CIS 833 – Information Retrieval and Text Mining **Name:_______________**

Homework Assignment 1 (20 points) – due September 16th at 11:59PM

Note: Please remember that you are allowed to discuss the assigned exercises, but you should write your own solution.

Exercise 1 (5 points)

Assume you are given a task to determine the most popular Web site (domain name) given a log of access requests, each of the form "URL requestor". Furthermore, assume you are given a function String getDomain(URL) that retrieves the domain name from each URL. Show pseudocode for a MapReduce program (i.e., map and reduce functions) that does this computation. Clearly explain what the input, intermediate and output <key,values> are.

```
Map(URL, requestor)
Let d ≔ getDomain(URL)
Emit (d, 1)

Reduce(domain, {sequence of counts})
Let L ≔ length of the sequence
Emit(d, L)
```

In order to get all counts in one file, we need to ensure we use one reducer. To get the most popular domain (i.e., the domain with max count), we could use a function to scan through the Reduce output file; alternatively, we could write another MR job that sorts the counts, and thus the domain(s) with the highest counts would be first in the output file (assuming we sort the counts in decreasing order).

Exercise 2 (5 points)

You are given an input file which contains comprehensive information about a social network that has asymmetrical (directed) links, i.e., a network where users 'follow' other users but not necessarily vice-versa (e.g., twitter). Each record in this input file is (userid-a, userid-b), where userid-a 'follows' userid-b (i.e., points to it). Note that this record tells you nothing about whether or not userid-b follows userid-a. Write pseudocode for a MapReduce program (i.e., map and reduce functions) that outputs all pairs of userids who follow each other.

Function 1 MAP (*userid-a*, *userid-b*)

```
if $userid\text{-}a < userid\text{-}b$ then        //lexicographic ordering
    $string \leftarrow$ "userid-a, userid-b"
else
    $string \leftarrow$ "userid-b, userid-a"
end if
return ($string$, 1)
```

Function 2 REDUCE (*key*, *list of values*)

```
if $sum(list\ of\ values) = 2$ then
    return $key$
else
    return
end if
```

Exercise 3 (10 points)

Google manages the GMail e-mail service, and they would like to filter out as many spammers as possible. In this problem you will implement a simple spam filtering idea with MapReduce, so that it can be efficiently applied to the multitudes of e-mail messages in GMail.

We want to produce a "blacklist" of e-mail addresses that are spamming GMail users. Spam messages are sent to many addresses at once, and so a spam message can be identified by having one of the most commonly-used subject lines. If an address sends many messages with the most common subject lines, it is likely a spammer address. We would like to find the top ten addresses that have sent the most messages with frequently-recurring subject lines.

The input data is a collection of e-mail records in the file GMail-messages. Each e-mail record is a list with the format

```
(from-address to-address subject-line email-body)
```

where each element is a double-quoted string. The mapper function will be applied to each e-mail record. A small example set of e-mail records is shown below:

("dcaragea" "cis890" "mapreduce" "mapreduce is great! lucky students!")
("bot1337" "cis-grad" "free ipod now!" "buy herbal ipod enhancer!")
("bot1338" "cis-ugrad" "free ipod now!" "buy herbal ipod enhancer!")

(i) Our first step is to produce a table with `subject-lines` as keys and counts of occurrences as values. Identify the intermediate key-value pairs and write pseudocode for the map and reduce functions.

```
method MAP(record_id, record_line)
subject_line <- get_subject_line_from_record_line        //third string between quotes
EMIT (subject_line, 1)

method REDUCE (subject_line, ones [1,...,1])             //intermediate key,value pairs
count = 0
for all one from ones [1,...,1] do
    count <- count + 1
EMIT (subject_line, count)
```

(ii) From our tabulation of `subject-line|count of occurrences` in (i), we want to find the most common `subject-lines` in the table. We can perform a sort by using the fact that MapReduce sorts by intermediate keys into the reducer groups. Identify the intermediate key-value pairs and write the map and reduce functions.

```
//we are reading the file produced in (i) – i.e., each line contains subject, count
method MAP(line_id, line_content)
split_line_content_into_subject_line_and_count
EMIT (count, subject_line)

//MapReduce will sort by intermediate keys, i.e., count, subject_line pairs
//will be sorted by count

method REDUCE (count, list of subject_lines)      //intermediate key,value pairs
//we don't need to do any aggregation here, i.e., reduce is the identity function
for each subject_line in list
    EMIT (subject_line, count)
```

(iii) Finally, assuming you've moved the ten most common `subject-lines` into a list, we want to make a table with `from-addresses` as keys and counts of e-mails sent with common subject lines as values. You don't need to sort the table, since the procedures would be identical to those in (ii). Identify the intermediate key-value pairs and write the map and reduce functions.

We have a list of the ten most common subject_lines – let's assume that they are stored in a hashtable H, which is visible to the MAP class.

```
method MAP(record_id, record_line)
from_address <- get_address_from_record_line               //first string between quotes
subject_line <- get_subject_line_from_record_line        //third string between quotes
if (subject_line in H)
EMIT(from_address, 1)

method REDUCE(from_address, ones [1,...,1])              //intermediate key,value pairs
count = 0
for all one from ones [1,...,1] do
    count <- count + 1
EMIT (from_address, count)
```