HW02: Filing and Piling
CSE2050 Fall 2017
Instructor: Jeffrey Meunier & Wei Wei
TA: Jenny Blessing, Param Bidja, Yamuna Rajan & Zigeng Wang

1. Introduction

In this exercise, we will complete three functions to simulate three different methods for file organization.

2. Objectives

The purpose of this assignment is to help you:

- Refresh knowledge on list and function in Python.
- Refresh knowledge on reading files in Python.
- Do a little bit study on different file organizing methods.

***Note**: Before you start, if you are not familiar with list, for loop or function in Python, you are recommended to review the sample codes we covered in lecture first.*

3. Background

3.1. Filing and Piling

In our daily lives, we are always dealing with a lot of different files. As a college student, we need to organize our graded paper homework, exams, lecture notes etc. Some of us may just insert documents randomly into a big drawer and whenever we need to fetch a specific document in our storage, we may do a ***linear search***[1] starting from the leftmost document to the right. If we are careful enough, we cannot miss the one we want to find. After using the document, we may randomly insert the document back into the drawer. And whenever we need a certain document again, we can redo the ***linear search*** and ***random insert*** process, which is simple but reliable.

Let's consider the time complexity of the ***random insert*** and ***linear search***. ***Random insert*** takes constant time since what we need to do is just to open the drawer and put the file in randomly. Nothing seems easier than that. But for ***linear search***, given a pile of documents, in order to find a specific one, we have to check each of them ***linearly*** starting from leftmost one up to the one we want. So, how many documents do we have to pass before finding the one we want? If the documents are ***randomly inserted*** into the pile, a certain document has an equal possibility of locating at each position in the pile. So, if there are in total $\boldsymbol{n}$ documents, the expected number of documents we have to pass before finding the specific one is around $\boldsymbol{n/2}$.

[1] In this assignment, we assume ***linear search*** is always conducted from the leftmost location of the pile to right.

Here we can see that the time complexity of ***linear search*** is strongly related to the way how documents are inserted into the pile. We may wonder, are there any other ways of insertions? Just simply recap how we are organizing things, the answer has to be yes. In this assignment, we will implement three other types of insertions (filing schemes) and compare their performance in file organization.

4. Assignment

In the skeleton zip file, you can find a skeleton code, named *filing.py*. All you need to do is to complete the skeleton code based on the following instructions and submit the *filing.py* (together with ***data.txt***) to the Mimir system.

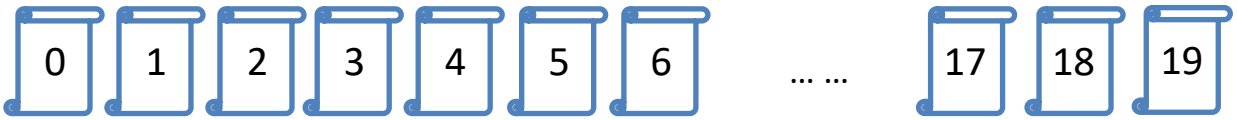
4.1. Leftmost Filing

Now, open the skeleton code. There are three functions for you to complete. The first function is the leftmostfiling function, where people always insert the document to the leftmost position of the pile. The function's skeleton is as the following:

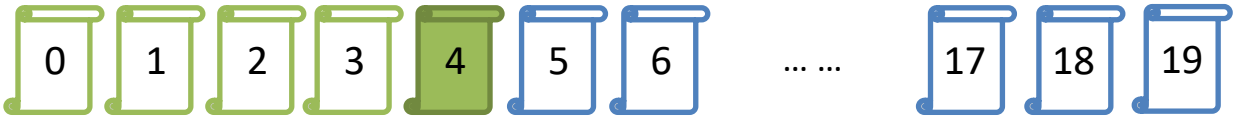
```
def leftmostfiling(L):          # L: list of document IDs acquired from the pile
    M = [x for x in range(20)]  # initialize a pile of 20 documents: [0, 1, 2, …, 19]
    count = 0                   # sum of total number of docs checked before finding each doc in L
    for l in L:
        ### add your code here
    return count
```

In the leftmostfiling function, we first initialize a pile of 20 documents in the form of list. The 20 documents in list **M** are with their unique document IDs from 0 to 19 that initially piled up with ascending order from left to right. The parameter variable **L** is the input of the function. **L** contains a list of document IDs to be acquired from the pile. Variable **count** denotes the sum of total number of documents checked and passed before finding each required document in **L**. In the for loop, we pick each document ID in **L**, and update the **count** and the pile list **M** with leftmost filing scheme. The following is a simple example with **L** = [4, 2, 2], where we will request Doc 4, Doc 2 and Doc 2 one by one.

First, we initialize the documents in **M** with IDs from 0 to 19:

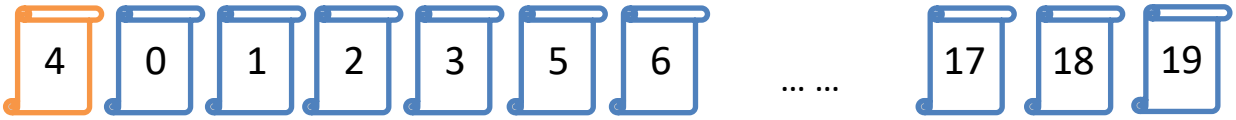


Then, we need to find Doc 4 by using ***linear search*** from left to right in **M** as the following:

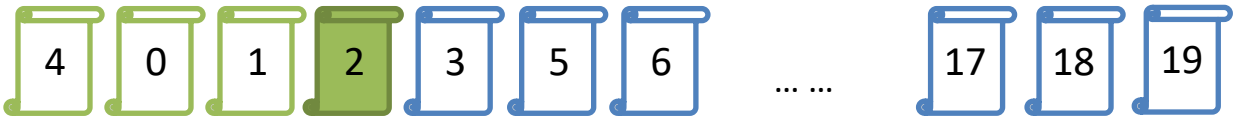


Here, we passed 4 documents Doc 0, Doc 1, Doc 2 and Doc 3 until we found Doc 4. So, we update the value of **count** = 0 + 4 = 4.

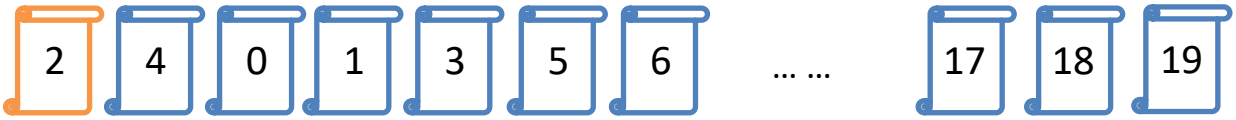
After using Doc 4, we insert Doc 4 back to **M** at the leftmost position,



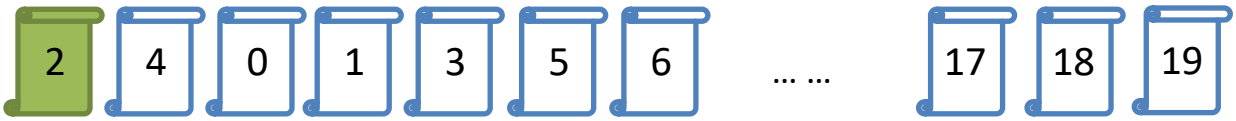
Then, we need to find Doc 2 in the new pile.



Here, we passed 3 documents Doc 4, Doc 0, Doc 1 until we found Doc 2. And we update the value of **count** = 4 + 3 = 7. Similarly, we insert Doc 2 back to **M** as the following,



According to **L**, we request Doc 2 again, where we find Doc 2 is at the leftmost location of **M**. so, we can simple update **count** = 7 + 0 since we do not need to pass any document before we find Doc 2.



After going through the documents in **L** one by one, we return the **count** value 7.

In the test case, there can be a much longer **L**, and what we need to do is to follow the same procedure, request each document in **L**, update **M** and **count** in each iteration.

4.2. Rightmost Filing

The second function we need to implement is the rightmostfiling. The only difference between the leftmost filing and the rightmost filing is that we always insert the document to the rightmost location of the pile. We will also need to calculate the **count** and return its value.

4.3. Fixed Filing

The third function we need to implement is the fixedfiling. Different from leftmost filing and rightmost filing, fixed filing always inserts the document to its original position in the pile. In term of time complexity, the fixed filing needs to pay efforts in both insertion and searching,

while leftmost and rightmost filing only takes time in searching since their insertion is too simple to count.

In fixed filing, for a certain requested document, we need to use ***linear search*** to find the document in **M**. When we end up using the document, we have to use ***linear search*** again to get the file's original location in the pile and put it back. So, here, in each iteration, we need to use ***linear search*** twice and the **count** is somehow doubled.

After implementing the three filing methods, we can compare their performance with different file request patterns with different given test cases in the skeleton code. It is meaningful to mention that, in the first test case, the given code uses a file operation to read each line in file **data.txt** and save them in list **L**. Very likely, in the near future, we need to write our own code to implement different file operations.

```
L = [int(line) for line in open('data.txt')]
```

5. Submit your work to Mimir

Submit your code filing.py (together with **data.txt**) to Mimir after you complete your code. The Mimir will automatically grade your submission based on different unit tests. You can submit your code to Mimir up to **30 times** to refresh your existing score before the submission deadline.

6. Due date

This lab assignment is worth **4 points** in the final grade. It will be due by **11:59pm on Tues. Sep 26th 2017**. A penalty of **10% per day** will be deducted from your grade, starting at 12:00am.

7. Getting help

Start your project early, because you will probably not be able to get timely help in the last few hours before the assignment is due.

- Go to the office hours of instructors and TAs.
    - Prof. Wei Wei: Mon. 2:30 - 3:15pm, Wed. 2:30 - 3:15pm, Thurs. 2:30 - 3:15pm, Fri. 2:30 - 3:15pm @ITE258
    - Jenny Blessing: Fri. 12pm - 2pm @ITE140
    - Param Bidja: Tues. 2pm - 3pm @ITE140
    - Yamuna Rajan: Tues. 11am - 12pm, Wed. 9:30am – 10:30am @ITE140
    - Zigeng Wang: Mon. 3pm - 4pm @ITE140
- Post your questions on Piazza. TAs and many of your classmates may answer your questions.
- Search the answer of your unknown questions on the Internet. Many questions asked by you might have been asked and answered many times online already.