STSCI 5065 - Spring 2019 Homework 2

(Solutions)

Question 1 (total 18 points)

Let's assume that there exists a file of size 5 GB. For all below sub parts, assume that transfer rate is 50 MB/s and seek time is 3 milliseconds. Remember from class, seek time is the time to find the start position of Each block to read the data from. For simplicity assume that 1GB = 1000 MB, 1 MB = 1000 KB and so on and the complete file is stored on just one system.

a) If the file is stored on a simple file system with a block size of 100 Kilobytes, how much time does it take to read the complete file? (5 points)

File Size in KB = 5000 MB * 1000 KB/MB= 5,000,000 KB
Total Number of Blocks = 5,000,000 KB / 100 KB/block= 50,000 blocks
Total Seek Time = 50,000 blocks* 3 ms/block = 150 seconds
Total time to read all the blocks = 5000 MB / 50 MB/s = 100 seconds
Total Time = Seek Time + Transfer Time = 150 seconds + 100 seconds = 250 seconds

b) If the file is stored on a Hadoop file system with a block size of 100 MB, how much time does it take to read the complete file? (5 points)

File Size in MB = 5000 MB
Total Number of Blocks = 5000 MB / 100 MB/block = 50 blocks
Total Seek Time = 50 blocks * 3 ms/block = 0.15 seconds
Total time to read all the blocks = 5000 MB / 50 MB/s = 100 seconds
Total Time = Seek Time + Transfer Time = 0.15 seconds + 100 seconds = 100.15 seconds

c) What is the percentage gain of time in the second case as compared to the first case? (5 points)

Time Gain = 250 – 100.15 = 149.85 seconds
Percentage = (149.85/250) * 100 = 59.94 %

d) Give reasons and situations where block size in question 1a is better than block size in 1b, and vice versa. (3 points)

We can clearly see that whenever we want to read a file block, a decent amount of time is spent in seeking the block itself. Hence, a larger Block size is better, where we want to read the entire large file and perform some processing on the entire data, thus minimizing the time to seek the blocks. Hadoop exploits this concept. So, in this case 1b is better than 1a.

However, in the situations where we just want to read some small number of parts of the whole file, a small block size is better. In situations like these, the number of seeks is much less since you are not reading all the blocks and you only need to transfer the blocks that you need. Example of such cases would be your normal filesystem, such as the Linux filesystem. So, in this case 1a is better than 1b.

Question 2 (total 12 points)

In this question we will be working with command line interface on the Horton Sandbox and use commands for working in Hadoop ecosystem. Remember the username to login is "root" and password is "hadoop". For each of the below question, write the command you executed to perform the task.

a) **(1 point) Create an HDFS directory called "course-data" at the root of the hadoop filesystem.**

```
# hadoop fs -mkdir /course-data
```

b) **(1 point) Create a normal directory "data-set" at the root of the file system.**

```
# mkdir /data-set
```

c) **(5 points) Download the file from the below location in the directory data-set and unzip it.**

http://data.gdeltproject.org/events/1990.zip
http://data.gdeltproject.org/events/1991.zip

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

d) **(1 point) After extraction of the above file, copy the above file in Hadoop Filesystem.** The below commands are executed from directory /data-set. The two ways shown below can be used interchangeably.

```
# hadoop fs -copyFromLocal 1990.csv /course-data
```

OR

```
# hadoop fs -put 1991.csv /course-data
```

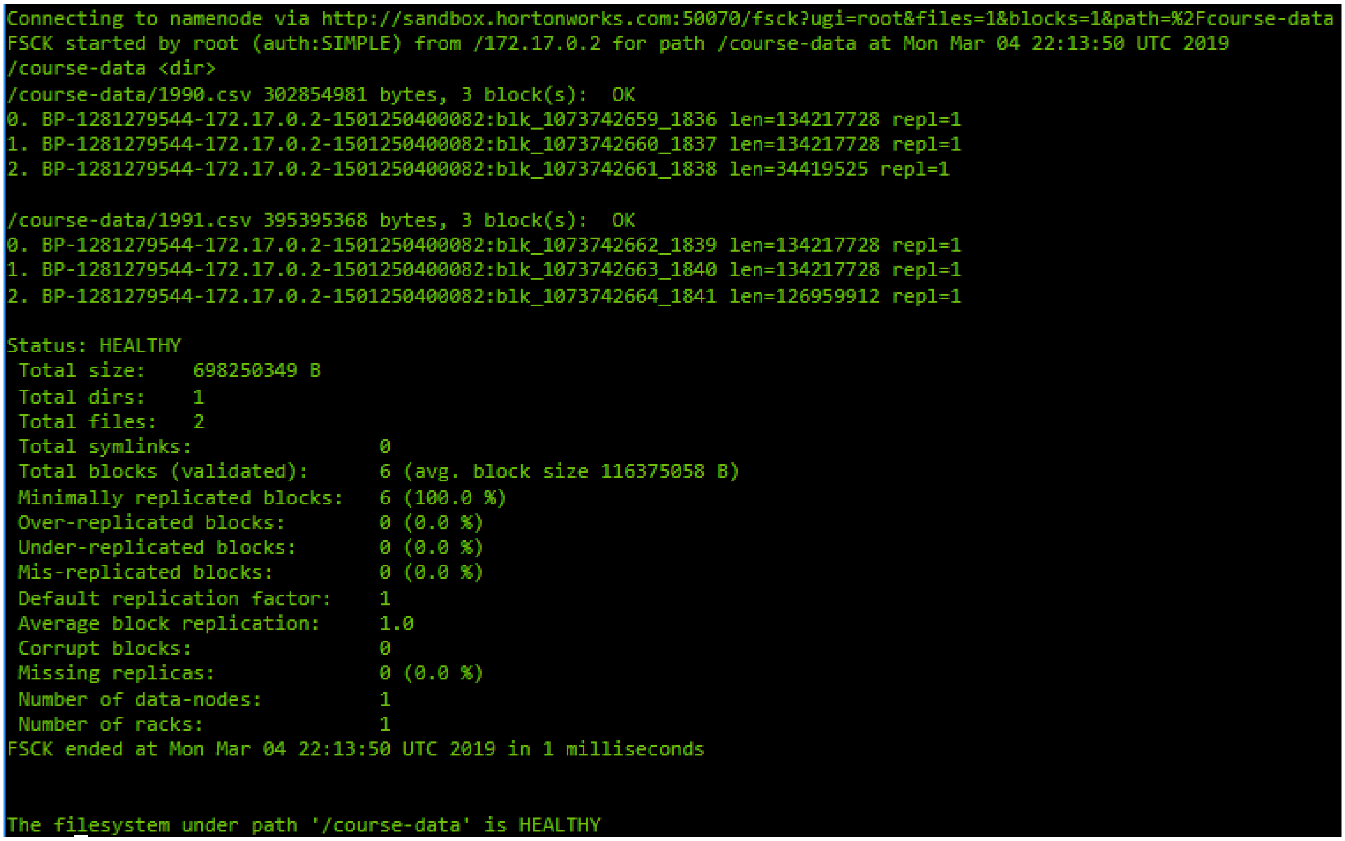
e) **(1 point) Write the command to validate if the file actually loaded in the system.**

```
# hadoop fs -ls /course-data
```

f) (3 points) Write the command to examine the file storage statistics of your Hadoop filesystem for the directory course-data. How many block(s) is/are allocated for it? Attach a screenshot of the command's output

```
# hadoop fsck /course-data –files –blocks | less
```

No of blocks = 6. This may vary in your case. As long as your answer matches with the screenshot its fine. The screenshot below explains all the data.

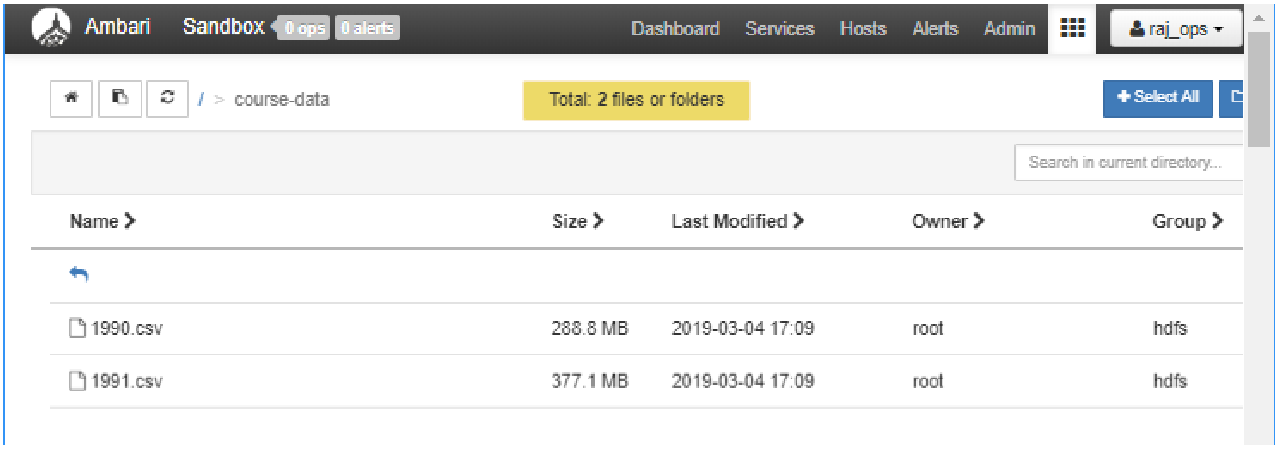


```
Connecting to namenode via http://sandbox.hortonworks.com:50070/fsck?ugi=root&files=1&blocks=1&path=%2Fcourse-data
FSCK started by root (auth:SIMPLE) from /172.17.0.2 for path /course-data at Mon Mar 04 22:13:50 UTC 2019
/course-data <dir>
/course-data/1990.csv 302854981 bytes, 3 block(s):  OK
0. BP-1281279544-172.17.0.2-1501250400082:blk_1073742659_1836 len=134217728 repl=1
1. BP-1281279544-172.17.0.2-1501250400082:blk_1073742660_1837 len=134217728 repl=1
2. BP-1281279544-172.17.0.2-1501250400082:blk_1073742661_1838 len=34419525 repl=1

/course-data/1991.csv 395395368 bytes, 3 block(s):  OK
0. BP-1281279544-172.17.0.2-1501250400082:blk_1073742662_1839 len=134217728 repl=1
1. BP-1281279544-172.17.0.2-1501250400082:blk_1073742663_1840 len=134217728 repl=1
2. BP-1281279544-172.17.0.2-1501250400082:blk_1073742664_1841 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 Mon Mar 04 22:13:50 UTC 2019 in 1 milliseconds


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

Question 3 (total 4 points)

Remember in Question 2, we copied two files in Hadoop file system. For the below questions, you will use file browser, via Apache Ambari, to view the files and perform operation.

a) **Attach the screenshot of the file browser showing the two files copied in Hadoop filesystem. The screenshot should show the file size and permissions as well. (2 points)**



Name	Size	Last Modified	Owner	Group
1990.csv	288.8 MB	2019-03-04 17:09	root	hdfs
1991.csv	377.1 MB	2019-03-04 17:09	root	hdfs

b) **Delete the file “1990.csv” using command line and not via file browser. Attach the screenshot of the file browser after the operation. Please write down the Hadoop command that performs the same operation. (2 points)**

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

Question 4 (total 66 points)

Q4a. (30 points)

Linux commands used to create HW2 under the system root and to create WDmapper1.py:

```
mkdir /HW2

vi /HW2/WDmapper.py
```

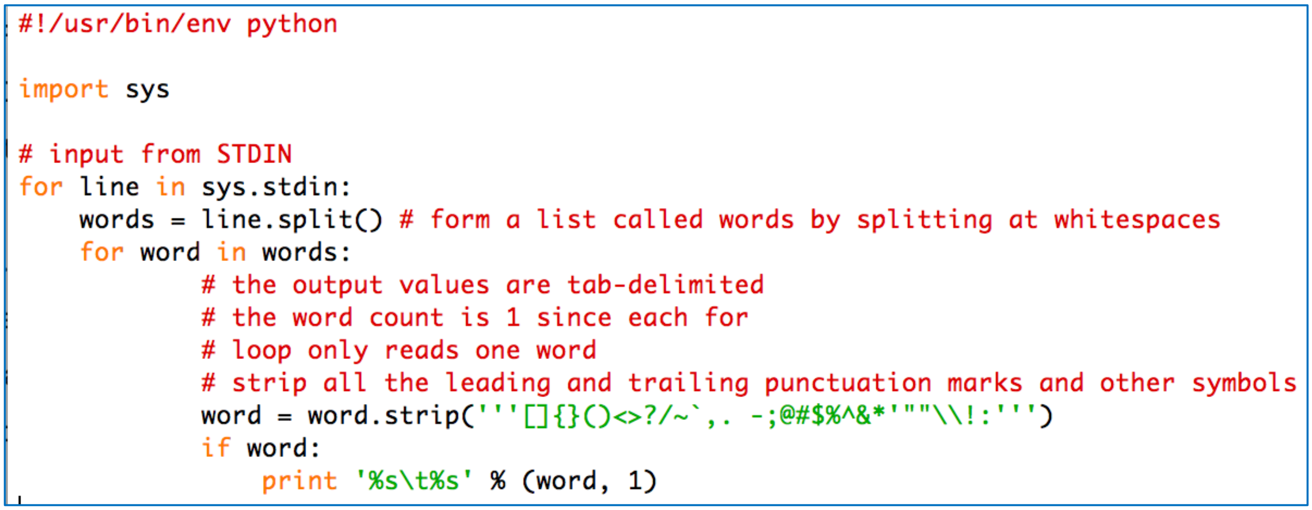
(the commands to create WDmapper1.py, WDmapper2.py, WDreducer.py, WDreducer1.py and WDreducer2.py are the same to this command, except the file names)

It is a good idea to test the WDmapper.py and WD reducer.py files with Linux shell scripting before running MapReduce in HDFS but it is not a requirement in this assignment.

Use the following command to run the MapReduce procedure (assuming you are currently in the /HW2 directory):

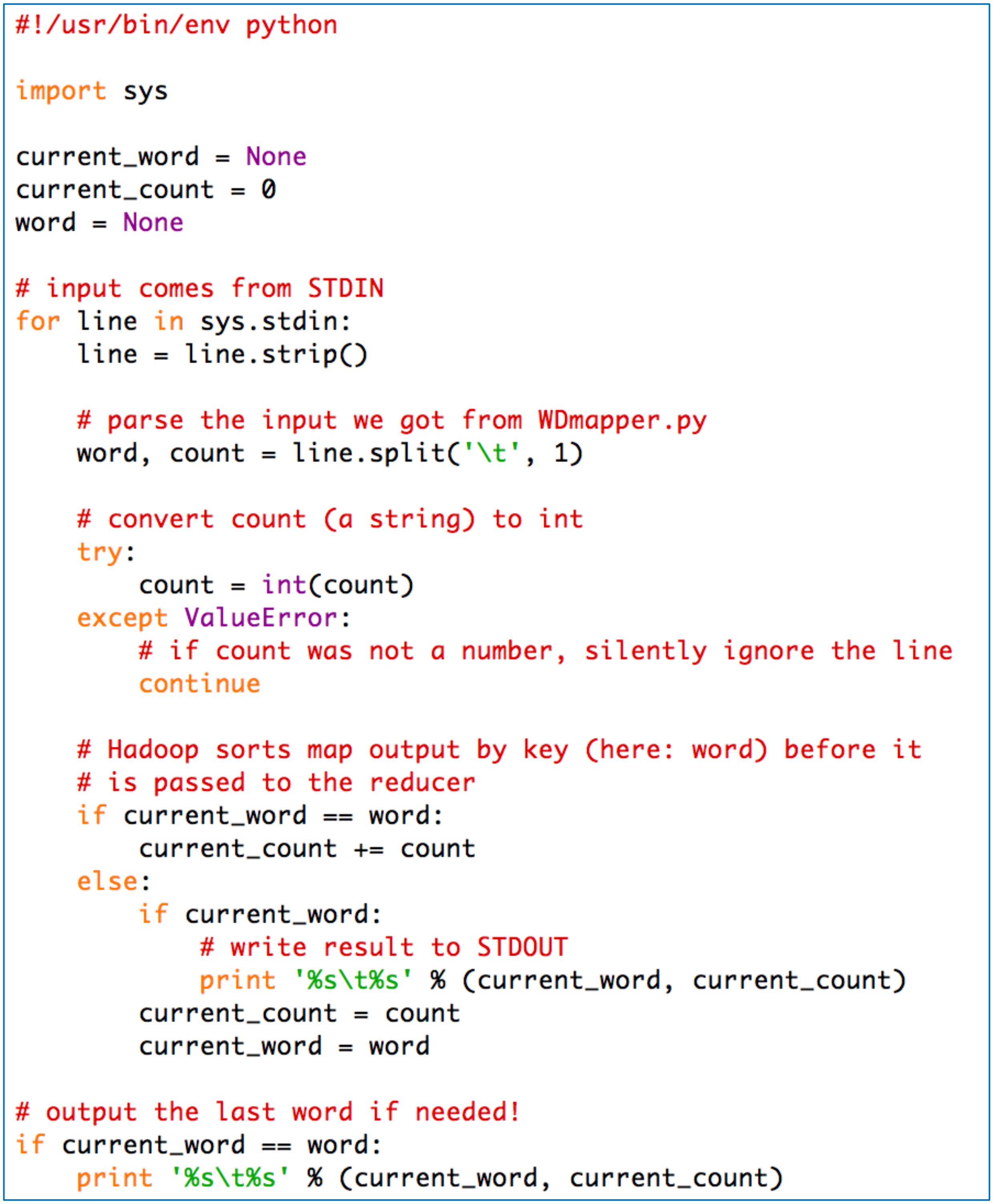
```
hadoop jar /usr/hdp/2.5.0.0-1245/hadoop-mapreduce/hadoop-streaming.jar \

-file WDmapper.py \

-file WDreducer.py \

-mapper WDmapper.py \

-reducer WDreducer.py \

-input /stsci5065/data/shakespeare.txt \

-output /stsci5065/HW2_5a
```

The content of WDmapper.py



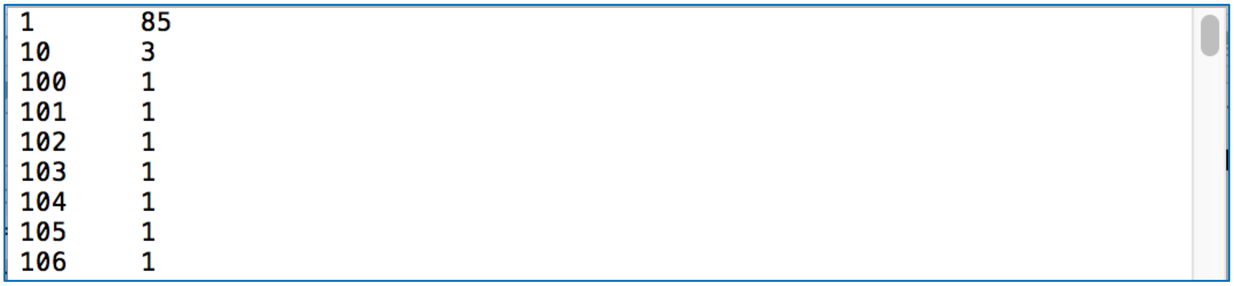
```
#!/usr/bin/env python

import sys

# input from STDIN
for line in sys.stdin:
    words = line.split() # form a list called words by splitting at whitespaces
    for word in words:
            # the output values are tab-delimited
            # the word count is 1 since each for
            # loop only reads one word
            # strip all the leading and trailing punctuation marks and other symbols
            word = word.strip('''[]{}()<>?/~`,. -;@#$%^&*'""\\!:''')
            if word:
                print '%s\t%s' % (word, 1)
```

The content of WDreducer.py



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

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 string) 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)
```

The contents of Q4a_Reducer-output.txt are shown with the following screenshots (the beginning portion and the end portion (it is also OK if the whole file is pasted in).

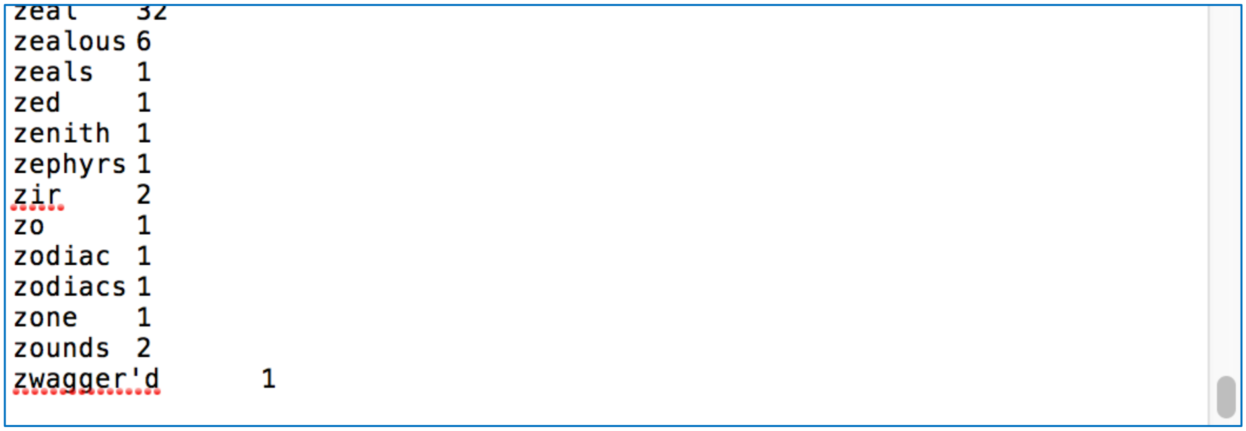


```
1       85
10      3
100     1
101     1
102     1
103     1
104     1
105     1
106     1
```

.

.

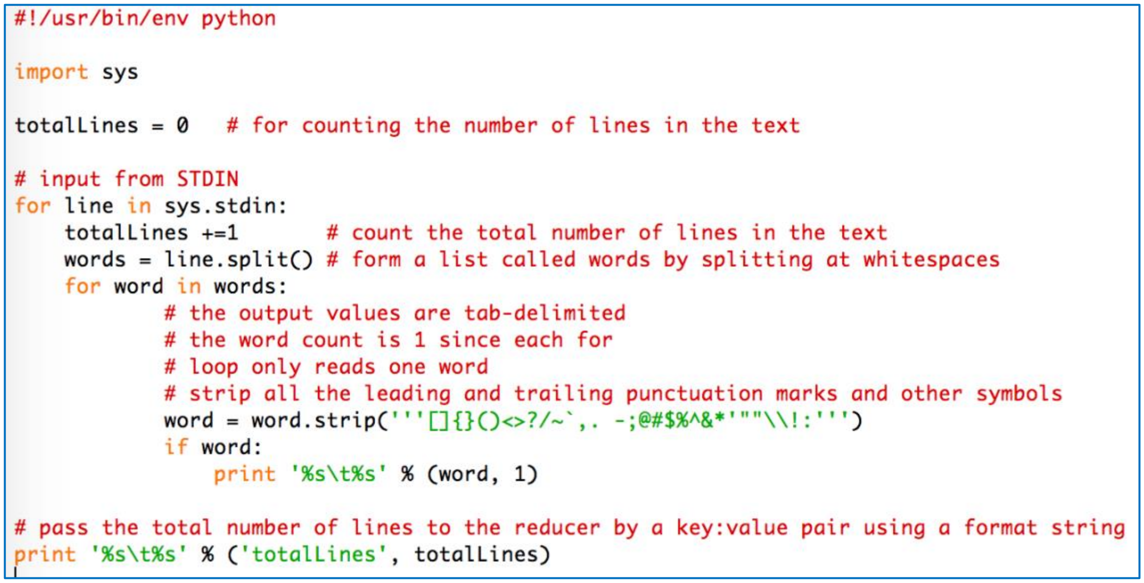
.



```
zeal    32
zealous 6
zeals   1
zed     1
zenith  1
zephyrs 1
zir     2
zo      1
zodiac  1
zodiacs 1
zone    1
zounds  2
zwagger'd       1
```

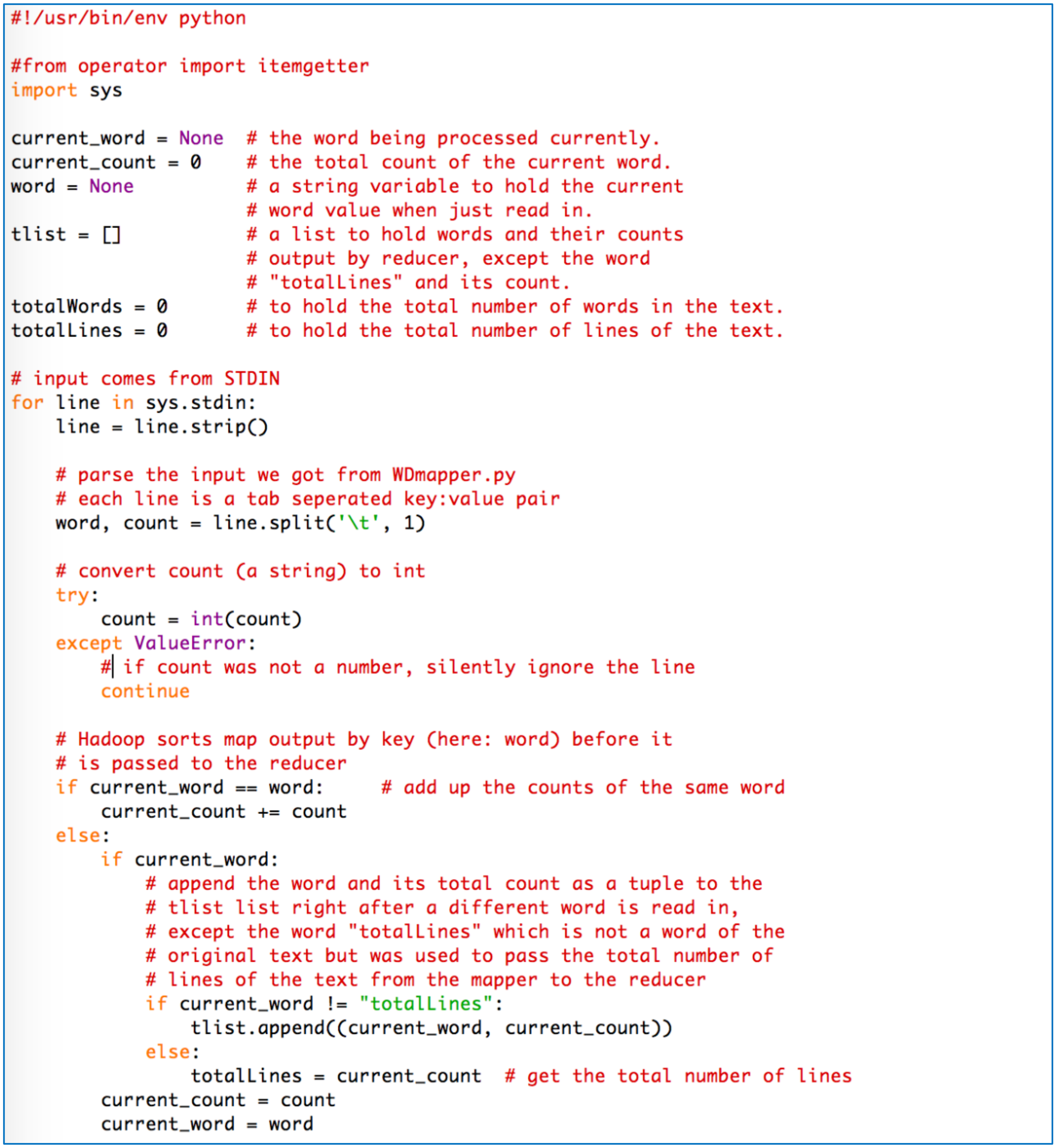
Q4b. (26 points)

The content of WDmapper1.py



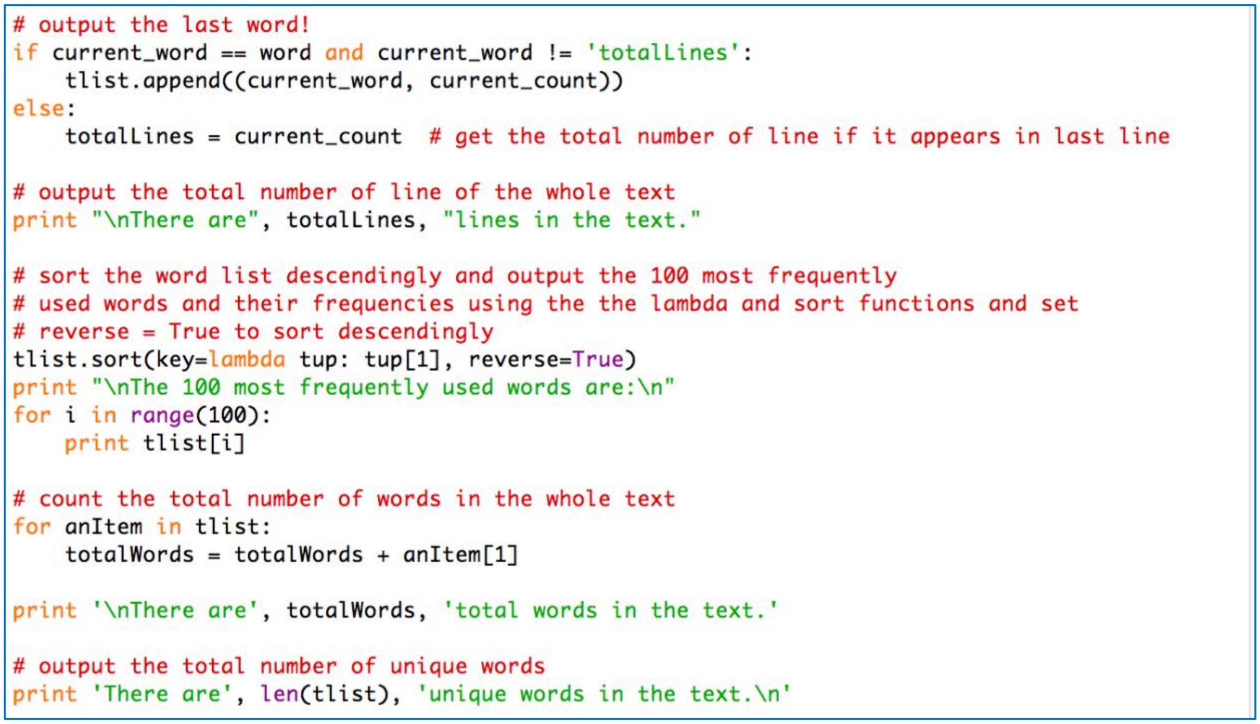
```python
#!/usr/bin/env python

import sys

totalLines = 0   # for counting the number of lines in the text

# input from STDIN
for line in sys.stdin:
    totalLines +=1       # count the total number of lines in the text
    words = line.split() # form a list called words by splitting at whitespaces
    for word in words:
            # the output values are tab-delimited
            # the word count is 1 since each for
            # loop only reads one word
            # strip all the leading and trailing punctuation marks and other symbols
            word = word.strip('''[]{}()<>?/~`,. -;@#$%^&*'""\\!:''')
            if word:
                print '%s\t%s' % (word, 1)

# pass the total number of lines to the reducer by a key:value pair using a format string
print '%s\t%s' % ('totalLines', totalLines)
```

The content of WDreducer1.py



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

#from operator import itemgetter
import sys

current_word = None  # the word being processed currently.
current_count = 0    # the total count of the current word.
word = None          # a string variable to hold the current
                     # word value when just read in.
tlist = []           # a list to hold words and their counts
                     # output by reducer, except the word
                     # "totalLines" and its count.
totalWords = 0       # to hold the total number of words in the text.
totalLines = 0       # to hold the total number of lines of the text.

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

    # parse the input we got from WDmapper.py
    # each line is a tab seperated key:value pair
    word, count = line.split('\t', 1)

    # convert count (a string) 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:     # add up the counts of the same word
        current_count += count
    else:
        if current_word:
            # append the word and its total count as a tuple to the
            # tlist list right after a different word is read in,
            # except the word "totalLines" which is not a word of the
            # original text but was used to pass the total number of
            # lines of the text from the mapper to the reducer
            if current_word != "totalLines":
                tlist.append((current_word, current_count))
            else:
                totalLines = current_count  # get the total number of lines
        current_count = count
        current_word = word
```

(continued)

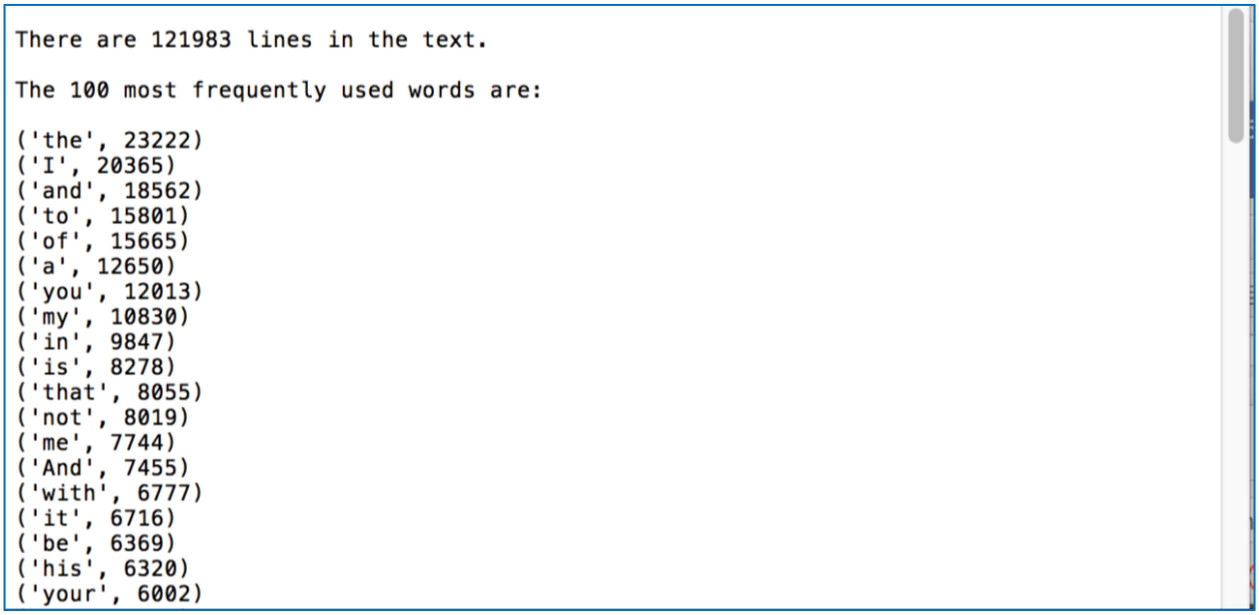


```
# output the last word!
if current_word == word and current_word != 'totalLines':
    tlist.append((current_word, current_count))
else:
    totalLines = current_count  # get the total number of line if it appears in last line

# output the total number of line of the whole text
print "\nThere are", totalLines, "lines in the text."

# sort the word list descendingly and output the 100 most frequently
# used words and their frequencies using the the lambda and sort functions and set
# reverse = True to sort descendingly
tlist.sort(key=lambda tup: tup[1], reverse=True)
print "\nThe 100 most frequently used words are:\n"
for i in range(100):
    print tlist[i]

# count the total number of words in the whole text
for anItem in tlist:
    totalWords = totalWords + anItem[1]

print '\nThere are', totalWords, 'total words in the text.'

# output the total number of unique words
print 'There are', len(tlist), 'unique words in the text.\n'
```

Use the following command to run the MapReduce procedure (assuming you are currently in the /HW2 directory):

```
hadoop jar /usr/hdp/2.5.0.0-1245/hadoop-mapreduce/hadoop-streaming.jar \

-file WDmapper1.py \

-file WDreducer1.py \

-mapper WDmapper1.py \

-reducer WDreducer1.py \

-input /stsci5065/data/shakespeare.txt \

-output /stsci5065/HW2_5b
```

The contents of Q4b_Reducer-output.txt are shown with the following screenshots (the beginning portion and the end portion (it is also OK if the whole file is pasted in).

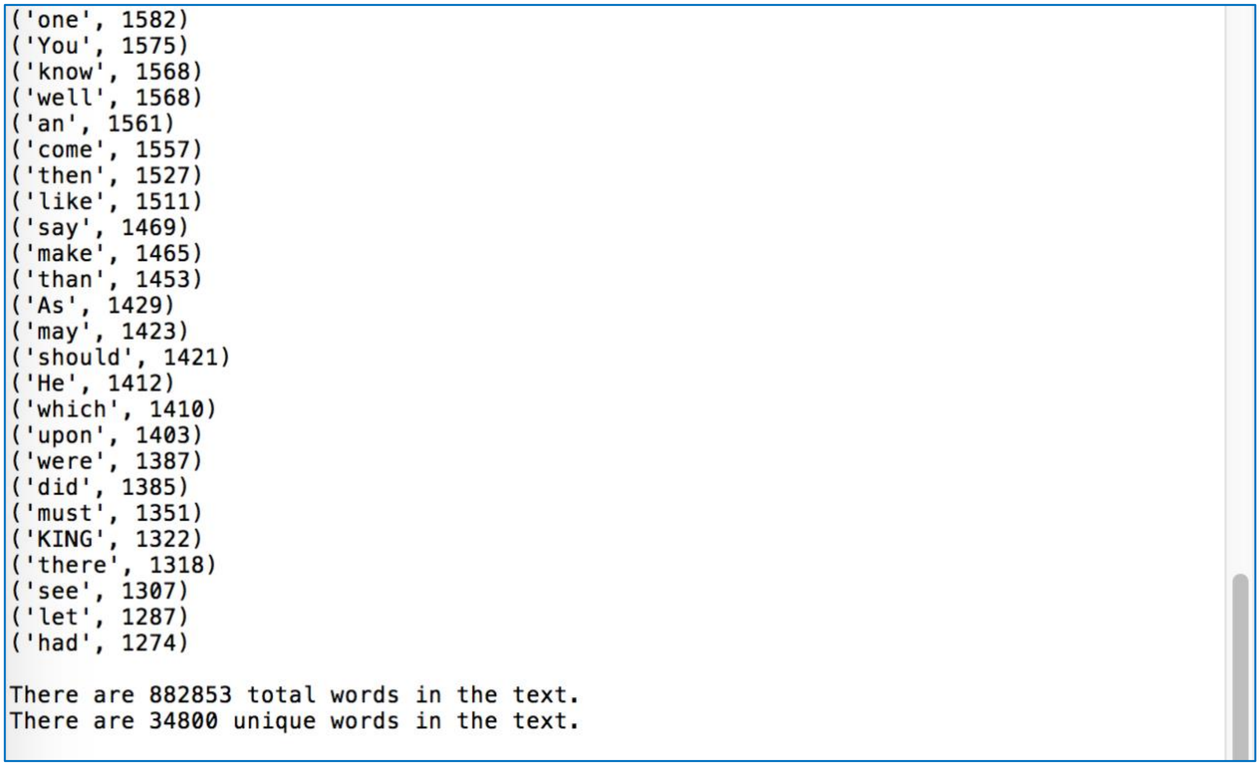


```
There are 121983 lines in the text.

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)
```

.

.

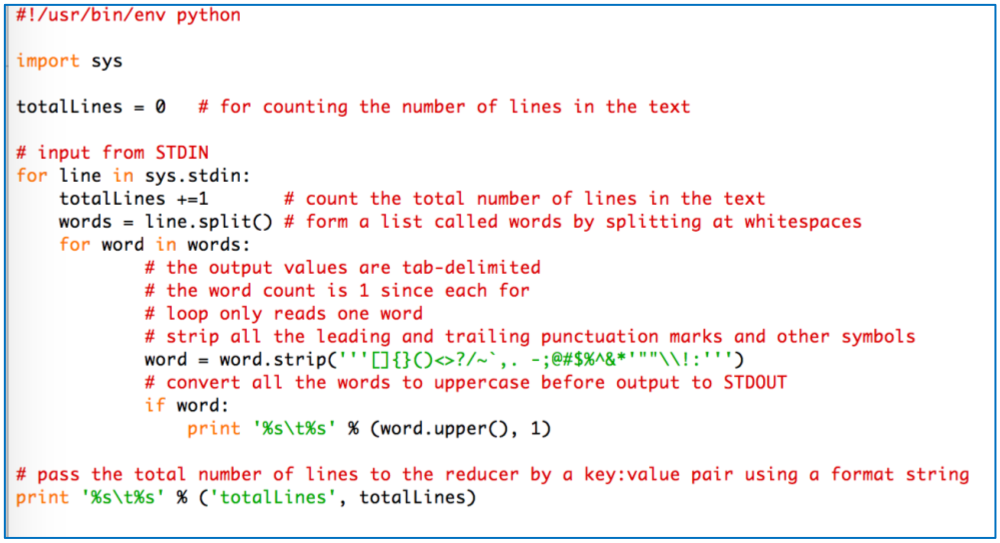
.



```
('one', 1582)
('You', 1575)
('know', 1568)
('well', 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 total words in the text.
There are 34800 unique words in the text.
```

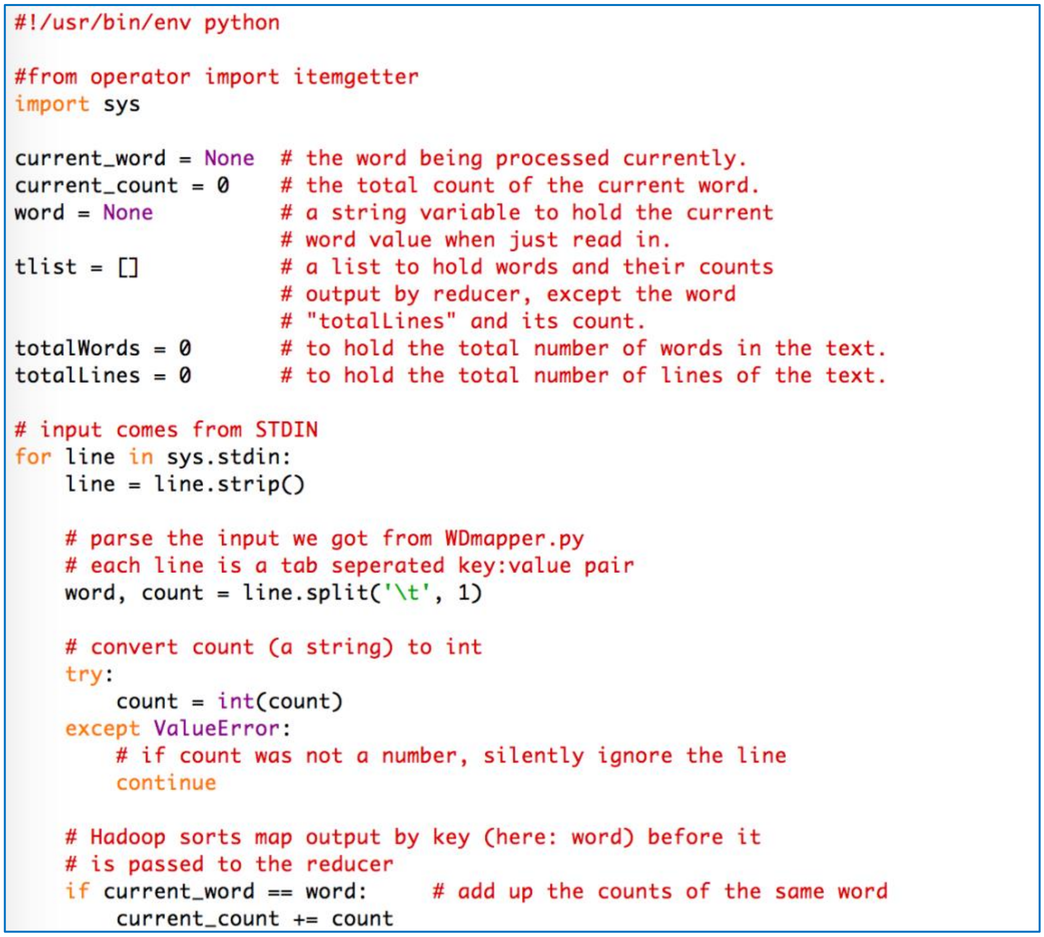
Q4c. (10 points)

The content of WDmapper2.py



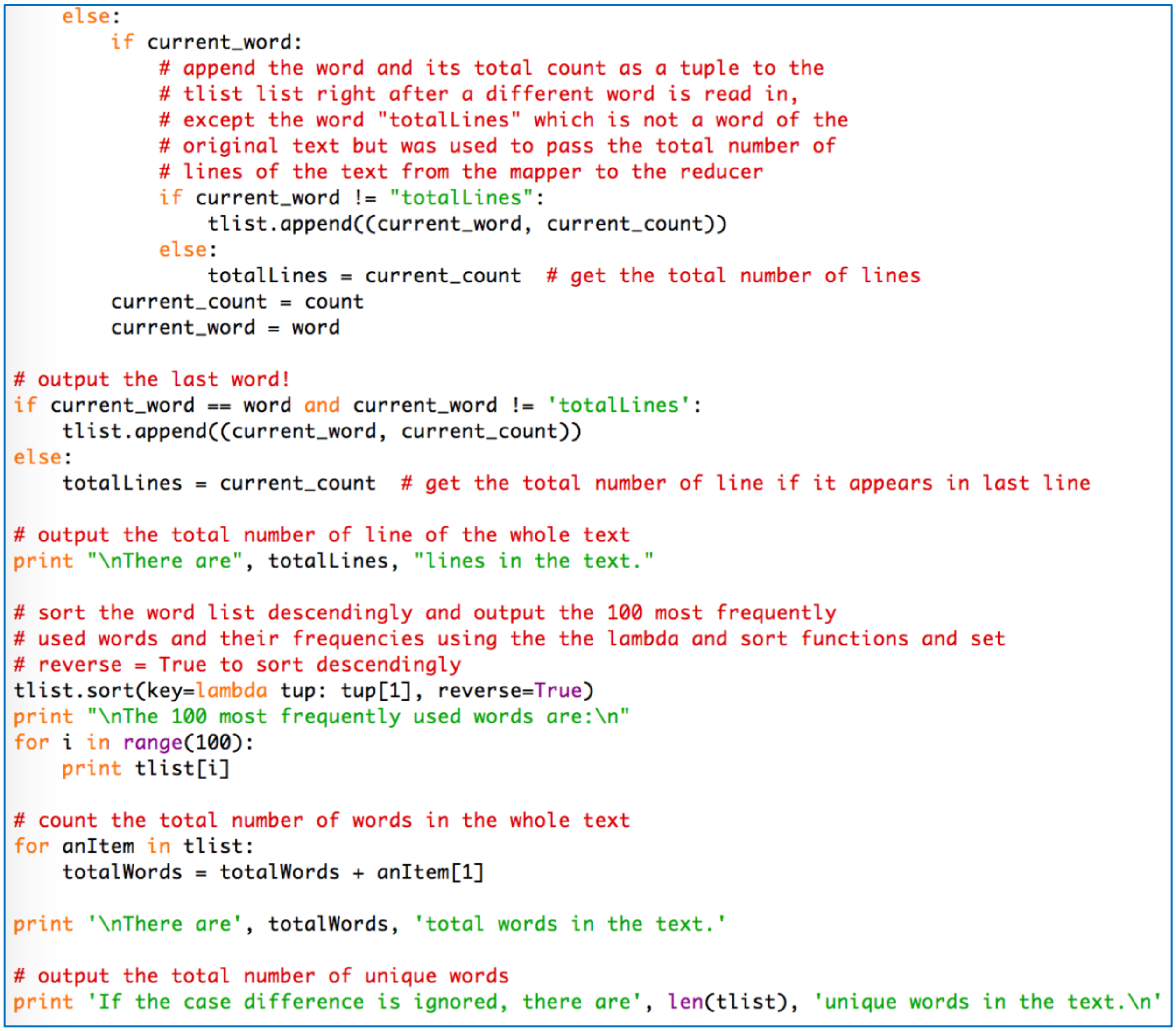
```
#!/usr/bin/env python

import sys

totalLines = 0   # for counting the number of lines in the text

# input from STDIN
for line in sys.stdin:
    totalLines +=1       # count the total number of lines in the text
    words = line.split() # form a list called words by splitting at whitespaces
    for word in words:
            # the output values are tab-delimited
            # the word count is 1 since each for
            # loop only reads one word
            # strip all the leading and trailing punctuation marks and other symbols
            word = word.strip('''[]{}()<>?/~`,. -;@#$%^&*'""\\!:''')
            # convert all the words to uppercase before output to STDOUT
            if word:
                print '%s\t%s' % (word.upper(), 1)

# pass the total number of lines to the reducer by a key:value pair using a format string
print '%s\t%s' % ('totalLines', totalLines)
```

The content of WDreducer2.py



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

#from operator import itemgetter
import sys

current_word = None  # the word being processed currently.
current_count = 0    # the total count of the current word.
word = None          # a string variable to hold the current
                     # word value when just read in.
tlist = []           # a list to hold words and their counts
                     # output by reducer, except the word
                     # "totalLines" and its count.
totalWords = 0       # to hold the total number of words in the text.
totalLines = 0       # to hold the total number of lines of the text.

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

    # parse the input we got from WDmapper.py
    # each line is a tab seperated key:value pair
    word, count = line.split('\t', 1)

    # convert count (a string) 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:     # add up the counts of the same word
        current_count += count
```

(continued)

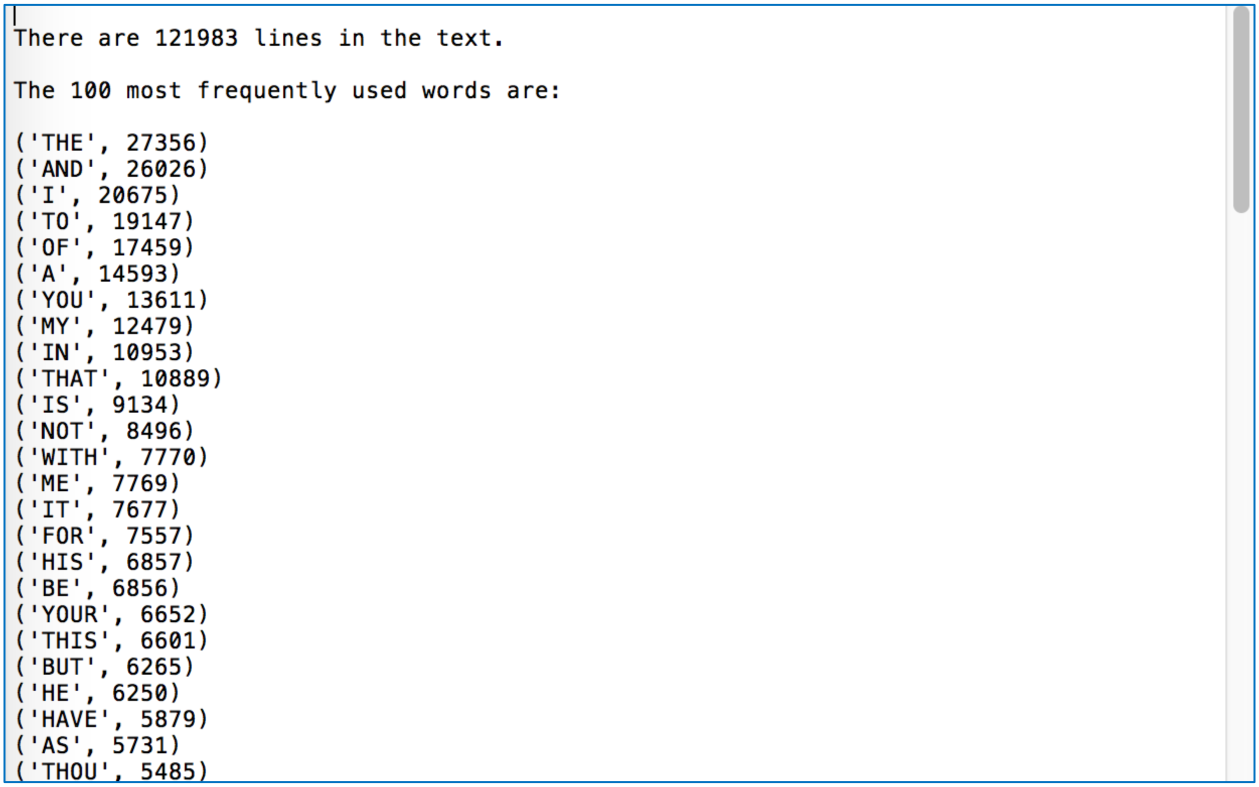


```
    else:
        if current_word:
            # append the word and its total count as a tuple to the
            # tlist list right after a different word is read in,
            # except the word "totalLines" which is not a word of the
            # original text but was used to pass the total number of
            # lines of the text from the mapper to the reducer
            if current_word != "totalLines":
                tlist.append((current_word, current_count))
            else:
                totalLines = current_count  # get the total number of lines
        current_count = count
        current_word = word

# output the last word!
if current_word == word and current_word != 'totalLines':
    tlist.append((current_word, current_count))
else:
    totalLines = current_count  # get the total number of line if it appears in last line

# output the total number of line of the whole text
print "\nThere are", totalLines, "lines in the text."

# sort the word list descendingly and output the 100 most frequently
# used words and their frequencies using the the lambda and sort functions and set
# reverse = True to sort descendingly
tlist.sort(key=lambda tup: tup[1], reverse=True)
print "\nThe 100 most frequently used words are:\n"
for i in range(100):
    print tlist[i]

# count the total number of words in the whole text
for anItem in tlist:
    totalWords = totalWords + anItem[1]

print '\nThere are', totalWords, 'total words in the text.'

# output the total number of unique words
print 'If the case difference is ignored, there are', len(tlist), 'unique words in the text.\n'
```

Use the following command to run the MapReduce procedure (assuming you are currently in the /HW2 directory):

```
hadoop jar /usr/hdp/2.5.0.0-1245/hadoop-mapreduce/hadoop-streaming.jar \

-file WDmapper2.py \

-file WDreducer2.py \

-mapper WDmapper2.py \

-reducer WDreducer2.py \

-input /stsci5065/data/shakespeare.txt \

-output /stsci5065/HW2_5c
```

The contents of Q4c_Reducer-output.txt are shown with the following screenshots (the beginning portion and the end portion (it is also OK if the whole file is pasted in).

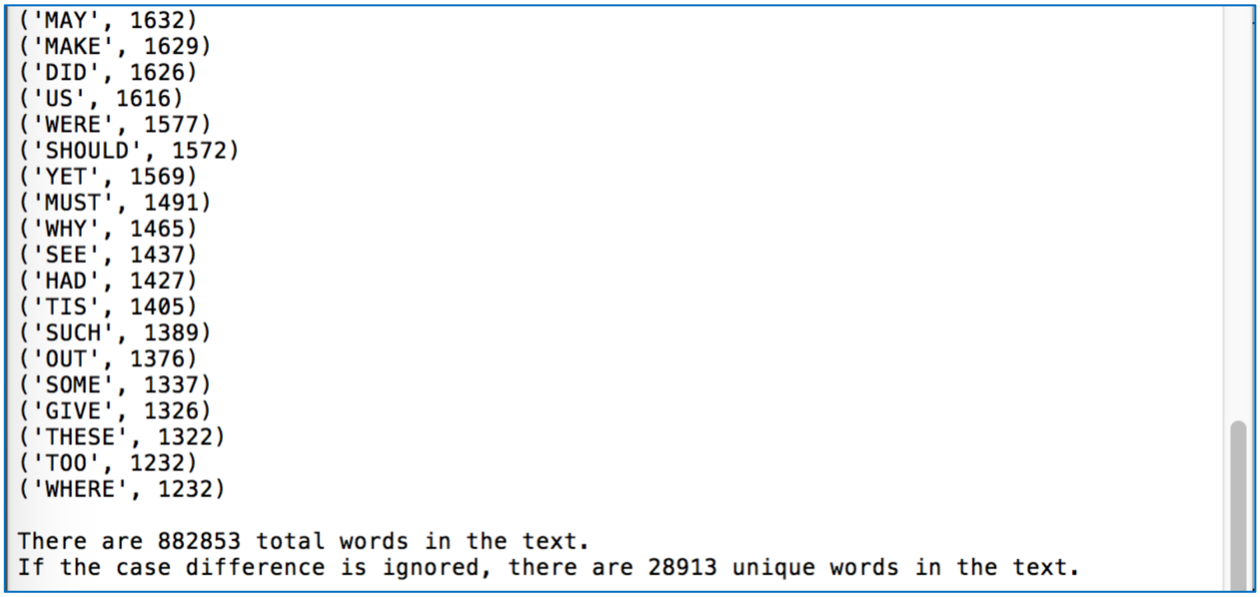


```
There are 121983 lines in the text.

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', 6652)
('THIS', 6601)
('BUT', 6265)
('HE', 6250)
('HAVE', 5879)
('AS', 5731)
('THOU', 5485)
```

.

.

.



```
('MAY', 1632)
('MAKE', 1629)
('DID', 1626)
('US', 1616)
('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 total words in the text.
If the case difference is ignored, there are 28913 unique words in the text.
```