PySpark HW report

Main task

Task: Create a map reduce word count application and run it

Project Structure:

```
main-task
├── main.py # Python script
├── target # Result directory, ignored in .gitignore
│   ├── _SUCCESS
│   ├── part-00000
│   └── part-00001
└── wnp.txt # War and Peace book text
```

Firstly, I installed PySpark via running `pip3 install pyspark`

Then I created a project and started exploring PySpark without any tutorials 😄

I didn't save the exact code I wrote but it was somewhat similar to final version:

```
def other_solution(self):
    '''
        My custom solution
    '''

    self.spark.read.text(INPUT_PATH).withColumn(
        'word',
        f.explode(
            f.split(
                # Splitting strings and getting separate words
                f.lower(f.col('value')),
                ' ',
            )
        )
    ).filter(
        # Filtering empty strings
        f.col('value') != '',
    ).groupBy(
        'word',
    ).count().sort(
        # Counting and sorting by count descending
        'count',
        ascending=False,
    ).show(n=WORDS_NUMBER)
```

Next I ran command `python3 main.py` and saw the following output:

```
+----+-----+
|word|count|
+----+-----+
| the|34270|
| and|21392|
|  to|16504|
|  of|14909|
|   a|10387|
|  he| 9298|
|  in| 8737|
| his| 7930|
|that| 7410|
| was| 7205|
+----+-----+
only showing top 10 rows
```

Quite good, however, I didn't use Reduce, so I checked out the tutorial and wrote next code based on it:

```
def main_solution(self):
    '''
        Solution based on presentation tutorial
    '''

    res = self.spark.sparkContext.textFile(
        self.input_path,
    ).flatMap(
        # Splitting words
        lambda line: line.split(' '),
    ).filter(
        # Filtering words: for instance, 'abc' is a word and 'abc123.,-'
is not
        lambda line: IS_WORD.match(line),
    ).map(
        lambda word: (word, 1),
    ).reduceByKey(
        lambda count1, count2: count1 + count2,
    ).sortBy(
        # Sorting by count descending
        lambda pair: pair[1],
        ascending=False,
    )

    res.saveAsTextFile(self.output_path)
    print(
        *map(lambda pair: f'{pair[0]}: {pair[1]}', res.collect()
[:WORDS_NUMBER]),
        sep='\n',
    )
```

So I ran the app and got target directory with all the words counts:

```
...
('from', 2517)
('you', 2422)
('said', 2406)
('were', 2352)
('by', 2316)
...
('scented,', 1)
('canceled.', 1)
('wearisome.”', 1)
('wound-up', 1)
('Novosíltsev’s', 1)
...
```

Also I saw top 10 words were written to terminal:

```
the: 31714
and: 20560
to: 16324
of: 14860
a: 10017
in: 8232
he: 7631
his: 7630
that: 7228
was: 7193
```

As you can see, the results are a bit different because in my solution I only filtered empty strings, while in the main one I wrote regular expression which checks that all of the symbols are English alphabet letters

Now we used MapReduce and are ready to work with Spark: `spark-submit main.py`

Spark when it just started:

Spark when it finished working:

There are so many jobs because I am running both of the solutions

Extra task

Task: Launch Spark cluster containing 1 master and 2 workers and run the previous app on it