**Project Documentation**

**Overview**

This project aims to analyze and process data from a Twitter dataset provided in a comma-separated file named "twitter.csv." The dataset represents relationships between Twitter users, where each row indicates a user following another user. The primary goals of the program are to identify the top influencers based on follower counts, display follower information, and provide a bonus function that recommends a group of Twitter users based on a specified threshold.

**Implementation Details**

**Data Structures**

MaxHeap Class

The program utilizes a **MaxHeap** class to efficiently maintain and retrieve the top influencers based on follower counts. The class is implemented with methods for pushing elements onto the heap, popping elements from the heap, and performing heapify operations to maintain the max heap property.

**Program Execution**

1. The program starts by reading the Twitter dataset from the file "twitter.csv."
2. It creates a dictionary, **accounts**, to store information about each Twitter account. The dictionary's keys are user IDs, and the values are lists containing the number of followers and a list of follower IDs.
3. The program uses a **MaxHeap** instance, **max\_heap**, to populate the heap with user IDs and their corresponding follower counts.
4. The user is prompted to choose whether to display follower IDs for each account.
5. The program then extracts and displays the top influencers (Twitter accounts with the highest follower counts) in descending order.
6. Additionally, the program calculates and prints the average number of followers per account and the standard deviation.

**Critical Point**

* The code is heavily depend on dictionaries. Time complexity of dictionary will affect badly on the total time complexity. Fortunately, dictionary lookup is O(1) the same apply to insert and delete. The reason behind that is the Python dictionary operates as a hashmap, and its worst-case time complexity is O(n) if the hash function is poorly designed, leading to numerous collisions. However, such scenarios are highly uncommon (rare case), where every added item shares the same hash and ends up in the same chain. This occurrence is exceedingly unlikely in major Python implementations. On average, the time complexity is O(1). In addition, The hash employed by the dictionary depends on the object utilized as a key. Each class has the ability to define its custom **\_\_hash\_\_()** method, and the specific value returned by this method for a given instance becomes the hash used for the dictionary. As a matter of fact, for string and tuple types, Python furnishes its own hash implementation.
* The implementation of dictionaries in CPython is specifically optimized for searching with string keys. Two distinct functions, namely lookdict and lookdict\_string (lookdict\_unicode in Python 3), are available for conducting lookups. Python opts for the string-optimized version initially, and only switches to the more general function when searching for non-string data.

**Bonus Function**

The bonus function, **Closest\_Group**, takes a Twitter account ID, the **accounts** dictionary, and a threshold as input. It aims to recommend a group of Twitter users who have a specified number of common followers with the target account.

1. The function first validates the input ID and creates a dictionary, **target\_dic\_followers**, to represent the followers of the target account.
2. It calculates a numerical threshold if the input threshold is less than 1, converting the percentage to an actual count of followers.
3. The function iterates through all Twitter accounts, excluding the target account, and identifies accounts with a sufficient number of common followers based on the threshold.
4. The recommended group is returned as a list.

**Time and Space Complexity Analysis**

MaxHeap Operations

* **Push Operation:** O(log N)
* **Pop Operation:** O(log N)
* **Heapify Operations:** O(log N)

Populating Max Heap

* **Reading and Processing File:** O(N)
* **Populating Max Heap:** O(N log N)

Bonus Function (Closest\_Group)

* **Creating Target Dictionary:** O(M), where M is the number of followers of the target account.
* **Iterating Over Accounts:** O(N \* F\_max), where N is the number of accounts, and F\_max is the maximum number of followers among all accounts. As a matter of fact, it will be O(N^2) in worst scenario when there is account which has all twitter accounts following him.

**Overall Time Complexity**

The overall time complexity is dominated by the operations related to populating the max heap and performing the Closest\_Group function. Therefore, it is approximately O(N log N) for populating the max heap and O(N \* F\_max) for the bonus function (it will be O(N^2) in worst case).

**Overall Space Complexity**

* **MaxHeap Class:** O(N), where N is the number of accounts in the heap.
* **Accounts Dictionary:** O(N), where N is the number of accounts.
* **Additional Variables:** O(1) (constant space).

The overall space complexity is O(N + N) = O(2N).

**Choosing threshold**

The choice of a specific percentage for the threshold is not fixed and depends on the goals and the characteristics of the dataset. The threshold will effect on **Sensitivity of Recommendations**

* A higher percentage will result in fewer recommendations, as it requires a larger common follower count to be considered a close group.
* A lower percentage will lead to more recommendations but might include users with weaker connections.

Determining an appropriate threshold for recommending new friends based on a common number of Twitter accounts depends on various factors, including the characteristics of the dataset and the desired balance between precision and recall.  
Steps to make this happen:

1. Examine the distribution of follower counts in the dataset.
2. Identify the average, median, and maximum number of followers per account.
3. Understanding the distribution will help in setting a reasonable threshold.

Let’s take a look for top and bottom ten results  
  
A screen shot of numbers

Description automatically generated

Top ten result

A screen shot of a number

Description automatically generated

Bottom ten result

Now, let’s get mean, median and standard deviation.  
Average followers per account: 21

The median index will be 40,653. The value of median is   
Account: 82992977, Followers Number: 8

The standard deviation: 1.0166923656331022   
  
The median of 8 provides insights into the central tendency of follower counts, indicating an asymmetrical distribution of the dataset. In such cases, the median may serve as a more representative measure of the center than the mean. A median of 8 implies that roughly half of the Twitter accounts possess a follower count of 8 or less, while the remaining half exceeds 8. This offers a sense of the central point around which follower counts are dispersed.

Observing that the mean significantly differs from the top ten and even from the median, it suggests that the upper echelons of the data represent special cases. Therefore, optimization of the threshold should prioritize the majority of the data rather than these exceptional cases. Notably, the standard deviation is nearly one, indicating that follower counts closely align with the mean, reflecting a concentrated distribution around the average.

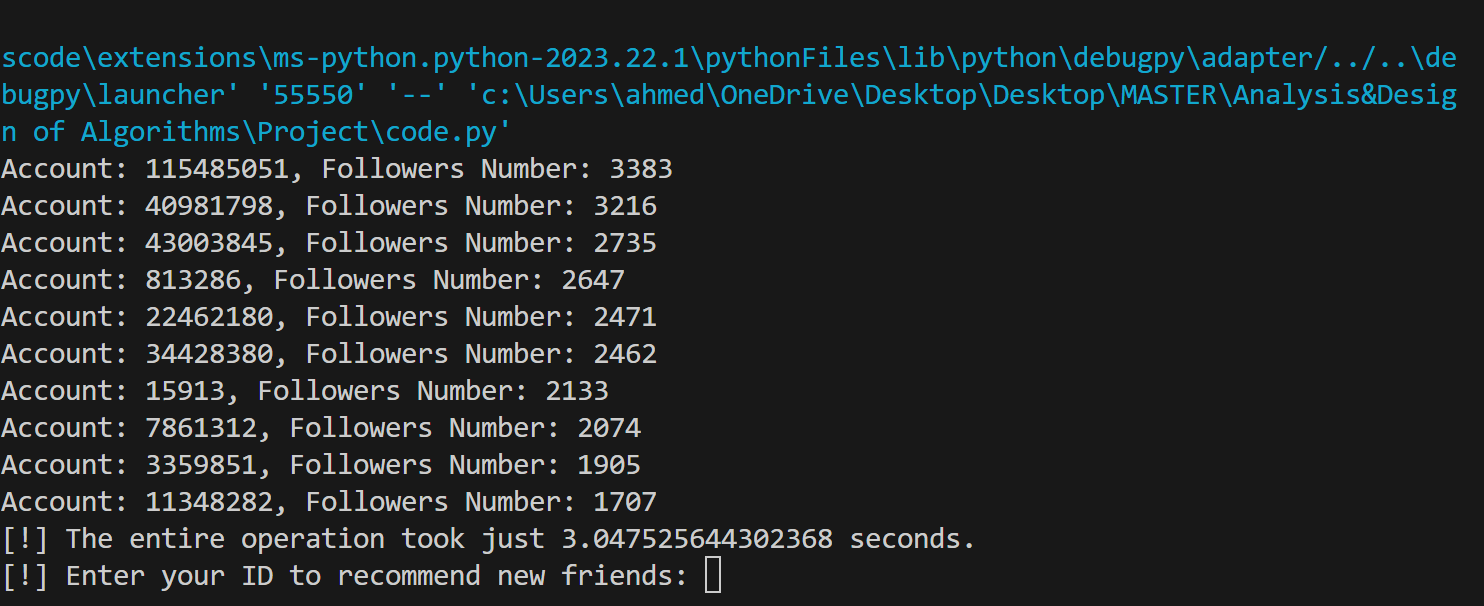
Considering our earlier assertion that "a higher percentage will result in fewer recommendations, especially for accounts with many followers," we should exclude higher percentages from the threshold equation. Indeed, depending on the mean and median of the data, it is advisable to choose a relatively small percentage as the threshold (while avoiding extremes) to ensure recommendations for individuals with varying follower counts. We can Start with a percentage of 10% to 15% as threshold. After a few trials, I believe that 10% is suitable.

**Output**

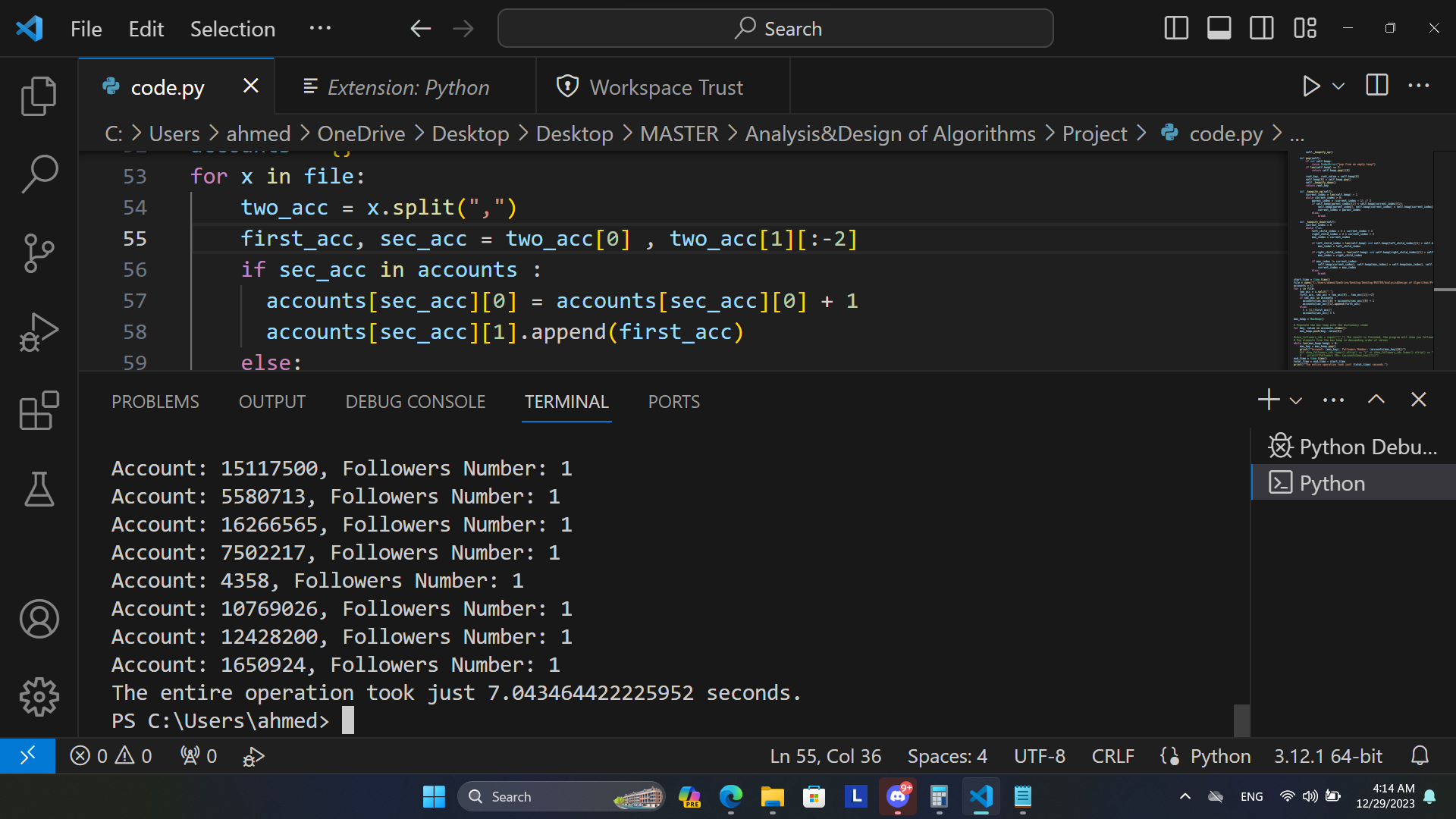
A black background with white text

Description automatically generatedTop of Form

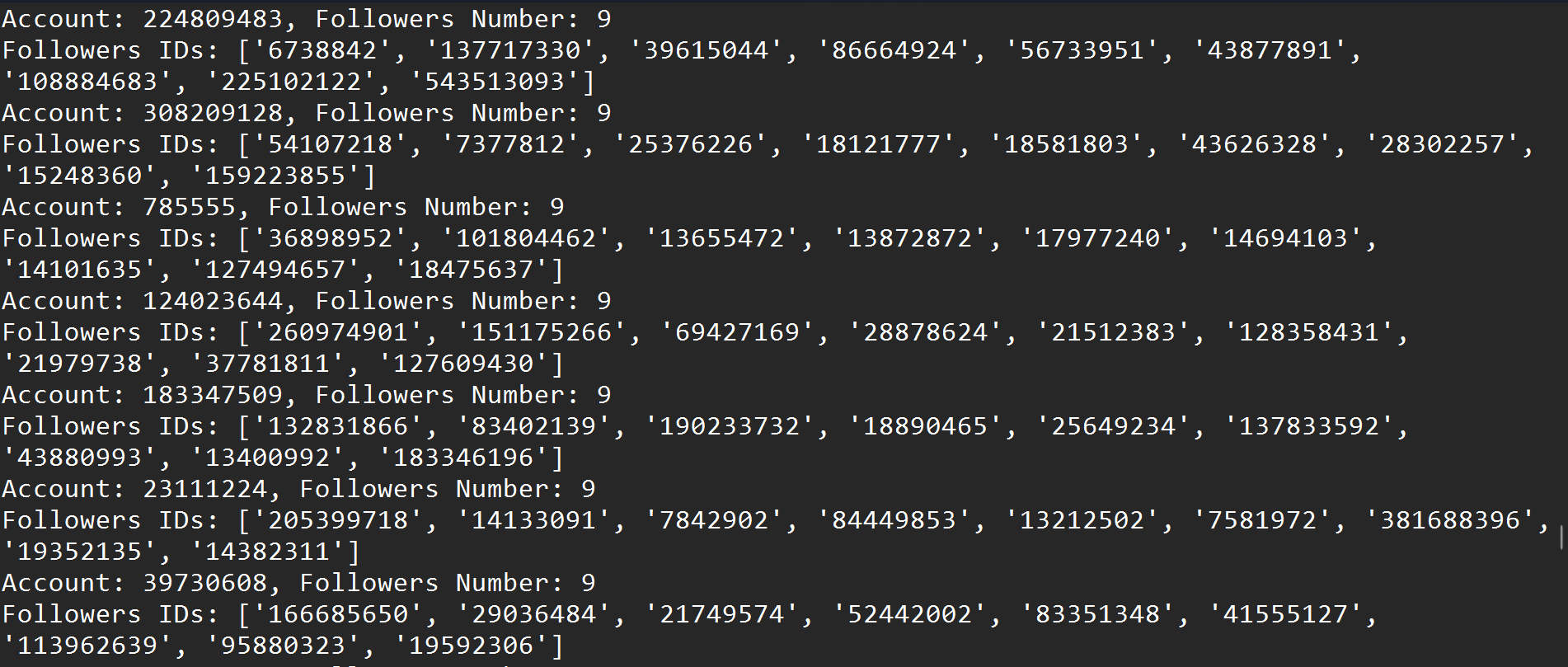
3 sec to get top 10. Of course, this time will vary from device to device. It depend on the CPU speed and storage speed, etc.



7 sec to print all accounts.



10 sec to print and store all these data and calculate mean and standard deviation to text file called Result.txt



Bonus function runs in 2 sec.  
