Big Data Algorithms
Lab-1

Due: 14.11.2021

1. (5pts) **Words importance in documents**
For this exercise, you can use word cloud generator from previous list. Console user interface can be replaced with simple lists of document paths hardcoded into the application.

   (a) Get yourself familiar with TF.IDF measure (*Term Frequency times Inverse Document Frequency*).

   (b) Modify your generator to support loading multiple documents - you should be able to print the n-most frequently appearing words per document and for all loaded files.

   (c) Implement the TF.IDF calculating function - lets assume that final output should be in form of *Map[DocumentId, Map[Word, TFIDF]]*, where *DocumentId* will be some value uniquely identifying the document loaded (ex. file name - we do not want to load any duplicated documents); *type Word = String* will represent word; *TFIDF* will be value of TF.IDF for word *Word* and document *DocumentId*.

   (d) Test your application on some two different books, for example printing 10 of the most frequent words for both, for whole collection and then words with the highest TF.IDF value. Do you see differences?

   (e) ($*$, bonus 3pts) Perform experiments. Prepare following documents:

      - 10 documents with similar topics (ex. articles about big data, chapters of a single book, several books from one single series etc.)
      - 10 documents about some unique topics (each document should has different topic; ex. one Tolkien book, one cooking book, one BD coursebook etc.)

      Try loading books with different configurations and analyze the results:

      i. all documents with similar topic,
      ii. half documents with similar topic, half with other topics,
      iii. all documents withh unique topics.

      Check how the primitive words frequencies give different results compared to TF.IDF - would you be able to match results to specific document?

2. (4pts) **MapReduce - edges inversion**
To solve this task you have to use MapReduce algorithm design - meaning you have to create two functions:

   - map function - which will transform input into the *(index, value)* format,
   - reduce function - which will combine the map function output into some specific output.

   The directed graph is defined by the list of neighbors in a following form:
```
[
(1, [2,3]),
(3, [1, 5]),
(2, [5]),
(5, [])
],
```
   where node 1 is connected to 2 and 3, node 3 to 1 and 5 and so on. Design MapReduce algorithm, taking as an input such defined graph and returning the graph with inverted edges (ex. `(1, [2,3])` should be converted to `(2, [1]), (3, [1])`).