Ex .1 - MapReduce

1. Check `ex1.Map` class for details.
2. Check `ex1.Reduce` class for details.
3. Check `ex1.run()` function for details.
4. The running time is shown below.

	100k	200k	300k	400k	1000k
Hadoop Streaming	20.541	23.202	27.428	35.672	73.504
Java	15.141	20.503	25.720	33.081	72.506

Ex .2 - Avro

1. The three API styles into which Avro can be used in MapReduce are
    1. **specific**:

        When using specific record, User class is generated, it includes additional methods with builders. It is more safe to use this strongly typed to avoid accessing fields by improper naming that could occur on Generic Records. Both DatumReader and DatumWriter are typed with User class.

    2. **generic**:

        When using Generic Record, there is no need to include plugins to generate the POJO from the schema. In this case, operations with Generic Record are the same as with the map - accessing values by the key name of the field. If the field is not set null value is returned.

    3. **reflect**:

        When using reflect record, Pojo record should be defined, and here Lombok library with its annotations to generate getter, setters and constructors is useful. Annotations @AvroDoc and @Nullable are from AVRO package and they are used to instrument Avro for schema generation and constraints. The important difference that schema on write is generated using reflection from the Pojo class on the fly. It uses special Avro annotations to fill additional info into the schema.

2. Check `ex2.java` for details.

Ex. 3 - Bloom Filter

1. A **Bloom filter** is a space-efficient probabilistic data structure conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not – in other words, a query returns either "possibly in set" or "definitely not in set". Bloom proposed the technique for applications where the amount of source data would require an impractically large amount of memory.

   An *empty Bloom filter* is a bit array of m bits, all set to 0. There must also be k different hash defined, each of which maps or hashes some set element to one of the m array positions, generating a uniform random distribution. Typically, k is a small constant which depends on the desired false error rate ε, while m is proportional to k and the number of elements to be added.

   To *add* an element, feed it to each of the k hash functions to get k array positions. Set the bits at all these positions to 1.

   To *query* for an element (test whether it is in the set), feed it to each of the k hash functions to get k array positions. If *any* of the bits at these positions is 0, the element is definitely not in the set; if it were, then all the bits would have been set to 1 when it was inserted. If all are 1, then either the element is in the set, *or* the bits have by chance been set to 1 during the insertion of other elements, resulting in a false positive. In a simple Bloom filter, there is no way to distinguish between the two cases, but more advanced techniques can address this problem.

2. The BloomFilter filters students whose ID ends with a "3". Check `ex3.Map` class for details.
3. First, I take an `or` operation to all BloomFilters in `Iterable<BloomFilter>` to combine all Bloom filters together in the reducer. Check `ex3.Reduce` class for details.

   The final result is shown in `result.txt` file with `avro-tools-1.10.2.jar`. We can see students whose ID ends with a "3" have been filtered out successfully.

Reference

1. Avro Generic, Specific and Reflection Writer/Reader | Roman Tsypuk Blog
2. Bloom filter - Wikipedia