

Big Data Algorithms

Lab-2

Due: 12.12.2021

1. (2pts) Graph analysis

Let $G = (V, E)$ be a directed graph (V - vertices, E - edges). For $v \in V$, we denote:

$$\text{inDeg}(v) = |\{u \in V : (u, v) \in E\}|$$

and

$$\text{outDeg}(v) = |\{u \in V : (v, u) \in E\}|.$$

Using Map-Reduce scheme, design procedure which will calculate:

$$\{(v, \text{inDeg}(v), \text{outDeg}(v)) : v \in V\}.$$

The input for the procedure will be in form of file, storing single edge $(v, u) \in E$ per line, ex.

```
1 2
1 5
5 1
```

will form the graph consisting of $V = \{1, 2, 5\}$ and $E = \{(1, 2), (1, 5), (5, 1)\}$. The output should be:

```
{
  (1, inDeg(1) = 1, outDeg(1) = 2),
  (2, inDeg(2) = 1, outDeg(2) = 0),
  (5, inDeg(5) = 1, outDeg(5) = 1),
}
```

Verify algorithm on [Stanford web graph](#). Solution should be efficient and define clear *map* and *reduce* functions, ex.

```
Source.fromFile(graphFile).getLines.map(mapFunction).reduce(reduceFunction)
should return the final set/map.
```

Hint 1: Theory required for following exercises you can find in [Mining of Massive Datasets - Chapter 3](#).

Hint 2: For following exercises prepare the results before the laboratories (as some calculations might take some time).

3. (2pts) Documents similarity

Prepare code calculating the Jaccard similarity between two documents. For example, given `bookA : String` and `bookB : String`:

- (a) For each book generate set of k -shingles
`(bookAShingles : Set[String], bookBShingles : Set[String]).`
- (b) Calculate the Jaccard similarity for these sets:
`JaccardFull(bookAShingles, bookBShingles) : Float`

Use it for collections of books from previous lists (ex. for 6 books, where 3 are similar, and 3 are unique). Check the results for shingle sizes $k \in \{4, \dots, 13\}$.

4. (6pts) Min-hashing

In this exercise you will extend the Jaccard similarity calculating application to use the *minhashing* technique for generating signatures for each document:

- (a) For each book generate set of k -shingles (as in previous exercise).

- (b) Prepare set of n hashing functions to use for the *minhashing* algorithm.
Hint: you can generate random hashing tables for simulating the set permutation (ex. for permutating the set of g elements, you can generate array h of size g , consisting of randomly shuffled integers from 0 to $g - 1$ - then you can use this array as hashing function $h(i)$ for integer i).
- (c) Prepare function calculating the minhash signatures for collection of input sets (ex. shingle sets) and set of hashing functions, ex.:
- ```
def Signatures(shingleSets:Array[Set[String]],hashFunctions:(Int)=>Int)
: Array[Array[Int]],
```
- where the result will be a signature array for each of the input set (you do not have to use exactly this signature - this is only some example), which for our exemplary input will return:
- ```
(bookASignature : Set[String], bookBSignature : Set[String]).
```
- (function should be able to calculate the signature for more then two documents)
- (d) Calculate the Jaccard similarity approximation using created signatures:
- ```
JaccardSignatures(bookASignature, bookBSignature) : Float
```

Verify the results for books from previous exercises. Check different shingle sizes  $k \in \{4, \dots, 13\}$  and numbers of hashing functions  $n \in \{10, 100, 250, 500\}$ . Compare the results with full Jaccard similarity from previous exercises - are your results similar?