Assignment 4

Summary of Instructions and Overview

Note	Read the instructions carefully and follow them exactly
Assignment Weight	As outlined in the syllabus, this assignment is worth 5% of your final grade
Due Date	This assignment is due 8am on Monday, Nov 20, 2017
Important	As outlined in the syllabus, late submissions will not be accepted.
	Any Python files with syntax errors will automatically be excluded from grading. Be sure to test your code before you submit it.
	For all functions, make sure you've written good docstrings that include type contract, function description and the preconditions if any.
	This is an individual assignment. Please review the Plagiarism and Academic Integrity policy presented in the first class, i.e. read in detail pages 15 – 19 of course outline. You can find that file on Brightspace under Course Information. While at it, also review Course Policies on pages 13 and 14.

The goal of this assignment is to learn and practice the concepts covered thus far. In particular you will get more practice with (2D) lists and functions. For one of the functions you will also need to learn how to prevent syntax errors i.e. crashes by learning about `try/except` concept for handling exceptions. Given the goals of this assignment, you **cannot** user: dictionaries, sets, deque, bisect module. You **can** though, and in fact should, use `.sort` or `sorted` functions.

As always, you can make multiple submissions, but only the last submission before the deadline will be graded.

For this assignment, you only need to submit one file, called `a4_xxxxxx.py`, where you changed `xxxxxx` to your student number. (No need to submit `a4_xxxxxx.txt` as proof that you tested your function. By now we trust that you learnt and understand the need and importance for testing your functions and code in general).

For this assignment, I provided you with starter code in file called `a4_xxxxxx.py`. Begin by replacing `xxxxxx` in the file name with your student number. Then open the file. Your solution (code) for the assignment must go into that file in the clearly indicated spaces. The file has `main` completely coded for you. Nothing else will go into the mail. It also has some functions completely precoded for you. Your task will me to code the remaining functions. You are not allowed to delete or comment-out any parts of the provided code. The only exception to that rule is the keyword `pass`. Some functions have that keyword. You can remove it once you are done coding that function. You also must follow the instructions given in comments and implied by docstrings. You are however allowed to add your own additional (helper) functions. In, fact you should add at least one more function.

I have provided 5 text files to test and debug your code with as explained in the next section.

As always, your program must run without syntax errors. In particular, when grading your assignment, TAs will first open your file `a4_xxxxxx.py` with IDLE and press Run Module. If pressing Run Module causes any syntax error, the grade for the assignment becomes zero. Furthermore, for each function whose code is missing, I have provided below one or more tests to test your functions with. To obtain a partial mark for these function your solutions may not necessarily give the correct answer on these tests. But if your function gives any kind of Python error when run on the tests provided, that function will be marked with zero points. Finally, each function has to be documented with docstrings.

There is also `a4-more-example-runs.txt` file, giving additional example runs to those given in the next section. The behaviour of all example runs below and in `a4-more-example-runs.txt` should be considered as an implied requirement for the assignment – as always.

Using global variables inside of functions is not allowed. If you do not know what that means, for now, interpret this to mean that inside of your functions you can only use variables that are created in that function. For example, the following code fragment would not be allowed, since variable `x` is not a parameter of function `a_times(a)` nor is it a variable created in function `a_times(a)`. It is a global variable created outside of all functions.

```
def a_times(a):
   result=x*a
   return result
```

```
x=float(input("Give me a number: "))
print(a_times(10))
```

1 Social Networks: friends recommendations and more – 100 points

Have you ever wondered how social networks, such as Facebook, recommend friends to you? Most of the social networks use highly sophisticated algorithms for this, but for this assignment you will implement a fairly naive algorithm to recommend the most likely new friend to users of a social network. In particular, you will recommend the most probable user to befriend based upon the intersection of your common friends. In other words, the user that you will suggest to Person A is the person who has the most friends in common with Person A, but who currently is not friends with Person A.

Five text files have been provided for you to run your program with. Each represents a social network. Three are small test files containing a made-up set of users and their friendships (these files are net1.txt, net2.txt and net3.txt). The two are a subset of a real Facebook dataset, which was obtained from: `fhttps://snap.stanford.edu/data/egonets-Facebook.html`

The format of all five files is the same:
The first line of the file is an integer representing the number of users in the given network.
The following lines are of the form: `user_u` `user_v` where `user_u` and `user_v` are the (non-negative integer) IDs of two users who are friends.
In addition `user_u` is always less than `user_v`
For example, here is a very small file that has 5 users in the social network: 5
0 1
1 2
1 8
2 3
The above is a representation of a social network that contains 5 users.
User ID=0 is friends with User IDs = 1
User ID=1 is friends with User IDs = 0, 2, 8
User ID=2 is friends with User IDs = 1, 3
User ID=3 is friends with User IDs = 2
User ID=8 is friends with User IDs = 1

Spend time studying the above small example to understand the model. For example, notice that since friendship is a symmetric relationship the social media networks in this assignment, if `user_u` is friends with `user_v`, that means that `user_v` is also friends with `user_u`. Such "duplicate" friendships are not present in the file. In particular each friendship is listed once in such way that `user_u` < `user_v`

Also note that, while you can assume that user IDs are sorted, you **cannot** assume that they are consecutive integers differing by one. For example the user IDs above are: 0,1,2,3,8.

You can also assume that in each file the users are sorted from smallest to largest (in the above example you see that users appear as: 0 1 1 2). Specifically, friendships of `user_u` appear before friendships of `user_v` if and only if `user_u` < `user_v`. And also for each user its friends appear sorted, for example for user 1 friendship with friend 2 appears before friendship with friend 4.

To complete the assignment you will have to code the following 9 functions. I strongly recommend you code the in the order given below and do not move onto coding a function until you complete all before. The function descriptions, including what they need to do, are given in `a4_xxxxxx.py`.

1. `create_network(file_name)` (35 points) This is the most important (and possibly the most difficult) function to solve. The function needs to read a file and return a list of tuples representing the social network from the file. In particular the function returns a list of tuples where each tuple has 2 elements: the first is an integer representing an ID of a user and the second is the list of integers representing his/her friends. In the `a4_xxxxxx.py` I refer the list that `create_network` function returns as a `2D-list for friendship network` (although one can argue that is is a 3D list). In addition the `2D-list for friendship network` that **must** `create_network` function returns **must be sorted** by the ID and a list of friends in each tuple also **must be sorted**.

   So for the example above, this function should return the following 2D-list for 2D-list for friendship network:
   `[(0, [1]), (1, [0,2,8]), (2,[1,3]), (3,[2]), (8,[1])]`

   More examples:

```
>>> net1=create_network("net1.txt")
>>> net1
[(0, [1, 2, 3]), (1, [0, 4, 6, 7, 9]), (2, [0, 3, 6, 8, 9]), (3, [0, 2, 8, 9]), (4, [1, 6, 7, 8]),
(5, [9]), (6, [1, 2, 4, 8]), (7, [1, 4, 8]), (8, [2, 3, 4, 6, 7]), (9, [1, 2, 3, 5])]
```

```
>>> net2=create_network("net2.txt")
>>> net2
[(0, [1, 2, 3, 4, 5, 6, 7, 8, 9]), (1, [0, 4, 6, 7, 9]), (2, [0, 3, 6,8, 9]), (3, [0, 2, 8, 9]), (4, [0, 1, 6, 7, 8]),
(5, [0, 9]), (6, [0, 1, 2, 4, 8]),  (7, [0, 1, 4, 8]), (8, [0, 2, 3, 4, 6, 7]), (9, [0, 1, 2, 3, 5])]
>>> net3=create_network("net3.txt")
>>>
[(0, [1, 2, 3, 4, 5, 6, 7, 8, 9]), (1, [0, 4, 6, 7, 9]), (2, [0, 3, 6,8, 9]), (3, [0, 2, 8, 9]), (4, [0, 1, 6, 7, 8]),
 (5, [0, 9]), (6, [0, 1, 2, 4, 8]), (7, [0, 1, 4, 8]), (8, [0, 2, 3, 4, 6, 7]), (9, [0, 1, 2, 3, 5]),
 (100, [112]), (112, [100, 114]), (114, [112])]
>>> net4=create_network("big.txt")
>>> net4[500:502]
[(500, [348, 353, 354, 355, 361, 363, 368, 373, 374, 376, 378, 382, 388, 391, 392, 396, 400, 402, 404, 408, 409, 410,
```

2. `getCommonFriends(user1, user2, network)` (15 points)

```
>>> getCommonFriends(3,1,net1)
[0, 9]
>>> getCommonFriends(0,112,net3)
[]
>>> getCommonFriends(217,163,net4)
[0, 100, 119, 150]
```

3. `recommend(user, network)` (15 points)
Read the docstrings to understand how this function should work. Understand why the given friends are recommended in the examples below including why no friend is recommended for 0 in `net2` and 112 in `net 3`.

```
>>> recommend(6,net1)
7
>>> recommend(4,net2)
2
>>> recommend(0,net2)
>>> recommend(114, net3)
100
>>> recommend(112,net3)
>>> recommend(217,net4)
163
```

4. `k_or_more_friends(network, k)` (5 points)

```
>>> k_or_more_friends(net1, 5)
3
>>> k_or_more_friends(net2, 8)
1
>>> k_or_more_friends(net3, 12)
0
>>> k_or_more_friends(net4, 70)
33
```

5. `maximum_num_friends(network)` (5 points)

```
>>> maximum_num_friends(net1)
5
>>> maximum_num_friends(net2)
9
>>> maximum_num_friends(net3)
9
>>> maximum_num_friends(net4)
347
```

6. `people_with_most_friends(network)` (5 points)

2. She needs to solve `getCommonFriends(user1, user2, network)` with running time $O(n_1 + n_2 + \log n)$ were n is the the total number of users in the network, n_1 is the number of friends of `user1` and n_2 is the number of friends of `user2`. In other words, n_1 =`len(user1)`, n_2 =`len(user2)` and n =`len(network)`.
   Note that on a typical network $O(n_1+n_2+\log n)$ is **much** better than $O(n)$ since a network like Facebook has n roughly 2 billion and the average number of friends per user is 338. Thus the number of operations an $O(n)$ solution would do, would be in the order of a billion, roughly. While the number of operations an `O(num_friends_user1 + num_friends_user2 + log n)` solution would do, would be in the order of, $O(338 + 338 + 21)$, thousand, roughly
   Thus $O(n)$ solutions will not be accepted for the bonus.
   To determine the running times of Python's functions on lists you can use this link (although it is not quite correct as it is amortized, which they incorrectly call average, analysis and not the worst case analysis).
   `https://wiki.python.org/moin/TimeComplexity`
   Again you cannot use sets nor dictionaries nor deque nor bisect module.
3. Finally, to obtain the bonus the student needs to come to my office hours between November 27th (or in extra office hours I will give that week) in order to explain her faster solution.