Report on the Project of COMP1002

Group 7
Group Members:
Alibek KUANTKHAN (20041645D)
Gaukhar TURGAMBEKOVA (20065366D)
Yang KE (20100884D)

Problem Description

There are three main problems in this project: Input Stage, Query Functions, KOL Analysis.

I. Input Stage

The input stage is the basis of all the other functions. It involves reading from ".pickle" files, creating and logging into an account, inputting articles and saving the data. The input stage consists of ***reg()***, ***log()***, ***addFriend()***, ***addArticle()***, ***pr()***, ***save()*** and ***startup()*** functions. Since the relationships of the users and articles are stored in dictionaries (refer to "Data abstraction"), the input stage is responsible for inserting new data into the dictionaries. Every dictionary has unique key-value pairs, e.g. dict6 and dict2 contain articles and their reports; however, dict2 stores only direct reports, while the dict6 stores all reports. Therefore, the ***addArticle()*** function does not simply append the new values, but it checks for data's correctness, data's relationship with other values and then decides how to store it.

II. Query Functions

Since we use dictionaries to store data, query functions just search necessary keys and values in the dictionaries. Check of input's correctness is performed in the Interface part of the program.

isFriend(X,Y):

The program searches for X in the dict5 (contains friends of users) and then searches for Y in the value of the X. If it is found, the function returns True, and False otherwise.

isDirectSource(A, B):

The program searches for B among values of A key in the dict2 (contains only direct reports). If B found, it returns True, and False otherwise.

isSource(A, B):

The program searches for B among values of the key A in the dict6 (contains both direct and indirect reports). If B found, it returns True, and False otherwise.

Anchor(A):

The program takes each element in the List1 (contains only Anchors) and uses them as keys in the dict2 to find the A in the values. The program returns the appropriate key.

DirectReport(A):

The program returns all the values of the key A of the dict2.

Report(A):

The program returns all the values of the key A of the dict6.

III. KOL Analysis

(To be clear, the word ‘influence’ mentioned in this report means the number of times an article or an author is quoted (both directly and indirectly) .)

In this project, the function kol(reports,authors_articles, threshold, percentage) use the input data to give a list of key opinion leaders.

To give a key opinion leader report, dict6 and dict1 need to be imported into the function. Besides, the administrator will be asked to provide a threshold of influence and the percentage of users that are qualified as KOL.

Firstly, we need to get the influence of each article. The influence of each article can be obtained by using len() to each value of the keys in dict6. The obtained influences are stored in the dictionary anchor_influence {}.

After we have the influences of all anchors, we can acquire the influence of all authors. A dictionary named influence_author0 {} is created, and all usernames are the keys. The initial influences are set to be 0. By linking the anchors to their authors using dict1{}, we can add up the influence of the anchors to their authors respectively. For-loops are used to match the anchors to their authors.

Then, we need to arrange the authors by their influences. A dictionary influence_author {} is created to contain the arranged authors and their influence. Initial maximum influence is set to be -1 as the minimum influence of the users is 0. Through for-loops, the most influential pairs in the influence_author0 {} are added to the new dictionary and are deleted from the previous one. This process repeats until no pairs are stored in influence_author0 {}.

Subsequently, we use the percentage input by the administrator to get the number of users that should be qualified as KOL.
Then a list called author_list [] is created, and the users are put in the order of their influences. By calling the position of the list, a respective author is returned, and we can get the author’s influence by calling the author in the dictionary. In this way, we can return the influence of an author based on its position in the dictionary.

Lastly, we work on how a list of key opinion leaders can be returned. The main problem is whether we should follow the threshold influence or the percentage of users when the two results are not identical. The problem is simplified into a choice between the minimum influence of the qualified author using input percentage and the threshold influence. We choose either one that has a higher value to satisfy both requirements. This value is considered as the final threshold. Any author with influence above the final threshold is appended to the final kol_list []. The list of key opinion leaders will be printed if the number 15 is input in the main menu.

Data Abstraction

Input Stage:

1. dict1 {...} - Key: username ; value: article_names [...].
2. dict2 {...} - Key: article_name(element from article names); value: reports[...] (only direct ones)
3. dict3 {...} - Key: article_name; value: 'name of the file storing the articles [...].
4. dict4 {...} - Key: username; Value: name (full), password (...),
5. dict5 {...} - Key: username; Value: friend_list [...]
6. list1 [...] - list of all Anchors
7. dict6 {...} - Key: article_name(element from article names); Value: reports[...] (all reports (will be repetition))

KOL Analysis:

8. anchor_influence {...} - Key:article_name; Value: its number of influences
9. influence_author0 {...} - Key: username; Value: its number of influences
10. influence_author {...} -Key: username; Value: its number of influences (arranged)
11. author_list [...] - a list of all users in the order of their influence
12. kol_list [...] - a list of qualified key opinion leaders

In this project, we used dictionaries in Python to abstract the data. Every dictionary represents the required relationships, e.g. user and user's articles, article and article's reports. Dictionaries are used to store data values in key: value pairs. A dictionary is a collection which is unordered, changeable and does not allow duplicates.

There are ten premade users and 15 articles, which are used for testing. The graphs are used to represent their relationship. While using the program, the user can create data, and it will be saved using 'pickle'.

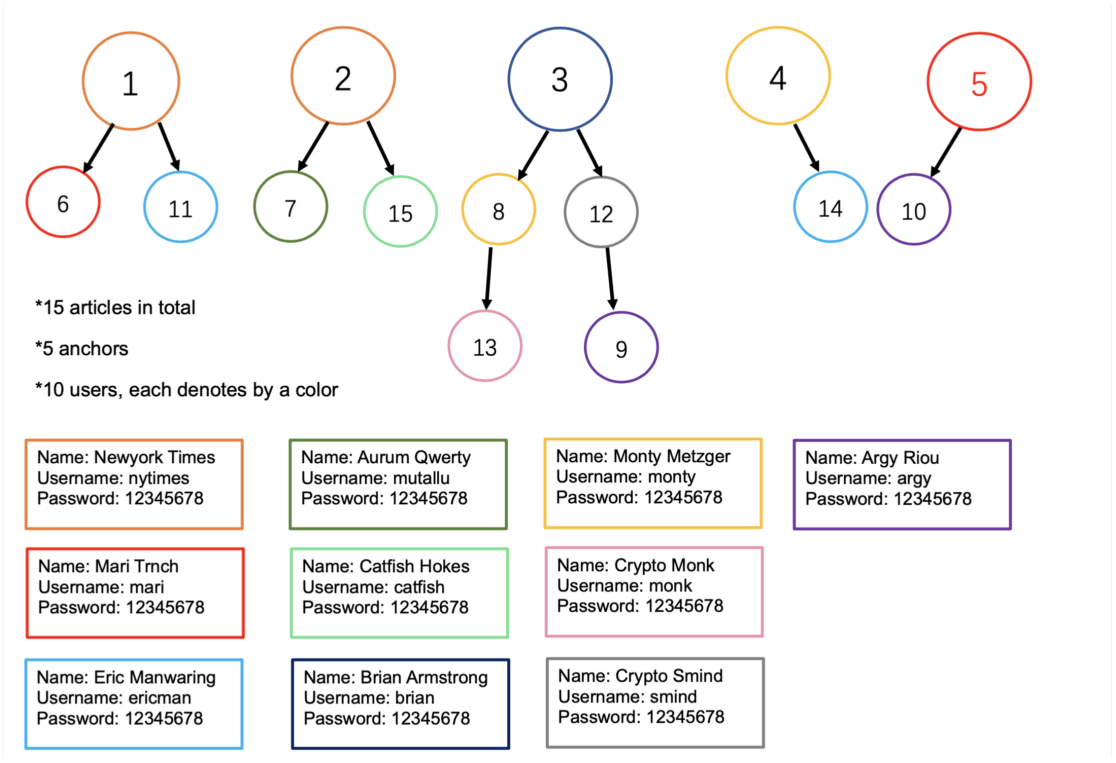


Figure 1. Relationship of articles and users information

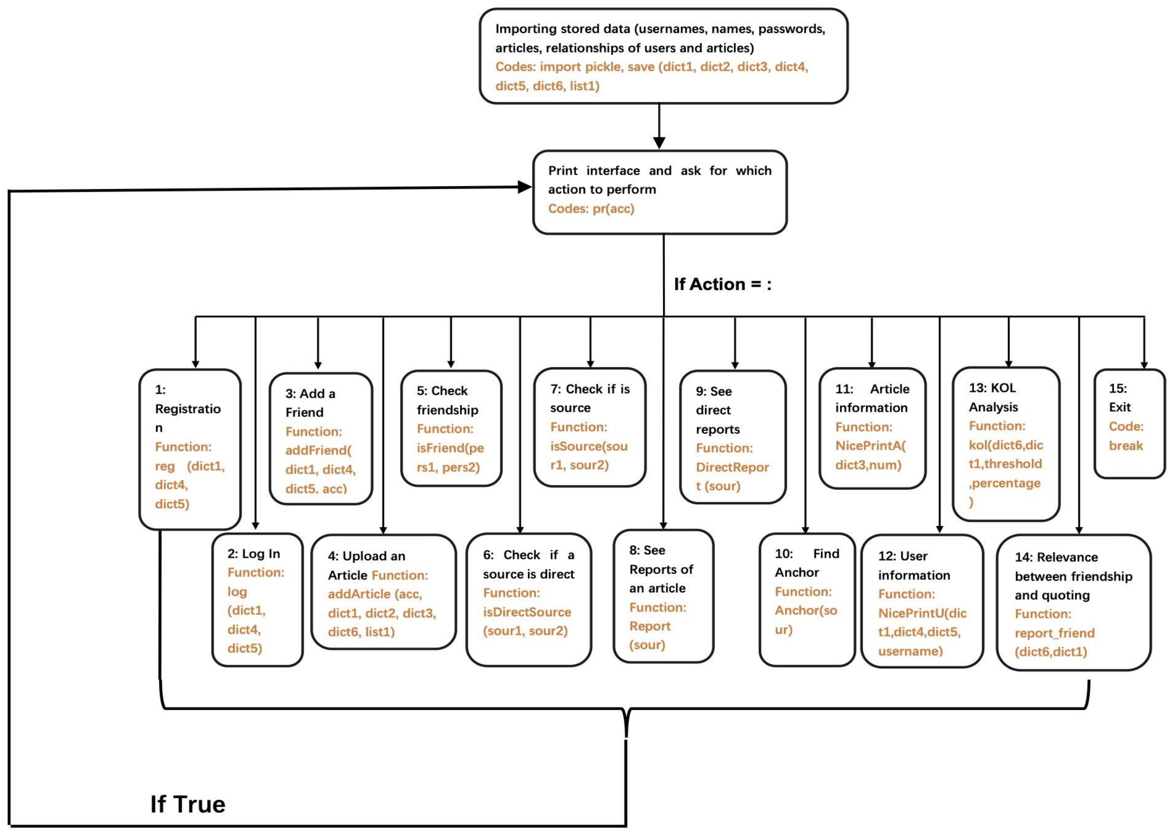
Python Implementation of the Data Types

The data type that is used in this program is dictionaries and lists.

At the input stage, two kinds of data are matched in a dictionary when they are input as keys and values. For example, all articles and their reports are linked together in dict2{}. Lists are usually used as values of dictionaries when the values have more than one item. The reports of an article may be more than one. So in dict2{}, the reports are stored in a list, and the list is the value of dict2{}.

At the output stage, dictionaries enable the values of the dictionaries to be called by their keys. When needed, the called values can be used as keys to called values in another dictionary. For instance, in function kol(dict6, dict1, threshold, percentage), articles are called by the author using dict1{}. Lists of sequenced dictionary keys enable the values of dictionaries to be called using the positions of the keys of the list. In the last part of kol function, the authors are stored in author_list[ ] in the sequence of influence. Then the least influence is called using the position of the respective author in author_list[ ].

Modular Design of the Program



Importing stored data (usernames, names, passwords, articles, relationships of users and articles)
Codes: import pickle, save (dict1, dict2, dict3, dict4, dict5, dict6, list1)
Print interface and ask for which action to perform
Codes: pr(acc)
If Action = :
1: Registration Function: reg (dict1, dict4, dict5)
2: Log In Function: log (dict1, dict4, dict5)
3: Add a Friend Function: addFriend(dict1, dict4, dict5, acc)
4: Upload an Article Function: addArticle (acc, dict1, dict2, dict3, dict6, list1)
5: Check friendship Function: isFriend(pers1, pers2)
6: Check if a source is direct Function: isDirectSource (sour1, sour2)
7: Check if is source Function: isSource(sour1, sour2)
8: See Reports of an article Function: Report (sour)
9: See direct reports Function: DirectReport (sour)
10: Find Anchor Function: Anchor(sour)
11: Article information Function: NicePrintA(dict3,num)
12: User information Function: NicePrintU(dict1,dict4,dict5, username)
13: KOL Analysis Function: kol(dict6,dict1,threshold ,percentage)
14: Relevance between friendship and quoting Function: report_friend (dict6,dict1)
15: Exit Code: break
If True

Figure 2. Modular design chart

Other Highlights

About Pickle

Since this program should save and provide data, it must be able to store the data outside of the program. However, variables are not capable of that because, after the finish of the program, the variables lose their content. Therefore, we used the 'pickle' library functions, which can serialise any objects from a code and save them into the '.pickle' files. Two functions use the 'pickle' - save() and startup().

1. ***Startup()*** is invoked only once in the beginning to assign the values to the dictionaries (they are empty at the beginning of the program).

2. ***Save()*** is invoked in other functions when the dictionaries are modified. It allows keeping the data in the dictionary actually. In other words, when a user creates an account or uploads an article, the save() is called.

Bonus: Whether friends are more likely to quote from each other?
The function report_friend(reports, author_articles) aims at discovering the relevance between friendships and quotes.
First, create a new dictionary new_reports{} to only include articles with reports. Set two variables called ‘is_friend’ and ‘not_friend’ as counters. Their initial values are 0. For each anchor and for each of its reports, see if their respective authors are friends. If yes, then increase the value of is_friend by 1, else increase that of not_friend by 1.
Finally, if the value of is_friend is greater than that of not_friend, a conclusion of ‘friends tend to quote each other’ will be drawn, and vice versa. (Notice that the converse situation includes is_friend=not_friend.)