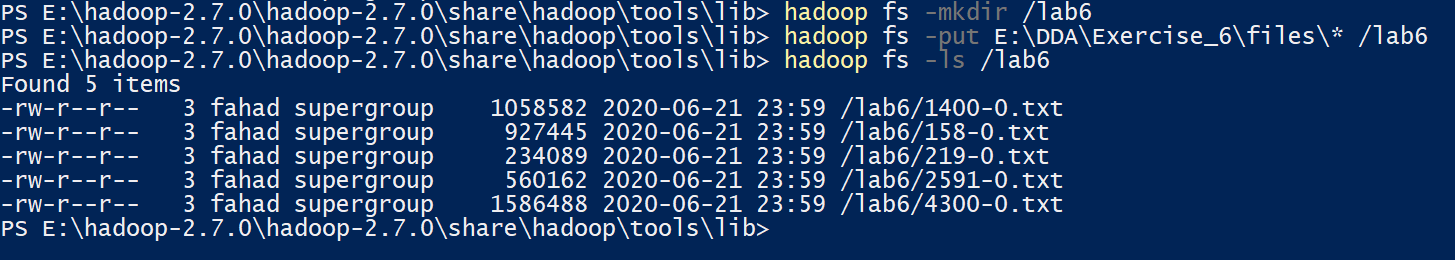
**Fahad Fiaz – (**303141**) – G2**

**System Info:**

|  |  |
| --- | --- |
| Processor | i7-5500U , 2.40GHz |
| Cores | 4 |
| Operating system | Windows 64 Bit |
| Ram | 8GB |
| Programming Language | Python 3.7.7 |

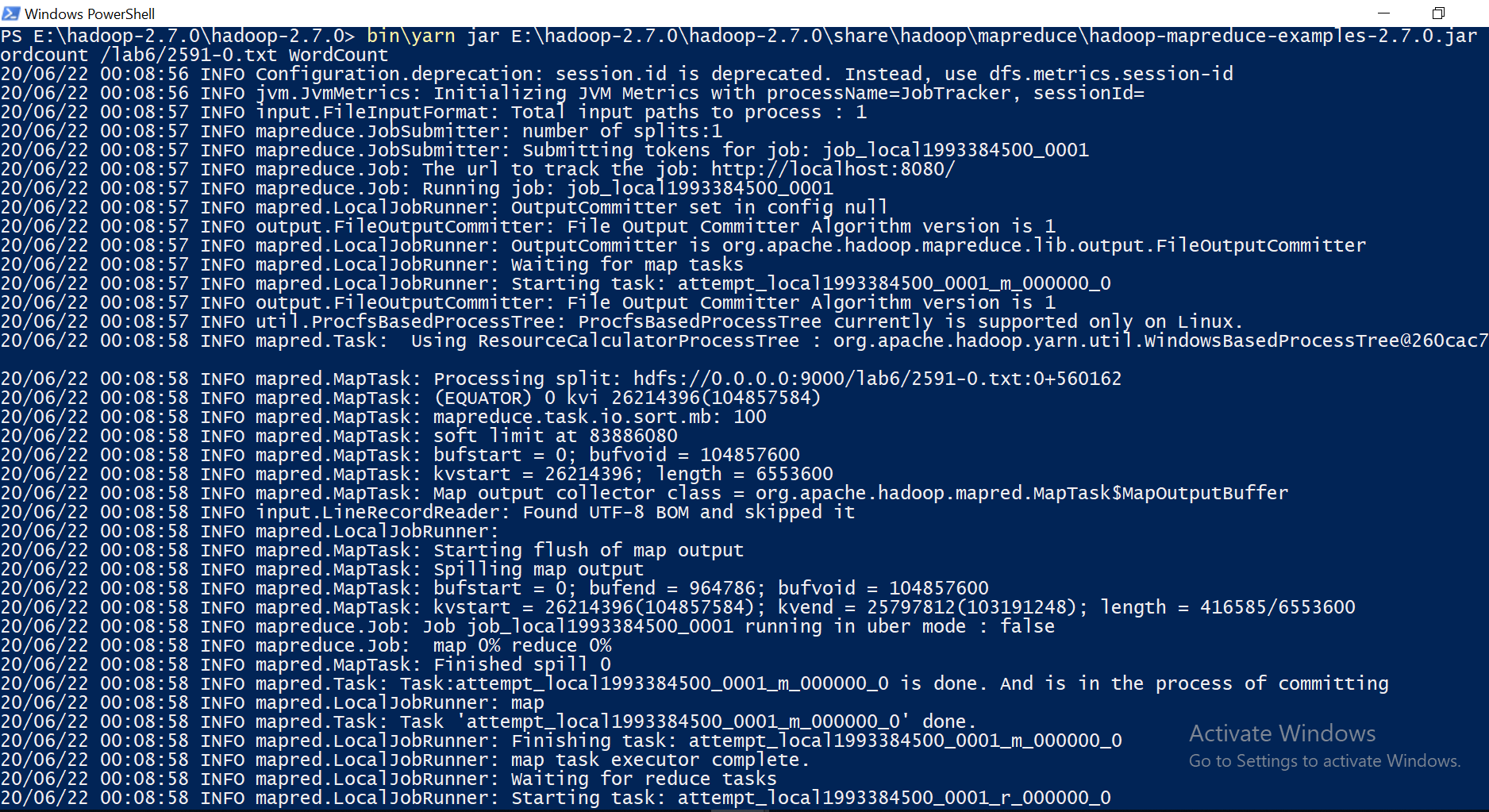
**Q1:**

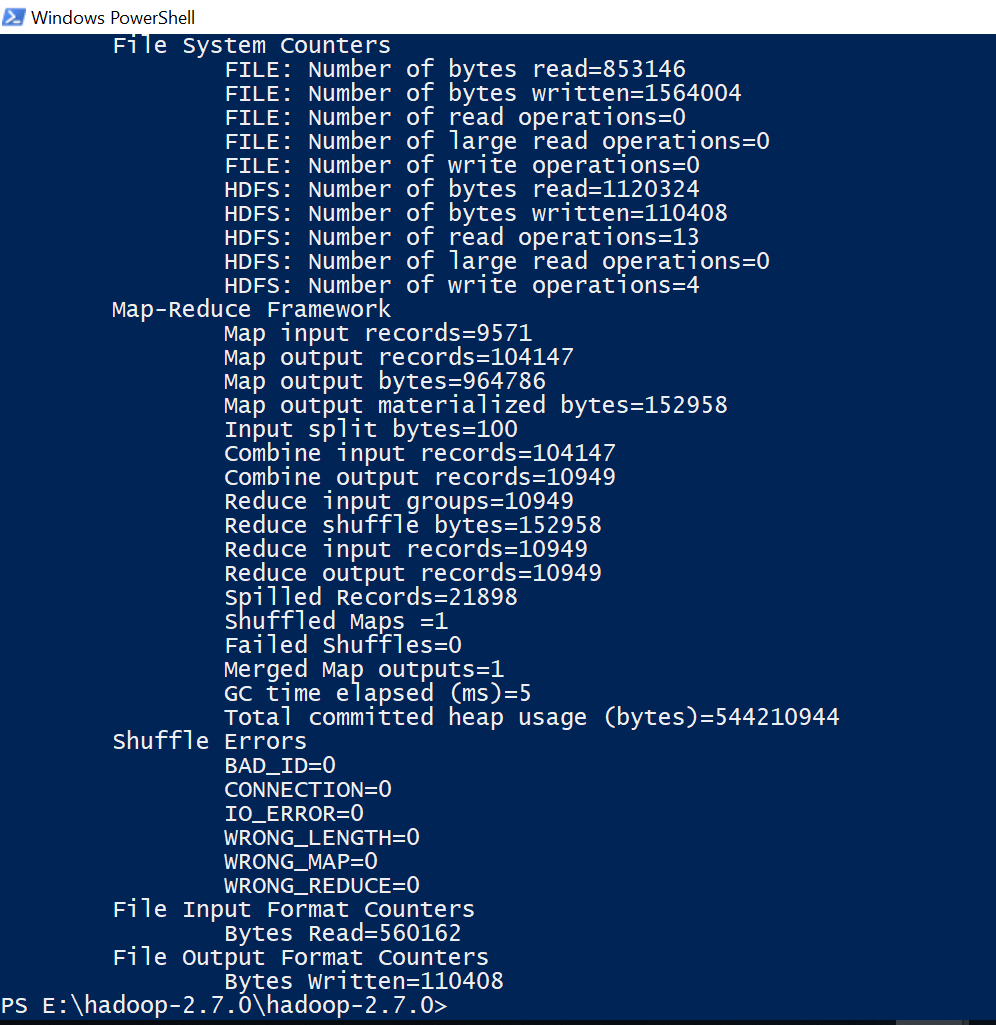
Making directory in HDFS and copied all the files I will use in exercise 1, 2, 3



Running the prebuild World count program to count occurrence of words.

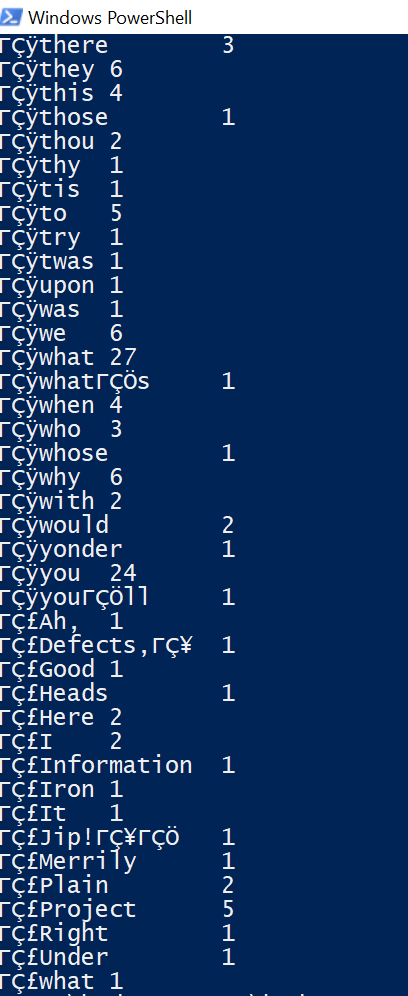
I have not attached full output because output was very lengthy instead I have taken a screenshot of running the command and end of output.



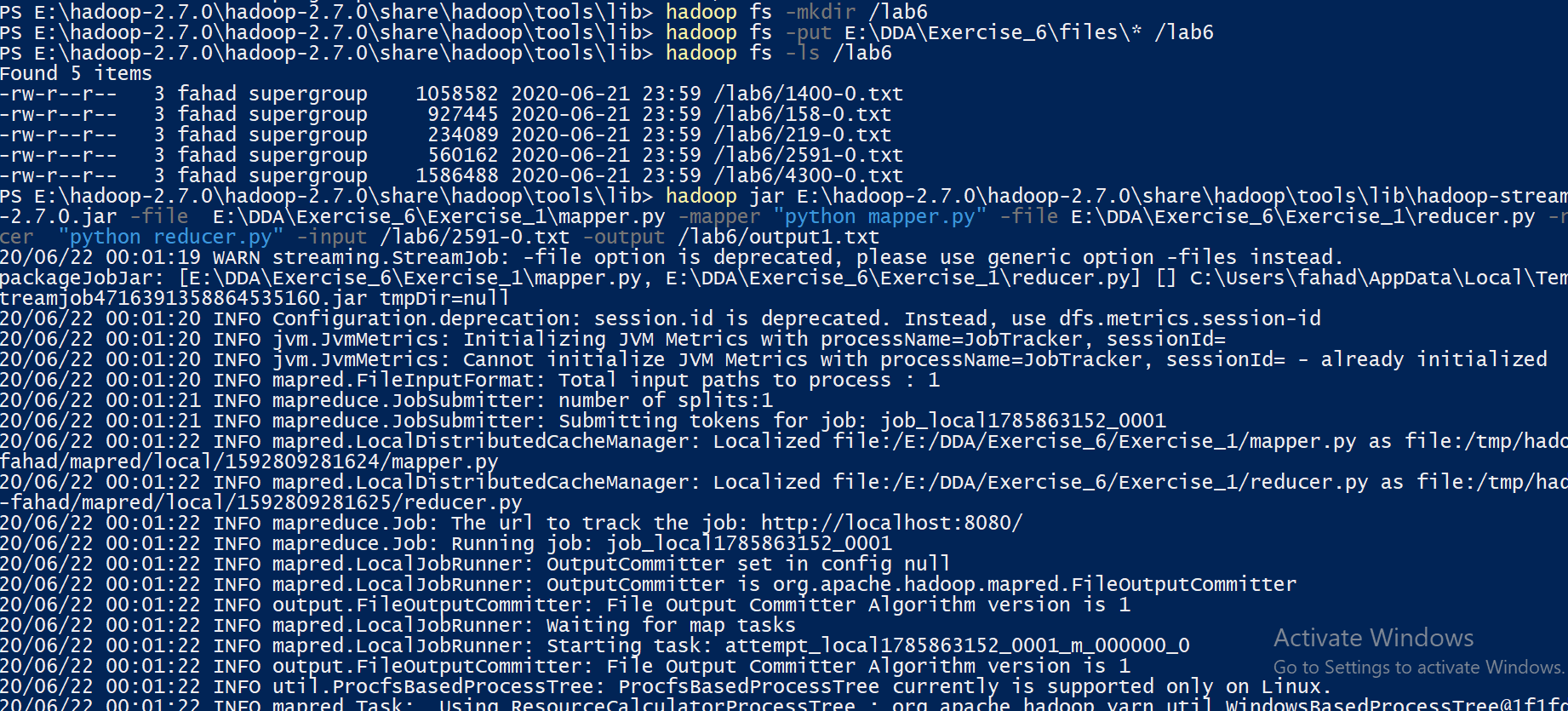


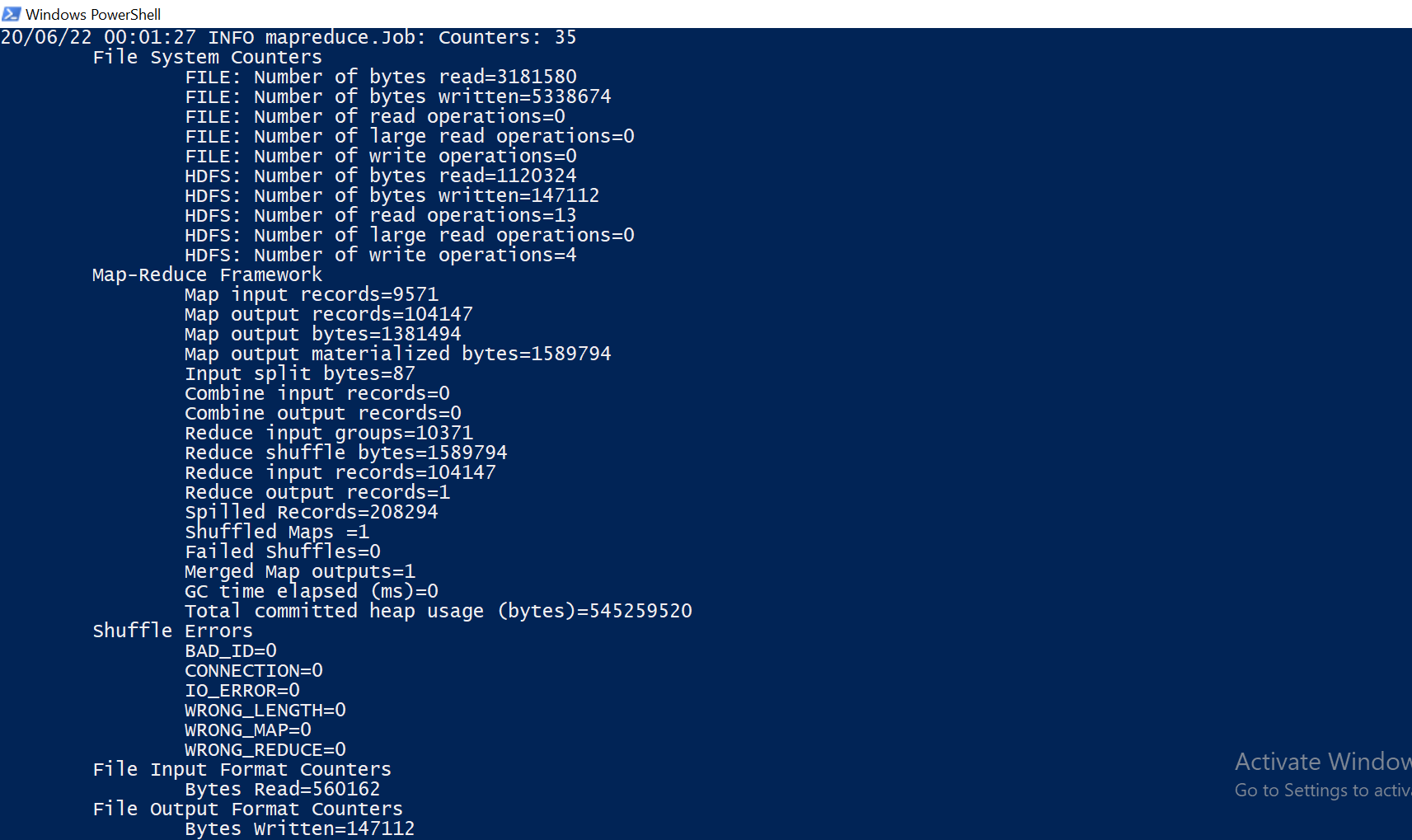
Printing output of prebuild word count program

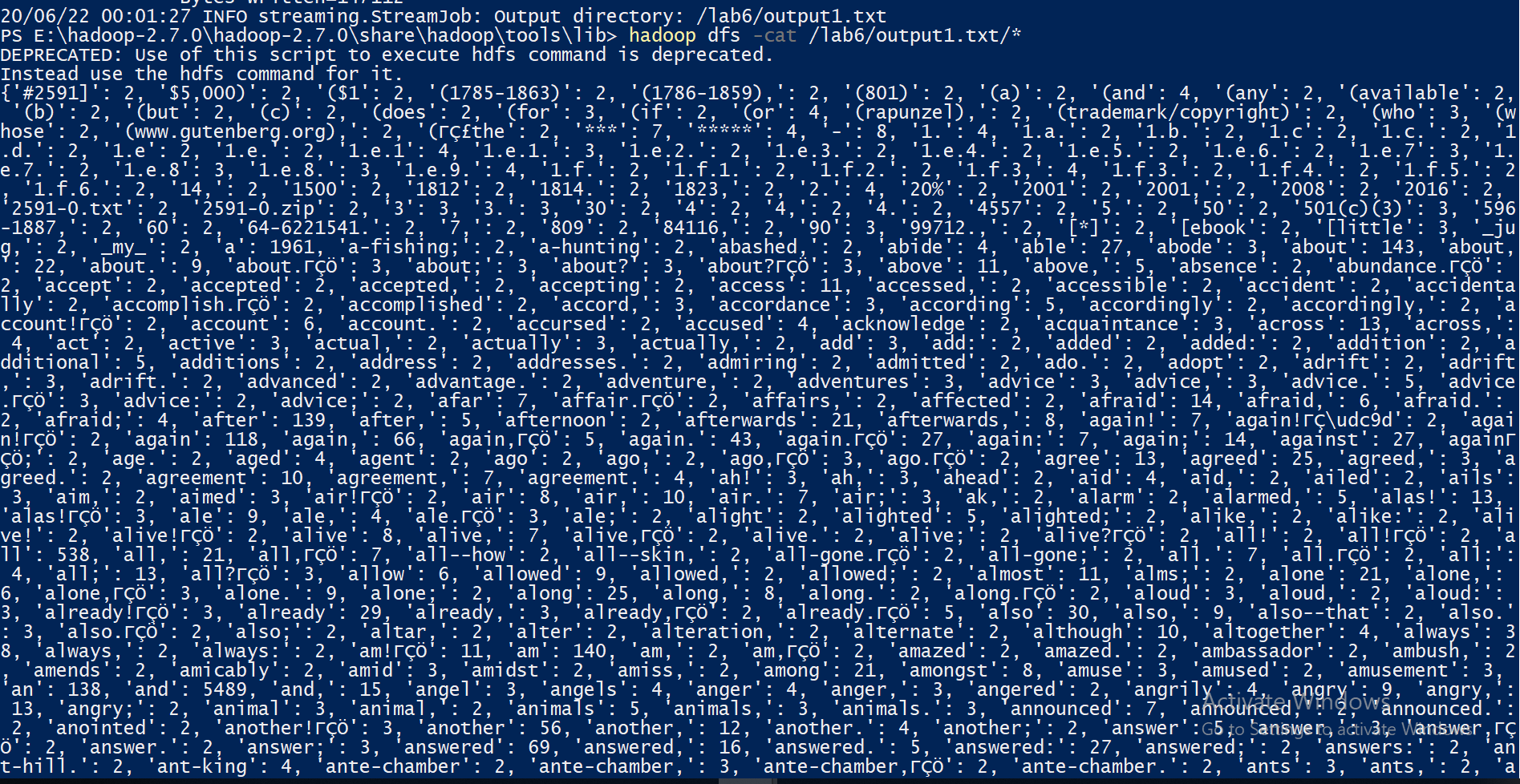
E:\DDA\Exercise_6\Exercise_1\8.PNG

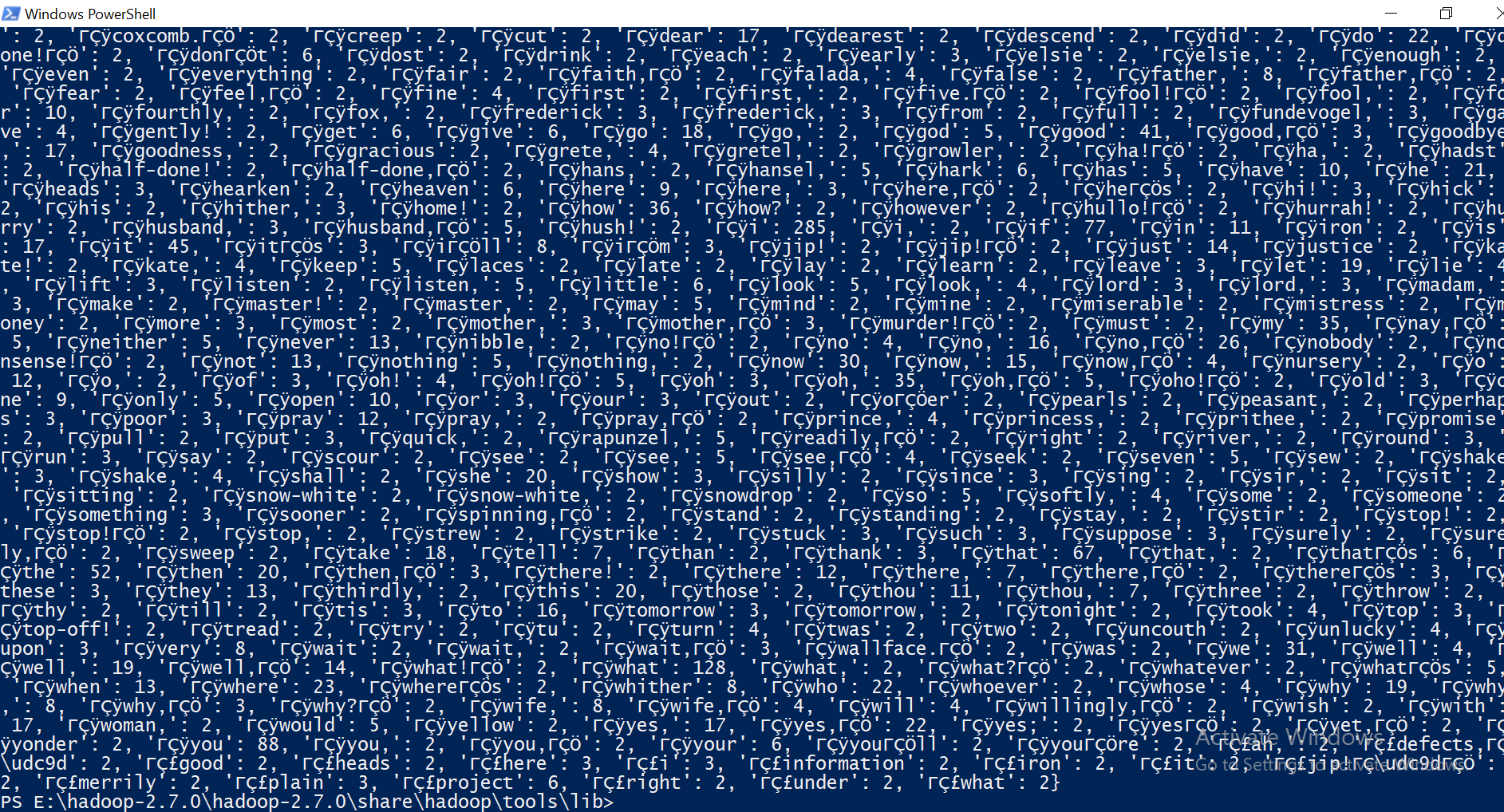


**Running my word count program and printing its output**





****

****

**Mapper:**

It will read data from STDIN, remove any leading or trailing whitespaces, split rows into words and output tuple containing word and its count to STDOUT

import sys  
  
for line in sys.stdin:  
 line = line.strip*()* words = line.split*()* # split the line into words  
 for word in words:  
 word = *(*word, 1*)* print*(*word*)*

**Reducer:**

It will read the results of mapper.py from STDIN (the output format of mapper.py and the expected input format of reducer.py must match).

Since the value received from STDIN is in string format so first we need to convert this string tuple to tuple.

dic = *{}*for line in sys.stdin:  
 key = literal\_eval*(*line*)[*0*]*

Then I check if word exists in dictionary or not. If word already exist then take its previous count and add 1 to it and save new count. If word does not exist in dictionary then I simply add word as key and count value as 1 in dictionary.

dic*[*key*]* = dic.get*(*key, 0*)* + 1

**Q2:**

**Mapper:**

It will read data from STDIN and remove any leading or trailing whitespaces. Then it read line character by character and skips any punctuations and number and concatenates other characters. Then split the line in words, check if that word exists in stop word dictionary or not. If not then output specific word with count to STDOUT

punctuations = '''!()-[]{};:'"\,<>./?@#$%^&\*\_~'''

numbers='0123456789'

stop\_words=['a','able','about','across','after','all','almost','also','am','among','an','and','any','are','as','at','be','because','been','but',

'by','can','cannot','could','dear','did','do','does','either','else','ever','every','for','from','get','got','had','has','have','he',

'her','hers','him','his','how','however','i','if','in','into','is','it','its','just','least','let','like','likely','may','me','might',

'most','must','my','neither','no','nor','not','of','off','often','on','only','or','other','our','own','rather','said','say','says',

'she','should','since','so','some','than','that','the','their','them','then','there','these','they','this','tis','to','too','twas',

'us','wants','was','we','were','what','when','where','which','while','who','whom','why','will','with','would','yet','you','your']

no\_punct\_and\_number = ""

for line in sys.stdin:  
 line = line.strip*()* for char in line:  
 if char not in punctuations and numbers:  
 no\_punct\_and\_number = no\_punct\_and\_number + char  
 words = no\_punct\_and\_number.split*()*# split the line into words  
 for word in words:  
 if word not in stop\_words:  
 word = *(*word.lower*()*, 1*)* print*(*word*)*

Following line uses build in function of translation. maketrans first two parameters tells translate function to translate nothing to nothing and translate any punctuation or numbers to None (i.e. remove them). This function worked faster so I used this in my code. My implementation of removing punctuation and number was taking a lot of time.

line = line.translate*(*str.maketrans*(*'','',punctuation*))*line = line.translate*(*str.maketrans*(*'','','1234567890'*))*line = line.strip*()*# split the line into words

**Reducer:**

It will read the results of mapper.py from STDIN (the output format of mapper.py and the expected input format of reducer.py must match).

Since the value received from STDIN is in string format so first we need to convert this string tuple to tuple.

dic = *{}*for line in sys.stdin:  
 key = literal\_eval*(*line*)[*0*]*

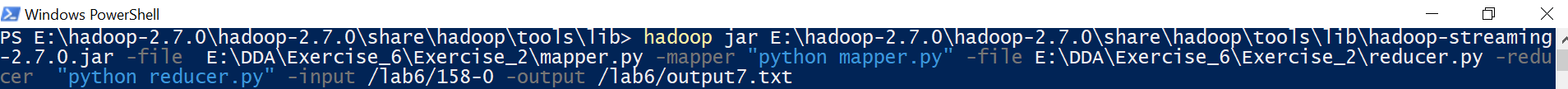
Then I check if word exists in dictionary or not. If word already exists then take its previous count and add 1 to it and save new count. If word does not exist in dictionary then I simply add word as key and count value as 1 in dictionary.

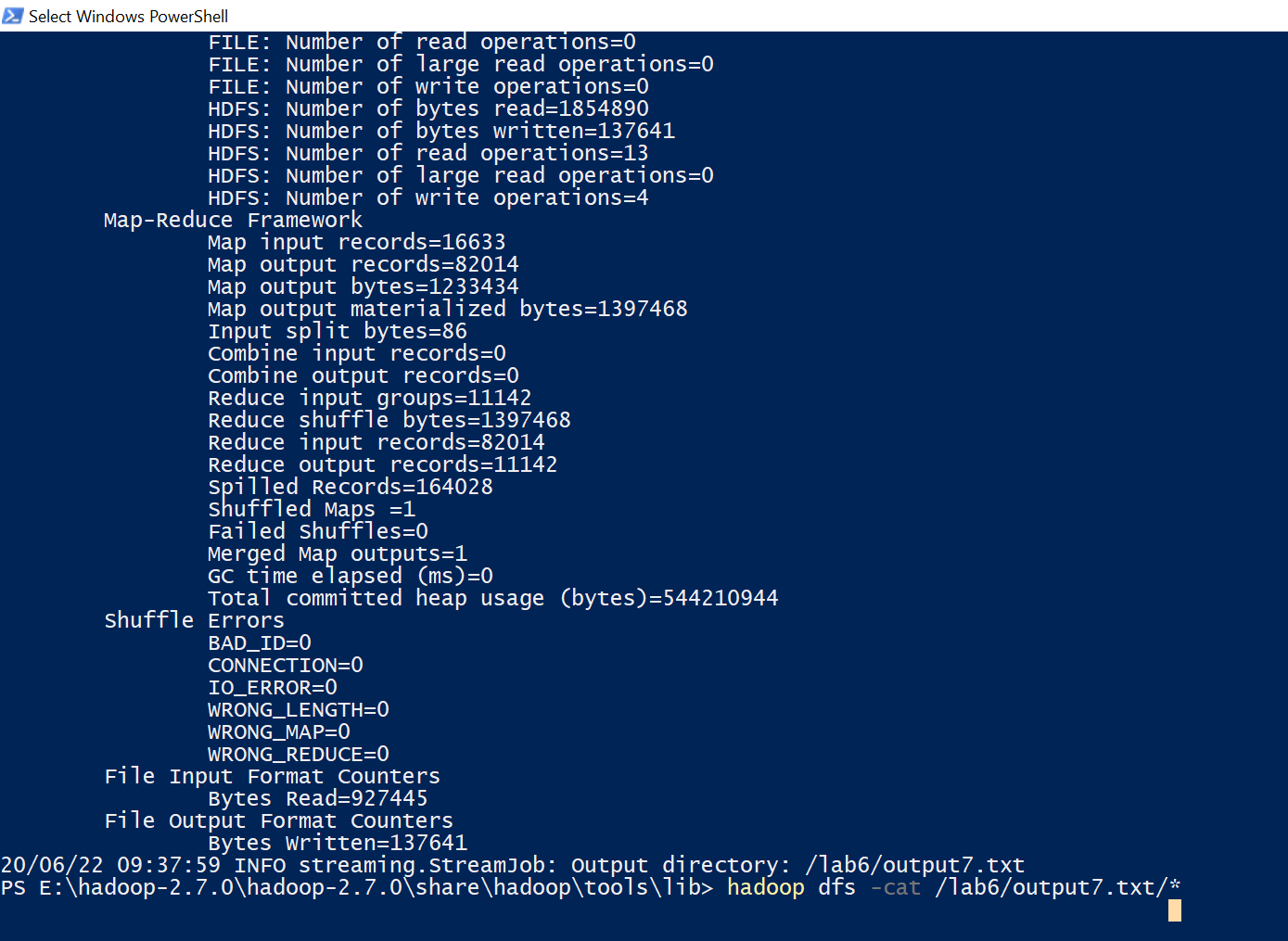
dic*[*key*]* = dic.get*(*key, 0*)* + 1

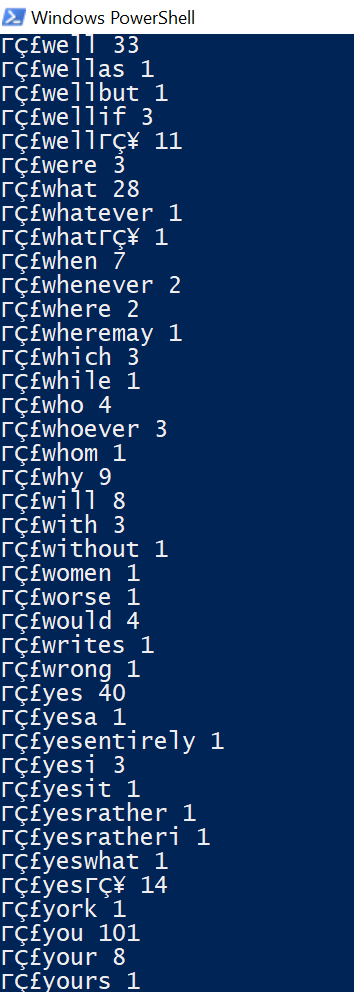
**Show the final results:**

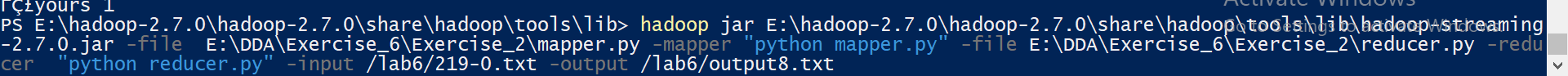
for k,v in dic.items*()*:  
 print*(*'{} {}'.format*(*k,v*))*

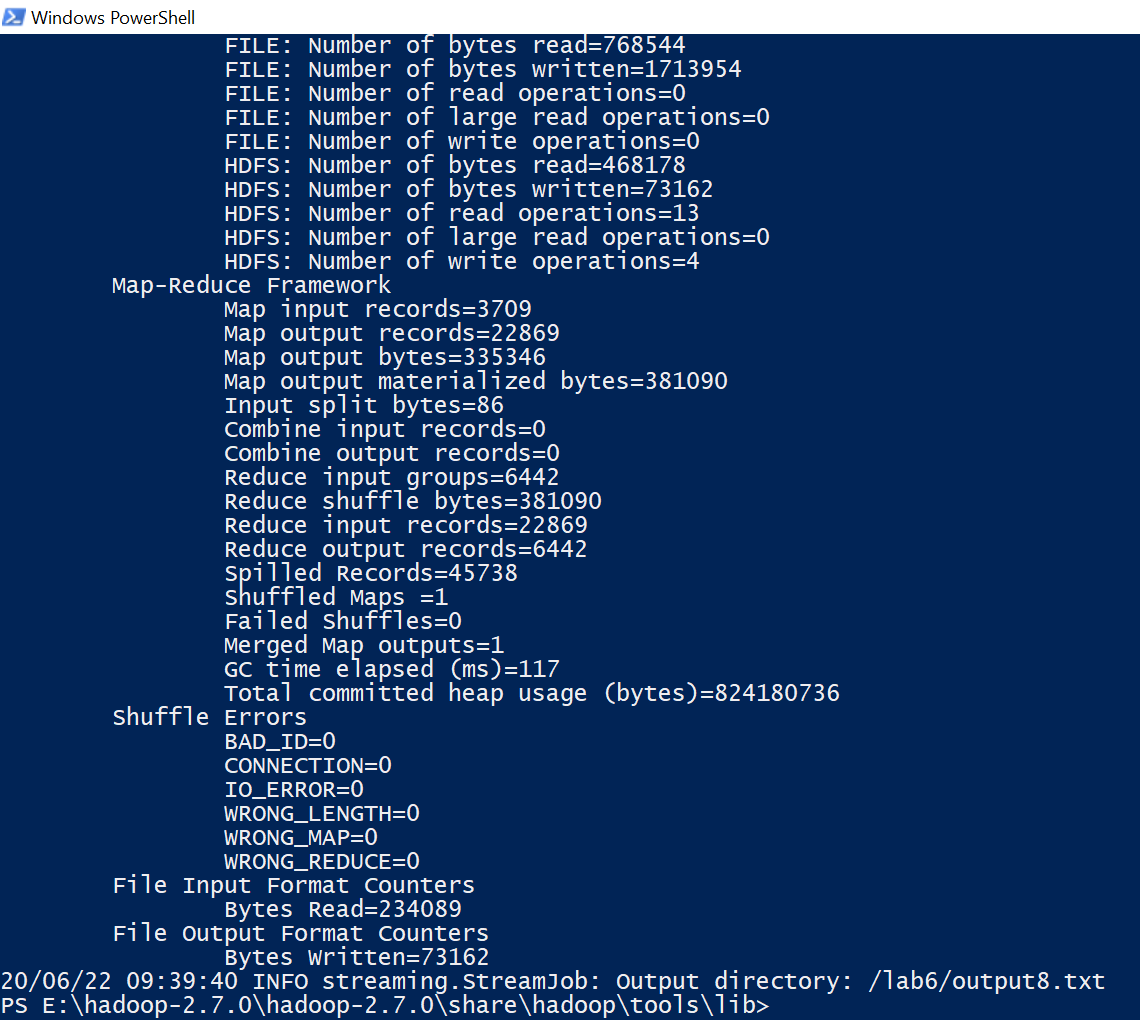
**Running commands: All 5 files**

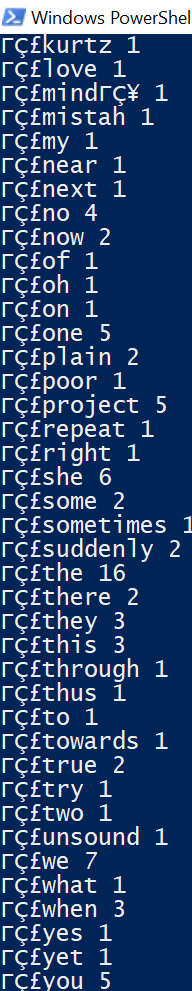
****

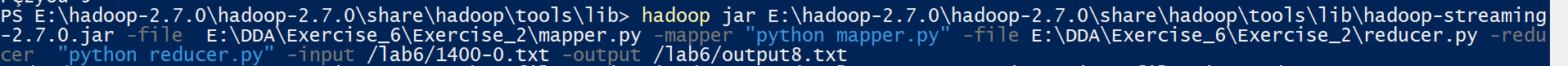
****

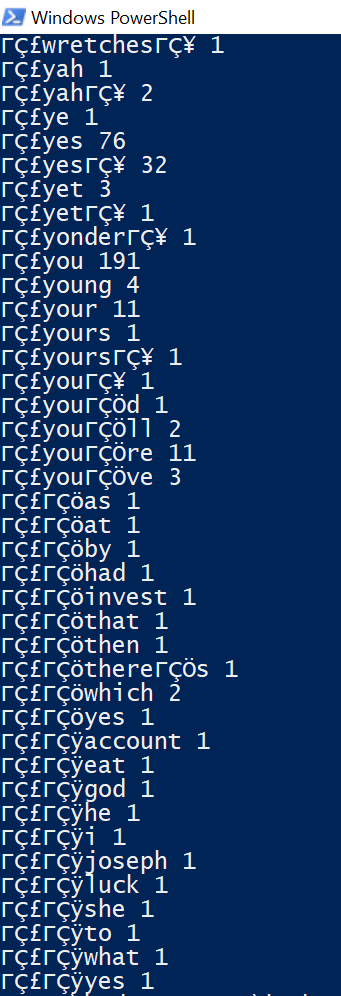
****

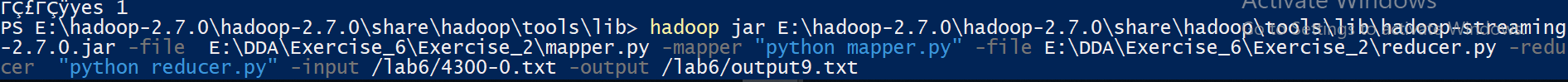
****

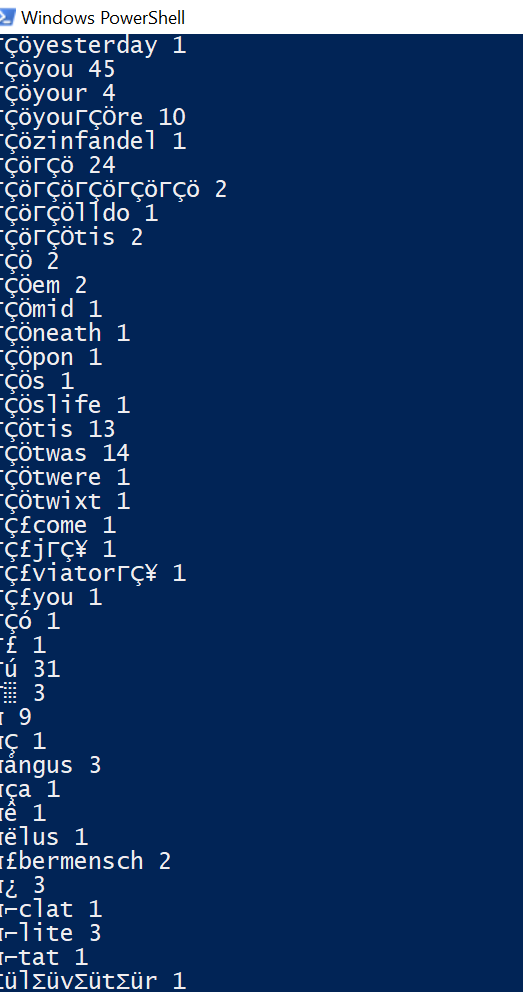
****

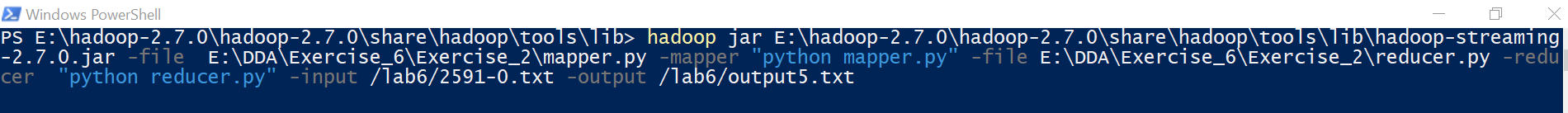
****

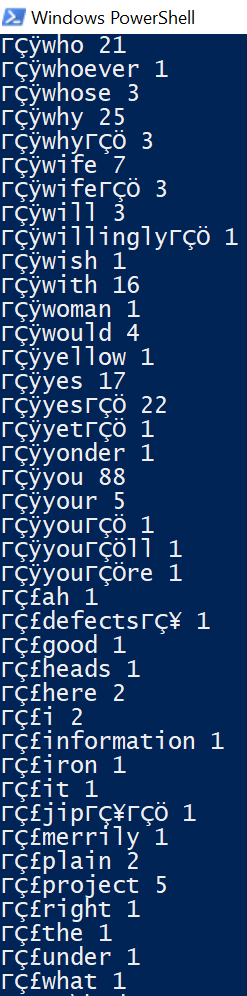
****

****

****

****





**Q3:**

**Mapper:**

It will read data from STDIN and get file names from environment variable “map\_input\_file”. These are input files names which are passes as command line argument. Then it remove any punctuations or number from line. Then remove any leading or trailing whitespaces, split the line in words, check if that word exists in stop word dictionary or not. If not then output specific word with count to STDOUT

for line in sys.stdin:  
 filename = os.environ*[*"map\_input\_file"*]* line = line.translate*(*str.maketrans*(*'','',punctuation*))* line = line.translate*(*str.maketrans*(*'','','1234567890'*))* line = line.strip*()*# remove leading and trailing whitespace  
 words = line.split*()* # split the line into words  
 for word in words:  
 if word not in stop\_words:  
 print*((*filename,word,1*))*

**Reducer:**

It will read the results of mapper.py from STDIN (the output format of mapper.py and the expected input format of reducer.py must match).

Since the value received from STDIN is in string format so first we need to convert this string tuple to tuple. After converting it to tuple we only take first two arguments (filename, word) of tuple as third argument is just 1 in all cases.

import sys  
from ast import literal\_eval  
dic = *{}*for line in sys.stdin:  
 key = literal\_eval*(*line*)[*:2*]*

Then I check if word exists in dictionary or not. If word already exists then take its previous count and add 1 to it and save new count. If word does not exist in dictionary then I simply add word as key and count value as 1 in dictionary.

dic*[*key*]* = dic.get*(*key , 0*)* + 1

Then we output the tuple as following format (filename, word, count of words in that filename).

for k,v in dic.items*()*:  
 print*((*k*[*0*]*,k*[*1*]*,v*))*

**Mapper2:**

It will read data from STDIN (output of reducer1). Since the value received from STDIN is in string format so first we need to convert this string tuple to tuple. Then this tuple (filename, word, count of words in that filename) is passed as output to STDOUT

for line in sys.stdin:   
 filename\_word\_count = literal\_eval*(*line*)* print*(*filename\_word\_count*)*

**Reduce2:**

It will read the results of mapper.py from STDIN (the output format of mapper.py and the expected input format of reducer.py must match).

Since the value received from STDIN is in string format so first we need to convert this string tuple to tuple. After converting it to tuple we calculate total words in each file. Also we save tuple that we received from STDIN which will help use later.

for line in sys.stdin:  
  
 filename, word, count = literal\_eval*(*line*)* count = int*(*count*)* if prev\_filename == filename:  
 N = N + count  
 else:  
 if prev\_filename != None:  
 file\_name\_with\_total\_wordCount*[*prev\_filename*]* = N  
 # print(file\_name\_with\_total\_wordCount[prev\_filename])  
  
 N = 0  
 prev\_filename = filename  
 saved\_previous\_data.append*(*line*)*file\_name\_with\_total\_wordCount*[*prev\_filename*]* = N #saved last file count

Then we map the list of tuple that we saved previously to total words in the specific files from which the word in tuple belong. Finally we output the tuple as following format (filename, word, count of words in that filename, total\_words\_count\_in\_specific file).

for line in saved\_previous\_data:  
 filename, word, count = literal\_eval*(*line*)* for k in file\_name\_with\_total\_wordCount.keys*()*:  
 if filename == k:  
 print*((*word, filename, count, file\_name\_with\_total\_wordCount*[*k*]))*

**Mapper3:**

It will read data from STDIN (output of reducer1). Since the value received from STDIN is in string format so first we need to convert this string tuple to tuple. Then this tuple (filename, word, count of words in that filename, total\_words\_count\_in\_specific file,1) is passed as output to STDOUT

for line in sys.stdin:   
 word\_filename\_count\_totalcount = literal\_eval*(*line*)* print*((*word\_filename\_count\_totalcount*[*0*]*,word\_filename\_count\_totalcount*[*1*]*,word\_filename\_count\_totalcount*[*2*]*,word\_filename\_count\_totalcount*[*3*]*,1*))*

**Reduce3:**

It will read the results of mapper.py from STDIN (the output format of mapper.py and the expected input format of reducer.py must match).

Since the value received from STDIN is in string format so first we need to convert this string tuple to tuple. After converting it to tuple we calculate number of documents that  
contains token t. Also we save tuple (word, filename) that which will help use later.

for line in sys.stdin:  
 word, filename, wordCount, total\_WordCount, count = literal\_eval*(*line*)* if prev\_word == word:  
 total\_count = total\_count + int*(*count*)* else:  
 if prev\_word != None:  
 df*[*prev\_word*]* = *(*wordCount, total\_WordCount, total\_count*)* word\_filename = *(*prev\_word, filename*)* saved\_previous\_data.append*(*word\_filename*)* total\_count = 1  
 prev\_word = word  
  
df*[*prev\_word*]* = *(*wordCount, total\_WordCount, total\_count*)*word\_filename = *(*prev\_word, filename*)*saved\_previous\_data.append*(*word\_filename*)*

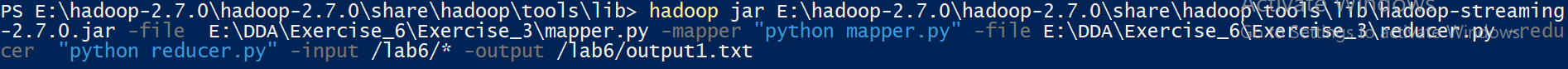
Now we have a list of tuple which contain word and filename. Then we have dictionary that has word as key and tuple (wordCount, total\_WordCount, total\_count\_of\_tokenK\_i n document) as value.

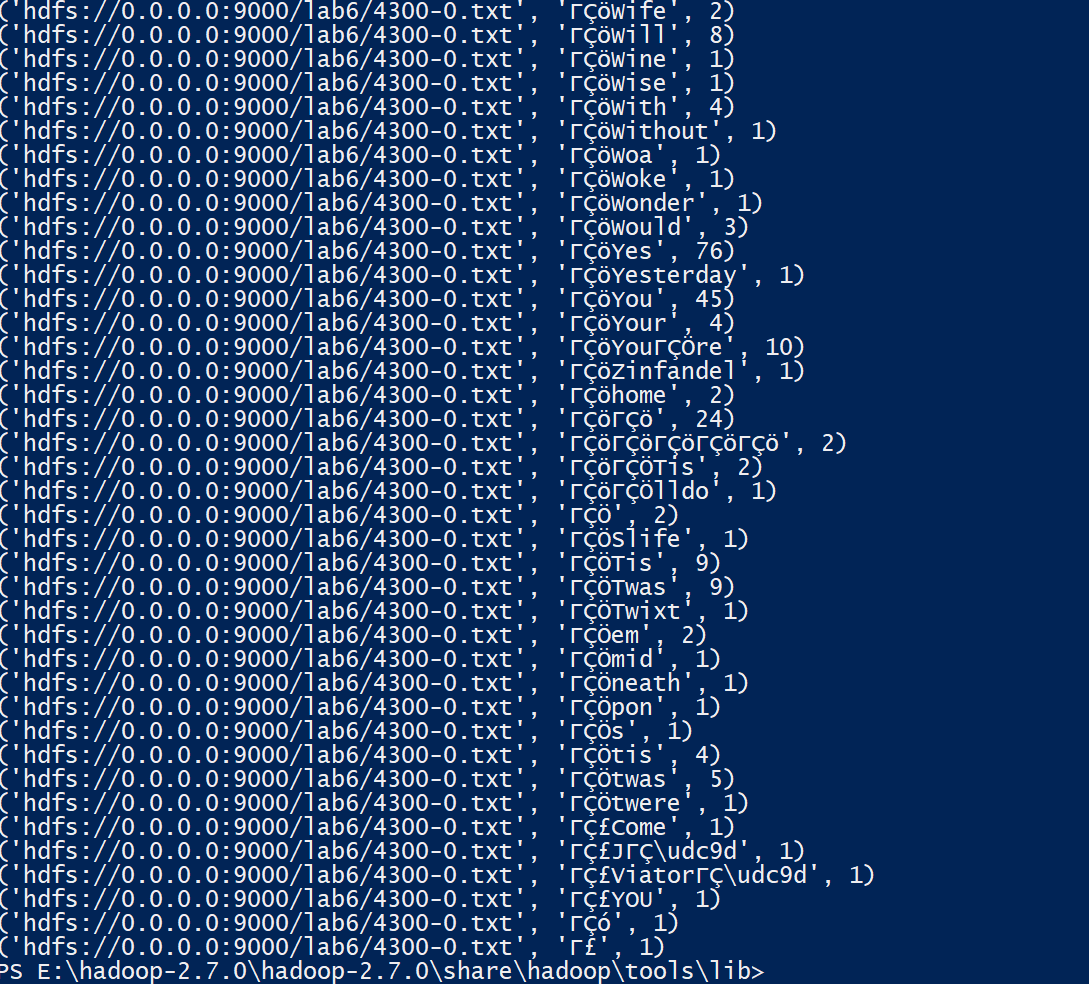
Then first I take token (word) and get its data from dictionary.Then I used this data to find tfifd of token using formula given in exercise

for line in saved\_previous\_data:  
 word, filename = line  
 for k in df.keys*()*:  
 if word == k:  
 wordCount, total\_WordCount, total\_count\_of\_word\_in\_different\_document = df*[*k*]* tfidf = *(*wordCount / total\_WordCount*)* \* log10*(*5 / total\_count\_of\_word\_in\_different\_document*)* print*(*'{}: {}'.format*(*word, tfidf*))*

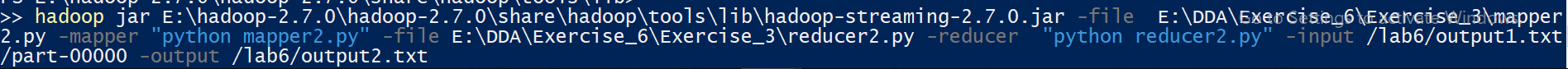
Running commands:

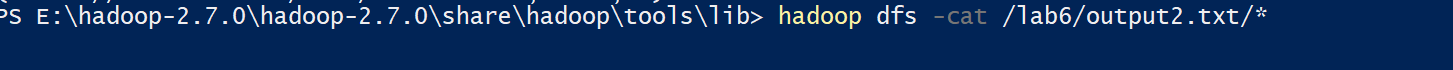
First mapper, reducer:



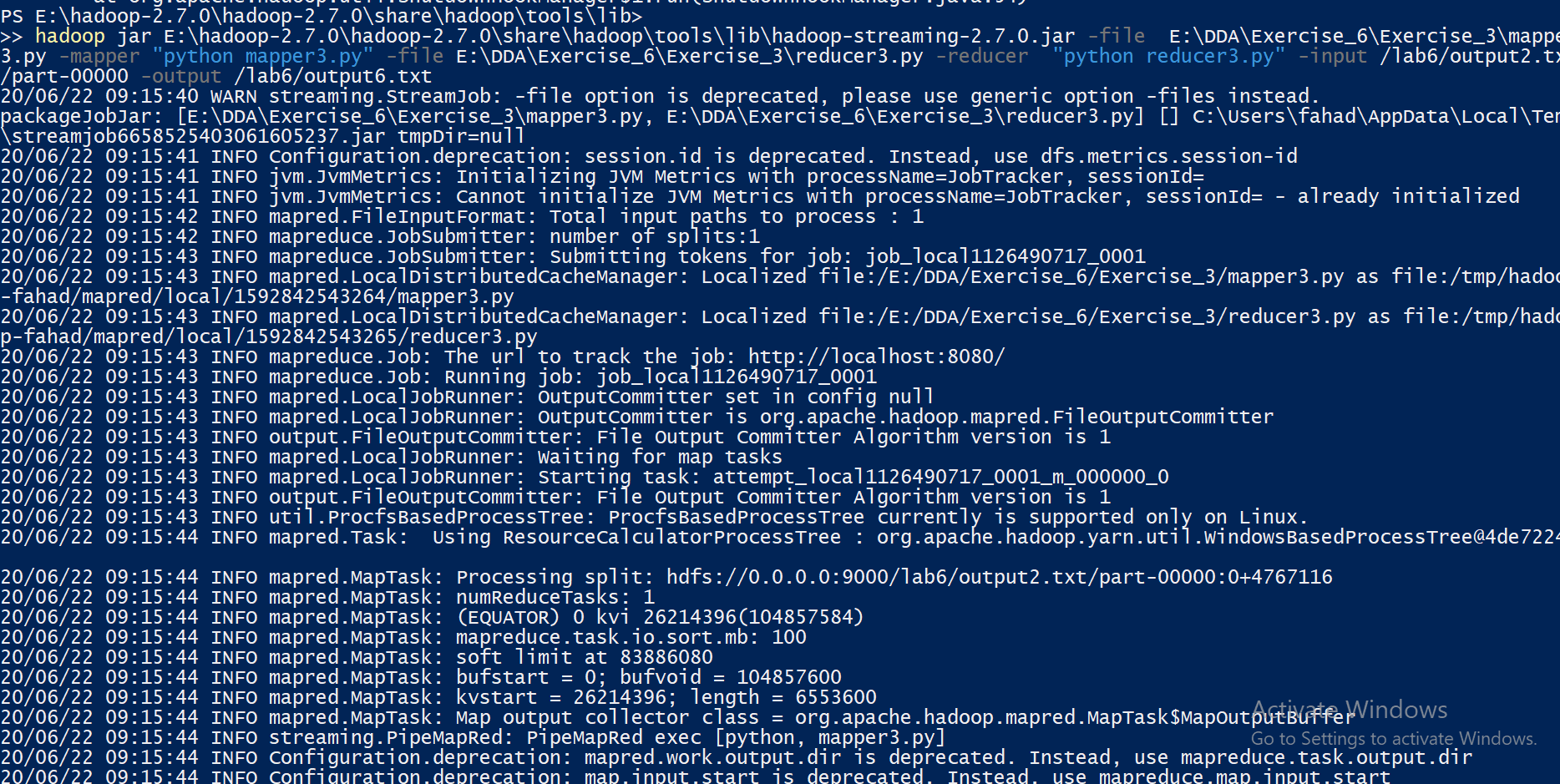


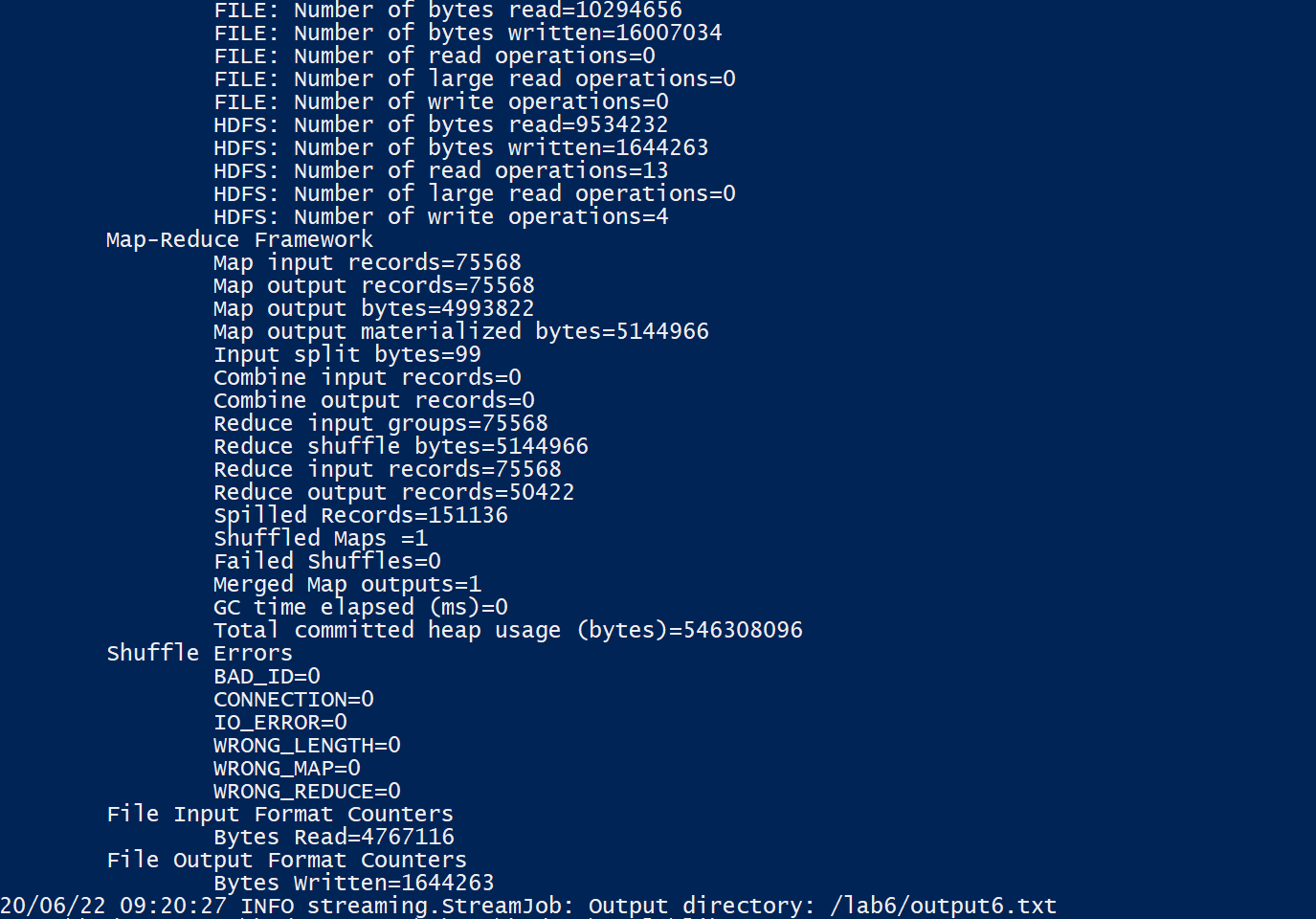
Second mapper reducer:





Third mapper reducer:





E:\DDA\Exercise_6\Exercise_3\fninp.PNG

