Exercise 3

3B

Using the output from EX 2 in the form of `word<tab>counter` pairs
we did the following steps:

step 1 DATA

We loaded the merged output file `data.txt` of the word counter to another folder into the hadoop dfs, allowing us to use the it in another hadoop MapReduce framework operation.

step 2 MAP-REDUCE

We needed to use the counter to be key key and anything to be the value as we never really use it in the reducer code.
We use an incrementing accumulator instead, when the next key kicks in the accumulator goes back to 0 counting the next (automatically sorted in the shuffle phase) key.

```
import sys


# input comes from STDIN (standard input)
# in the form of <word><\t><count>
for line in sys.stdin:
    word, count = line.split('\t')
    word = word.strip()
    if len(word) > 0 :
        try:
            count = int(count)

        except ValueError:
            # count was not a number, so silently
            # ignore/discard this line
            print "int cast failed!"
            continue

        # swapping key-value to take advantage of the
        # map reduce framework shuffle reordering by key
        print '%d\t%s' % (count, 1)
```

The reducer funtion aggregate the data ordered in the automatic shuffle phase. The ordering is done automatically on the key which in this case is the counter, allowing us to fill a list of words having the same counter.

```
"""reducer.py"""

from operator import itemgetter
import sys

current_count = 0
acc = 0

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # parse the input we got from mapper.py
    # <count> <1>
    count, _ = line.split('\t')

    # convert count (currently a string) to int
    try:
        count = int(count)

    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # if first value
    if current_count == 0:
        current_count = count

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_count == count:
        acc += 1

    else:
        print '%d\t%d'%(current_count,acc)
        current_count = count
        acc = 1

# do not forget to output the last record <--
if current_count == count:
    print '%d\t%d'%(current_count,acc)
```

step 3 RUNNIN TASK

this time we anted to make sure the algorithm worked in the example data first so we simulated it locally:

```
[user_lsc_3@it B]$ cat example.txt
car     3
the     6
house   3
phone   5
pen     3
glass   3
battery 5
[user_lsc_3@it B]$ cat example.txt | python mapper.py | sort -k1,1 | python reducer.py | sort -k2,2 -nr
3       4
5       2
6       1
```

We ran the task the usual way using the exercise 3 data already loaded

```
hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-*streaming*.jar -mapper mapper.py -file ./mapper.py -r
```

step 4

After merging with `hdfs dfs getmerge` we sorted by occurence of the key (which is a word frequency) and observed the obteined occurrence of frequencies in the harry potter books:

```
[user_lsc_3@it B]$ sort output.txt -k2,2 -nr | head
1       5158
2       2045
3       1172
4       838
5       548
6       435
7       344
8       313
9       280
10      220
```