COMS 4701 Artificial Intelligence

Homework - Coding

Due date: Feb 2nd, 2021

Read Carefully:

- You must name your file `homework1.py`. Any submission that does not follow this naming will not be graded and will receive a zero.
- Do not import any libraries other than `NumPy`.
- A skeleton of each function has been provided to you. You are expected to ONLY write code in the functions and blocks specified. In any case, DO NOT modify the function signatures, or any code that is not specified to be modifiable for any reason. Also, though you may modify the main function to test other cases, do not turn in an assignment with a modified main function. This will be run by our autograder, so any unexpected modifications that make it malfunction will receive a zero. Test cases have been provided in the main function in the skeleton code.
- Only Python 3.x versions will be graded.
- To receive points, make sure your code runs. We recommend using Spyder, Pycharm or Google Colab. They all allow you to download .py files. Be aware that if you write your code in some platforms like Codio and copy and paste it in a text file, there may be spurious characters in your code, and your code will not compile. Always ensure that your .py compiles. Code that does not run will not receive any points.
- **Submission Instructions:** Please submit the `homework1.py` file on Gradescope under the assignment Homework1 Programming.

Problem 1: Comma Separated Factorials

Write a Python function to return a string of first `k` factorials (starting from `1!`), separated by commas, in reverse order. Use a single for loop to achieve this. You may use a helper function.

For example, for `k = 8`, the output would be the following string: `"40320,5040,720,120,24,6,2,1"`

Problem 2: List Manipulation

Write a single Python function that takes two lists (`x` and `y`) as inputs for each of the following:

a. Return the resulting list after the following operations: (i) sort `y` by descending order, and (ii) delete the last element of this sorted list.

b. Return the reverse list of `x`.

c. Return the list of unique elements that you get by concatenating `x` and `y`, in ascending order.

d. Return a single list consisting of `x` and `y` as its two elements.

Problem 3: Set Manipulation

Write a single Python function that takes three sets, `x`, `y`, and `z`, as inputs for each of the following:

a. Return the union set of all three sets.

b. Return the intersection set of all three sets.

c. Return a set of elements that belong to only a single set out of the three sets.

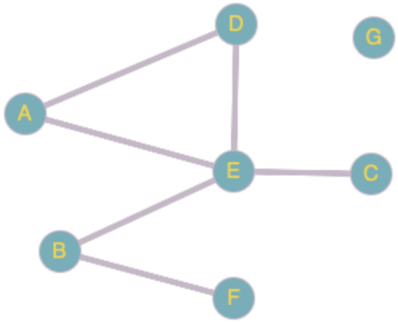
Problem 4: Under Attack

a. Consider the following 5x5 mini-chess board. Assuming that the top left corner is `(0, 0)` and any cell in the board is represented with `(i, j)` where `i` is the row and `j` is column, create a 2D `NumPy` array where black knights are represented with integer `1`, white pawn is represented with `2`, and empty spaces are represented with `0`. Return this array in the associated function.

b. Now suppose you are controlling the white pawn. Write a Python function that takes in **any** such board configuration `x` (i.e. a `NumPy` array) as the only argument and returns a list of indices of the black knights that currently threaten the white pawn (e.g. `[(0, 1), (0, 3), ...]`). This should work for all cases (i.e. any 5x5 board with black knights as 1s and a single white pawn as 2). You do not have to worry about the order of tuples in the list. To see how a knight moves in chess, check https://www.youtube.com/watch?v=VGoT8FR0O$_8$.

Problem 5: Graphs

Consider the following **undirected graph**.



We can represent this graph with the following Python dictionary, often referred to as an adjacency list:

```
graph = {
   "A":  ["D", "E"],
   "B":  ["E", "F"],
   "C":  ["E"],
   "D":  ["A", "E"],
   "E":  ["A", "B", "C", "D"],
   "F":  ["B"],
   "G":  []
 }
```

Write a single Python function that takes any such dictionary x as input, for each of the following:

a. Return the number of isolated nodes (i.e. nodes without any connections).

b. Return the number of non-isolated nodes (i.e. nodes with connections).

c. Return the unique edges as a list of tuples (e.g. `[("A", "D"), ("A", "E"), ...]`). You do not have to worry about the order of tuples in the list.

d. Return a 2D `NumPy` adjacency matrix representing the same graph. Use a `1` if there is an edge between two nodes, and a `0` if there is not. For example, the `(0, 0)` index of the adjacency matrix will have the value `0` because there is no edge between `A` and `A`, and the `(0, 3)` index will have the value `1` because there is an edge between `A` and `D`.

Problem 6: Supermarket

Consider the following products and prices in a supermarket:

apple	$5.0
banana	$4.5
carrot	$3.3
kiwi	$7.4
orange	$5.0
mango	$9.1
pineapple	$9.1

Create a Python class implementing **priority queue** from scratch (i.e. do not use any libraries) with `is_empty()`, `push(element)` and `pop()` methods. Remember that a priority queue is a queue where elements with higher priority are dequeued before elements with lower priority. In case priorities are equal, the elements are dequeued based on their order in the queue. For this problem, assume that the priorities are based on the prices of the products.

For example, let's insert 3 elements into the priority queue, and pop elements until there is none left:

```
pq = PriorityQueue()
pq.push("apple")
pq.push("kiwi")
pq.push("orange")
while not pq.is_empty():
   print(pq.pop())
```

1 FAQs

1. Q: Can we use Python built-in functions?

   A: Yes! For this homework, we are leaving the decision on when to use a built-in function and when to implement something from scratch to you. However, it is important to note that implementing things from scratch is often a good practice, and it will make you become more familiar with Python.

2. Q: Can we use our functions as helper functions for other questions?

   A: Yes, this is allowed! We like modularized code.

3. Q: Can we define as many helper functions as we want?

   A: Yes!

4. Q: Do we need to be worried about the time-complexity of our implementations?

   A: You don't need to be worried about the time and space complexities of your solutions. With that being said, your program should always return in a reasonable time. Unless explicitly specified, you wouldn't lose points for a O(log(n)) vs. O(n) but you would lose points if the program takes forever to run. Some homework down the line might have time-constraints, so keep an eye out for them when applicable.

5. Q: Should we implement solutions for edge cases and/or invalid user input?

   A: No, you can safely assume that the user input will always be valid.

6. Q: If our output matches the expected output file, can we expect 100% from the auto-grader?

A: No. The expected output file is simply there for you to understand the output format; please think about all the possible (valid) cases and try to test your code as much as possible. We have hidden test cases. We can however promise that the input will always be reasonable :)

7. Q: Does it matter if the functions modify the input arguments without working with the copy of the arguments?

   A: One example this question applies to is lists. Python lists are mutable, meaning they can be changed in place. If you alter the input lists within the function scope, the changes will still be reflected in the outer scope of the program.

8. Q: For question <X>, can I implement it using <APPROACH>?

   A: The implementation itself does not matter, as long as you are abiding by the rules and expectations posted in the homework and the functionality is correct.

9. Q: For problem 4B, what can I assume about the board configurations we are trying to solve?

   A: The board configuration (i.e. NumPy array) will always be 5x5, with any number of black knights represented with 1s and a single white pawn represented with 2. The white pawn need not be on the center of the board. Please implement your solution accordingly.