CS7070 Big Data Analytics

Programming Project Assignment

Guidelines followed while handling data:

- All the words in the data file are converted into lower case and punctuations present at the leading and trailing of the words are stripped.
  For example: “We won’t let you go!” => <we,1> <won’t,1><let,1><you,1><go,1>
- The punctuation within the word is left so that the two actual meanings of the words “its” and “it’s” is preserved.

Note: However, the results of the raw data are also presented in the **appendix** of this document.

Note: The code presented in this document is the modified code from the source: http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

1. Design and execute a MapReduce program to produce the frequencies of all the words in the book collection, retaining only those words whose frequencies are greater than 5. Submit the following:
    a. Your commented code for the Mapper and Reducer

    Mapper Code: wc_1_1_map.py

```
#!/usr/bin/env python
import sys
import string

def read_input(file):
    for line in file:
        # strip the leading and trailing spaces
        # And, split the line into words
        yield line.strip().split()

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_input(sys.stdin)
    for words in data:
        # Each word in words is converted into lower case
        # and leading an trailing punctuations are removed;
        # write each word tab-delimited with the trivial count 1 to STDOUT (standard output)
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        for word in words:
            word = word.lower().strip(string.punctuation)
            if word:
                print '%s%s%d' % (word, separator, 1)

if __name__ == "__main__":
    main()
```

By:
Varun Raj Rayabarapu

Reducer Code: wc_1_1_red.py

```
#!/usr/bin/env python
"""A more advanced Reducer, using Python iterators and generators."""

from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 1)

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_mapper_output(sys.stdin, separator=separator)
    # groupby groups multiple word-count pairs by word,
    # and creates an iterator that returns consecutive keys and their group:
    # current_word - string containing a word (the key)
    #   group - iterator yielding all ["&lt;current_word&gt;", "&lt;count&gt;"] items
    for current_word, group in groupby(data, itemgetter(0)):
        try:
            total_count = sum(int(count) for current_word, count in group)
            # write only words whose total count > 5
            if total_count>5:
                print "%s%s%d" % (current_word, separator, total_count)
        except ValueError:
            # count was not a number, so silently discard this item
            pass

if __name__ == "__main__":
    main()
```

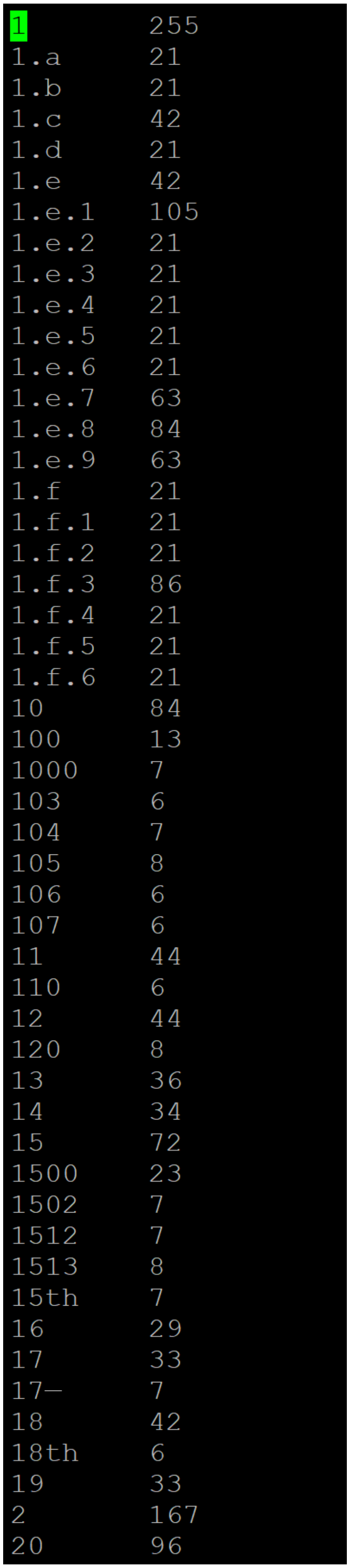
Command used to execute the above code in Hadoop:

```
$ hadoop jar /usr/hdp/3.0.0.0-1634/hadoop-mapreduce/hadoop-streaming.jar -file wc_1_1_map.py -mapper wc_1_1_map.py -file wc_1_1_red.py -reducer wc_1_1_red.py -input /user/kumarlt/books/input/* -output BDA/PPA_1/results/wc_1_1_cres
```

b. First 50 words and their frequencies from your program’s output file.

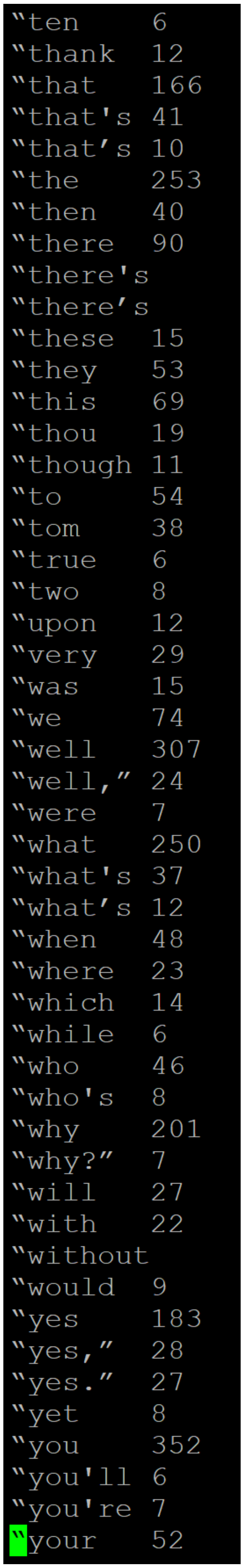
The first 50 words from the program output file are numericals as show in the screen-shot on the next page.

Each word is tab delimited with its frequency.



```
1           255
1.a         21
1.b         21
1.c         42
1.d         21
1.e         42
1.e.1       105
1.e.2       21
1.e.3       21
1.e.4       21
1.e.5       21
1.e.6       21
1.e.7       63
1.e.8       84
1.e.9       63
1.f         21
1.f.1       21
1.f.2       21
1.f.3       86
1.f.4       21
1.f.5       21
1.f.6       21
10          84
100         13
1000        7
103         6
104         7
105         8
106         6
107         6
11          44
110         6
12          44
120         8
13          36
14          34
15          72
1500        23
1502        7
1512        7
1513        8
15th        7
16          29
17          33
17—         7
18          42
18th        6
19          33
2           167
20          96
```

c. Last 50 words and their frequencies from your program’s output file.



```
“ten     6
“thank   12
“that    166
“that's  41
“that’s  10
“the     253
“then    40
“there   90
“there's
“there’s
“these   15
“they    53
“this    69
“thou    19
“though  11
“to      54
“tom     38
“true    6
“two     8
“upon    12
“very    29
“was     15
“we      74
“well    307
“well,”  24
“were    7
“what    250
“what's  37
“what’s  12
“when    48
“where   23
“which   14
“while   6
“who     46
“who's   8
“why     201
“why?”   7
“will    27
“with    22
“without
“would   9
“yes     183
“yes,”   28
“yes.”   27
“yet     8
“you     352
“you'll  6
“you're  7
“your    52
```

By:
Varun Raj Rayabarapu

2. Design and execute a MapReduce program to produce frequencies of all 2-gram word-pairs in the book collection, retaining only those 2-grams whose frequencies are greater than 5. Submit the following:

   a. Your commented MapReduce program to produce the frequencies of all the 2-grams in all the books.

Mapper Code: wc_2_1_map.py

```
#!/usr/bin/env python
import sys
import string

def read_input(file):
   for line in file:
      # split the line into words
      yield line.strip().split()

def main(separator='\t'):
   # input comes from STDIN (standard input)
   data = read_input(sys.stdin)
   # initializing lastword to empty string
   lastword = ""
   for words in data:
      # Each word in words is converted into lower case
      # and leading an trailing punctuations are removed;
      # write each word tab-delimited with the next word and again
      # tab-delimited with trivial count 1 to STDOUT (standard output)
      # what we output here will be the input for the
      # Reduce step, i.e. the input for reducer.py

      for i in range(len(words)-2):
         word_1 = words[i].lower().strip(string.punctuation)
         word_2 = words[i+1].lower().strip(string.punctuation)
          if word_1 and word_2:
             print '%s%s%s%s%d' % (word_1, separator, word_2,separator,1)
      # This is code below deals with producing the bigrams with the word
      # from current line last word and next line first word
      if lastword:
         print '%s%s%s%s%d' % (lastword, separator, words[0].lower()
         \.strip(string.punctuation),separator,1)
         lastword=words[-1].lower().strip(string.punctuation)

if __name__ == "__main__":
   main()
```

Reducer Code: wc_2_1_red.py

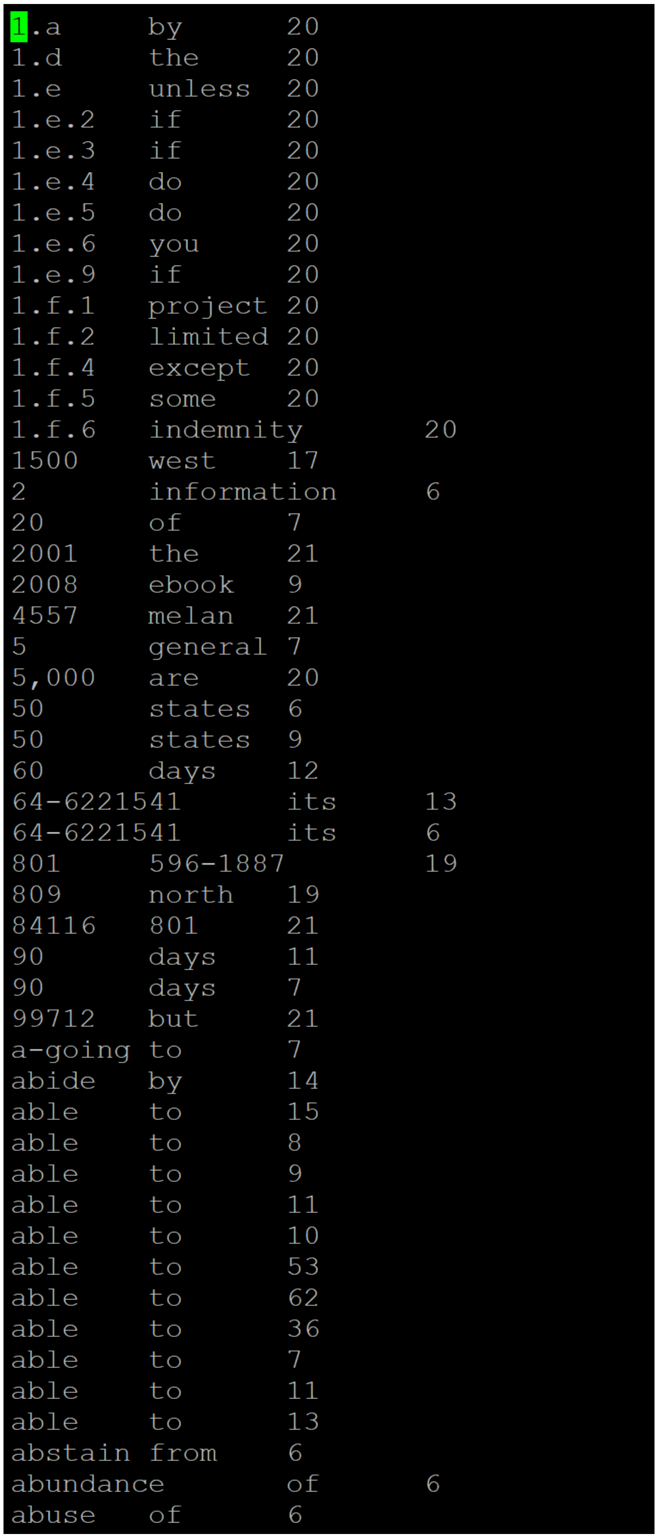
```
#!/usr/bin/env python
from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file, separator='\t'):
   for line in file:
      yield line.rstrip().split(separator, 2)

def main(separator='\t'):
   # input comes from STDIN (standard input)
   data = read_mapper_output(sys.stdin, separator=separator)
   # groupby groups multiple word-word-count pairs by firstword,
   # and creates an iterator that returns consecutive keys and their group:
   # current_firstword - string containing a word (the key)
   #   group - iterator yielding all ["&lt;current_firstword&gt;",
  # "&lt;current_secondword&gt;", "&lt;count&gt;"] items
   for current_firstword, group_1 in groupby(data, itemgetter(0)):
      try:
        # groupby groups multiple word-word-count pairs by secondword,
        # and creates an iterator that returns consecutive keys and their group:
        # current_secondword - string containing a word (the key)
        #   group - iterator yielding all ["&lt;current_firstword&gt;",
        # "&lt;current_secondword&gt;", "&lt;count&gt;"] items
         for current_secondword, group_2 in groupby(group_1,itemgetter(1)):
            try:
               total_count = sum(int(count) for firstword,secondword, count in group_2)
               if total_count>5:
                  print "%s%s%s%s%d" % (current_firstword, separator,\
                  current_secondword, separator,total_count)
            except ValueError:
               pass
      except ValueError:
         # count was not a number, so silently discard this item
         pass

if __name__ == "__main__":
   main()
```

Command used to execute the above code in Hadoop:

```
$ hadoop jar /usr/hdp/3.0.0.0-1634/hadoop-mapreduce/hadoop-streaming.jar -file
wc_2_1_map.py -mapper wc_2_1_map.py -file wc_2_1_red.py -reducer wc_2_1_red.py -
input /user/kumarlt/books/input/* -output BDA/PPA_1/results/wc_2_1_cres
```

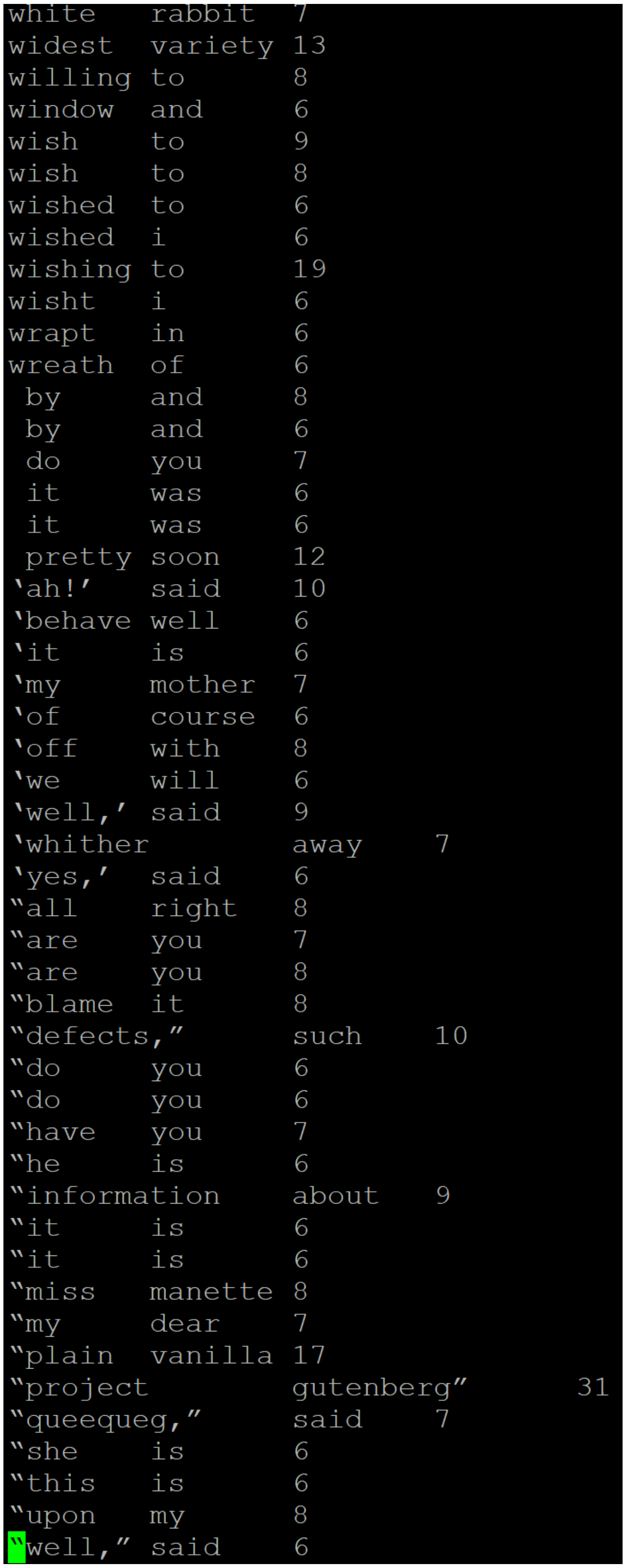
b. First 50 2-grams and their frequencies from your program’s output file.

Each line contains bigram(two words delimited by tab)and its frequency.



```
1.a     by      20
1.d     the     20
1.e     unless  20
1.e.2   if      20
1.e.3   if      20
1.e.4   do      20
1.e.5   do      20
1.e.6   you     20
1.e.9   if      20
1.f.1   project 20
1.f.2   limited 20
1.f.4   except  20
1.f.5   some    20
1.f.6   indemnity       20
1500    west    17
2       information     6
20      of      7
2001    the     21
2008    ebook   9
4557    melan   21
5       general 7
5,000   are     20
50      states  6
50      states  9
60      days    12
64-6221541      its     13
64-6221541      its     6
801     596-1887        19
809     north   19
84116   801     21
90      days    11
90      days    7
99712   but     21
a-going to      7
abide   by      14
able    to      15
able    to      8
able    to      9
able    to      11
able    to      10
able    to      53
able    to      62
able    to      36
able    to      7
able    to      11
able    to      13
abstain from    6
abundance       of      6
abuse   of      6
```

c. Last 50 2-grams and their frequencies from your program’s output file.



```
white   rabbit  7
widest  variety 13
willing to      8
window  and     6
wish    to      9
wish    to      8
wished  to      6
wished  i       6
wishing to      19
wisht   i       6
wrapt   in      6
wreath  of      6
 by     and     8
 by     and     6
 do     you     7
 it     was     6
 it     was     6
 pretty soon    12
‘ah!’   said    10
‘behave well    6
‘it     is      6
‘my     mother  7
‘of     course  6
‘off    with    8
‘we     will    6
‘well,’ said    9
‘whither        away    7
‘yes,’  said    6
“all    right   8
“are    you     7
“are    you     8
“blame  it      8
“defects,”      such    10
“do     you     6
“do     you     6
“have   you     7
“he     is      6
“information    about   9
“it     is      6
“it     is      6
“miss   manette 8
“my     dear    7
“plain  vanilla 17
“project        gutenberg”      31
“queequeg,”     said    7
“she    is      6
“this   is      6
“upon   my      8
“well,” said    6
```

3. Design and execute a MapReduce program to produce the top 100 most frequent words in the book collection. You may need two rounds of Map and Reduce processors. Submit the following:

    a. Your commented code for the Mappers and Reducers.

    I have used output file of the first question as input to the below mapper.

Mapper Code: wc_3_1_map.py

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

def read_input(file,separator = '\t'):
   for line in file:
      # split the line into word and value
      yield line.strip().split(separator,1)

def main(separator='\t'):
   # input comes from STDIN (standard input)
   data = read_input(sys.stdin)
   for line in data:
      # collect word and value and reverse its role
      word = line[0]
      value= int(line[1])
      print '%d%s%s' % (value, separator, word)

if __name__ == "__main__":
   main()
```

Reducer Code: wc_3_1_red.py

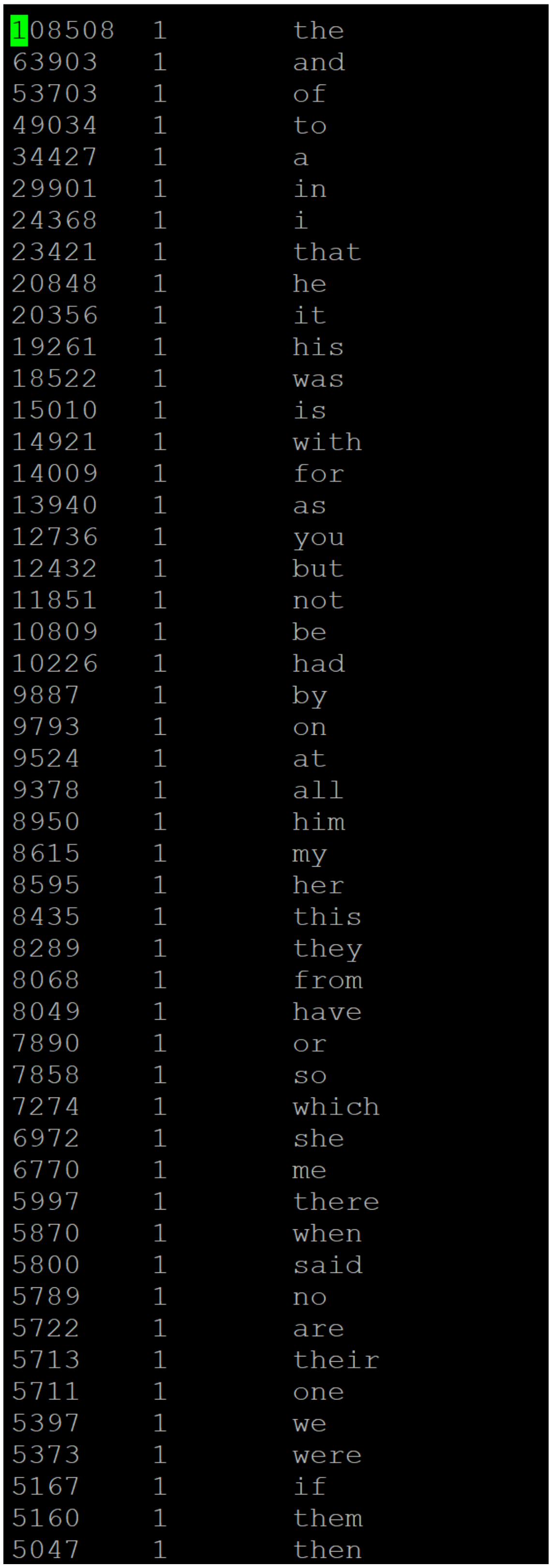
```
#!/usr/bin/env python
"""A more advanced Reducer, using Python iterators and generators."""

from itertools import groupby
from operator import itemgetter
import sys
import collections

def read_mapper_output(file, separator='\t'):
   for line in file:
      yield line.rstrip().split(separator, 1)

def main(separator='\t'):
   # input comes from STDIN (standard input)
   data = read_mapper_output(sys.stdin, separator=separator)
   # groupby groups multiple value-word pairs by value,
   # and creates an iterator that returns consecutive words and their group:
    #   current_word - string containing a word (the key)
    #   group - iterator yielding all ["&lt;current_word&gt;", "&lt;count&gt;"] items
    # combine all the words in this group by separator tab
    # and output the frequency-# of words - words group separated by tab
    for current_value, group in groupby(data, itemgetter(0)):
        try:
            total_word_group = "\t".join(word for current_value,word in group)
            print "%d%s%d%s%s" % (int(current_value), separator, \
            int(len(total_word_group.split(separator))),separator,total_word_group)
        except ValueError:
            # count was not a number, so silently discard this item
            pass

if __name__ == "__main__":
    main()
```

NOTE: I have used the output of the first question solution as input to this above mapper and reducer. And also, Hadoop inbuilt key comparator options are set to sort decreasing order of the frequency.

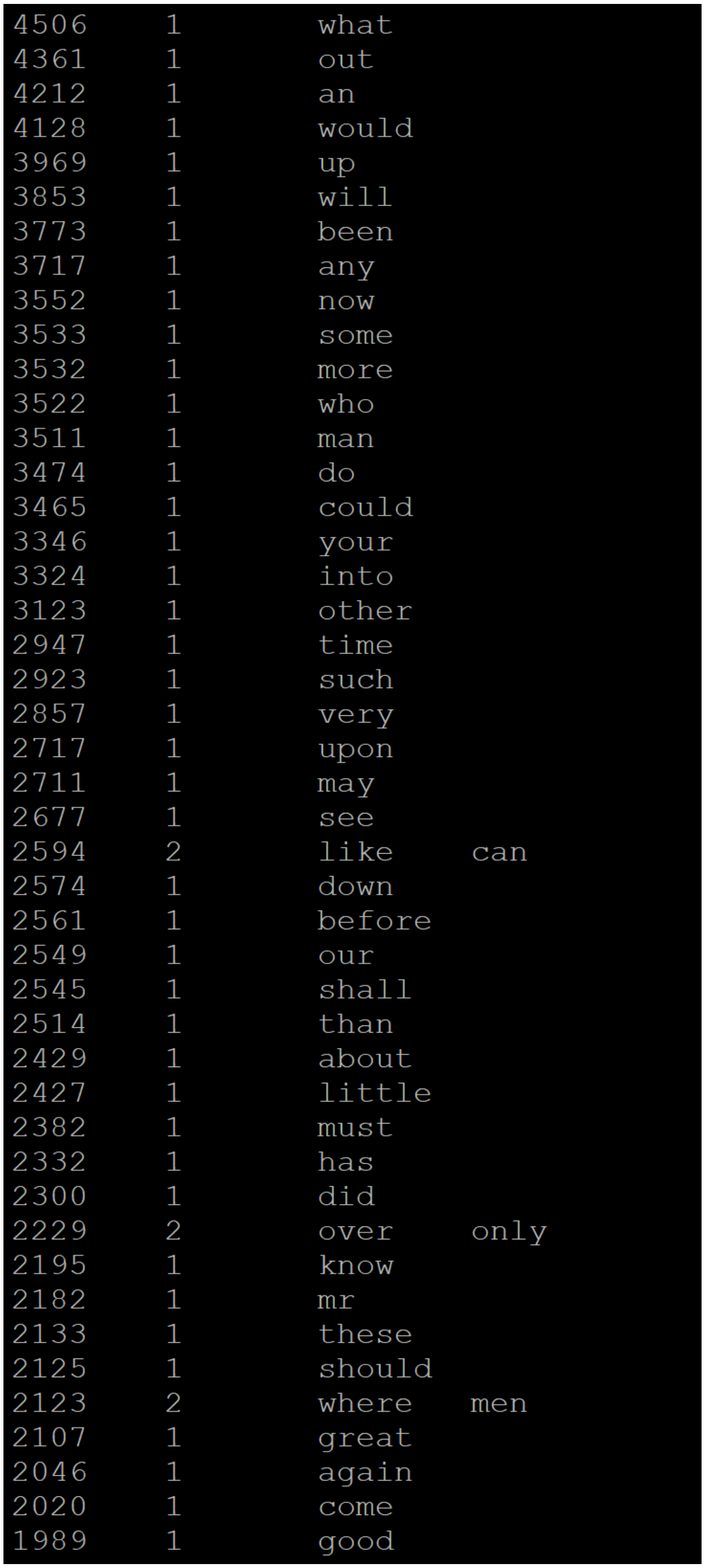
Command used to execute the above code in Hadoop:

```
$ hadoop jar /usr/hdp/3.0.0.0-1634/hadoop-mapreduce/hadoop-streaming.jar -D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyFieldBasedComparator -D mapred.text.key.comparator.options=-k1,1nr -file wc_3_1_map.py -mapper wc_3_1_map.py -file wc_3_1_red.py -reducer wc_3_1_red.py -input BDA/PPA_1/results/wc_1_1_cres/part-00000 -output BDA/PPA_1/results/wc_3_1_cres
```

b. The list of 100 most frequent words and their frequencies in the book collection.



```
108508  1       the
63903   1       and
53703   1       of
49034   1       to
34427   1       a
29901   1       in
24368   1       i
23421   1       that
20848   1       he
20356   1       it
19261   1       his
18522   1       was
15010   1       is
14921   1       with
14009   1       for
13940   1       as
12736   1       you
12432   1       but
11851   1       not
10809   1       be
10226   1       had
9887    1       by
9793    1       on
9524    1       at
9378    1       all
8950    1       him
8615    1       my
8595    1       her
8435    1       this
8289    1       they
8068    1       from
8049    1       have
7890    1       or
7858    1       so
7274    1       which
6972    1       she
6770    1       me
5997    1       there
5870    1       when
5800    1       said
5789    1       no
5722    1       are
5713    1       their
5711    1       one
5397    1       we
5373    1       were
5167    1       if
5160    1       them
5047    1       then
```



```
4506    1       what
4361    1       out
4212    1       an
4128    1       would
3969    1       up
3853    1       will
3773    1       been
3717    1       any
3552    1       now
3533    1       some
3532    1       more
3522    1       who
3511    1       man
3474    1       do
3465    1       could
3346    1       your
3324    1       into
3123    1       other
2947    1       time
2923    1       such
2857    1       very
2717    1       upon
2711    1       may
2677    1       see
2594    2       like    can
2574    1       down
2561    1       before
2549    1       our
2545    1       shall
2514    1       than
2429    1       about
2427    1       little
2382    1       must
2332    1       has
2300    1       did
2229    2       over    only
2195    1       know
2182    1       mr
2133    1       these
2125    1       should
2123    2       where   men
2107    1       great
2046    1       again
2020    1       come
1989    1       good
```

The above output represents the frequency-tab-#of words of that frequency-tab-words list. These are the top 100 words.

By:
Varun Raj Rayabarapu

Appendix:

This appendix consists solutions to the above questions when the raw data is handled.

1. a. **Code:**

Mapper Code: wc_1_1_map.py

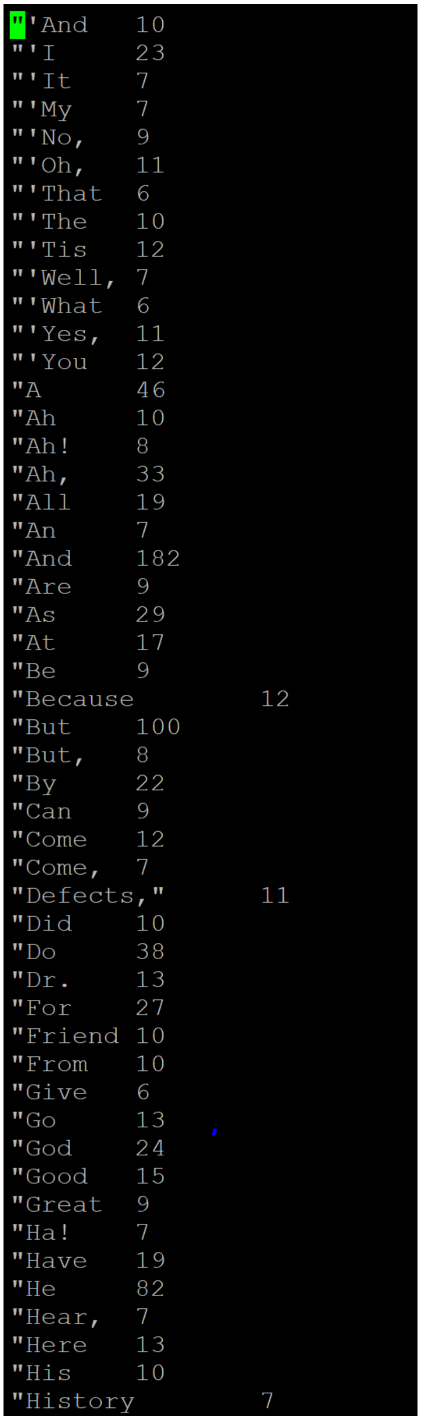
```
#!/usr/bin/env python
import sys
def read_input(file):
    for line in file:
        # split the line into words
        yield line.strip().split()
def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_input(sys.stdin)
    for words in data:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        # tab-delimited; the trivial word count is 1
        for word in words:
            print '%s%s%d' % (word, separator, 1)
if __name__ == "__main__":
    main()
```

Reducer Code: wc_1_1_red.py

```
#!/usr/bin/env python
from itertools import groupby
from operator import itemgetter
import sys
def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 1)
def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_mapper_output(sys.stdin, separator=separator)
    # groupby groups multiple word-count pairs by word,
    # and creates an iterator that returns consecutive keys and their group:
    #   current_word - string containing a word (the key)
    #   group - iterator yielding all ["&lt;current_word&gt;", "&lt;count&gt;"] items
    for current_word, group in groupby(data, itemgetter(0)):
        try:
            total_count = sum(int(count) for current_word, count in group)
            # output only those words with total_count>5
            if total_count>5:
```

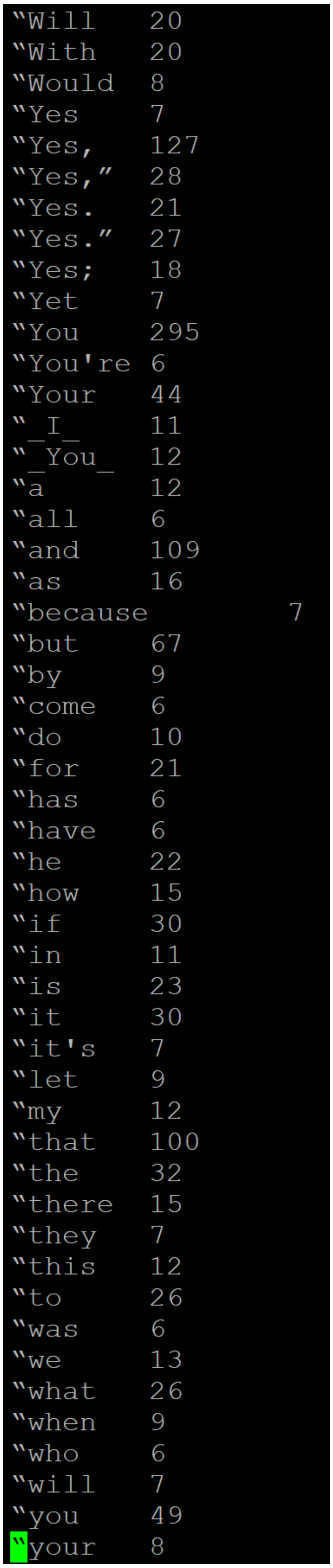
```
        print "%s%s%d" % (current_word, separator, total_count)
    except ValueError:
      # count was not a number, so silently discard this item
      pass
if __name__ == "__main__":
  main()
```

1.b. First 50 words of output file:



```
"'And   10
"'I     23
"'It    7
"'My    7
"'No,   9
"'Oh,   11
"'That  6
"'The   10
"'Tis   12
"'Well, 7
"'What  6
"'Yes,  11
"'You   12
"A      46
"Ah     10
"Ah!    8
"Ah,    33
"All    19
"An     7
"And    182
"Are    9
"As     29
"At     17
"Be     9
"Because        12
"But    100
"But,   8
"By     22
"Can    9
"Come   12
"Come,  7
"Defects,"      11
"Did    10
"Do     38
"Dr.    13
"For    27
"Friend 10
"From   10
"Give   6
"Go     13
"God    24
"Good   15
"Great  9
"Ha!    7
"Have   19
"He     82
"Hear,  7
"Here   13
"His    10
"History        7
```

1.c. Last 50 words of output file:



```
“Will    20
“With    20
“Would   8
“Yes     7
“Yes,    127
“Yes,”   28
“Yes.    21
“Yes.”   27
“Yes;    18
“Yet     7
“You     295
“You're  6
“Your    44
“_I_     11
“_You_   12
“a       12
“all     6
“and     109
“as      16
“because         7
“but     67
“by      9
“come    6
“do      10
“for     21
“has     6
“have    6
“he      22
“how     15
“if      30
“in      11
“is      23
“it      30
“it's    7
“let     9
“my      12
“that    100
“the     32
“there   15
“they    7
“this    12
“to      26
“was     6
“we      13
“what    26
“when    9
“who     6
“will    7
“you     49
“your    8
```

By:
Varun Raj Rayabarapu

2.a. **Code:**

Mapper Code: wc_2_1_map.py

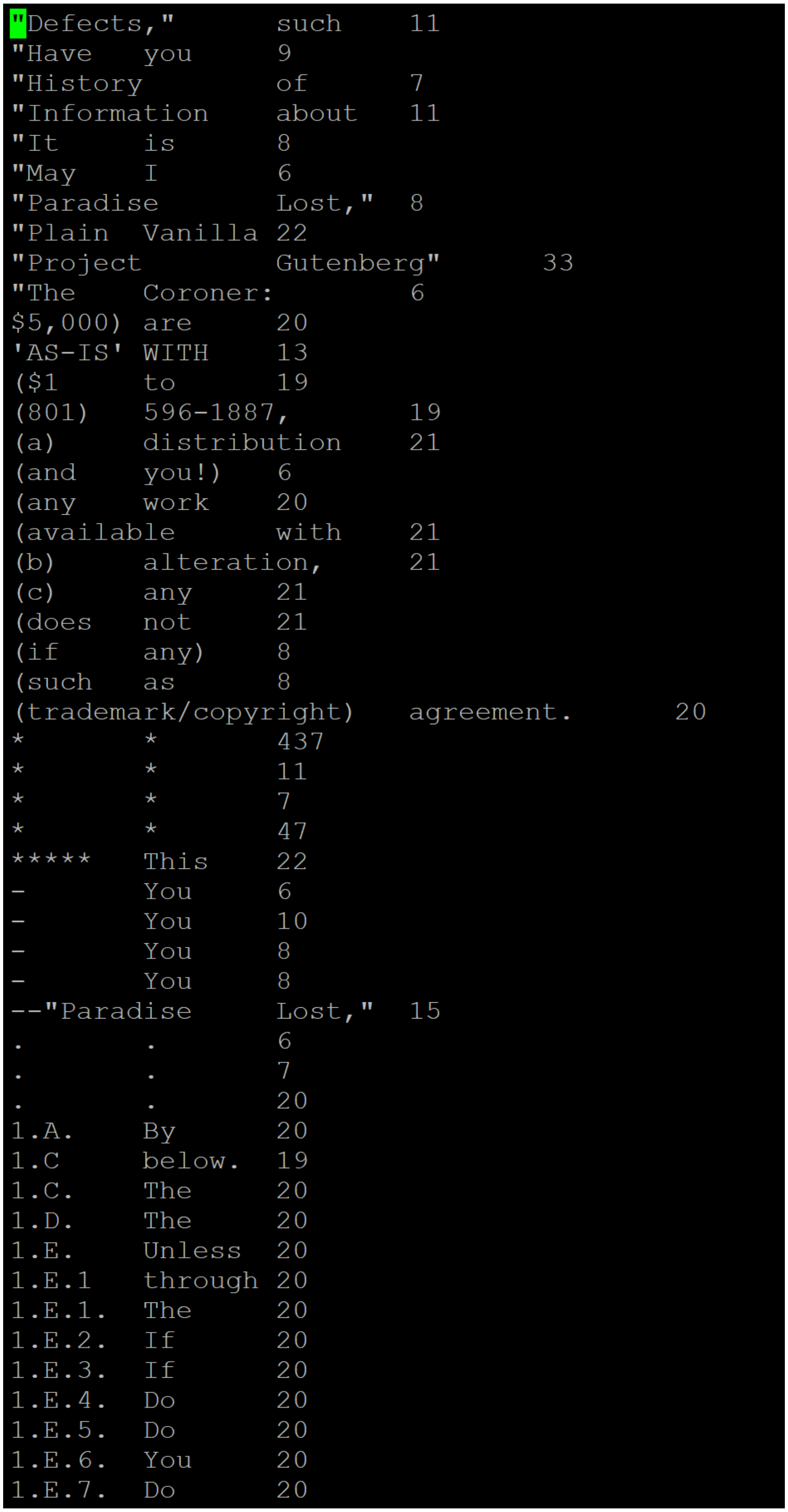
```
#!/usr/bin/env python
import sys

def read_input(file):
    for line in file:
        # split the line into words
        yield line.strip().split()

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_input(sys.stdin)
    lastword = ""
    for words in data:
        # write each word tab-delimited with the next word and again
        # tab-delimited with trivial count 1 to STDOUT (standard output)
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        for i in range(len(words)-2):
            print '%s%s%s%s%d' % (words[i], separator, words[i+1],separator,1)
        # This is code below deals with producing the bigrams with the word
        # from current line last word and next line first word
        if lastword:
            print '%s%s%s%s%d' % (lastword, separator, words[0],separator,1)
            lastword=words[-1]

if __name__ == "__main__":
    main()
```

Reducer Code: wc_2_1_red.py

```
#!/usr/bin/env python
from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file, separator='\t'):
    for line in file:
        yield line.rstrip().split(separator, 2)

def main(separator='\t'):
    # input comes from STDIN (standard input)
    data = read_mapper_output(sys.stdin, separator=separator)
```

```
              # groupby groups multiple word-word-count pairs by firstword,
              # and creates an iterator that returns consecutive keys and their group:
              # current_firstword - string containing a word (the key)
              #   group - iterator yielding all ["&lt;current_firstword&gt;",
              # "&lt;current_secondword&gt;", "&lt;count&gt;"] items
    for current_firstword, group_1 in groupby(data, itemgetter(0)):
        try:
            for current_secondword, group_2 in groupby(group_1,itemgetter(1)):
              # groupby groups multiple word-word-count pairs by secondword,
              # and creates an iterator that returns consecutive keys and their group:
              # current_secondword - string containing a word (the key)
              #   group - iterator yielding all ["&lt;current_firstword&gt;",
              # "&lt;current_secondword&gt;", "&lt;count&gt;"] items

                try:
                    total_count = sum(int(count) for firstword,secondword, count in group_2)
                    if total_count>5:
                        print "%s%s%s%s%d" % (current_firstword, separator,
current_secondword,separator,total_count)
                except ValueError:
                    pass
        except ValueError:
            # count was not a number, so silently discard this item
            pass

if __name__ == "__main__":
    main()
```

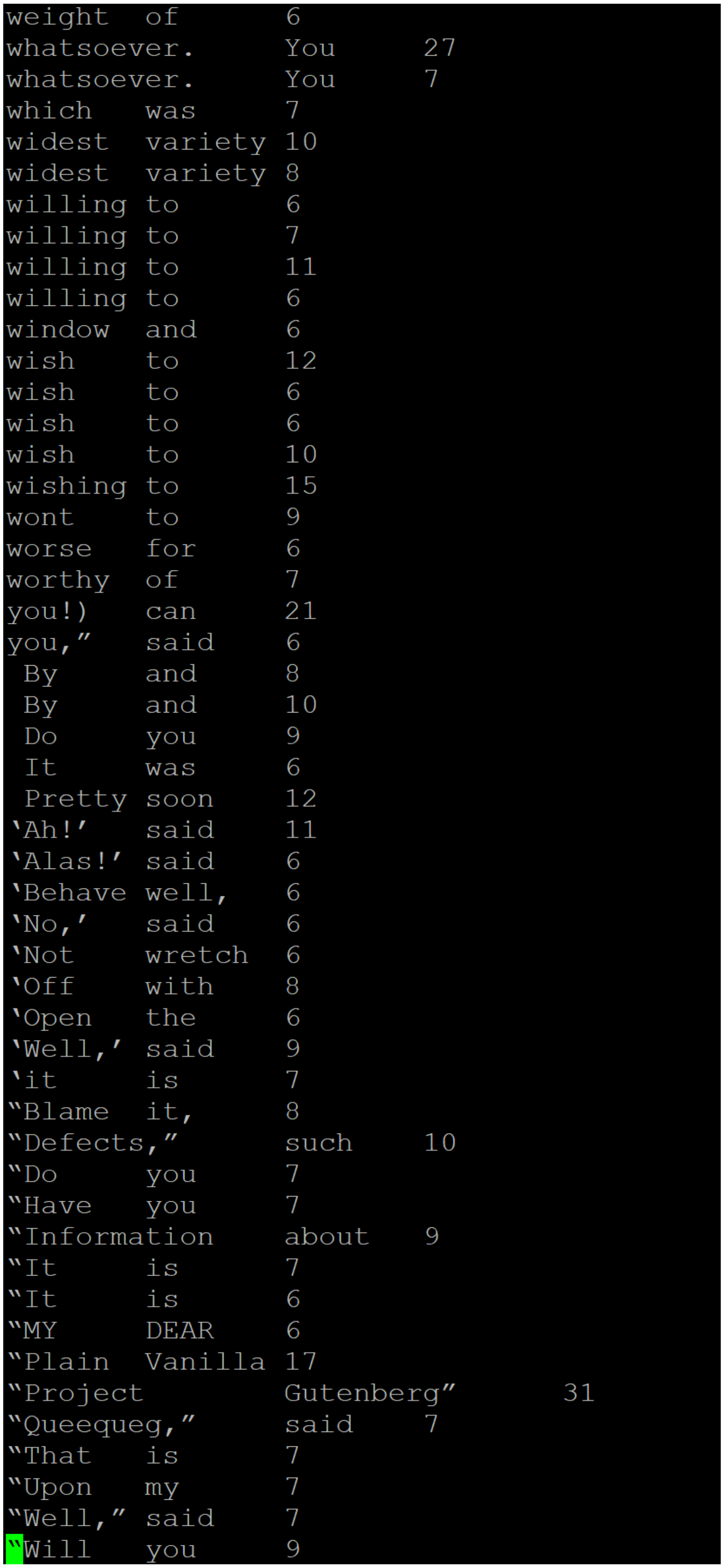
2.b. First 50 results of output file:



```
"Defects,"      such    11
"Have   you     9
"History        of      7
"Information    about   11
"It     is      8
"May    I       6
"Paradise       Lost,"  8
"Plain  Vanilla 22
"Project        Gutenberg"      33
"The    Coroner:        6
$5,000) are     20
'AS-IS' WITH    13
($1     to      19
(801)   596-1887,       19
(a)     distribution    21
(and    you!)   6
(any    work    20
(available      with    21
(b)     alteration,     21
(c)     any     21
(does   not     21
(if     any)    8
(such   as      8
(trademark/copyright)   agreement.      20
*       *       437
*       *       11
*       *       7
*       *       47
*****   This    22
-       You     6
-       You     10
-       You     8
-       You     8
--"Paradise     Lost,"  15
.       .       6
.       .       7
.       .       20
1.A.    By      20
1.C     below.  19
1.C.    The     20
1.D.    The     20
1.E.    Unless  20
1.E.1   through 20
1.E.1.  The     20
1.E.2.  If      20
1.E.3.  If      20
1.E.4.  Do      20
1.E.5.  Do      20
1.E.6.  You     20
1.E.7.  Do      20
```

By:
Varun Raj Rayabarapu

2.c. Last 50 results of output file:



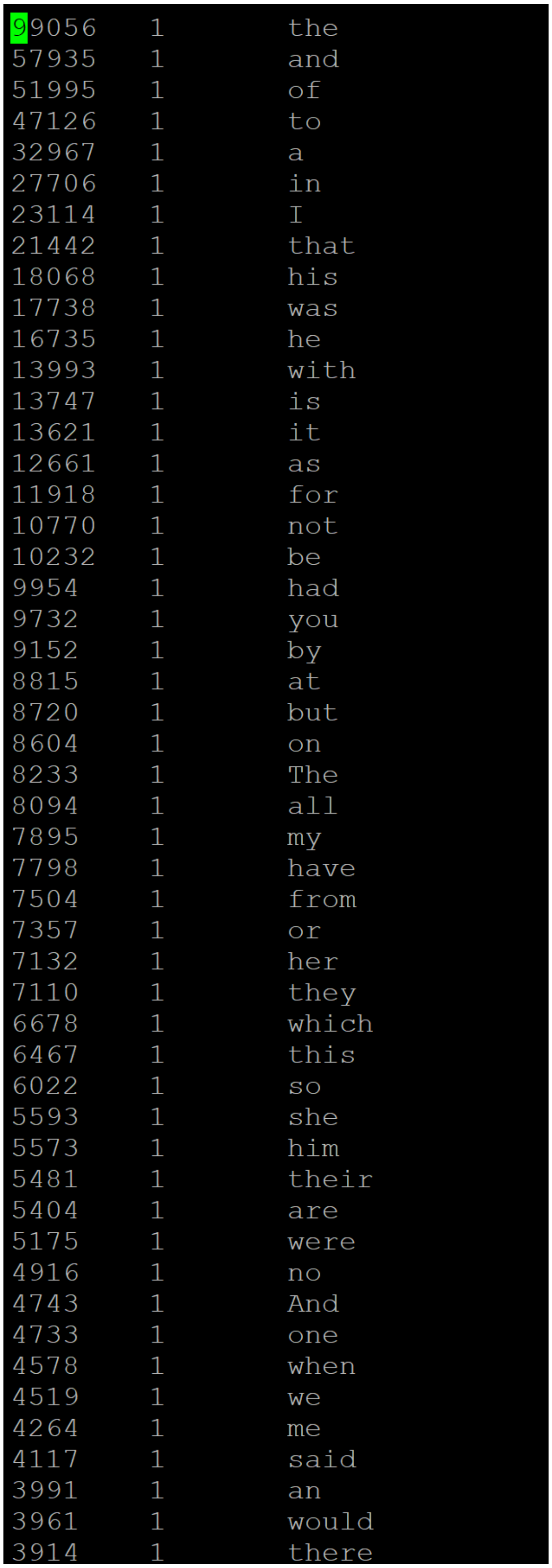
```
weight  of      6
whatsoever.     You     27
whatsoever.     You     7
which   was     7
widest  variety 10
widest  variety 8
willing to      6
willing to      7
willing to      11
willing to      6
window  and     6
wish    to      12
wish    to      6
wish    to      6
wish    to      10
wishing to      15
wont    to      9
worse   for     6
worthy  of      7
you!)   can     21
you,”   said    6
 By     and     8
 By     and     10
 Do     you     9
 It     was     6
 Pretty soon    12
‘Ah!’   said    11
‘Alas!’ said    6
‘Behave well,   6
‘No,’   said    6
‘Not    wretch  6
‘Off    with    8
‘Open   the     6
‘Well,’ said    9
‘it     is      7
“Blame  it,     8
“Defects,”      such    10
“Do     you     7
“Have   you     7
“Information    about   9
“It     is      7
“It     is      6
“MY     DEAR    6
“Plain  Vanilla 17
“Project        Gutenberg”      31
“Queequeg,”     said    7
“That   is      7
“Upon   my      7
“Well,” said    7
“Will   you     9
```

By:
Varun Raj Rayabarapu

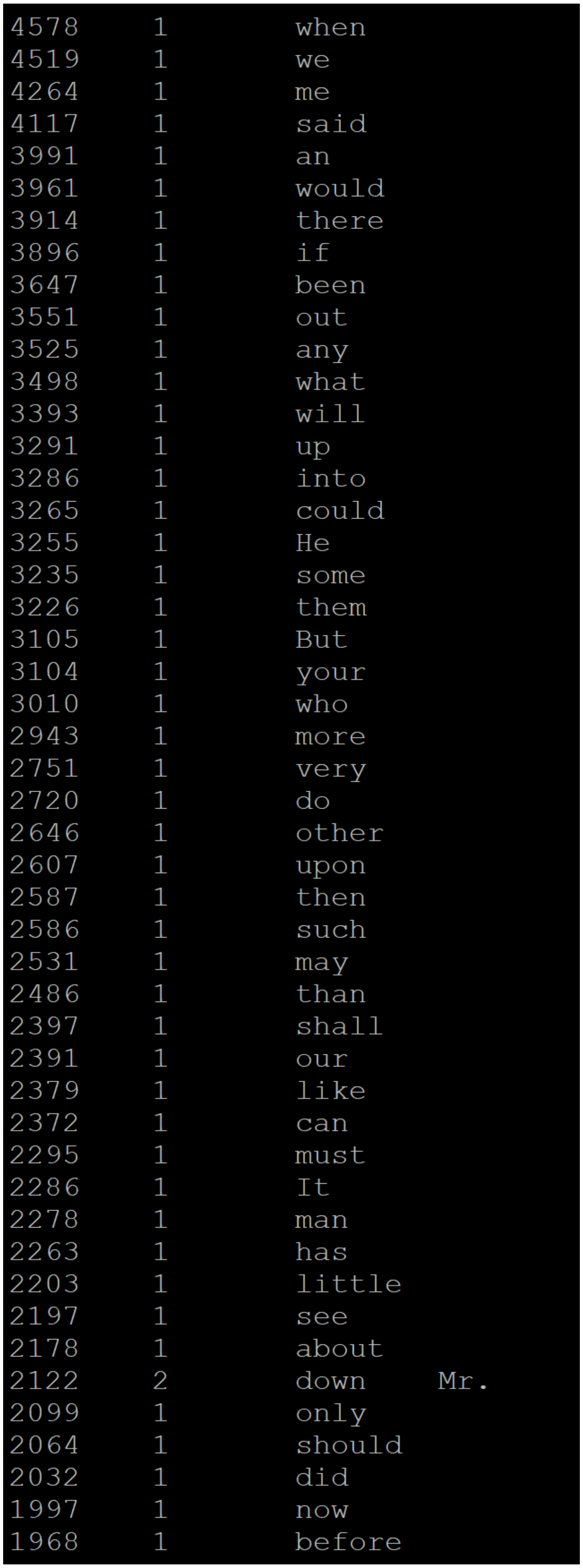
3.a **Code:**

The code is same as the one for that is discussed in 3.a. with cleaned data.

3.b. Top 100 Results:



```
99056   1       the
57935   1       and
51995   1       of
47126   1       to
32967   1       a
27706   1       in
23114   1       I
21442   1       that
18068   1       his
17738   1       was
16735   1       he
13993   1       with
13747   1       is
13621   1       it
12661   1       as
11918   1       for
10770   1       not
10232   1       be
9954    1       had
9732    1       you
9152    1       by
8815    1       at
8720    1       but
8604    1       on
8233    1       The
8094    1       all
7895    1       my
7798    1       have
7504    1       from
7357    1       or
7132    1       her
7110    1       they
6678    1       which
6467    1       this
6022    1       so
5593    1       she
5573    1       him
5481    1       their
5404    1       are
5175    1       were
4916    1       no
4743    1       And
4733    1       one
4578    1       when
4519    1       we
4264    1       me
4117    1       said
3991    1       an
3961    1       would
3914    1       there
```



```
4578    1       when
4519    1       we
4264    1       me
4117    1       said
3991    1       an
3961    1       would
3914    1       there
3896    1       if
3647    1       been
3551    1       out
3525    1       any
3498    1       what
3393    1       will
3291    1       up
3286    1       into
3265    1       could
3255    1       He
3235    1       some
3226    1       them
3105    1       But
3104    1       your
3010    1       who
2943    1       more
2751    1       very
2720    1       do
2646    1       other
2607    1       upon
2587    1       then
2586    1       such
2531    1       may
2486    1       than
2397    1       shall
2391    1       our
2379    1       like
2372    1       can
2295    1       must
2286    1       It
2278    1       man
2263    1       has
2203    1       little
2197    1       see
2178    1       about
2122    2       down    Mr.
2099    1       only
2064    1       should
2032    1       did
1997    1       now
1968    1       before
```