



**1. Intellectual Property and Software Development (25 points):** The purpose of this problem is to get you to think about issues that are relevant today for software developers. Get together with a small group of 2-4 students and discuss issues surrounding intellectual property, software patents, and open source. Discuss the following points:

- How has the Open Source movement changed the availability of software and dissemination of software? (Think about the software stacks and tools on which many companies today are built).
- How would your group go about creating a startup (focus on the technical side of things, and not on the product aspect of a startup)? What software would you use, how would you investigate prior work, and how would you protect your ideas?
- In what ways can you give back some of the value that you create (new software, new algorithms, etc.), without undermining your business?
- Companies that produce proprietary software need to be very careful about open source. Introducing open source software into a codebase can jeopardize the entire product. Working at a such a company often comes with limiting your exposure to open source software to avoid issues of ‘tainting.’ This is very unlike the situation when you’re in school. How would you go about investigating potential solutions to problems at these companies?
- If you go work for a software company you will likely sign away your rights to intellectual property that you create (the code you write, techniques you invent, etc.). How do you feel about these tradeoffs (getting a good salary but being unable to own your ideas)?

Write up two to three paragraphs summarizing the discussion your group had. We’re not looking for a simplistic “open source is awesome!” solution; there are many tradeoffs to consider. Hopefully these questions stimulate thoughtful conversation and get you thinking about situations you will likely encounter in the next few years.

**2. MapReduce (50 points):**

In this problem you are going to explore writing a non-trivial map-reduce program for network extraction. Rather than setting up and using the Hadoop framework, you will emulate the distributed process using only command line tools and simple Python scripts.

Recall the following steps of the Map-Reduce process:

### MAP STEP:

Input Data → Map program → Mapper output. You can use a command like this to perform this step: “cat mydata.txt | python my\_map\_function.py > map\_output.txt”

The map program should read lines from STDIN and print out whatever key-value pairs you want to STDOUT. See the example program **wc\_mapper.py** for how to do this.

### SHUFFLE STEP:

In an actual map-reduce system key-value pairs get assigned to a particular reducer task based on the hash of the key. Since we’re only using a single reduce task, this isn’t necessary.

However, in preparation for applying the reduce function, we do need to sort the key-value pairs by key so that we see all of the values for the same key sequentially. You should investigate using the Unix **sort** command, and the various options that allow you to finesse its operation.

Sample command: sort map\_output.txt > suffled\_map\_output.txt

### REDUCE STEP:

For the reduce step, we simply pipe the suffled mapper output into the reduce program: “cat shuffled\_map\_output.txt | python my\_reduce\_function.py > reducer\_output.txt” Once again, the reducer should read lines from STDIN and write key-value pairs (or just output) to the STDOUT (see **wc\_reducer.py** for an example).

### Putting it together into a one-liner:

Using the three files from the assignment directory, we can run the following command:

```
cat wc_sample_input.txt | python wc_mapper.py | sort | python wc_reducer.py
```

### Output:

```
is      3
this    3
an      2
example  2
as      1
but.    1
case    1
class   1
count   1
file    1
for     1
fun.    1
input   1
input.  1
is.     1
```

not 1  
punctuation. 1  
really 1  
relevant. 1  
serves 1  
text 1  
that 1  
word 1  
your 1

**The task:** You will write mappers and reducers **in Python** to address two problems. Suppose that we have a directed network of who-mentions-whom in Twitter. We use this as a proxy for one user paying attention to another. Additionally, we have all of the hashtags that each user has used. First, we want to understand the degree distribution in this network (this doesn't involve hashtags at all). We also want to investigate how certain hashtags spread through this attention network. Our goal is to extract the subgraph of users who used a particular hashtag, and the edges between them, **for every hashtag at once**.

**Data formats:**

Edge file format: one "edge U V" per line indicating there is an edge from U to V in the graph. U and V are integers, but you shouldn't make assumptions about them being contiguous on [0, N-1].

Hashtag file format: one "ht U **hashtag**" entry per line indicating that U used **hashtag**. You may assume that hashtags have been normalized so that '#OBAMA' and '#obama' are both going to appear as '#obama' in the data file.

**The problems, in detail:**

**Degree Distribution (15 points)-** Write a map-reduce task that outputs the in- and out-degree distribution for every node in the graph. The output should have one "Node\_id in\_degree out\_degree" entry per line. If either the in- or out-degree is zero, your output should reflect this.

**The real deal (35 points)-** Write a map-reduce task that outputs two lines per hashtag that appears in the hashtags file. One line should read "**hashtag** users A B C ..." where **hashtag** is some hashtag, and A, B, C ... etc. are the users who used **hashtag**. The other line should read "**hashtag** edges U1 V1 U2 V2 U3 V3 ... Uk Vk" where again **hashtag** is a particular hashtag and the pairs  $U_i V_i$  are edges for which both  $U_i$  and  $V_i$  used the hashtag.

Your map-reduce process should digest data from both files together. That is, your pipeline should look something like

"cat edges.txt hashtags.txt | python mapper\_1.py | ..."

**What to turn in:** You should turn in a .zip or a .tar.gz that contains your map and reduce programs, as well as two shell scripts named **degree\_count.sh** and **ht\_graphs.sh** that run the pipeline end-to-end. These scripts should accept input on the standard input. See the example shell script for doing word counts which can be run as “cat wc\_sample\_input.txt | bash wc\_pipeline.sh”

We will run your programs as follow: “cat edge\_file.txt | bash degree\_count.sh” and “cat edge\_file.txt hashtag\_file.txt | bash ht\_graphs.sh” You can use these exact commands to verify that your code runs from end to end!

**Hints:**

- Don't do everything at once. Consider a multi-step process in which each step gets you closer to the solution.
  - Write out intermediate files so that you can inspect what is happening at each phase of the map-reduce process.
  - More hints to come!
3. **Topic Identification and Aggregation (25 points):** Describe an approach for solving the ‘message aggregation problem’ in which you want to group together messages (Tweets, FB status updates, news articles, etc.) talking about some particular topic. There are two problems to consider here: content aggregation and topic identification. You should address both issues in your solution. Try to borrow ideas from the entirety of the class- we’ve studied network structure, web search, recommendation systems, large-scale data processing, and NLP.
  4. **Bonus (up to 100 points):** Implement your idea above on some real data source. We suggest looking at the Twitter API as it’s pretty easy to understand and use, but you are free to use any data source you want. You can work on this with a partner and you have through the end of the semester to complete it. You’ll meet with Bren to show off what you did. Points will be assigned based on novelty, presentation of results, and quality of results.