

CME 323: Distributed Algorithms and Optimization

Instructor: Reza Zadeh (rezab@stanford.edu)

HW#3 – Due at the beginning of class Tuesday May 22

1. **Intro to Spark** Download the following materials.

- Slides
- Spark and Data

Now, answer the following questions.

- (a) Checkpoint on slide 11.
- (b) Checkpoint on slide 55.
- (c) Checkpoint on slide 60. Note: slide 59 references the file `CONTRIBUTING.md` which is in the provided zip file. Instead, use the file `website/getting-started.md`

Submit your code and answers.

2. **Intro to Map Reduce** Warmup question. Assume you are given a typical MapReduce implementation where you only have to write the Map and Reduce functions. The Map function you will write takes as input a (key, value) record and returns either a (key, value) record or nothing. The Reduce function you will write takes as input (key, list of all values for that key) and returns either a record or nothing. The framework already takes care of iterating the Map function over all the records in the input file, key-based intermediate data transfer between Map and Reduce, and storing the returned value of Reduce you do not have to worry about these. You are now given an input file which contains comprehensive information about a social network that has asymmetrical (directed) links, i.e., a network where users follow other users but not necessarily vice-versa (e.g., Twitter). Each record in this input file is (userid-a, userid-b), where userid-a follows userid-b (i.e., points to it). Note that this record tells you nothing about whether or not userid-b follows userid-a. Write a MapReduce program (i.e., Map function and Reduce function) that outputs all pairs of users who follow each other. Pseudocode is OK.
3. **Word Count Shuffle** Warmup question. Consider counting the number of occurrences of words in a collection of documents, where there are only k possible words. Write a MapReduce to achieve this, and analyze the shuffle size with and without combiners being used (assuming B mappers are used).
4. **Prefix Sum** The *prefix-sum* operator takes an array a_1, \dots, a_n and returns an array s_1, \dots, s_n where $s_i = \sum_{j < i} a_j$. For example, starting with an array `[17 0 5 32]` it returns `[17 17 22 54]`. Describe (in detail) how to implement *prefix-sum* in MapReduce, where the input is stored as $\langle i, a_i \rangle$. That is, the key is the position in the array, and the value is the value at that position. Analyze the shuffle size and the reduce-key space and time complexity.

5. **Shallow Graphs** For an undirected graph $G = (V, E)$ with n vertices and m edges ($m \geq n$), we say that G is shallow if for every pair of vertices $u, v \in V$, there is a path from u to v of length at most 2 (i.e. using at most two edges).

- (a) Give an algorithm that can decide whether G is shallow in $O(n^{2.376})$ time.
- (b) Given an $n \times r$ matrix A and an $r \times n$ matrix B where $r \leq n$, show that we can multiply A and B in $O((n/r)^2 r^{2.376})$ time. Hint: use the fact that we can multiply two $r \times r$ matrices in $O(r^{2.376})$ time.
- (c) Give an algorithm that can decide whether G is shallow in $O(m^{0.55} n^{1.45})$ time. Hint: consider length-2 paths that go from low-degree vertices and length-2 paths that go through high-degree vertices separately. Use result from part (b).