TI2736-B: Assignment 6
Big Data Processing

Due date: 15.01.2017 @ 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 goal of this assignment is to familiarize yourself with the concepts used for graph processing in Pregel/Giraph. The first three exercises are pen-and-paper exercises. The last exercise asks you to implement a piece of code in Giraph - this exercise will take time, as Giraph is not the easiest tool to work with. Please take this advice into account when scheduling your work!

1. Explain how *Aggregators* can be used to take care of dangling nodes in the PageRank computation. Assume the following:

   - several aggregators can exist at any time,
   - in superstep S a node can send values to one or more aggregators, and,
   - in superstep $S + 1$ the aggregated values from S are available to all nodes.

2. Given a number of items (e.g. points in a 2D space), the goal of the *K-means clustering* algorithm is to assign each item to one of k clusters (the number k is fixed in advance).
   Below you find the pseudo-code of K-means. A visualization of a toy example is shown in Figure 1.

```
1  Input: items to be clustered, number k (#clusters)
2  Output: cluster label of each item

3  Initialise:
4    - Pick k items randomly (the initial cluster centroids)
5    - For each item:
6       - Compute distance to all centroids
7       - Assign item to the cluster with minimum distance

8  Repeat until no more label changes or 1000 iterations reached:
9    - Re-compute cluster centroids (the mean of assigned items)
10   - For each item:
11      - Compute distance to all centroids
12      - Assign item to the cluster with minimum distance
```

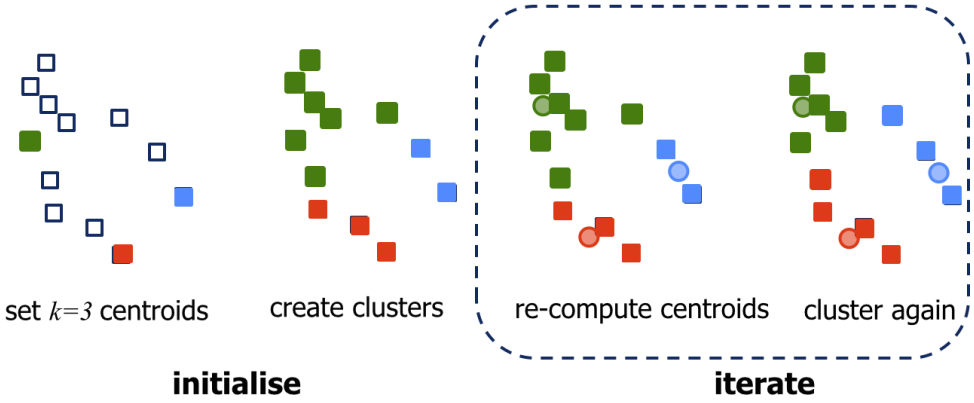

Figure 1: K-means example with $k = 3$ and 12 items to cluster.

How can K-means be computed in the Giraph/Pregel framework of thinking? Clearly indicate how you transform this problem space into a graph (what are vertices and what are edges here?). Then, explain superstep by superstep the data contained in each vertex, the local computation(s), messages send/received and votes to halt.

3. The following paragraphs define the so-called **local clustering coefficient** of a node in a graph; intuitively, the larger the local clustering coefficient of a node, the more connected the node and its direct neighbours are to each other. Such a statistic is useful for instance to compute how tightly or loosely connected members of different communities are (a community can consist of all authors of a particular research field, or of all Twitter users from a particular geographic area, etc.).

   Let k_i be the number of outgoing edges of a node N_i. Let NH_i be the set of all nodes which have an incoming edge from N_i (this is also called the neighbourhood of N_i). Finally, let ℓ_i be the number of edges between the nodes in NH_i.

   Then, the local clustering coefficient of N_i is defined as the number of edges between the nodes in the neighbourhood divided by the total number of possible edges between them (this value is $k_i \times (k_i - 1)$):

$$c_i = \frac{\ell_i}{k_i(k_i - 1)} \tag{1}$$

   As an example, lets look at node $N2$ in Figure 2: it has outgoing edges to $k_2 = 4$ other nodes with $NH_2 = \{N3, N4, N5, N8\}$. We also find one edge between nodes in NH_2: between $N4$ an $N8$. Thus,

$$c_2 = \frac{1}{4 \times 3}. \tag{2}$$

How can the clustering coefficient of a node be computed in the Giraph/Pregel framework of thinking? Explain superstep by superstep the local computation(s), messages send and votes to halt. Assume the following input format: the graph is represented in adjacency list format with each vertex containing an list of its outgoing edges.

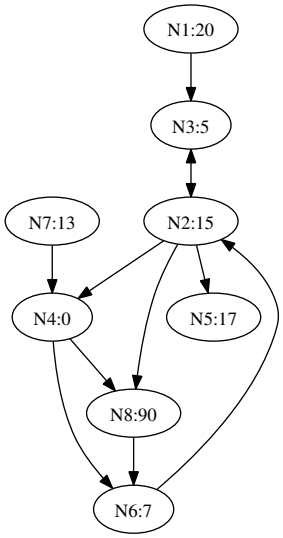

Figure 2: A directed graph with eight nodes. The label in each node shows the node identifier, e.g. $N5$, and the node's value, e.g. 17 for $N5$.

4. Finally, we turn to using Giraph.

   (a) You first need to install Giraph as outlined in the procedure `setup-giraph.pdf` available on Blackboard. As part of the setup, you will also run your first Giraph job (single source shortest path) on a toy dataset. Note that the installation of Giraph is not an easy one, it will take you some time to get it right. Having run the single source shortest paths algorithm on the provided toy graph, **report** the `Giraph Stats` and `Giraph Timers` output of the job you ran. You find those statistics among the program statistics that are printed out to console (reporting these numbers shows that you were able to install Giraph successfully).

   (b) Implement the local clustering coefficient from the previous exercise in Giraph. The simplest way to add your own code to Giraph is to take one of the examples as your blueprint and adapt it where necessary. If you followed the setup as described in 5(a) you can find the shortest path source code at:
   `/usr/local/giraph/giraph-examples/src/main/java/org/apache/giraph/examples/SimpleShortestPathsComputation.java`
   Save the new class within the same folder as the original and recompile the

Maven project. You are now able to use your own class in the same manner as the examples provided by Giraph.

(c) To test your implementation, use **one directed graph data set** of your own choice from `http://snap.stanford.edu/data/`. Note that in most cases you will have to adjust the data slightly to a format that is compatible with your Giraph code (you can use any tool of your choice for that step).
What is the average clustering coefficient of your chosen data set? Is it similar to the one listed on the dataset's SNAP webpage? How long did it take Giraph to compute the local clustering coefficient?
Submit your code alongside the answers to these questions.