Q1 [120 marks]: Community Detection in Online Social Networks

Problem Description:

Community detection has drawn lots of attention these years in the machine learning field. With the popularity of online social networks, we can acquire many valuable datasets and based on which develop community detection algorithms. In this homework, you will implement such a community detection algorithm for 3 blogs network datasets.

Blogs/microblogs are typical applications of online social networks. Basically, blogs are more likely to be in the same community when they have higher similarities (i.e., sharing many common links on their pages). There are two types of relationships in the blogs network. One is `symmetric`, i.e., blog X has embedded a link to blog Y, and blog Y also provides a link directing to blog X; the other is `asymmetric`, which means blog Y may not link to blog X even though blog X has a link to blog Y. In the latter case, there are two roles involved in this relationship: `follower` and `followee` (like the case when using Twitter and Weibo). When blog X has a link to Y, X is the follower and Y is the followee.

follower：粉丝
followee：关注

To detect communities, we need to calculate the `similarity` between any pair of blogs. In this homework, for a given pair of blogs, the similarity is measured by the number of *common followees* divided by the *total* (deduplicated) number of the two blogs' *followees*. Specifically, the formal definition of `similarity` is as follows:

公共关注者

If $|\mathbf{out}(A) \cup \mathbf{out}(B)|$ >0, we define

out(A)表示blog A的所有关注者

$$\text{Similarity}(A,B) = \frac{|\mathbf{out}(A) \cap \mathbf{out}(B)|}{|\mathbf{out}(A) \cup \mathbf{out}(B)|} \text{ ,}\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\text{(*)}$$

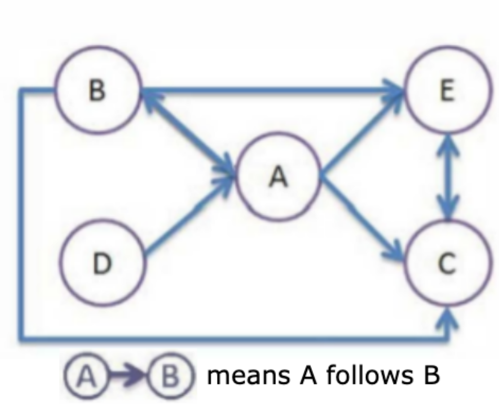
交集 / 并集

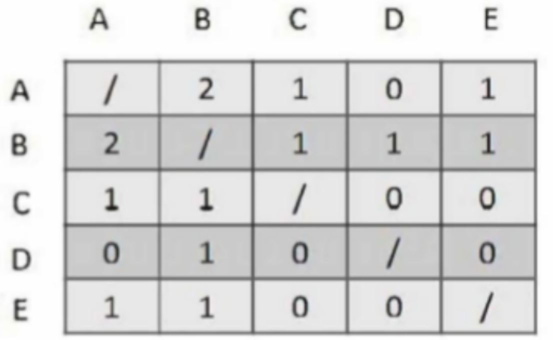
where `out(A)` is the set of all followees of blog A, and |S| is the cardinality of S.

If $|\mathbf{out}(A) \cup \mathbf{out}(B)| = 0$, we set the similarity to be 0.

Motivating Example:

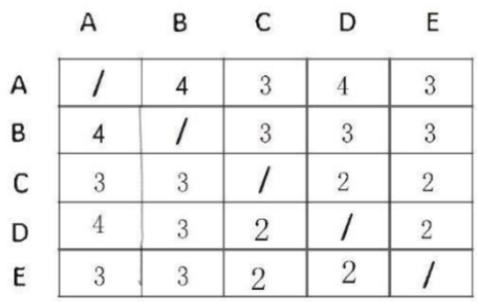
The following figure illustrates the process of calculating the (pair-wise) `similarity`. Five blogs A,B,C,D,E are involved in this example.





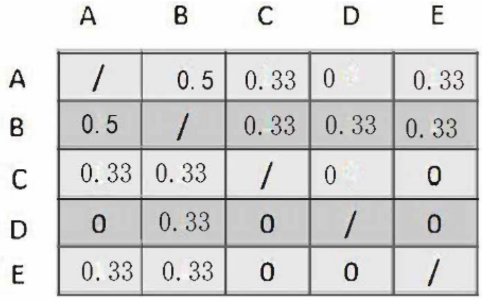
	A	B	C	D	E
A	/	2	1	0	1
B	2	/	1	1	1
C	1	1	/	0	0
D	0	1	0	/	0
E	1	1	0	0	/

交集 Common followees 共同的关注对象



	A	B	C	D	E
A	/	4	3	4	3
B	4	/	3	3	3
C	3	3	/	2	2
D	4	3	2	/	2
E	3	3	2	2	/

并集 Total number of followees 两者的所有关注对象



	A	B	C	D	E
A	/	0. 5	0. 33	0	0. 33
B	0. 5	/	0. 33	0. 33	0. 33
C	0. 33	0. 33	/	0	0
D	0	0. 33	0	/	0
E	0. 33	0. 33	0	0	/

Pair-wise Similarity

The set of followees of A is {B, C, E} and the set of followees of B is {A, C, E}. There are 2 common followees between A and B (i.e. C and E), and the number of the union of their followees is 4. The similarity between A and B is therefore 2/4 = 0.5.

We provide three datasets with different sizes. The small dataset contains around 4K blogs; the medium one contains around 80K blogs, and the large one contains over 100K blogs. Each blog is represented by its unique ID number. The download links of all datasets are listed in reference [1]-[3]. The small dataset is provided to facilitate your initial debugging and testing. The design of your program should be *scalable* enough to handle all the datasets.

Sample Input:

The format of the data file in the above example is as follows:

A B A follow B
A D
B A B follow A
C A
C B
C E
E A
E B
E C

Note: Pay attention to the input format, particularly how it corresponds with the upper left graph.

Sample Output:

Our expectation for the community detection output is that for *EVERY* blog, we want to know the TOP K most similar blogs. An example of the output format (K=4) can be as follows (different similar blogs separated by space; blogs with 0 similarity omitted):

A: B C E
B: A C D E
C: A B
D: B
E: A B

Objective:
Solve the above community detection problem using MapReduce. Five tasks with instructions can be found below. Note that the output format of each specific task may vary.

[Task Requirements]

1. You can either use the IE DIC or the Hadoop cluster you built for HW#0 to run your MapReduce program(s).
2. You are free to use any programming languages (e.g., Java [4] or Python [5] or C/C++) to implement the required MapReduce components, i.e., mapper(s), reducer(s), etc. (e.g. by leveraging the "Hadoop Streaming" capability [6]).
3. Again, the design of your program should be scalable enough to handle all three datasets.

[Submission Requirements]

1. Submit the codes and outputs of your programs in one single PDF file. Besides, you should also present the descriptions (in words) and illustrations (in figures/ screenshots) that help convey and clarify your ideas in solving the problem.
2. As for part (a), (b) and (e), submit the community detection results of all those blogs whose IDs share the same last 4 digits with your CUHK student ID. For example, if your student ID is 1155004321, then you need to submit the community members for blogs with ID = 4321, 14321, 24321, 34321, …, 114321, …
   Note that this requirement only serves as a *filter* for the original outputs. In other words, you still need to run your MapReduce jobs on every blog and harvest all outputs, and you need this extra step to trim the outputs according to your CUHK ID and paste into your report.

找出与每个博客common followees最多的博客

Tasks.

a. **[25 marks]** For EVERY blog, recommend the blog with the maximal number of common followees in the **`medium`**-sized dataset [2]. If multiple blogs share the same number, pick the one with the largest ID. Your output should consist of m lines, where m is the total number of blogs. Each line follows the format below: 返回m行结果

A:B, {C,E}, 2

where "A:B" is the blog pair, "{C,E}" is the set of their common followees (no special requirement for the elements' order, i.e., {E,C} is acceptable), "2" is the count of common followees.

b. **[30 marks]** Find the TOP K (K=3) most similar blogs of EVERY blog as well as their common followees for the **`medium`**-sized dataset [2]. If multiple blogs have the same similarity, randomly pick three of them. For each pair of blogs, output a line with the following format:

A:B, {C,E}, simscore ·····································(F1)

where "simscore" is the similarity score between A and B.

Hints:

1. To facilitate the computation of the similarity as defined in (*), you can use the inclusion-exclusion principle $|S \cup T| = |S| + |T| - |S \cap T|$.
2. You are allowed to use the "sort" command on Linux to get the top K similar blogs after you have computed similarity scores for all pairs.

c. **[25 marks]** In fact, each blog is annotated with a label indicating its community. In each dataset, a label file is provided, with the first column indicating the blog ID and the second column indicating the label value. For example, the small dataset has seven different labels (the value ranges from 0 to 6), which means that each blog is from one of the seven communities.

For each community in the **`medium`** dataset, please figure out how many (unique) members act as the common followees of other blogs. (For example, suppose that A, B, C, D, E are labeled with community 0, 1, 2, 1, 2, respectively. Then, for community 0, one of its members (blog A) acts as the common followee of others (blog B and D). As for community 1, none of its members is the common followee of others.) Your reported results should be formatted like the following example:

Community 0: 1
Community 1: 0
Community 2: 2

d. **[20 marks]** Run part (a) for the **`medium`** dataset multiple times while modifying the number of mappers and reducers for your MapReduce job(s) each time. You need to examine and report the performance of your program for at least 4 different runs. Each run should use a different combination of the number of mappers and reducers.

For each run, performance statistics to be reported should include: (i) the time consumed by the entire MapReduce job(s); (ii) the maximum, minimum and average time consumed by mapper and reducer tasks; (iii) tabulate the time consumption for each MapReduce job and its tasks. (One example is given in the following table.) Moreover, describe (and explain, if possible) your observations.

Example:

1st Run:

#Job	Mapper num	Reducer num	Max mapper time	Min mapper time	Avg mapper time	Max reducer time	Min reducer time	Avg reducer time	Total time
1	3	2	60s	40s	50s	60s	40s	50s	2.5 min
2	…	…	…	…	…	…	…	…	…

Note:

1. The number of rows in the table depends on the number of jobs (one distinct map and reduce function for each job) you chain to complete part (a).
2. The elapsed time for each Map/Reduce Task can be found on Job History Service Web UI (port 19888).

   For students using their own VMs, start the service by running `./sbin/mr-jobhistory-daemon.sh start historyserver`. Remember to set up proper firewall rules/ use ssh port forwarding so as to access the Web UI on your local browser.

   For students using IE DIC Cluster, the Web UI is served at http://dicvmd3.ie.cuhk.edu.hk:8088/cluster/apps. Moreover, users can find the details of a particular job via http://dicvmd3.ie.cuhk.edu.hk:8088/cluster/app/application_1642685809406_0001, where 1642685809406_0001 is the ID of the job you created.

e. **[20 marks]** Find the TOP K (K=3) most similar blogs and the list of common followees for each blog in the **`large`** dataset in [3] using the format of Q1(b). (Hints: To reduce the memory consumption of your program, you may consider using the composite key design pattern and secondary sorting techniques as discussed in [7] and [8]).

General Hints:

1. Going through HW#0 Q1(c),(d)'s source codes may help you understand how to construct your mapper & reducer code here. However, unlike WordCount, you cannot always expect to finish all your work with only one single mapper and reducer code.

   Chaining multiple (i.e. a series of different) MapReduce jobs to handle complex community detection problems is a standard approach. It may be difficult to just use one MapReduce program to get the final results for each problem due to memory exhaustion and the parallel & distributed nature of data in Mappers/Reducers.
2. For each dataset, there are two files included. The one with the suffix '_relation' indicates the mutual relations of each pair of blogs. The one with the suffix '_label' indicates the community label for each blog.
3. For students who did not manage to set up their Hadoop cluster in HW#0, please contact the TAs. You can either choose to set up the cluster with TAs' help or you can run your MapReduce programs on the IE DIC Cluster, where Hadoop has already been installed. Once the IE DIC cluster is ready, TAs will inform you of the account information. More details will be provided in the tutorial/ on CUHK blackboard.
4. If you use Java, you can specify the number of mappers with the following code: `job.setNumMapTasks(20)`. If you use Hadoop streaming with Python, you can specify it via the following command option: `-D mapred.map.tasks=20`. If this does not work, you may need to modify the split size in `$hadoop/etc/hadoop/mapred-site.xml`: `mapred.min.split.size=268435456`. Refer to [9] for more information.
5. As for the large dataset, you may want to set `mapreduce.map.output.compress=true` to compress the intermediate results, in case you don't have enough local hard disk space (to hold the intermediate tuples).

6. Tackling the problems may take much longer than you expect. Particularly, the large dataset may take a long time to process even if everything is correct. **Please start doing this assignment as early as possible**.

References:

[1] Small-scale dataset
https://mobitec.ie.cuhk.edu.hk/iems5730Spring2023/static_files/assignments/small.tar.gz
[2] Medium-scale dataset
https://mobitec.ie.cuhk.edu.hk/iems5730Spring2023/static_files/assignments/medium.tar.gz
[3] Large-scale dataset
https://mobitec.ie.cuhk.edu.hk/iems5730Spring2023/static_files/assignments/large.tar.gz
[4] Write a Hadoop program in Java
https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
[5] Write a Hadoop program in Python2
http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/
[6] Hadoop Streaming
https://hadoop.apache.org/docs/r2.9.2/hadoop-streaming/HadoopStreaming.html
[7] Composite Key
https://techmytalk.com/2014/11/14/mapreduce-composite-key-operation-part2/
[8] Secondary Sort
http://codingjunkie.net/secondary-sort/
[9] How many Mappers and Reducers?
https://cwiki.apache.org/confluence/display/HADOOP2/HowManyMapsAndReduces