Lab/Homework 4

Deadline: 23:59 pm, Sunday, Dec 3.

What to submit:

A report with answers to each exercise and corresponding python program (.py file), packaged into a zip file. Named zip as “class_name_HW4 ",for example, “AI1-刘晔-HW4”. Please submit to TA.

Requirements on Coding:

1. Adding header to each .py file.

```
""""
xxxx.py
author:
date:
description:
""""
```

2. Please add a space around the operator and after the comma.
3. Add a blank line between code of different functions
4. Indent your code blocks with 4 spaces. Never use tabs or mix tabs and spaces.

Exercise 4.0 MovieAwards (10pts)

Write the function MovieAwards(oscarResults) that takes a set of tuples, where each tuple holds the name of a category and the name of the winning movie, then returns a dictionary mapping each movie to the number of the awards that it won. For example, if we provide the set:

```
 {
   ("Best Picture", "Green Book"),

   ("Best Actor", "Bohemian Rhapsody"),

   ("Best Actress", "The Favourite"),

   ("Film Editing", "Bohemian Rhapsody"),

   ("Best Original Score", "Black Panther"),

   ("Costume Design", "Black Panther"),

   ("Sound Editing", "Bohemian Rhapsody"),

   ("Best Director", "Roma")

 }
```

the program should return:

```
 {
   "Black Panther" : 2,

   "Bohemian Rhapsody" : 3,

   "The Favourite" : 1,

   "Green Book" : 1,

   "Roma" : 1

 }
```

Note 1: Remember that sets and dictionaries are unordered! For the example above, the returned dictionary may be in a different order than what we have shown, and that is ok.

Note 2: Regarding efficiency, your solution needs to run faster than the naive solution using lists instead of some other appropriate, more-efficient data structures.

Exercise 4.1 FriendsOfFriends(10pts)

Background: we can create a dictionary mapping people to sets of their friends. For example, we might say:

```
d = { }

d["jon"] = set(["arya", "tyrion"])

d["tyrion"] = set(["jon", "jaime", "pod"])

d["arya"] = set(["jon"])

d["jaime"] = set(["tyrion", "brienne"])

d["brienne"] = set(["jaime", "pod"])

d["pod"] = set(["tyrion", "brienne", "jaime"])

d["ramsay"] = set()
```

With this in mind, write the function FriendsOfFriends(d) that takes such a dictionary mapping people to sets of friends and returns a new dictionary mapping all the same people to sets of their friends-of-friends. For example, since Tyrion is a friend of Pod, and Jon is a friend of Tyrion, Jon is a friend-of-friend of Pod. This set should exclude any direct friends, so

Jaime does not count as a friend-of-friend of Pod (since he is simply a friend of Pod) despite also being a friend of Tyrion's. Additionally, a person cannot be a friend or a friend-of-friend of themself.

Thus, in this example, FriendsOfFriends should return:

```
{
 'tyrion': {'arya', 'brienne'},
 'pod': {'jon'},
 'brienne': {'tyrion'},
 'arya': {'tyrion'},
 'jon': {'pod', 'jaime'},
 'jaime': {'pod', 'jon'},
 'ramsay': set()}
```

Note 1: you may assume that everyone listed in any of the friend sets also is included as a key in the dictionary.

Note 2: you may **not** assume that if Person1 lists Person2 as a friend, Person2 will list Person1 as a friend! Sometimes friendships are only one-way.

Note 3: Regarding efficiency, your solution needs to run faster than the naive solution using lists instead of some other appropriate, more-efficient data structures.

Exercise 4.2 ContainsPythagoreanTriple(10pts)

Create a program ContainsPythagoreanTriple.py, write the function ContainsPythagoreanTriple(L) that takes a list of positive integers and returns True if there are 3 values (a, b, c) anywhere in the list such that (a, b, c) form a Pythagorean Triple (where a**2 + b**2 == c**2). So [1, 3, 6, 2, 5, 1, 4] returns True because of (3,4,5): 3**2 + 4**2 == 5**2. [1, 3, 6, 2, 1, 4] returns False, because it contains no triple.

A naive solution would be to check every possible triple (a, b, c) in the list. That runs in O(N**3). To make the program more efficient, you may consider the solution run in no worse than O(N**2) time.

Exercise 4.3 MostCommonName(30pts)

Write the function MostCommonName(L), that takes a list of names (such as ["Jane", "Aaron", "Cindy", "Aaron"], and returns the most common name in this list (in this case, "Aaron"). If there is more than one such name, return a set of the most common names. So mostCommonName(["Jane", "Aaron", "Jane", "Cindy", "Aaron"]) returns the set {"Aaron", "Jane"}. If the set is empty, return None. Also, treat names case sensitively, so "Jane" and "JANE" are different names.

Example:

```
print(mostCommonName(["Jane", "Aaron", "Cindy", "Aaron"])
        == "Aaron")
print(mostCommonName(["Jane", "Aaron", "Jane", "Cindy", "Aaron"])
        == {"Aaron", "Jane"})
print(mostCommonName(["Cindy"]) == "Cindy")
print(mostCommonName(["Jane", "Aaron", "Cindy"]) == {"Aaron",
"Cindy", "Jane"})
print(mostCommonName([]) == None)
```

Exercise 4.4 SparseMatrix(40pts)

For this exercise, you should use Python dictionaries.

A sparse matrix or sparse array is a matrix in which most of the elements are zero. For more information about sparse matrices, please refer to the material. In this exercise, create a program named SparseMatrix.py, you may use Python dictionaries to represent sparse matrices and deal with computation of sparse matrices.

1. Define a function named ToSparseMatrix(sparsemat), which takes a list representing the sparse matrix, and returns a dictionary with <key, value> pairs representing the index and corresponding value in the sparse matrix.

   Example:

   spmatrix1 = ToSparseMatrix([[0,0,1,0],[1,0,0,0]])

   print(spmatrix1.items())

   >>>dict_items([((0, 2), 1.0), ((1, 0), 1.0)])

2. Define a function named AddSparseMatrix(sparsemat1, sparsemat2, size), which takes two dictionaries representing two sparse matrices, size=N representing NxN matrix, and returns a dictionary of the summation of two sparse matrices.

   Note: please check size = N is valid for input sparsemat

   Example1:

   sparsemat1 = {

   (0,2):1,

   (1,0):1

```
}
sparsemat2 = {
(0,2):1,
(1,0):1
}
spmat = AddSparseMatrix(sparsemat1, sparsemat2, 3)
print(spmat.items())
>>>dict_items([((0, 2), 2.0), ((1, 0), 2.0)])

spmat = AddSparseMatrix(sparsemat1, sparsemat2, 1)
print(spmat.items())
>>>matrix size is not valid!
```

3. Define a function named SubSparseMatrix(sparsemat1, sparsemat2, size), which takes two dictionaries representing two sparse matrices, size=N representing NxN matrix, and returns a dictionary of (sparsemat1-sparsemat2)

   Note: please check size = N is valid for input sparsemat

   .

   Example:

```
sparsemat1 = {
(0,2):1,
(1,0):1
}
sparsemat2 = {
(0,1):1,
(1,0):1
}
spmat = SubSparseMatrix(sparsemat1, sparsemat2, 3)
print(spmat.items())
>>>dict_items([((0, 2), 1.0), ((1, 0), 0.0), ((0, 1), -1.0)])
```

4. Define a function named MulSparseMatrix(sparsemat1, sparsemat2, size), which takes two dictionaries representing two sparse matrices, size=N representing NxN matrix, and returns a dictionary of sparsemat1*sparsemat2.

   Note: please check size = N is valid for input sparsemat

```
sparsemat1 = {
(0,2):1,
(1,0):1
}
sparsemat2 = {
(0,1):1,
(1,0):1,
(3,0):1,
(3,1):1
}
spmat = MulSparseMatrix(sparsemat1, sparsemat2, 4)
print(spmat.items())
>>>dict_items([((1, 1), 1.0)])
```