**Computational Thinking and Problem Solving (COMP1002) and Problem Solving Methodology in Information Technology (COMP1001)**

Course Project
(Due at noon on **7 December 2020**)

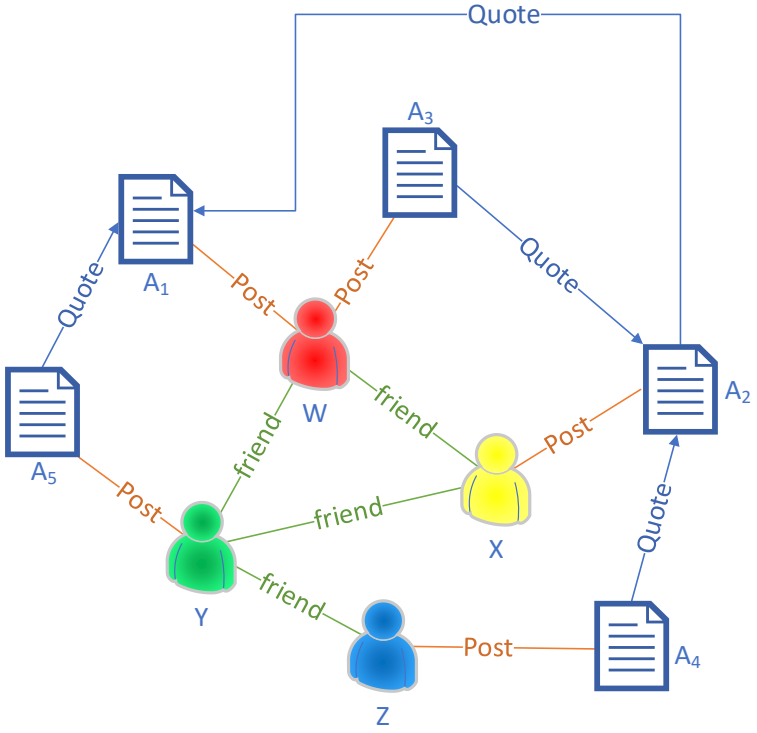
In this project, your group is requested to develop a simple system to maintain information about users and their posted articles in a social media. Based on the information, you are to find out some interesting facts. There are several key functions in this system.

A **user** needs to first register with the system, by providing the name (first name and last name), a username, and a password. Auxiliary information like date-of-birth, phone number is optional.

Each user may post a number of **articles**. Each article contains a title, a textual content and a date when it is posted.

Users may have friends. Note that **`friend`** is a *symmetric* relation here, i.e. if *Y* is a friend of *X*, then *X* is also a friend of *Y*. However, it is *not transitive*, i.e. if *Y* is a friend of *X* and *Z* is a friend of *Y*, then it is not necessarily true that *Z* is also a friend of *X*.

Articles may form a thread via quoting part of another article. This is similar to the thread of postings in **BlackBoard Discussion**. Let us simplify the quoting scenario that each article can quote at most one article. Of course, an article may not need to quote any article. Note that in reality, social media posts may include multiple references and that will make the situation much more complicated. An article without quoting any article is called an *anchor*. An article that quotes another article is called a *report*, and the article that got quoted is called a *source*. For example, it is possible that article $A_1$ is an anchor, which is a source quoted by article $A_2$. Therefore, article $A_2$ is a report for $A_1$. However, $A_2$ can also serve as a source to be quoted by $A_3$ and $A_4$. So $A_3$ and $A_4$ are reports for $A_2$ as well as $A_1$ (transitive relationship). Another article $A_5$ may also quote $A_1$ as source. The following diagram illustrates the basic idea with an example involving five articles as explained above posted by 4 users.



To solve this problem, there is a need to perform data abstraction, namely, determine the type of data to be used and how they are being stored inside the program. In this project, you can see that there are some important *entities*, namely, **users** and **articles**. You need to decide on how a user is being represented inside the Python program and there are many different ways. The particular way you have chosen to represent your data will form the basis of data abstraction. Some representations are better in certain contexts or applications, but may be worse in some other.

After designing for the key data structure or representation, you have to implement some functions in your system to read the data (from keyboard or from files) and store in the memory (i.e. Python variables) for further processing by additional functions. This is the algorithmic part of the project.

It is interesting in social networks to identify KOL (key opinion leaders) and their relationship, or to investigate into whether friends tend to quote articles posted by each other, within a social circle. To support this kind of investigations, you are to develop a system to store the data for the users and articles. Then your system will also support different data access requests to the collected set of data. Note that in real social network, there is often a need to analyze the content of posts to identify their similarity, but we are not doing this complex procedure in this project.

Query functions to be supported:

1. **`isFriend`**(X,Y): return True if user X and Y are friends, and False otherwise.
2. **`isDirectSource`**(A,B): return True if article A is a direct source of B, and False otherwise.
3. **`isSource`**(A,B): return True if A is either a direct source of B or an indirect source of B, and False otherwise.
4. **`Anchor`**(A): returns the article K that is the anchor for A, i.e. K is an anchor and A directly or indirectly quote K.
5. **`DirectReport`**(A): return the list of articles that directly quote A as source.
6. **`Report`**(A): returns the list of articles that directly or indirectly quote A as source.

Useful auxiliary functions:

7. **`NicePrintU`**(X): print out the information for a user X in a nice way.
8. **`NicePrintA`**(A): print out the information for an article A in a nice way.
9. **`GetU`**(X): read in information for user X, from keyboard or from file, as appropriate
10. **`GetA`**(A): read in information for article A, from keyboard or from file, as appropriate (note that you will need also to know who is posting this article)

Let us consider the first investigation, i.e. to identify a KOL. Let us define the *influence* of an article $A$ to be the total number of users posting an article that is a report of $A$. A KOL is then a person whose posted articles have strong influence. When the number of total articles increases in the network, more and more people can become a KOL. In our very simple treatment, we make use of two parameters to define KOL. To qualify as a KOL, the person should (1) have posted more influential articles than others AND (2) the total influence should reach a minimum threshold $T$. Suppose there can only be at most $p$% of users who can qualify as a KOL. For example, if $T = 10$ and $p = 25$. Then in a system with 50 users, there can be at most 12 KOL (rounding down from 12.5). We compute the influence of each user and only consider those with influence over 10. From those with sufficient influence, only the top 12 users can be a KOL. If there are no more than 12 users with influence over 10, then all those users are considered as KOL. If there are more than 12 users with influence over 10 and there is a tie in terms of the influence, it is possible to have more than 12 KOL. Among those KOL, does any friendship exist between them?

With your query functions, implement an application to find the list of KOL in a social network and find out the existence of any friendship. Note that here, $T$ and $p$ are parameters that the system

administrator can choose. You might specify them via the parameters to the **`main`**() function when you execute the program.

Bonus level: Investigate the second problem by first designing a set of criteria to check whether friends tend to quote articles from one another. This is followed by an implementation of your design in the system as one or more additional functions. This bonus part could be placed in appendix and is not counted towards the page limit of the report.

## Deliverables

1. A report in **`.pdf`** format documenting the process of solving the problem.
   a. Five pages maximum.
   b. The report must contain the following information and sections. There is no need to include the code, because it is already in the **`.py`** file.
      i. Your group number, member names, and student ID
      ii. The problem description
      iii. Data abstraction (including a description of the key data types and representations, perhaps via examples)
      iv. A description of the Python implementation of the data types
      v. A modular design of the program via the definition and use of a few key functions, e.g. **`computeInfluence`**(X) for user *X* (explanation to auxiliary functions is not required)
      vi. Any special observation or approach you would like to highlight
2. A well-documented Python program in a single **`.py`** file.
   a. You must use *docstring* to describe each function in the form of comments.
   b. By using *appropriate* comments and variable names, your program must be easy to follow and understand.
   c. Give proper *reference* to the source of the code that you adopt for your program.

## Submission Instructions

Follow the steps below:

1. Create a folder and name it as **`G`**<group no>_<your names>, e.g., **`G11_CHANTaiMan_WongTaiSin_CheungMeiYiu`**
2. Submit the source file (**`.py`**). Name the **`.py`** files as **`G`**<group no>_<subject code>**`.py`**. Note that here subject code is either 1002 (for most students) or 1001 (for some students), e.g., **`G11_1002.py`** or **`G4_1001.py`**
3. Submit your report in a **`.pdf`** file. Name the single **`.pdf`** file as **`G`**<group no>_<subject code>**`.pdf`**, e.g., **`G11_1002.pdf`** or **`G4_1001.pdf`**
4. Put all the **`.py`** and **`.pdf`** files into the folder.
5. Compress the folder (**`.zip`**, **`.7z`**, or **`.rar`**).
6. Submit the compressed file to Blackboard.

A maximum of **3 attempts** for submission are allowed. **Only the last attempt will be graded**. A late penalty of 5% per hour will be imposed.

## Assessment Criteria

This project is assessed based on the following rubrics:

| | Excellent | Good | Satisfactory | Weak | Fail |
|---|---|---|---|---|---|
| **Report (40 marks)** | Identify all requirements and data abstraction, and provide a comprehensive system design and sound justifications. (31-40 marks) | Identify most of the requirements and data abstraction, and provide good system design and justifications. (21-30 marks) | Identify some of the requirements and data abstraction, and provide some system design and justifications. (11-20 marks). | Identify a few of the requirements and data abstraction, and provide little system design and justifications. (6-10 marks) | Very little or no requirements, data abstraction, system design and justifications are provided. (0-5 marks) |
| **System Implementation (50 marks)** | Develop an application that satisfies all the system requirements and well documented. (41-50 marks) | Develop an application that satisfies most of the system requirements and properly documented. (31-40 marks) | Develop an application that satisfies some of the system requirements and reasonably documented. (21-30 marks) | Develop an application that satisfies a few of the system requirements and fairly documented. (11-20 marks) | Unable to develop an application that satisfies the system requirements and poorly documented. (0-10 marks) |
| **System Usability (10 marks)** | The system provides an excellent user experience. (9-10 marks) | The system provides a good user experience. (7-8 marks) | The system provides a reasonable user experience. (5-6 marks) | The system provides a fair user experience. (3-4 marks) | The user interface is non-existent or provides little/no clue to the system functions. (0-2 marks) |