Extreme Computing
First Assignment

Due **16:00 on 28 October**. All questions should go on Piazza

https://piazza.com/ed.ac.uk/fall2016/infr11088

in the "hw1" folder. If your question reveals an answer, ask privately.

The assignment is worth 25 marks in total. We mark for correctness, efficiency, and proper use of tools. That includes, but is not limited to, running your solution in a scalable way. In many cases, it is possible to get the right answer with a less scalable implementation, in which case you may receive partial marks. In all problems, you must use `HDFS` and `Hadoop`.

As a hint, all of the solutions take less than an hour to run. You should kill any jobs that take longer:

```
mapred job -kill $jobid
```

We may use a bot to kill long-running jobs so that everyone can run on the cluster.

This assignment is divided into two parts and eight tasks. The first part deals with taking messy Web data, cleaning it up and performing simple queries over it. The second part focuses on how huge matrices can be transposed by using MAPREDUCE. Finally, the last part deals with how a relational join operation should be done by using MAPREDUCE. In all parts, it is ok to use the output of a previous task as input to subsequent tasks.

You should use the teaching Hadoop Cluster[1] and any programming language you want. If you are logging in from outside Informatics, first

```
ssh s12345678@student.ssh.inf.ed.ac.uk
```

or use the Informatics VPN:

http://computing.help.inf.ed.ac.uk/openvpn

Once inside Informatics, connect to a random machine in the cluster:

```
ssh scutter0$((RANDOM%7+1))
```

For each part there are different data sets (on HDFS). There are two versions of each input file that should be used in your program, a small one and a larger one. The **small** version file is for developing and testing your code; when you are happy that it works, you should use the **large** version.

Reference outputs for small data can be found in `/data/assignments/samples/`. Note that your output can be different if you used a different number of reducers or partition function. Such output will still be considered correct.

The files (on HDFS) that you will need for each part follows:

Part 1

File	Number of lines	Number of words
`/data/assignments/ex1/webSmall.txt`	95350	6311838
`/data/assignments/ex1/webLarge.txt`	1897987	123588873

[1] You can run your own, but we won't support it and you need to copy output back to the teaching one.

Part2

File	Number of lines	Number of words
`/data/assignments/ex1/uniSmall.txt`	2000	6987
`/data/assignments/ex1/uniLarge.txt`	2000000	6985354

All your actual results should be produced using the larger files and for all tasks you should use Hadoop MAPREDUCE and HDFS.

1 Tasks

1.1 Processing Web data

Task 1

◁ **Task**

Take file `webLarge.txt`, and produce a version which is all *upper-case*. For example, the sentence `John loves Mary.` would become `JOHN LOVES MARY.` Call this the *upper-case* version. **For purposes of correctness marks, changing the order of the lines is permissible.** **(2 marks)**

Task 2

◁ **Task**

Using MAPREDUCE write a program that will filter the *upper-case* version (from task 1) such that only lines that appear **exactly** once will be in the output. Call this the *unique-only* version. Your approach should be exact (do not use approximate techniques). The output will be used for some of the following tasks. For example, for the file:

```
bob had a little lamb and a small cat
alice had one tiger
bob had a little lamb and a small cat
mary had some small dogs and a rabbit
mary had some small dogs and a rabbit
bob had a little lamb and a small cat
```

the output should be:

```
alice had one tiger
```

because `alice had one tiger` is the only line that appeared exactly once in the input. **(3 marks)**

Task 3

◁ **Task**

Compute the length of the longest token and the length of the longest line in the *unique-only* version. We define a token the same way Python's `split()` function does: anything separated by runs of whitespace (space `' '`, tab `'\t'`, carriage return `'\r'`, new line `'\n'`, or form feed `'\f'`). The length of a line does not include the newline character; see the example.

Your output should be a **single file** on HDFS. For example, the following file `example.txt`:

```
bob had a little lamb and a small cat and a tiny fish
alice had one kangaroo
mary had some small dogs and a rabbit
```

should have the following result (8 is the length of the longest token and 53 is the length of the longest line):

```
8 53
```

(2 marks)

Task 4

◁ **Task**

On the *unique-only* version, find all three-word sequences and their counts. For example, the two sentences:

```
mary had a little lamb
mary had a little tiger
```

have the following three-word sequences and counts:

Sequence	Count
`mary had a`	2
`had a little`	2
`a little lamb`	1
`a little tiger`	1

(5 marks)

Task 5

◁ **Task**

What are the top twenty most frequent three-word sequences? Produce a single output file. Each line in the output should contain a count of a three-word sequence followed by a space followed by the actual three-word sequence. The output should be sorted in decreasing frequency order (i.e. the most frequent three-word sequence should be first).

(2 marks)

Task 6

◁ **Task**

Using the three-word sequence counts you found in task 4, find the entropy of words for each two-word context. For a given two-word context, the entropy of the next word is given as

$$H(c) = -\sum_{w \in \text{followers}(c)} p(w \mid c) \cdot \log_2 \left(p(w \mid c)\right) \tag{1}$$

where $H(c)$ is the entropy of context c, $\text{followers}(c)$ are the words which were seen to follow context c and $p(w \mid c)$ is the probability that the word w follows context c and is defined as

$$p(w \mid c) = \frac{\text{count}(c\ w)}{\sum_{w' \in \text{followers}(c)} \text{count}(c\ w')} \tag{2}$$

For example, consider the following three-word sequences with their respective counts:

Sequence	Count
`big green apple`	7
`big green pear`	4
`big green watermelon`	3
`small red apple`	1
`small red car`	2
`tiny grey mouse`	5

There are three contexts: *big green*, *small red* and *tiny grey*. The context *big green* can be followed by *apple*, *pear* or *watermelon*. The context *small red* can be followed by *apple* or *car* and the context *tiny grey* can be only followed by *mouse*. The probability of a word depends on the context. For example, the probability of *apple* in the context *big green* is:

$$p(\textit{apple} \mid \textit{big green}) = \frac{7}{7+4+3} = \frac{1}{2} \tag{3}$$

whilst in the context of *small red* it is:

$$p(\textit{apple} \mid \textit{small red}) = \frac{1}{1+2} = \frac{1}{3} \tag{4}$$

The entropy of words for the context *big green* would be:

$$H(\textit{big green}) = -\left(\frac{1}{2}\right) \cdot \log_2 \left(\frac{1}{2}\right) - \left(\frac{2}{7}\right) \cdot \log_2 \left(\frac{2}{7}\right) - \left(\frac{3}{14}\right) \cdot \log_2 \left(\frac{3}{14}\right) \tag{5}$$

Finally, the output for the given example counts would be:

```
big green   1.4926
small red   0.9183
tiny grey   0
```

In your output, you are not required to round your entropy values to a specific number of decimal digits. **(5 marks)**

1.2 Relational Join using MAPREDUCE

In this task you will perform a join operation in Hadoop. Let us assume that we have the relations **student(**`studentId`**,** `name`**)** and **marks(**`studentId`**,** `courseId`**,** `mark`**)** as shown below:

students

studentId	name
1	George
2	Anna

marks

studentId	courseId	mark
1	EXC	70
2	EXC	65
1	TTS	70
1	ADBS	80

and need to join them on the `studentId` field. Traditionally, this is an easy task when we deal with relational databases and can be performed by using the relational **join operator**. However, the way this join operation is performed drastically changes, when we assume our input is into a **single file** that stores information from both relations.

Assume the format of such a single input file storing data from two relations is as follows:

```
student   1   George
mark      1   EXC     70
student   2   Anna
mark      1   ADBS    80
mark      2   EXC     65
mark      1   TTS     80
```

The first column is a **tag** that shows from which relation the data comes from. Depending on this tag, we can assign meaning to the other columns. When the tag used is `mark`, we know that the second column refers to the `studentId`, the third to the `courseId` and the fourth refers to the `grade` the student took in this specific course. On the other hand, if the tag is `student`, we know that there are only two other columns, one with the `studentId` and one with the student `name`.

Task 7

◁ **Task**

Use the `uniLarge.txt` file perform a join operation on the `studentId` key and produce an output that will have the grades of each student as follows:

`name --> (course1, mark1) (course2, mark2) (course3, mark3) ...`

For example, for the previous input file your algorithm should return:

```
George --> (ADBS, 80) (EXC, 70) (TTS, 80)
Anna --> (EXC, 65)
```

(3 marks)

Task 8

◁ **Task**

What is the fictional name of the student (or students in case of equality) with the highest average when the number of lessons examined is greater than three? **(3 marks)**

2 Submission

To submit your work, please do the following:

1. Make sure that you store the output of your MAPREDUCE jobs in HDFS. Please store the output for the Xth task in the folder: `/user/sXXXXXXX/assignment1/taskX` (replace `sXXXXXXX` with your student number). This way we can easily check whether you got the right output. When marking, we will only be interested in the output for the **large** version inputs. Do not put the output for the small version inputs in the same folder as it can be mistaken for your output for the large version. **Do not delete these files from HDFS until you are told it is OK to do so.**

2. To submit your code, please prepare a directory for submission (it can have any name) that will contain one directory for each task and name the directory for the Xth task `taskX`. Inside each task folder, please include:

   - One file with `.sh` extension. This should be the shell script that executes your MAPREDUCE job(s) for Xth task.
   - One file with `.out` extension. This file should contain the first 20 lines of your output. If Hadoop splits your output into multiple files, please use the first twenty lines from the first output file (usually called `part-00000`). You can use the command:

     ```
     hdfs dfs -cat filename | head -20
     ```

     for showing just the first 20 lines of the file. If the whole output has less then 20 lines together, then your `*.out` file should contain the whole output.
   - Files with your code. This includes all code that you wrote to solve the task. **Do not submit a Word document or a PDF file—only submit a plain text file**. It is not required, but it will make marking easier if the file name with mapper, combiner, reducer code starts with `mapper`, `combiner` and `reducer` respectively. You can see an example below:

     ```
     submissionFolder/
     -- task1/
        -- run.sh
        -- output.out
        -- mapper.py
        -- mapper2.py
        -- reducer.py
        -- reducer2.py
        -- combiner.py
        -- combiner2.py
     -- task2/
        -- run.sh
        -- output.out
     ...
     ```

     This is an artificial example, it does not mean that you need to code in python or that you need two MAPREDUCE jobs with combiners to solve the first task.

Once you have prepared your submission directory with the specified structure, please run

```
/afs/inf.ed.ac.uk/group/teaching/exc/submit-assignment1.sh path/to/submission_folder
```

This will give you friendly warnings to help you spot if a folder (e.g. `task8`) or a file is missing. You will be asked whether you want to submit your submission directory (regardless of whether any warnings were given). You can submit as many times as you want, but only the last submission will be marked.