigger than the upper cell
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
[(1, 1), (1, 2), (2, 2), (3, 2), (3, 3)]
```

Another test:

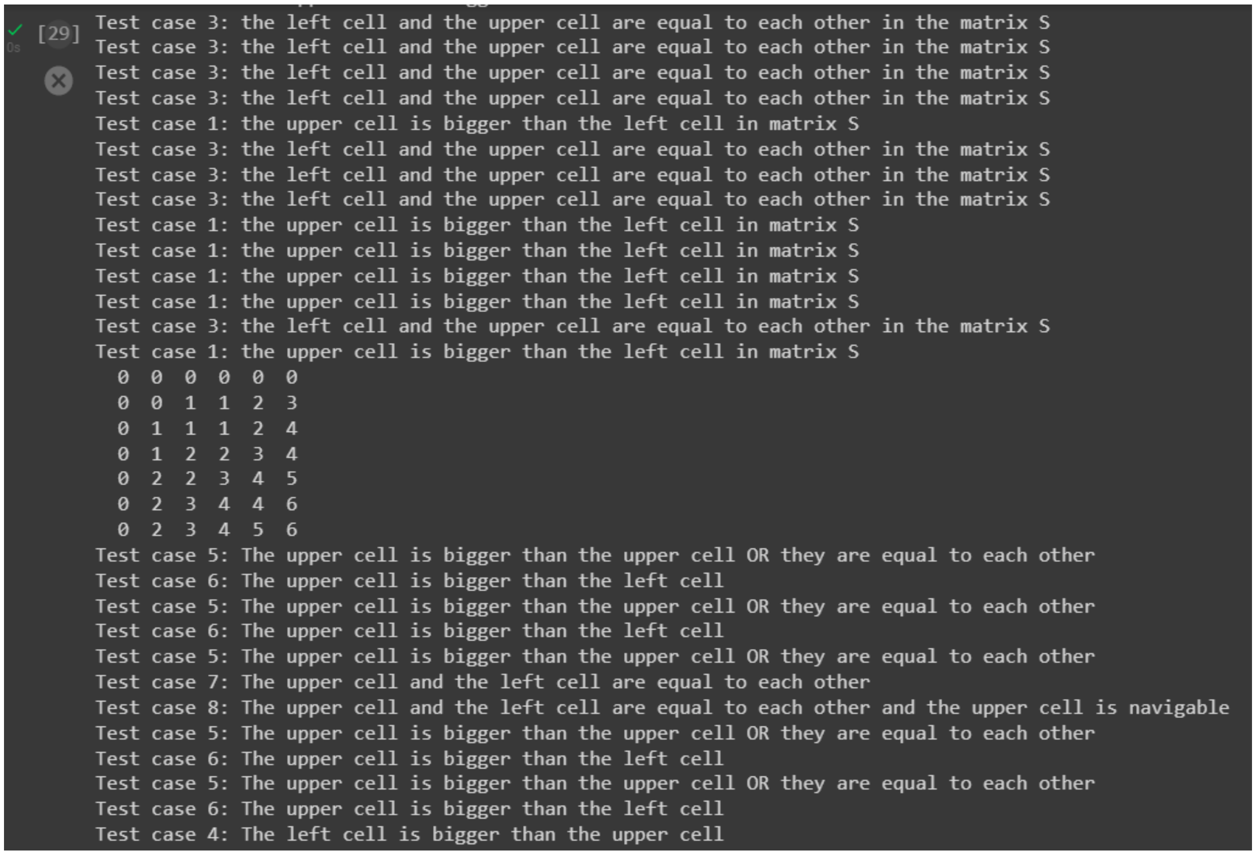


```
[29] rows = 6
     cols= 5
     A = settleWeeds(rows,cols)
     print('\n'.join([''.join(['{:3}'.format(item) for item in row]) for row in A]))
     print("\n")
     PWMW(A,rows,cols)
```

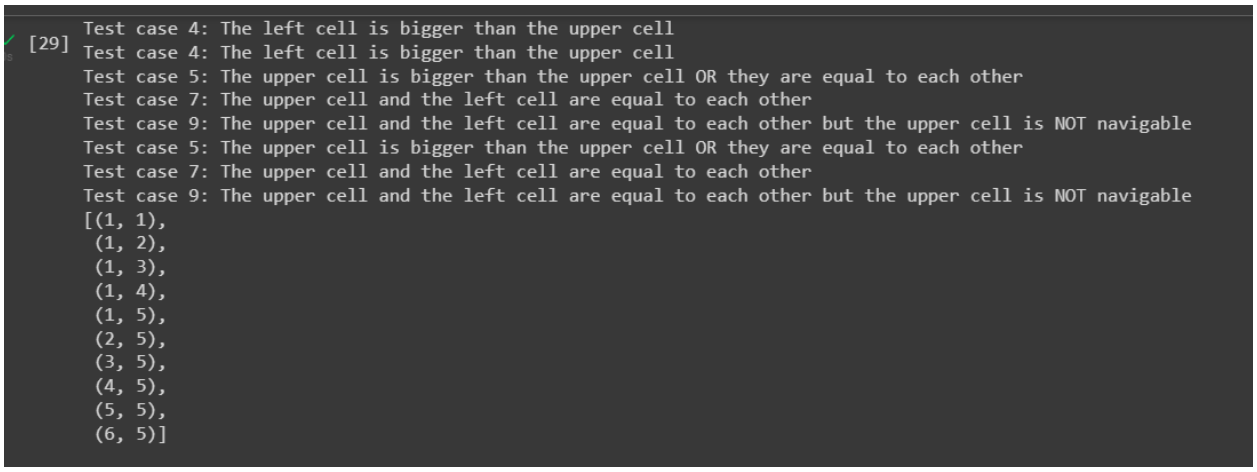
```
  0  1  0  1  1
  1  0  0  0  1
  0  1  0  1  0
  1  0  1  1  1
  0  1  1  0  1
  0  0  0  1  0
```



```
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 2: the left cell is bigger than the upper cell in the matrix S
Test case 2: the left cell is bigger than the upper cell in the matrix S
Test case 2: the left cell is bigger than the upper cell in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 2: the left cell is bigger than the upper cell in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
```

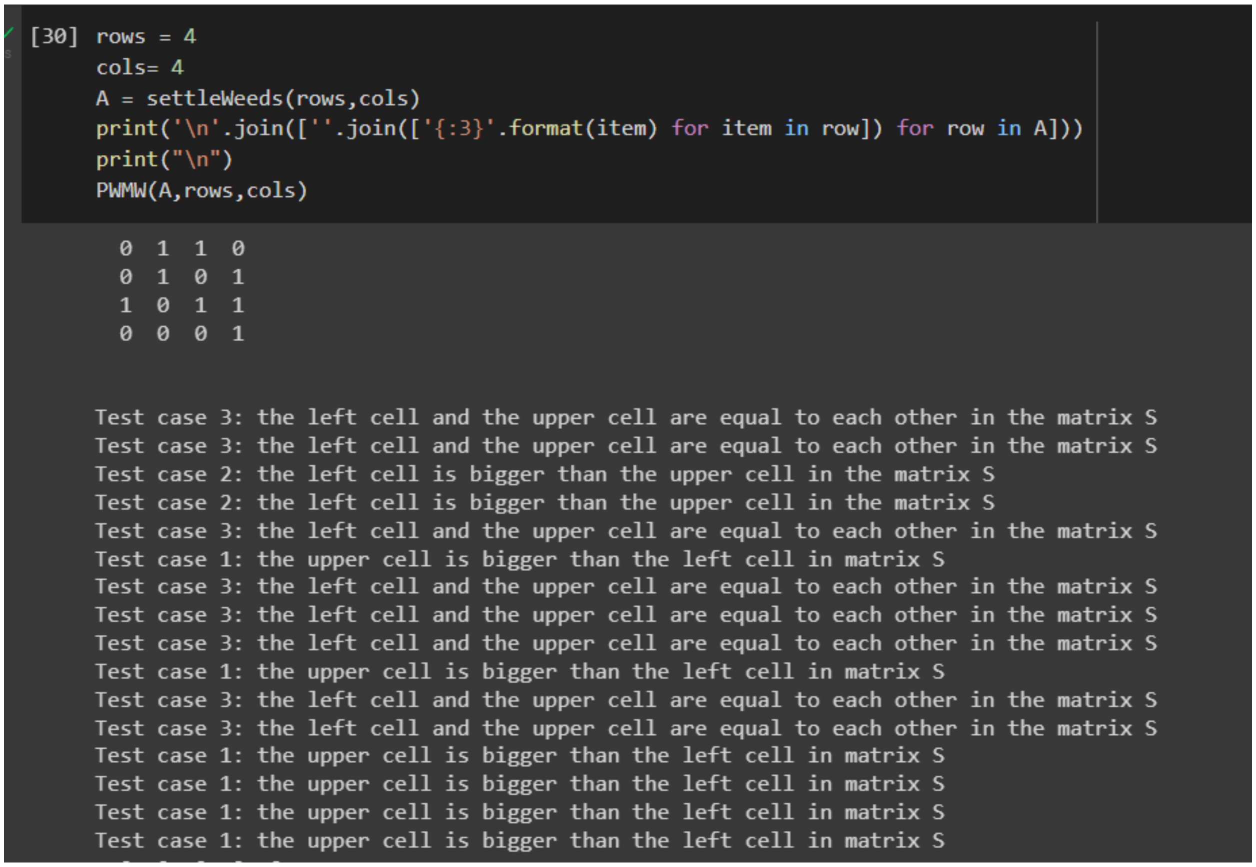


```
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
  0  0  0  0  0  0
  0  0  1  1  2  3
  0  1  1  1  2  4
  0  1  2  2  3  4
  0  2  2  3  4  5
  0  2  3  4  4  6
  0  2  3  4  5  6
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 6: The upper cell is bigger than the left cell
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 6: The upper cell is bigger than the left cell
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 8: The upper cell and the left cell are equal to each other and the upper cell is navigable
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 6: The upper cell is bigger than the left cell
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 6: The upper cell is bigger than the left cell
Test case 4: The left cell is bigger than the upper cell
```



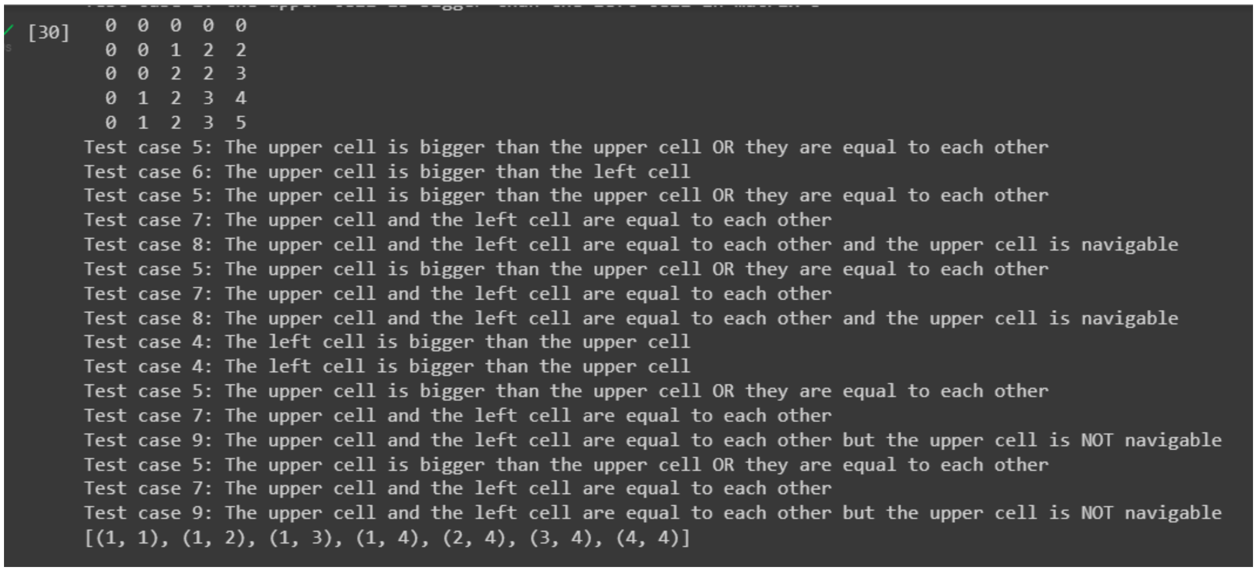
```
Test case 4: The left cell is bigger than the upper cell
Test case 4: The left cell is bigger than the upper cell
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
[(1, 1),
 (1, 2),
 (1, 3),
 (1, 4),
 (1, 5),
 (2, 5),
 (3, 5),
 (4, 5),
 (5, 5),
 (6, 5)]
```

Another test:



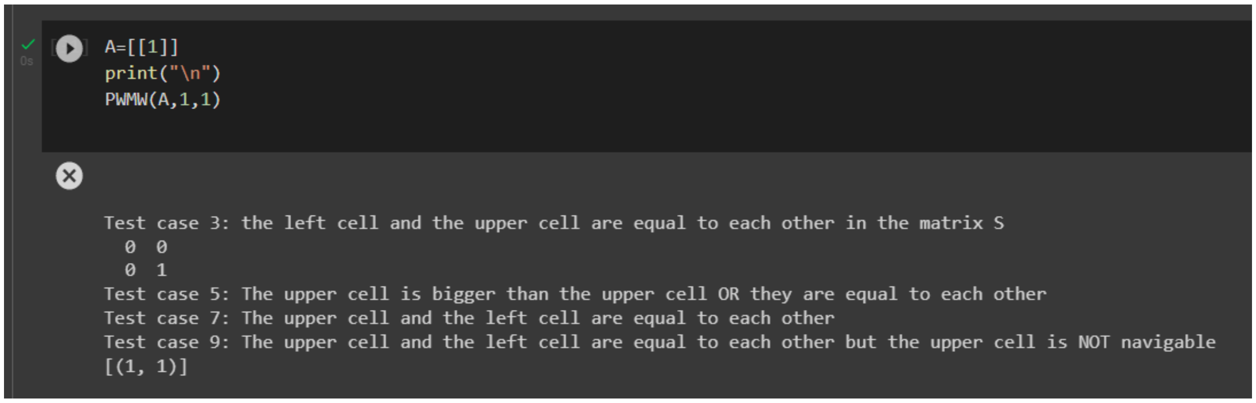
```
[30] rows = 4
     cols= 4
     A = settleWeeds(rows,cols)
     print('\n'.join([''.join(['{:3}'.format(item) for item in row]) for row in A]))
     print("\n")
     PWMW(A,rows,cols)
```

```
  0  1  1  0
  0  1  0  1
  1  0  1  1
  0  0  0  1


Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 2: the left cell is bigger than the upper cell in the matrix S
Test case 2: the left cell is bigger than the upper cell in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
Test case 1: the upper cell is bigger than the left cell in matrix S
```



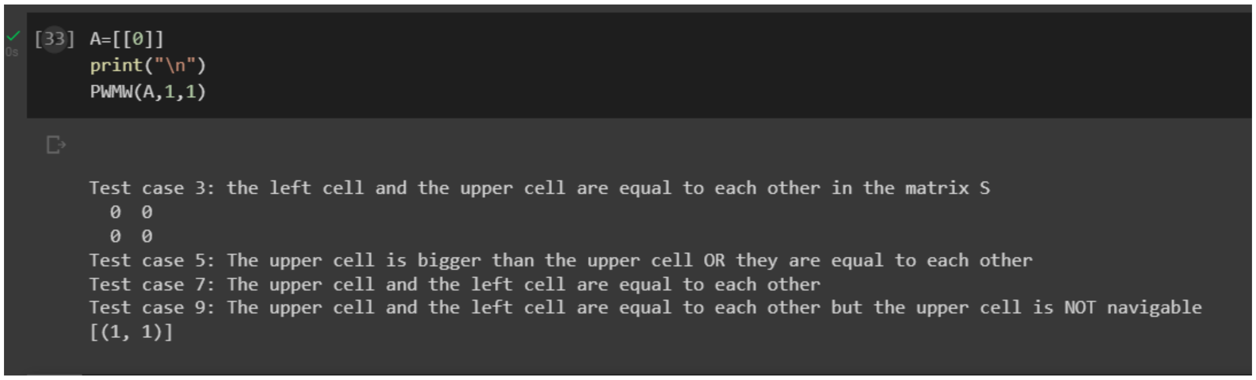
```
[30]
  0  0  0  0  0
  0  0  1  2  2
  0  0  2  2  3
  0  1  2  3  4
  0  1  2  3  5
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 6: The upper cell is bigger than the left cell
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 8: The upper cell and the left cell are equal to each other and the upper cell is navigable
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 8: The upper cell and the left cell are equal to each other and the upper cell is navigable
Test case 4: The left cell is bigger than the upper cell
Test case 4: The left cell is bigger than the upper cell
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
[(1, 1), (1, 2), (1, 3), (1, 4), (2, 4), (3, 4), (4, 4)]
```

The above covers all the cases. I will now test some other matrix which doesn’t cover all the cases.



```
A=[[1]]
print("\n")
PWMW(A,1,1)
```

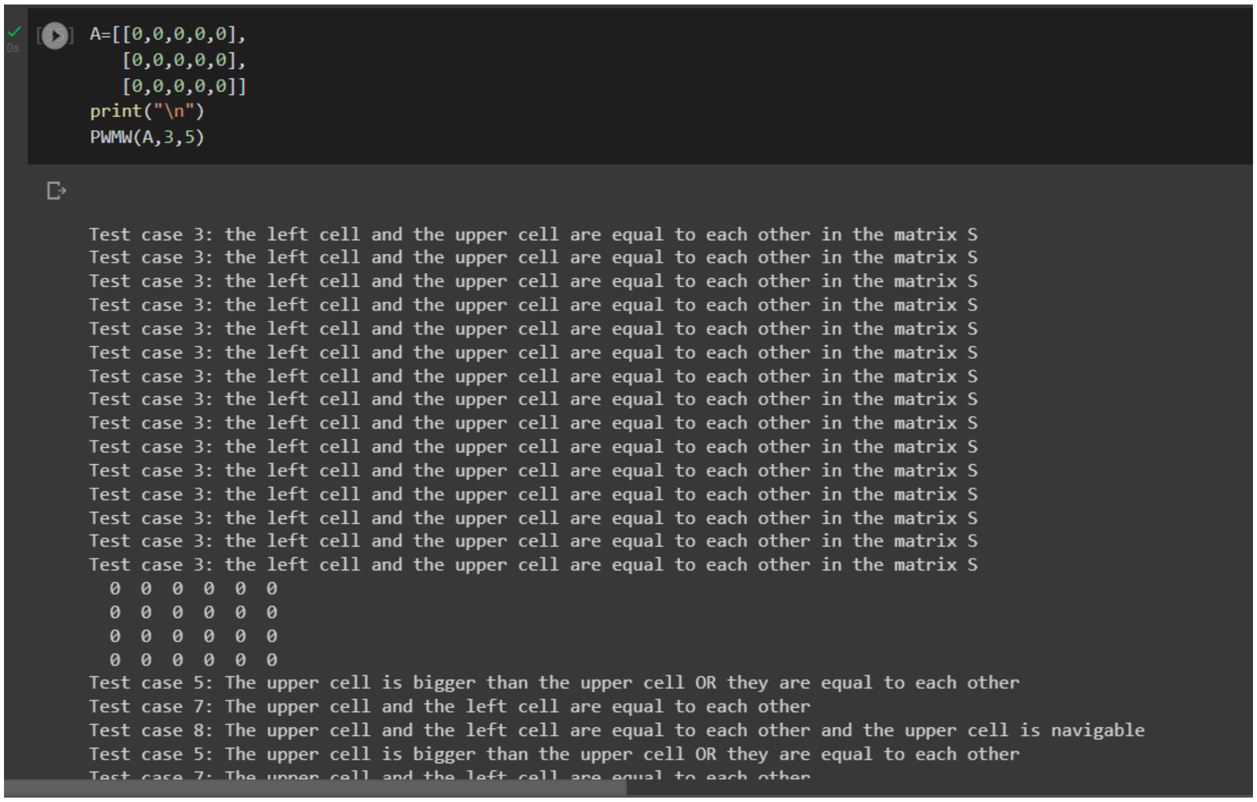
```


Test case 3: the left cell and the upper cell are equal to each other in the matrix S
  0  0
  0  1
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
[(1, 1)]
```



```
A=[[0]]
print("\n")
PWMW(A,1,1)
```

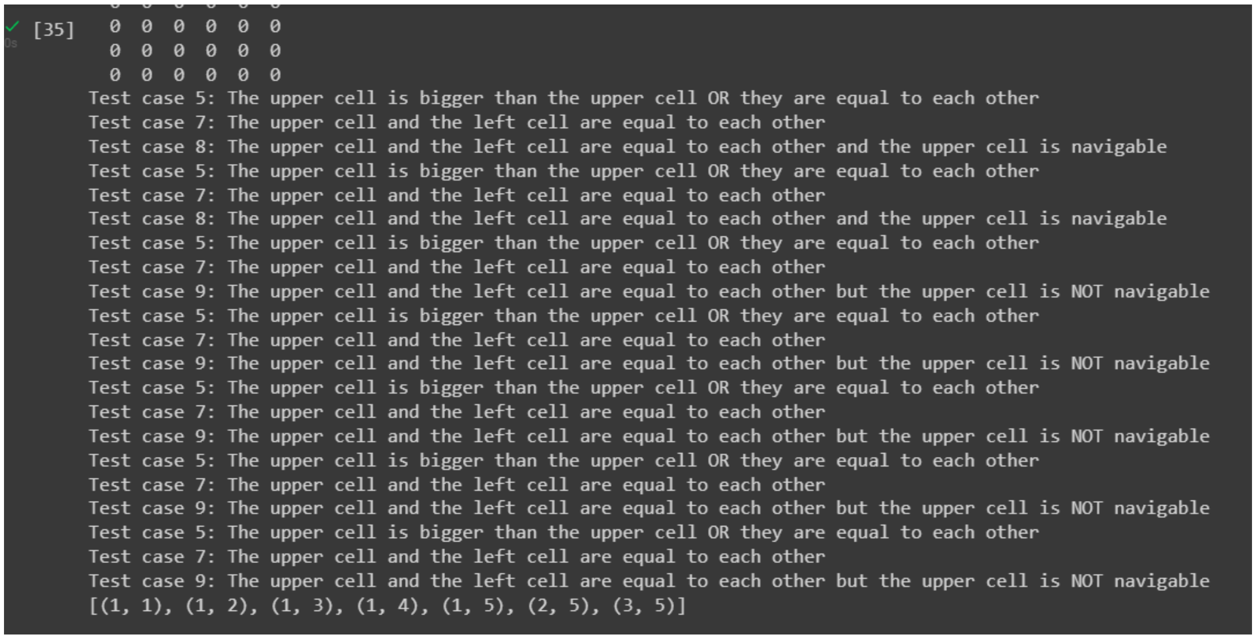
```


Test case 3: the left cell and the upper cell are equal to each other in the matrix S
  0  0
  0  0
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
[(1, 1)]
```

Another test:



```
A=[[0,0,0,0,0],
   [0,0,0,0,0],
   [0,0,0,0,0]]
print("\n")
PWMW(A,3,5)
```

```


Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
Test case 3: the left cell and the upper cell are equal to each other in the matrix S
  0  0  0  0  0  0
  0  0  0  0  0  0
  0  0  0  0  0  0
  0  0  0  0  0  0
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
Test case 8: The upper cell and the left cell are equal to each other and the upper cell is navigable
Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
Test case 7: The upper cell and the left cell are equal to each other
```



```
[35]   0  0  0  0  0  0
       0  0  0  0  0  0
       0  0  0  0  0  0
     Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
     Test case 7: The upper cell and the left cell are equal to each other
     Test case 8: The upper cell and the left cell are equal to each other and the upper cell is navigable
     Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
     Test case 7: The upper cell and the left cell are equal to each other
     Test case 8: The upper cell and the left cell are equal to each other and the upper cell is navigable
     Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
     Test case 7: The upper cell and the left cell are equal to each other
     Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
     Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
     Test case 7: The upper cell and the left cell are equal to each other
     Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
     Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
     Test case 7: The upper cell and the left cell are equal to each other
     Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
     Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
     Test case 7: The upper cell and the left cell are equal to each other
     Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
     Test case 5: The upper cell is bigger than the upper cell OR they are equal to each other
     Test case 7: The upper cell and the left cell are equal to each other
     Test case 9: The upper cell and the left cell are equal to each other but the upper cell is NOT navigable
     [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 5), (3, 5)]
```

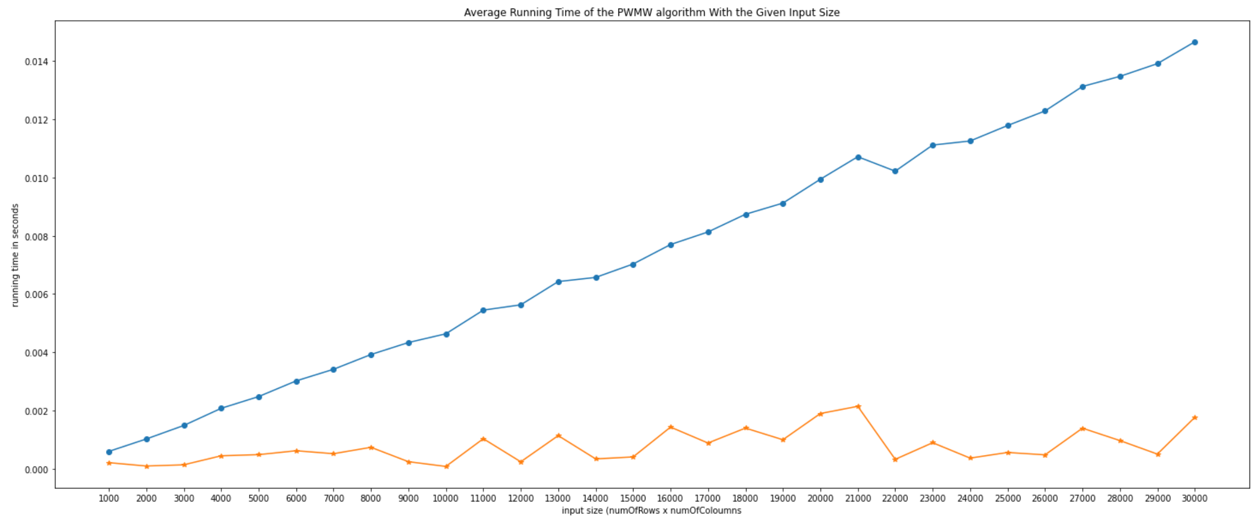
I checked all of the results for either the black box testing or the white box testing. My algorithm gives correct outputs.

Performance Testing

To do this I created 20 instances for each input size from 1000 to 30000. (input size is n x m)
I calculated the mean of the 20 running times for each 30 instances.
I plot the means.

Also I calculated the sds as I did for the means.

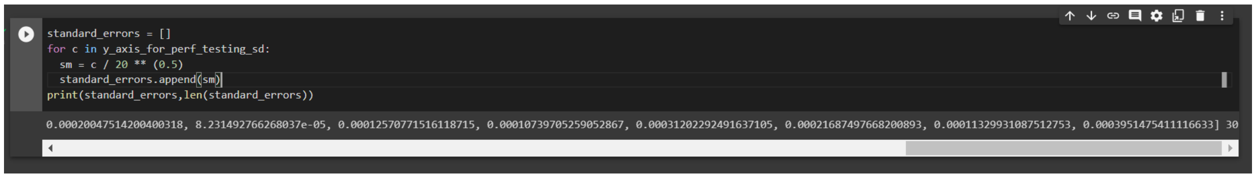
Here is what I get:



In contrast to the plot I got from the experimental results question, This line graph is a more smooth graph. This shows that the randomness and the number of execution times for some instance of an input size gives the probability of getting the guessed result O(mn).

The std's are very low. This means that I got good samples.

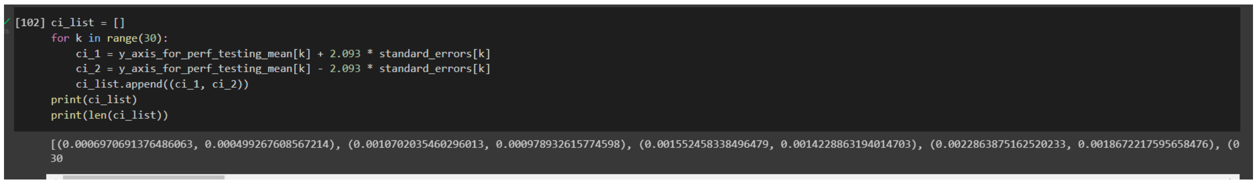
Now, I will calculate the estimated standard error for each 20 runs of the instances.
You can check my work from the ipynp notebook



```
standard_errors = []
for c in y_axis_for_perf_testing_sd:
  sm = c / 20 ** (0.5)
  standard_errors.append(sm)
print(standard_errors,len(standard_errors))
```

```
0.00020047514200400318, 8.231492766268037e-05, 0.00012570771516118715, 0.00010739705259052867, 0.00031202292491637105, 0.00021687497668200893, 0.00011329931087512753, 0.0003951475411116633] 30
```

I want a 95% confidence interval for all of the 30 instance types.
From the table, because n = 20, alpha = 0.95, t = 2.093.



```
[102] ci_list = []
      for k in range(30):
          ci_1 = y_axis_for_perf_testing_mean[k] + 2.093 * standard_errors[k]
          ci_2 = y_axis_for_perf_testing_mean[k] - 2.093 * standard_errors[k]
          ci_list.append((ci_1, ci_2))
      print(ci_list)
      print(len(ci_list))
```

```
[(0.0006970691376486063, 0.0004992676085672l4), (0.0010702035460296013, 0.000978932615774598), (0.001552458338496479, 0.0014228863194014703), (0.0022863875162520233, 0.0018672217595658476), (0
30
```

You can check the intervals from the notebook. For example, for the input size nm = 1000, the actual mean running time is within interval (0.000697069 , 0.000499267) with .95 probability.