1) Count Capital Letters

Write a one-liner that will count the number of capital letters in a file. Your code should work even if the file is too big to fit in memory.

Assume you have an open file handle object, such as: with open(SOME_LARGE_FILE) as fh:

count = # your code here

2) You left your computer unlocked and your friend decided to troll you by copying a lot of your files to

random spots all over your file system.

Even worse, she saved the duplicate files with random, embarrassing names ("this_is_like_a_digital_wedgie.txt" was clever, I'll give her that).

Write a function that returns a list of all the duplicate files. We'll check them by hand before actually deleting them, since programmatically deleting files is really scary. To help us confirm that two files are actually duplicates, return a list of tuples ↴ where:

• the first item is the duplicate file • the second item is the original file

For example:

[('/tmp/parker_is_dumb.mpg', '/home/parker/secret_puppy_dance.mpg'), ('/home/trololol.mov', '/etc/apache2/httpd.conf')]

You can assume each file was only duplicated once.

3) You wrote a trendy new messaging app, MeshMessage, to get around flaky cell phone coverage.

Instead of routing texts through cell towers, your app sends messages via the phones of nearby users, passing each message along from one phone to the next until it reaches the intended recipient. (Don't worry—the messages are encrypted while they're in transit.)

Some friends have been using your service, and they're complaining that it takes a long time for messages to get delivered. After some preliminary debugging, you suspect messages might not be taking the most direct route from the sender to the recipient.

Given information about active users on the network, find the shortest route for a message from one user (the sender) to another (the recipient). Return a list of users that make up this route.

There might be a few shortest delivery routes, all with the same length. For now, let's just return any shortest route.

Your network information takes the form of a dictionary mapping username strings to a list of other users nearby:

network = {

'Min' : ['William', 'Jayden', 'Omar'], 'William' : ['Min', 'Noam'],

'Jayden' : ['Min', 'Amelia', 'Ren', 'Noam'], 'Ren' : ['Jayden', 'Omar'],

'Amelia' : ['Jayden', 'Adam', 'Miguel'], 'Adam' : ['Amelia', 'Miguel', 'Sofia', 'Lucas'], 'Miguel' : ['Amelia', 'Adam', 'Liam', 'Nathan'], 'Noam' : ['Nathan', 'Jayden', 'William'], 'Omar' : ['Ren', 'Min', 'Scott'],

...

}

For the network above, a message from Jayden to Adam should have this route:

['Jayden', 'Amelia', 'Adam’]

4) In order to win the prize for most cookies sold, my friend Alice and I are going to merge our Girl Scout Cookies orders and enter as one unit.

Each order is represented by an "order id" (an integer).

We have our lists of orders sorted numerically already, in lists. Write a function to merge our lists of orders into one sorted list.

For example:

my_list = [3, 4, 6, 10, 11, 15]

alices_list = [1, 5, 8, 12, 14, 19]

# Prints [1, 3, 4, 5, 6, 8, 10, 11, 12, 14, 15, 19] print merge_lists(my_list, alices_list)

5) A building has 100 floors. One of the floors is the highest floor an egg can be dropped from without breaking.

If an egg is dropped from above that floor, it will break. If it is dropped from that floor or below, it will be completely undamaged and you can drop the egg again.

Given two eggs, find the highest floor an egg can be dropped from without breaking, with as few drops as possible.