# TI2736-B: Assignment 2
# Big Data Processing

Due date: 04.12.2016 @ 11.59pm

Please submit your report and code via Blackboard (please do **not** copy & paste your code directly into the report). Make sure to include your name and student number in your report.
The assignment has two parts: a pen-and-paper part, as well as a practical part. The Hadoop programming exercises aim to familiarize you with Hadoop. **Please submit your source code, and the output you get once you run your code alongside the answers to the exercises.**

1. Data streams

   (a) Assume we have a data stream of salary information across the Netherlands. The stream elements have the form: *(employer, department, employeeID, salary)*. Employers are unique, but department titles are only unique within a single employer - different employers may have the same department "Human Resources". Similarly, employeeID are unique within an employer, but different employers can use the same IDs to identify their employees.

   Suppose we want to answer certain queries approximately from a 1/20th sample of the data. For each of the following queries, indicate how you would construct the sample to end up with a good estimate:

      i. For each employer, estimate the average number of employees in a department.
      ii. Estimate the percentage of employees who earn more than 100,000 Euros per year.
      iii. Estimate the percentage of departments where all employees make less than 40,000 Euros a year.
      iv. Estimate the percentage of employers with more than 10 employees.
      v. For each employer, estimate the average salary the employees receive across all departments.

   (b) Suppose you have a stream of integers: 3, 1, 4, 1, 5, 9, 2, 6, 5. Use the FM-sketch to estimate the number of distinct elements in the stream. Our hash functions will all be of the form $h(x) = ax + b \; mod \; 32$ for some $a$ and $b$. You should treat the result as a 5-bit binary integer. Determine the tail length for each stream element and the resulting estimate of the number of distinct elements if the hash function is:

$$h_1(x) = 2x + 1 \; mod \; 32 \tag{1}$$

$$h_2(x) = 3x + 7 \; mod \; 32 \tag{2}$$

$$h_3(x) = 4x \; mod \; 32 \tag{3}$$

(c) Suppose you have a stream of integers: 3, 1, 4, 1, 5, 9, 2, 6, 5, 9, 9, 3, 1, 9, 2. Apply the AMS algorithm to the data stream to compute the following:

i. the 3rd order moment for 3 variables with random positions 3, 8, 13

ii. the 3rd order moment for 5 variables with random positions 2, 3, 8, 9, 10

For comparison, also provide the true value of the 3rd moment.

(d) Employ the DGIM algorithm. Shown below is a data stream with $N = 22$ and the current bucket configuration. New elements enter the window at the right. Thus, the oldest bit of the window is the left-most bit shown.

$$\boxed{1\,0\,1\,1\,0\,0\,0\,1}\,0\,\boxed{1\,1\,1\,0\,1}\,\boxed{1\,0\,0\,1}\,0\,\boxed{1}\,\boxed{1}\,0$$

i. What is the largest possible bucket size for $N = 22$?

ii. What is the estimate of the number of 1's in the latest $k = 15$ bits of this window?

iii. The following bits enter the window, one at a time: 1 0 1 1 1 0 0 1. What is the bucket configuration in the window after this sequence of bits has been processed by DGIM?

iv. After having processed the bits from (iii), what is now the estimate of the number of 1's in the latest $k = 15$ bits of the window?

v. In the lecture, we have also covered how to generalize the DGIM algorithm from a bit stream to positive integers. Analogously to the slide example, work out the bit streams for the following stream of 8 numbers (oldest first): $(125, 2, 77, 5, 13, 9, 99, 56)$. Compute the result for $k = 3$.

2. In order to work on the following **Hadoop** exercises, you need to have a working version of Hadoop. We assume that you are using Cloudera's distribution of Hadoop (*CDH 5.8*). The virtual machine image can be downloaded at: http://www.cloudera.com/downloads/quickstart_vms/5-8.html. The CDH runs Hadoop on a single machine and has all its components already set up, including those we will use in later lectures[1].

   The instructions for running it with Virtual Box are as follows: Select at least 2GB memory (4GB or 8GB are preferred), choose RedHat 64-bit as OS and then load Cloudera's image. You are now ready to interact with Hadoop.

[1] It is of course also possible to install Hadoop and all necessary additional tools on your laptop without the CDH - just realize that we do not offer IT support in this case.

(a) Familiarize yourself with Hadoop's command-line interface by working through the section **The Command-Line Interface** in Chapter 3 of Tom White's *Hadoop: The Definite Guide, 4th Edition*. The book is available online through the TU Delft network: `http://bit.ly/1HCumS6`.
– nothing to submit here –

(b) Cloudera offers a useful set of tutorials (**WordCount v1.0, v2.0, v3.0**) through which you will gain familiarity with Hadoop's boilerplate code for the mapper/reducer and how to deploy a Hadoop job. Work through the tutorials carefully - do not only read through the walkthrough of the source code, but also run the code yourself! You find the tutorials here: `http://bit.ly/2gkShZR`. Make sure to understand the input/output data types of the Mapper and Reducer.
– nothing to submit here –

(c) Download the following six works by Shakespeare from Project Gutenberg:

- `http://www.gutenberg.org/cache/epub/1524/pg1524.txt`
- `http://www.gutenberg.org/cache/epub/1112/pg1112.txt`
- `http://www.gutenberg.org/cache/epub/2267/pg2267.txt`
- `http://www.gutenberg.org/cache/epub/2253/pg2253.txt`
- `http://www.gutenberg.org/cache/epub/1513/pg1513.txt`
- `http://www.gutenberg.org/cache/epub/1120/pg1120.txt`

.

Run Cloudera's **WordCount v2.0** using the Shakespeare texts' as corpus to analyze. Among the status information you also find statistics about the Map/Combine/Reduce input and output records. Based on these statistics, what can you say about the Combiner? To what extent does it (or does it not) improve the efficiency of the program?

(d) Copy the output directory from HDFS to your local directory. Have a look at `part-r-00000`[2]; it contains the output of the Reducer (i.e. the final result). How many unique terms does the file contain?

(e) Make three changes to the tokenizer within `map()` to reduce the number of unique terms found (e.g. remove non-alphanumeric terms) and run the job again. Which changes did you make and how do they influence the number of unique terms found?

(f) Adapt the `map()/reduce()` functions to produce an inverted index in a way explained in lecture 4. Generate the inverted index.

[2]If you have more than one part-r-* file, concatenate them for the final result.