Mod 6 Homework: Search In Monotonic Matrices

Goal

In this assignment you will implement a search algorithm which looks for items in a partially sorted matrix of numbers. Let's define a "row-column monotonic" matrix of numbers, as one in which entries increase (or stay the same) along all rows and all columns. For example,

```
Q= [[0.1 1.0 1.7 2.0]
    [0.6 1.4 2.4 3.2]
    [0.7 1.4 3.2 3.6]
    [1.5 1.5 3.6 4.3]]
```

is such a matrix. Your task is to write a function, say, `my_search(Q, item)` which given a row-column monotonic matrix Q and an entry `item`, returns True when the item is present in Q, and False otherwise. There is one additional requirement: your implementation should perform better than simply "unrolling" the matrix into a list and then linearly search for the item.

Sub-problem 1: Implement A Comparison Counting Float

We will measure performance of your search algorithm by counting how many comparisons it makes. To that end, we will derive a custom float class in Python which behaves exactly the same way as an ordinary float, except that it keeps a counter that is incremented whenever a comparison ==, <, >, >=, <=, and != is made

```python
class my_float(float)
   comparison_counter = 0

   def __eq__(self, other):
      comparison_counter += 1
      return super().__eq__(other)

   ### other magic methods ###
```

- Note that in the code fragment above we do not override the `__init__` method. We do not need to because we do not want to alter behavior of `floats` except for the counting of comparisons that have been made.
- We have a variable `comparison_counter` that is outside of the body of the `__init__` constructor. What that means is that the unique address of a single `comparison_counter` variable is shared by all instances of the `my_float` class. This is what we want: whenever any pair of `my_floats` are compared, the unique counter will be bumped up. Such variables, i.e. variables which belong to a class but whose address is shared across all its instances are called *class* or *static* variables. Static variables are initialized before any instances of the class housing them have been created. As shown in the below code snippet, you refer to such variables by the class name followed by "." and the static variable name.

Your next task is to complete the implementaiton of `my_float` class with the remaining magic methods for comparisons >, <, >=, <=, and !=.

Once implemented, you could use `my_float` to measure the number of comparisons performed by a segment of code as follows:

```python
LL = [my_float(2.1), my_float(-3), my_float(7),
                     my_float(5), my_float(1.1)]
snap1 = my_float.comparison_counter
LL.sort()
snap2 = my_float.comparison_counter
```

```
print("{} comparisons performed".format(snap2 - snap1))
```

IMPORTANT NOTE. You are *NOT* allowed to assign to `my_float.comparison_counter` except when implementing the above magic functions. If you do that in questions requiring comparison counting, all your credit for that question will be taken away. In other words, outside of the implementation of the magic methods ==, >,<, >=, <=, !=, you can only read from `my_float.comparison_counter`, and are not allowed to modify it.

Sub-problem 2: Generate Some Monotonic Matrices

One can very easily generate numerical matrices in Python using the `numpy` module and its arrays. For example, this code snippet

```
import numpy as np
N = 5
M = np.zeros([N, N])

for i in range(N):
    for j in range(N):
        M[i,j] = (-1)**(i+j)
```

creates an N x N matrix M of zeros and then fills it with alternating +1 and -1's. A `numpy` array can be sliced in a very similar way as Python lists, but we can do the slicing in two dimentions:

```
M[3:10, :]  # rows with indices 3,...,9 and all columns
M[:, 3:10]  # columns with indices 3,...,9 and all rows
M[2:5, 7:]  # rows with indices 2,3,4 and all columns starting with index 7
# etc.
```

One important difference between `numpy` arrays and Python lists is that slicing returns a view into the array and not a copy as it was for the lists.

There is no need to know more about numpy arrays for this homework. Your very first task is to write a function which creates an example of a row-column monotonic matrix of integers that strictly increase along the rows and columns.

`my_generate_monotonic_matrices`

- input - a number (the number of rows and columns in this square matrix)
- output - a square monotonic matrix

Sub-problem 3: Implement The Simple Row-By-Row Linear Search

For comparison, implement a function called `my_search_linear` which scans a given matrix of `my_floats` entry-by-entry from left to right, top to bottom, until it finds the requested item in which case it returns True. If such item is not found the function should return False.

Sub-problem 4 (The Main Task): Implement An Efficient Search Function

Your method, which should be named `my_search` and take two arguments: a row-column monotonic matrix `Q` of `my_floats`, and an `item` (a number to search for). `my_search` should return True if `item` is in `Q`, and False otherwise. To obtain full credit, your function should perform noticeably better than the exhaustive row-by-row (or column-by-column) search.

You should take some inspiration from the binary search algorithm to solve this problem - we are effectively working with a 2-dimensional sorted list.

You do not have to generate examples of row-column monotonic matrices `Q` of `my_floats`. A function `generate_monotonic_matrix` is provided for that purpose in the starter code.

Summary

The main task of this assignment is the implementation of the `my_search` function which efficiently (i.e., better than the exhaustive row-by-row search) finds a required element in a row-column monotonic matrix, or declares that such an element is not present. In the process of completing this assignment you are asked to implement a comparison counting `float` class and create examples of row-column monotonic matrices.

Submitting

Your task is to implement the required classes/functions:

- `my_float`
- `my_generate_monotonic_matrix`
- `my_search_linear`
- `my_search`

Please implement these items in a single file called `my_functions.py`. The other file is provided for convenience.

Students must submit to Mimir individually by the due date (typically, the Wednesday after this module at 11:59 pm EST) to receive credit.

Grading

- 15 - `my_float` test 1
- 15 - `my_float` test 2
- 15 - `my_generate_monotonic_matrix`
- 15 - `my_search_linear`
- 20 - `my_search`
- 20 - `my_search`

Feedback

If you have any feedback on this assignment, please leave it here.

We check this feedback regularly, and it has resulted in many improvements.