STSCI 5065 HW2

Franklin Zhao (qz297)

03/11/2019

Question1

a) {100KB / [(50*1000/1000)KB/ms] + 3 ms} * (5 * 10^6KB / 100KB) = 5ms * 5 * 10^4 = 250s

b) {100 * 1000KB / [(50*1000/1000)KB/ms] + 3ms} * (5 * 1000MB / 100MB) = 20003ms * 50 = 100.15s

c) (100.15 - 250) / 250 = -59.94%

d) If the file size is small, then the total time spent will be roughly the same in both systems. In this case HDFS will not be necessary and large block size can be underutilized. So the block size in Q1a is better in this scenario.

However if the file size is large (like we calculated above), then large block size using HDFS will be much faster. So the block size in Q1b is better in this scenario.

Question2

a)
```
hadoop fs -mkdir /course-data
```

b)
```
mkdir /data-set
```

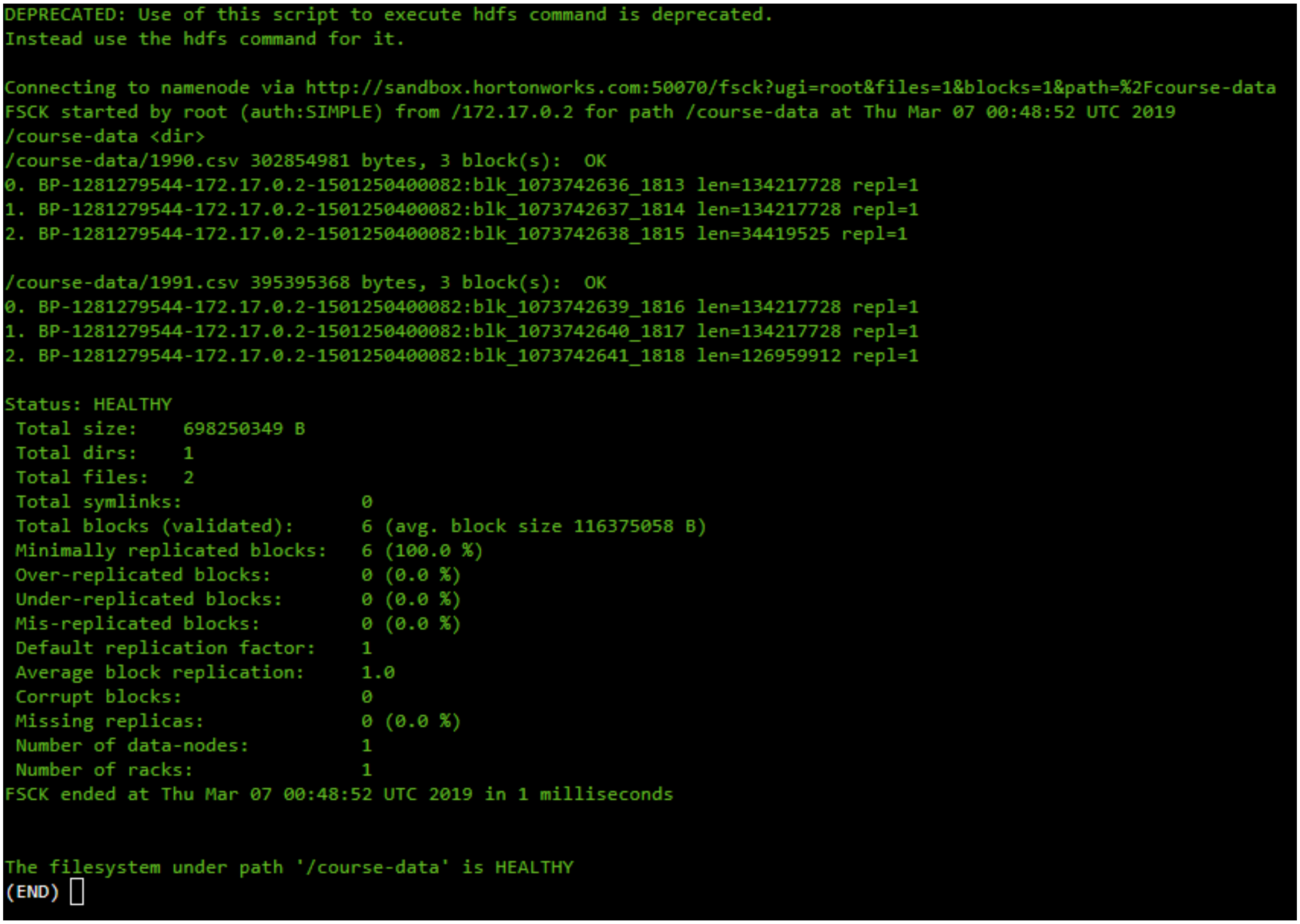
c)
```
cd /data-set/
wget http://data.gdeltproject.org/events/1990.zip
unzip 1990.zip
wget http://data.gdeltproject.org/events/1991.zip
unzip 1991.zip
```

d)
```
hadoop fs -put 1990.csv 1991.csv /course-data
```

e)
```
hadoop fs -ls /course-data
```

f)
```
hadoop fsck /course-data -files -blocks | less
```

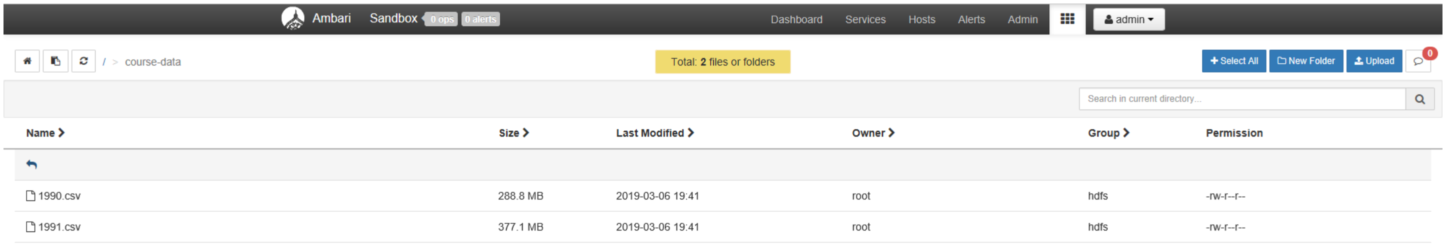
Both 1990.csv and 1991.csv use 3 blocks (**6 blocks in total**)



```
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Connecting to namenode via http://sandbox.hortonworks.com:50070/fsck?ugi=root&files=1&blocks=1&path=%2Fcourse-data
FSCK started by root (auth:SIMPLE) from /172.17.0.2 for path /course-data at Thu Mar 07 00:48:52 UTC 2019
/course-data <dir>
/course-data/1990.csv 302854981 bytes, 3 block(s):  OK
0. BP-1281279544-172.17.0.2-1501250400082:blk_1073742636_1813 len=134217728 repl=1
1. BP-1281279544-172.17.0.2-1501250400082:blk_1073742637_1814 len=134217728 repl=1
2. BP-1281279544-172.17.0.2-1501250400082:blk_1073742638_1815 len=34419525 repl=1

/course-data/1991.csv 395395368 bytes, 3 block(s):  OK
0. BP-1281279544-172.17.0.2-1501250400082:blk_1073742639_1816 len=134217728 repl=1
1. BP-1281279544-172.17.0.2-1501250400082:blk_1073742640_1817 len=134217728 repl=1
2. BP-1281279544-172.17.0.2-1501250400082:blk_1073742641_1818 len=126959912 repl=1

Status: HEALTHY
 Total size:    698250349 B
 Total dirs:    1
 Total files:   2
 Total symlinks:                0
 Total blocks (validated):      6 (avg. block size 116375058 B)
 Minimally replicated blocks:   6 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    1
 Average block replication:     1.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          1
 Number of racks:               1
FSCK ended at Thu Mar 07 00:48:52 UTC 2019 in 1 milliseconds


The filesystem under path '/course-data' is HEALTHY
(END)
```

Question3

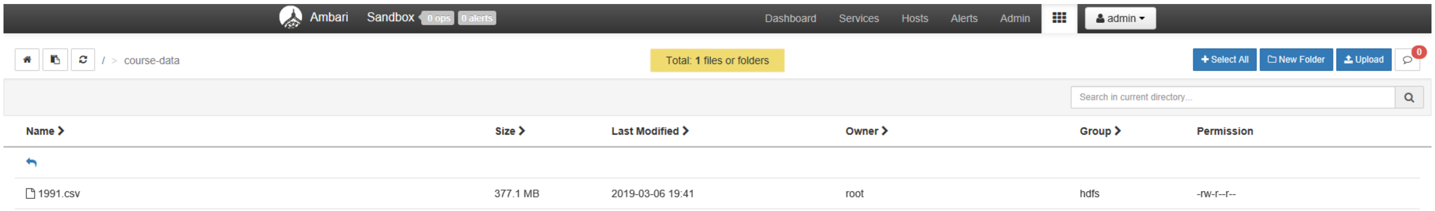
a)



Total: 2 files or folders

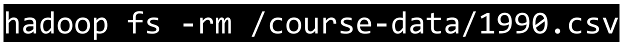
Name	Size	Last Modified	Owner	Group	Permission
1990.csv	288.8 MB	2019-03-06 19:41	root	hdfs	-rw-r--r--
1991.csv	377.1 MB	2019-03-06 19:41	root	hdfs	-rw-r--r--

b)



Total: 1 files or folders

Name	Size	Last Modified	Owner	Group	Permission
1991.csv	377.1 MB	2019-03-06 19:41	root	hdfs	-rw-r--r--



```
hadoop fs -rm /course-data/1990.csv
```

Question4

```
mkdir /HW2Q4
cd /HW2Q4/
scp -P 2222 shakespeare.txt root@127.0.0.1:/HW2Q4
```

a) `vi WDmapper.py`

```
#!/usr/bin/env python

'''
The script is used for mapping phase (word count)
'''
import sys
# regular expressions can increase the efficiency
# and avoid too many lines of code
import re

# input from STDIN: reading one line in one loop
for line in sys.stdin:
    # remove the front/end blanks of each line
    line = line.strip()
    # split words by whitespaces
    words = re.split('\s', line)
    # word processing in one loop
    for word in words:
        # the output values are tab-delimited
        # the word count is 1 since each for loop only reads one word
        # remove all charaters other than alphanumeric in a word
        # by doing this we may 'change' some words such as turning I'm into Im
        # but it won't influence the result of world count
        word = re.sub('\W+', '', word)
        if word:
            print('%s\t%s' % (word, 1))
```

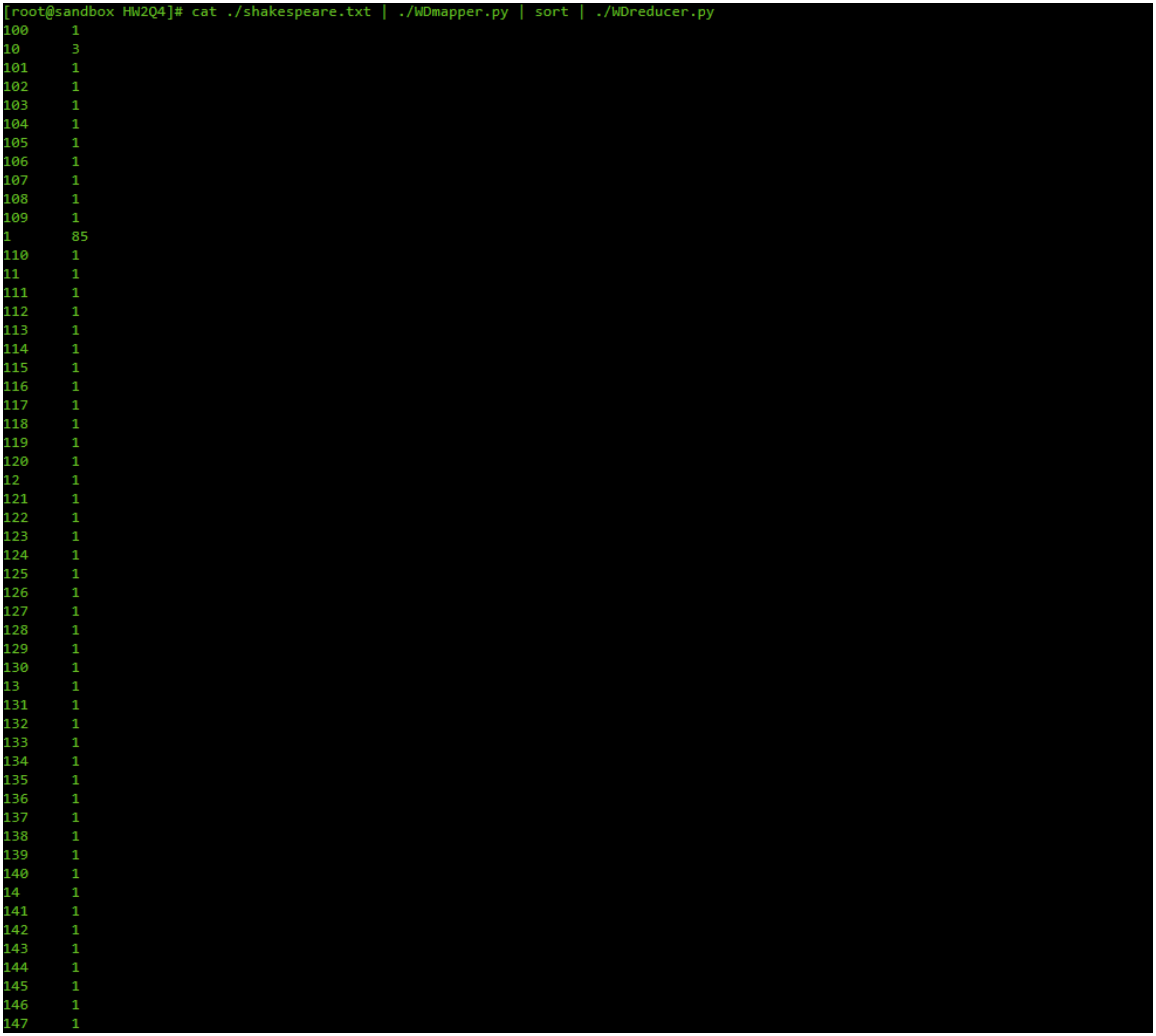
```
vi WDreducer.py
```

```
#!/usr/bin/env python

'''
The script is for reducing phase (word count)
For word count, since the task remains unchanged,
it is basically the same as the script provided by the instructor
'''
import sys
```

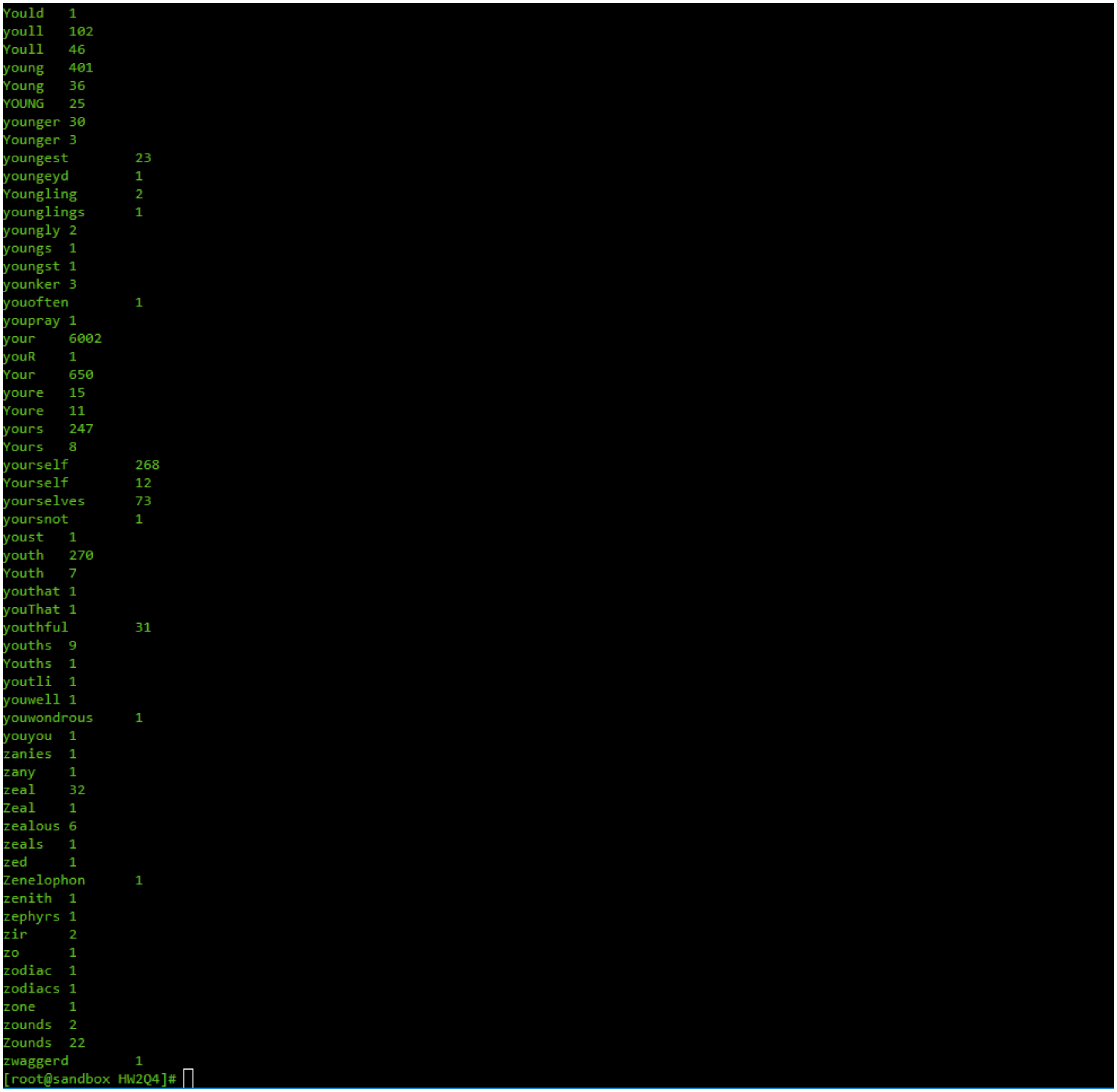
```
current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    line = line.strip()

    # parse the input we got from WDmapper.py
    word, count = line.split('\t', 1)

    # convert count (a tring) to int
    try:
        count = int(count)
    except ValueError:
        # if count was not a number, silently ignore the line
        continue

    # Hadoop sorts map output by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            #write result to STDOUT
            print('%s\t%s' % (current_word, current_count))
        current_count = count
        current_word = word

# output the last word if needed!
if current_word == word:
    print('%s\t%s' % (current_word, current_count))
```

```
chmod +x WDmapper.py
chmod +x WDreducer.py
cat ./shakespeare.txt | ./WDmapper.py | sort | ./WDreducer.py
```

Result screenshots (only first & last screen included):



```
[root@sandbox HW2Q4]# cat ./shakespeare.txt | ./WDmapper.py | sort | ./WDreducer.py
100     1
10      3
101     1
102     1
103     1
104     1
105     1
106     1
107     1
108     1
109     1
1       85
110     1
11      1
111     1
112     1
113     1
114     1
115     1
116     1
117     1
118     1
119     1
120     1
12      1
121     1
122     1
123     1
124     1
125     1
126     1
127     1
128     1
129     1
130     1
13      1
131     1
132     1
133     1
134     1
135     1
136     1
137     1
138     1
139     1
140     1
14      1
141     1
142     1
143     1
144     1
145     1
146     1
147     1
```

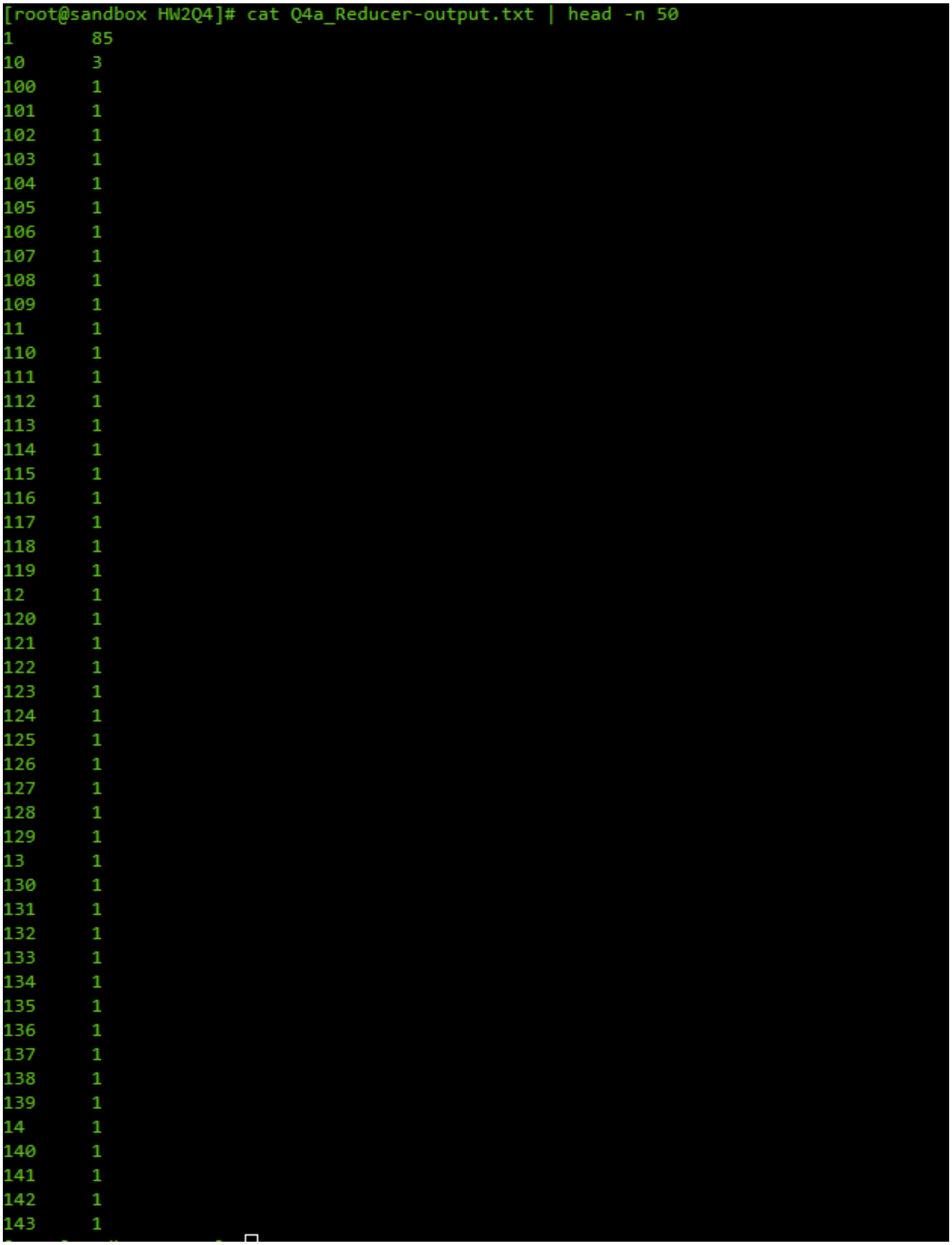


```
Yould   1
youll   102
Youll   46
young   401
Young   36
YOUNG   25
younger 30
Younger 3
youngest        23
youngeyd        1
Youngling       2
younglings      1
youngly 2
youngs  1
youngst 1
younker 3
youoften        1
youpray 1
your    6002
youR    1
Your    650
youre   15
Youre   11
yours   247
Yours   8
yourself        268
Yourself        12
yourselves      73
yoursnot        1
youst   1
youth   270
Youth   7
youthat 1
youThat 1
youthful        31
youths  9
Youths  1
youtli  1
youwell 1
youwondrous     1
youyou  1
zanies  1
zany    1
zeal    32
Zeal    1
zealous 6
zeals   1
zed     1
Zenelophon      1
zenith  1
zephyrs 1
zir     2
zo      1
zodiac  1
zodiacs 1
zone    1
zounds  2
Zounds  22
zwaggerd        1
[root@sandbox HW2Q4]#
```

Run in HDFS:

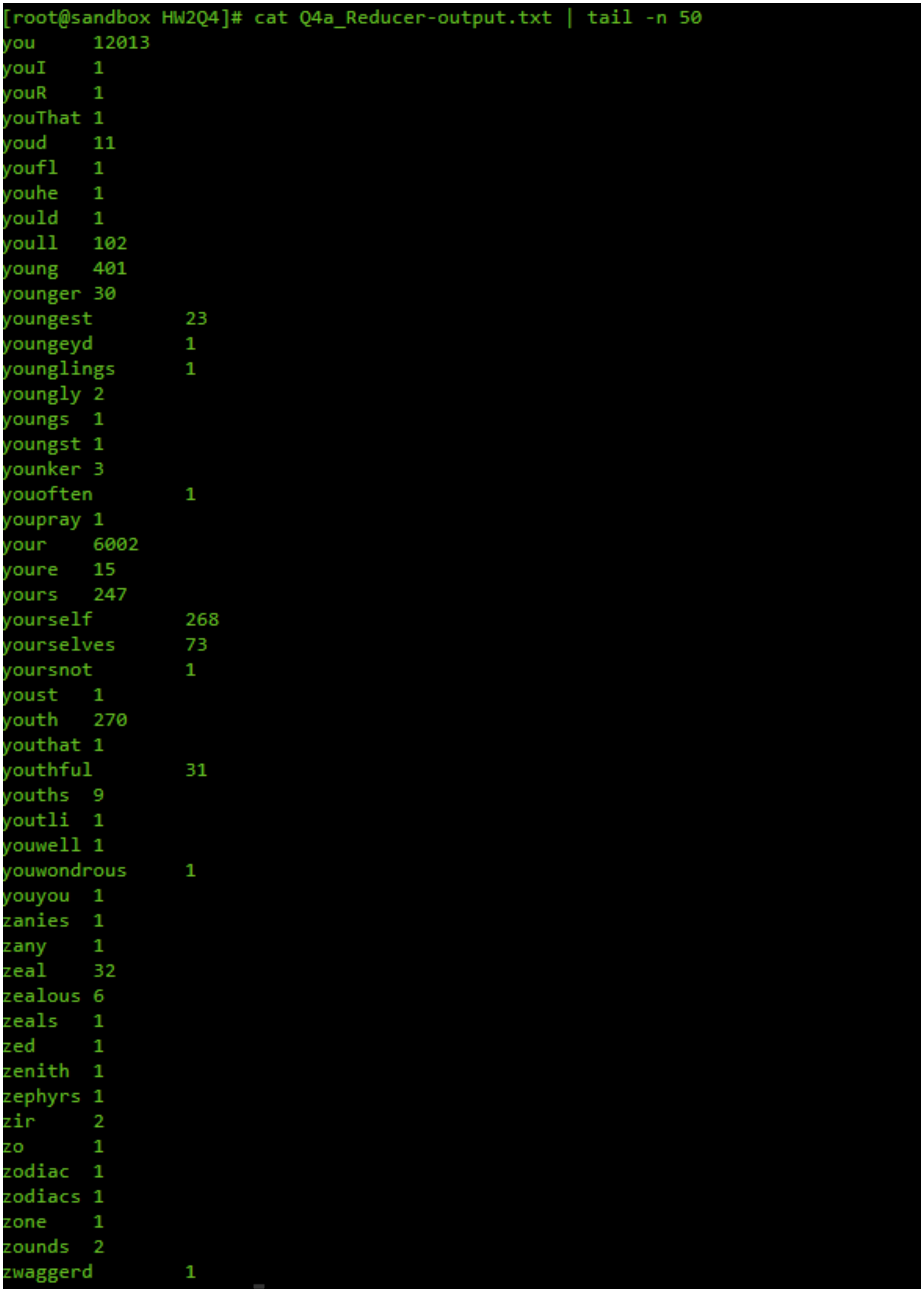
```
hadoop jar /usr/hdp/2.6.1.0-129/hadoop-mapreduce/hadoop-streaming.jar\
-file WDmapper.py -file WDreducer.py\
-mapper WDmapper.py -reducer WDreducer.py\
-input /course-data/shakespeare.txt -output /course-data/Q4a_reducer-output/
```

```
scp -P 2222 Q4b_Reducer-output.txt root@127.0.0.1:/HW2Q4
cat Q4a_Reducer-output.txt | head -n 50
```



```
[root@sandbox HW2Q4]# cat Q4a_Reducer-output.txt | head -n 50
1       85
10      3
100     1
101     1
102     1
103     1
104     1
105     1
106     1
107     1
108     1
109     1
11      1
110     1
111     1
112     1
113     1
114     1
115     1
116     1
117     1
118     1
119     1
12      1
120     1
121     1
122     1
123     1
124     1
125     1
126     1
127     1
128     1
129     1
13      1
130     1
131     1
132     1
133     1
134     1
135     1
136     1
137     1
138     1
139     1
14      1
140     1
141     1
142     1
143     1
```

```
cat Q4a_Reducer-output.txt | tail -n 50
```



```
[root@sandbox HW2Q4]# cat Q4a_Reducer-output.txt | tail -n 50
you     12013
youI    1
youR    1
youThat 1
youd    11
youfl   1
youhe   1
yould   1
youll   102
young   401
younger 30
youngest        23
youngeyd        1
younglings      1
youngly 2
youngs  1
youngst 1
younker 3
youoften        1
youpray 1
your    6002
youre   15
yours   247
yourself        268
yourselves      73
yoursnot        1
youst   1
youth   270
youthat 1
youthful        31
youths  9
youtli  1
youwell 1
youwondrous     1
youyou  1
zanies  1
zany    1
zeal    32
zealous 6
zeals   1
zed     1
zenith  1
zephyrs 1
zir     2
zo      1
zodiac  1
zodiacs 1
zone    1
zounds  2
zwaggerd        1
```

b) `vi WDmapper1.py`

```
#!/usr/bin/env python

'''
The script is used for mapping phase
'''
import sys
# regular expressions can increase the efficiency
# and avoid too many lines of code
import re

# input from STDIN: reading one line in one loop
for line in sys.stdin:
    print('%s\t%d' % ('#Line#', 1))
    # remove the front/end blanks of each line
    line = line.strip()
    # split words by whitespaces
    words = re.split('\s', line)
    # word processing in one loop
    for word in words:
        # the output values are tab-delimited
        # the word count is 1 since each for loop only reads one word
        # remove all charaters other than alphanumeric in a word
        # by doing this we may 'change' some words such as turning I'm into Im
        # but it won't influence the result of world count
        word = re.sub('\W+', '', word)
        if word:
            print('%s\t%s' % (word, 1))
```

`vim WDreducer1.py`

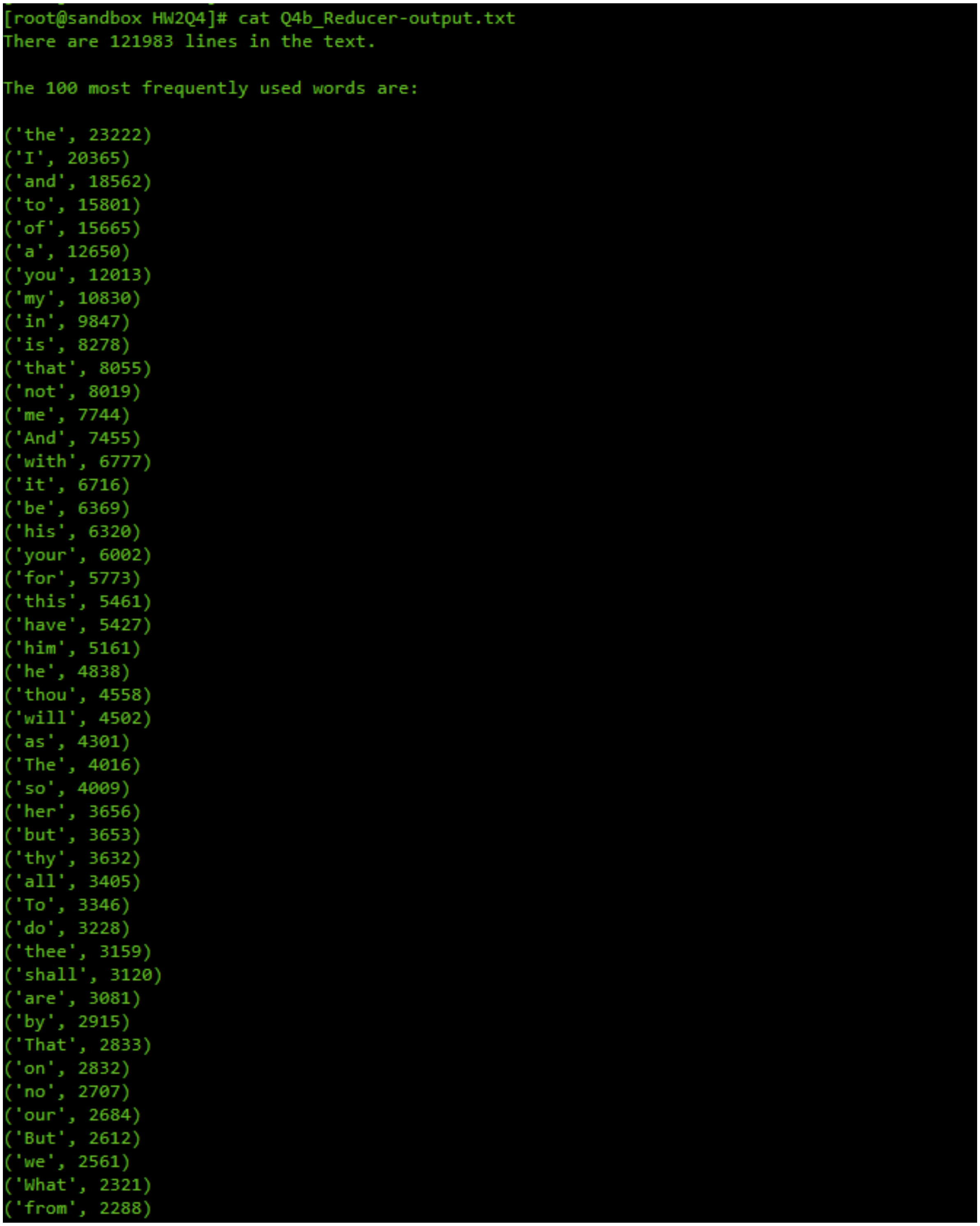
```
#!/usr/bin/env python

'''
The script is for reducing phase
'''
import sys

current_word = word = None
current_count = line_count = word_count = unique_word = 0
words = []


# input comes from STDIN
for line in sys.stdin:
    line = line.strip()
    # parse the input we got from WDmapper.py
    key, val = line.split('\t', 1)
    # if the input is a line, we count line; otherwise we count word
    if key == '#Line#':
        line_count += 1
        continue
    word, count = key, val

    # total word number add 1
    word_count += 1

    # convert count (a tring) to int
    try:
        count = int(count)
    except ValueError:
        # if count was not a number, silently ignore the line
        continue

    # Hadoop sorts map output by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        # store the (word, count) pair in a list
        if current_word:
            unique_word += 1
            words.append((current_word, current_count))
        current_count = count
        current_word = word

# output the total number of lines
print('There are ' + str(line_count) + ' lines in the text.\n')

# output the top 100 most frequently words
word_100 = sorted(words, key=lambda x: x[1], reverse=True)[:100]
print('The 100 most frequently used words are:\n')
for word, count in word_100:
    print((word, count))
print('\n')

# output the total word count
print('There are ' + str(word_count) + ' words in the text.\n')

# output the number of unique words
print('There are ' + str(unique_word) + ' unique words in the text')
```

```
chmod +x WDmapper1.py
chmod +x WDreducer1.py
```

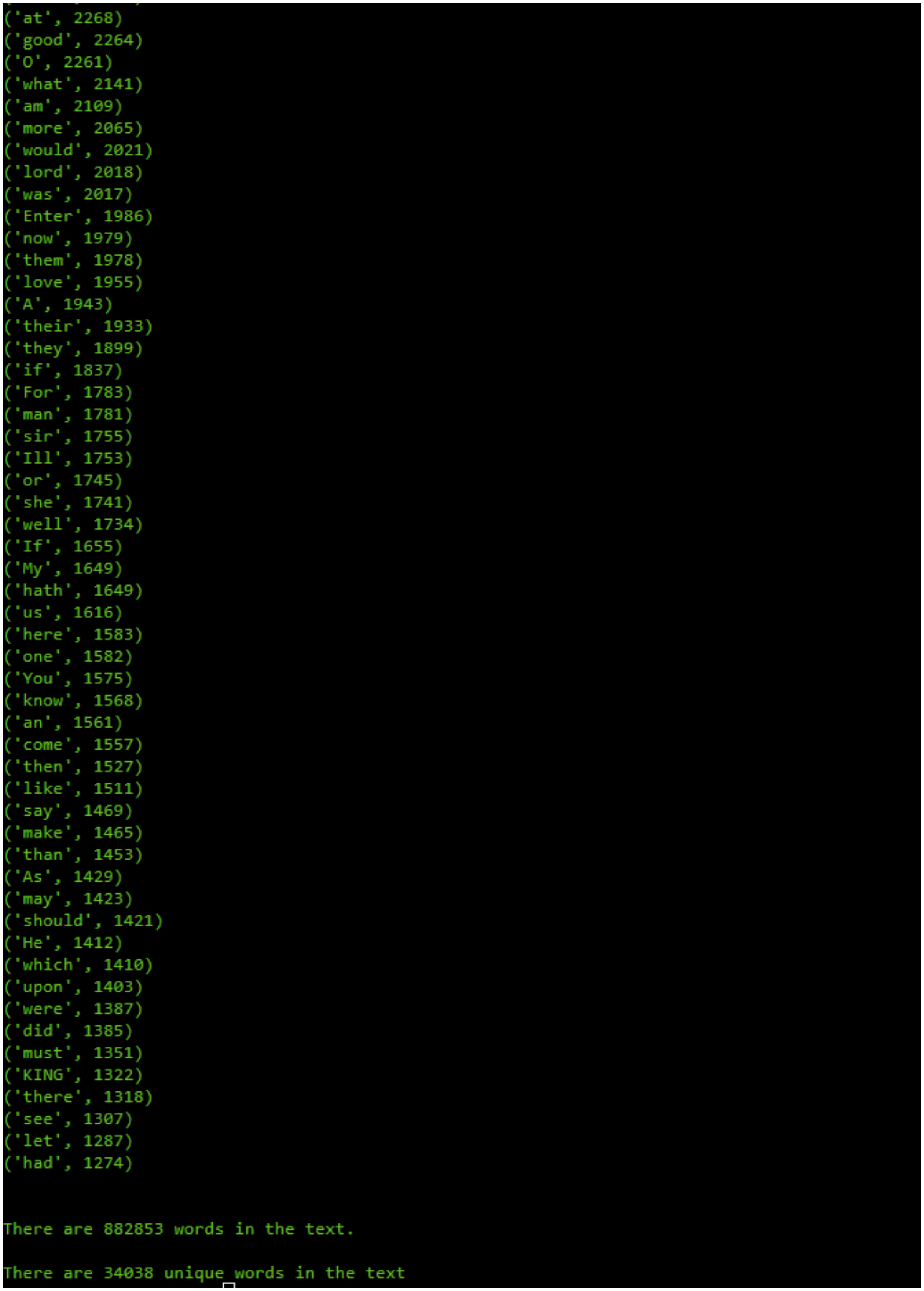
Run in HDFS:

```
hadoop jar /usr/hdp/2.6.1.0-129/hadoop-mapreduce/hadoop-streaming.jar\
-file WDmapper1.py -file WDreducer1.py\
-mapper WDmapper1.py -reducer WDreducer1.py\
-input /course-data/shakespeare.txt -output /course-data/Q4b_reducer-output/
```

```
scp -P 2222 Q4a_Reducer-output.txt root@127.0.0.1:/HW2Q4
cat Q4b_Reducer-output.txt
```



```
[root@sandbox HW2Q4]# cat Q4b_Reducer-output.txt
There are 121983 lines in the text.

The 100 most frequently used words are:

('the', 23222)
('I', 20365)
('and', 18562)
('to', 15801)
('of', 15665)
('a', 12650)
('you', 12013)
('my', 10830)
('in', 9847)
('is', 8278)
('that', 8055)
('not', 8019)
('me', 7744)
('And', 7455)
('with', 6777)
('it', 6716)
('be', 6369)
('his', 6320)
('your', 6002)
('for', 5773)
('this', 5461)
('have', 5427)
('him', 5161)
('he', 4838)
('thou', 4558)
('will', 4502)
('as', 4301)
('The', 4016)
('so', 4009)
('her', 3656)
('but', 3653)
('thy', 3632)
('all', 3405)
('To', 3346)
('do', 3228)
('thee', 3159)
('shall', 3120)
('are', 3081)
('by', 2915)
('That', 2833)
('on', 2832)
('no', 2707)
('our', 2684)
('But', 2612)
('we', 2561)
('What', 2321)
('from', 2288)
```



```
('at', 2268)
('good', 2264)
('O', 2261)
('what', 2141)
('am', 2109)
('more', 2065)
('would', 2021)
('lord', 2018)
('was', 2017)
('Enter', 1986)
('now', 1979)
('them', 1978)
('love', 1955)
('A', 1943)
('their', 1933)
('they', 1899)
('if', 1837)
('For', 1783)
('man', 1781)
('sir', 1755)
('Ill', 1753)
('or', 1745)
('she', 1741)
('well', 1734)
('If', 1655)
('My', 1649)
('hath', 1649)
('us', 1616)
('here', 1583)
('one', 1582)
('You', 1575)
('know', 1568)
('an', 1561)
('come', 1557)
('then', 1527)
('like', 1511)
('say', 1469)
('make', 1465)
('than', 1453)
('As', 1429)
('may', 1423)
('should', 1421)
('He', 1412)
('which', 1410)
('upon', 1403)
('were', 1387)
('did', 1385)
('must', 1351)
('KING', 1322)
('there', 1318)
('see', 1307)
('let', 1287)
('had', 1274)


There are 882853 words in the text.

There are 34038 unique words in the text
```

c) `vim WDmapper2.py`

```
#!/usr/bin/env python

'''
The script is used for mapping phase
To ignore the case difference,
we simply just convert each word into all-lowercase in the map phase
'''
import sys
# regular expressions can increase the efficiency
# and avoid too many lines of code
import re

# input from STDIN: reading one line in one loop
for line in sys.stdin:
    print('%s\t%d' % ('#Line#', 1))
    # remove the front/end blanks of each line
    line = line.strip()
    # split words by whitespaces
    words = re.split('\s', line)
    # word processing in one loop
    for word in words:
        # the output values are tab-delimited
        # the word count is 1 since each for loop only reads one word
        # remove all charaters other than alphanumeric in a word
        # by doing this we may 'change' some words such as turning I'm into Im
        # but it won't influence the result of world count
        # lower() will change all characters to lowercase
        word = re.sub('\W+', '', word).lower()
        if word:
            print('%s\t%s' % (word, 1))
```

`vim WDreducer2.py`

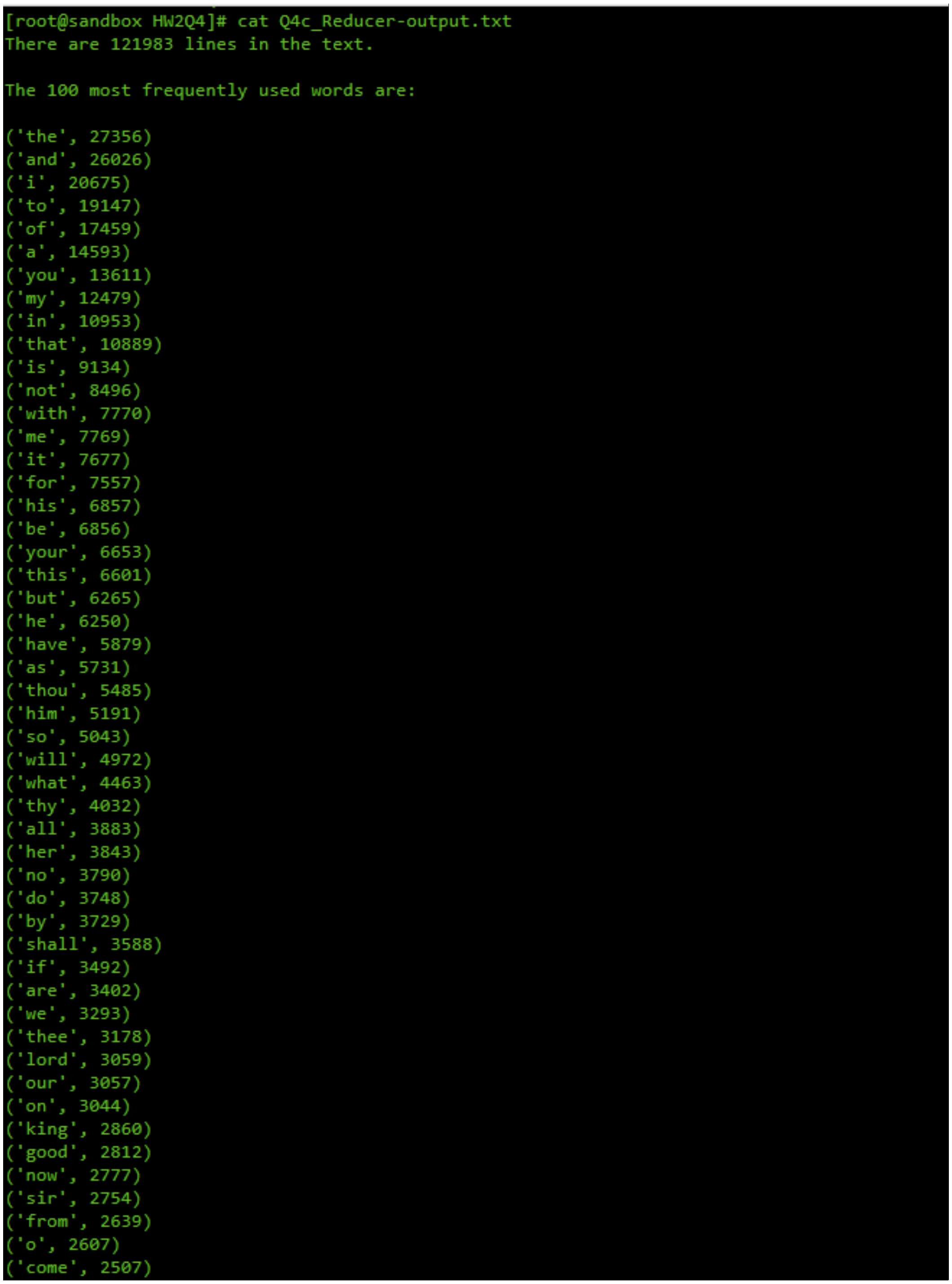
```
#!/usr/bin/env python

'''
The script is for reducing phase
Same script as the previous one
'''
import sys

current_word = word = None
current_count = line_count = word_count = unique_word = 0
words = []
```

```
# input comes from STDIN
for line in sys.stdin:
    line = line.strip()

    # parse the input we got from WDmapper.py
    key, val = line.split('\t', 1)
    # if the input is a line, we count line; otherwise we count word
    if key == '#Line#':
        line_count += 1
        continue
    word, count = key, val

    # total word number add 1
    word_count += 1

    # convert count (a tring) to int
    try:
        count = int(count)
    except ValueError:
        # if count was not a number, silently ignore the line
        continue

    # Hadoop sorts map output by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        # store the (word, count) pair in a list
        if current_word:
            unique_word += 1
            words.append((current_word, current_count))
        current_count = count
        current_word = word

# output the total number of lines
print('There are ' + str(line_count) + ' lines in the text.\n')

# output the top 100 most frequently words
word_100 = sorted(words, key=lambda x: x[1], reverse=True)[:100]
print('The 100 most frequently used words are:\n')
for word, count in word_100:
    print((word, count))
print('\n')

# output the total word count
print('There are ' + str(word_count) + ' words in the text.\n')

# output the number of unique words
```

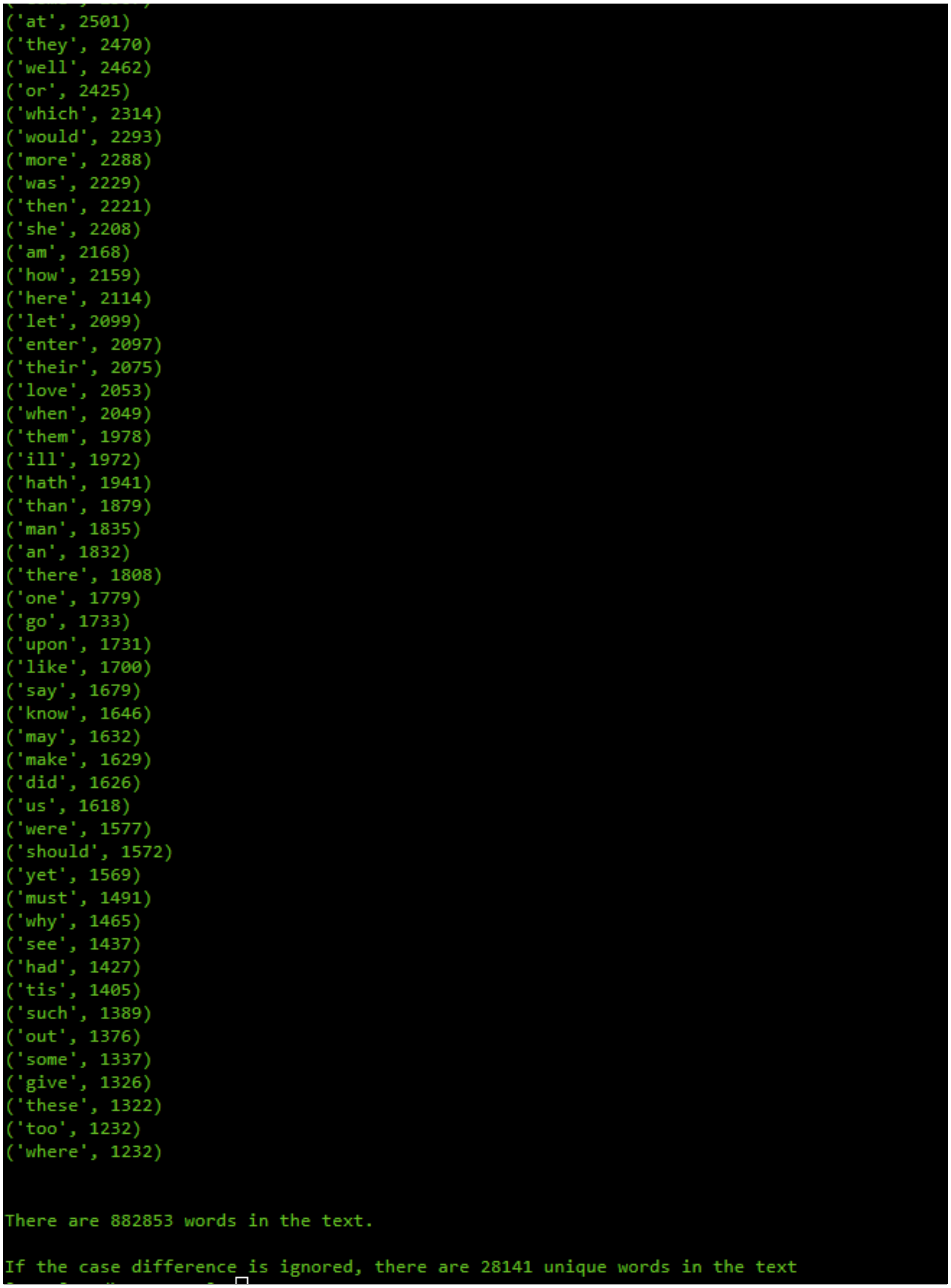
```
print('If the case difference is ignored, there are '
      + str(unique_word) + ' unique words in the text')
```

```
chmod +x WDmapper2.py
chmod +x WDreducer2.py
hadoop jar /usr/hdp/2.6.1.0-129/hadoop-mapreduce/hadoop-streaming.jar\
-file WDmapper2.py -file WDreducer2.py\
-mapper WDmapper2.py -reducer WDreducer2.py\
-input /course-data/shakespeare.txt -output /course-data/Q4c_reducer-output/
```

```
scp -P 2222 Q4c_Reducer-output.txt root@127.0.0.1:/HW2Q4
cat Q4c_Reducer-output.txt
```



```
[root@sandbox HW2Q4]# cat Q4c_Reducer-output.txt
There are 121983 lines in the text.

The 100 most frequently used words are:

('the', 27356)
('and', 26026)
('i', 20675)
('to', 19147)
('of', 17459)
('a', 14593)
('you', 13611)
('my', 12479)
('in', 10953)
('that', 10889)
('is', 9134)
('not', 8496)
('with', 7770)
('me', 7769)
('it', 7677)
('for', 7557)
('his', 6857)
('be', 6856)
('your', 6653)
('this', 6601)
('but', 6265)
('he', 6250)
('have', 5879)
('as', 5731)
('thou', 5485)
('him', 5191)
('so', 5043)
('will', 4972)
('what', 4463)
('thy', 4032)
('all', 3883)
('her', 3843)
('no', 3790)
('do', 3748)
('by', 3729)
('shall', 3588)
('if', 3492)
('are', 3402)
('we', 3293)
('thee', 3178)
('lord', 3059)
('our', 3057)
('on', 3044)
('king', 2860)
('good', 2812)
('now', 2777)
('sir', 2754)
('from', 2639)
('o', 2607)
('come', 2507)
```



```
('at', 2501)
('they', 2470)
('well', 2462)
('or', 2425)
('which', 2314)
('would', 2293)
('more', 2288)
('was', 2229)
('then', 2221)
('she', 2208)
('am', 2168)
('how', 2159)
('here', 2114)
('let', 2099)
('enter', 2097)
('their', 2075)
('love', 2053)
('when', 2049)
('them', 1978)
('ill', 1972)
('hath', 1941)
('than', 1879)
('man', 1835)
('an', 1832)
('there', 1808)
('one', 1779)
('go', 1733)
('upon', 1731)
('like', 1700)
('say', 1679)
('know', 1646)
('may', 1632)
('make', 1629)
('did', 1626)
('us', 1618)
('were', 1577)
('should', 1572)
('yet', 1569)
('must', 1491)
('why', 1465)
('see', 1437)
('had', 1427)
('tis', 1405)
('such', 1389)
('out', 1376)
('some', 1337)
('give', 1326)
('these', 1322)
('too', 1232)
('where', 1232)


There are 882853 words in the text.

If the case difference is ignored, there are 28141 unique words in the text
```