CSCI 3412 – Algorithms
Spring 2013
Problem Set 2 – Programming

An n-gram is a contiguous sequence of *n* words from a given sequence of text. In this programming assignment you will extract n-grams from tokenized text files.

All programming assignments require that a written report as well as your source code file(s) be turned in. For multi-part programming assignment, such as this one, you may have one written report covering all of the parts and may even have combined source code file(s) as long as I can easily tell what code goes with each part of the problem – some code may go with multiple parts.

The written report should include, for each part of the programming assignment covered, a description of the problem (basically the information from the assignment itself), an overview of your solution to the problem, a description of the relevant algorithm(s) – including an analysis of the algorithm(s), the results, and an analysis of the results.

The programs may be written in any relevant language. The languages C, C++, Java, Ada, Python, and Racket are acceptable. For other languages, please discuss your language choice with me ahead of time. Generally, you can – are encouraged to – use appropriate libraries for regular expressions, standard data structures, etc.

Please review the Collaboration Policy in the course syllabus. Remember that collaboration is encouraged, but that the work turned in must be your own. Also, please remember to include the names of others you collaborated with in your written report and be sure to cite any sources used.

This programming assignment builds on the n-gram extraction functionality that you developed for programming assignment 1.

This programming assignment has two parts.

Part 1

In text analysis, a number of English words are commonly ignored because they don't typically convey any useful information. One such list – in comma separated value (CSV) format – can be found at http://www.textfixer.com/resources/common-english-words.txt. This file is a single line of text with the words separated by commas.

```
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,g
ot,had,has,have,he,her,hers,him,his,how,however,i,if,in,into,is,it,its,just,least,l
et,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,th
em,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
```

This list is based on modern English and needs to be changed for our Shakespeare input. I used the 1-grams to find the equivalent common words in the Shakespeare text. Here is my list.

`a,able,about,across,after,all,almost,also,am,among,an,and,any,are,`**`art`**`,as,at,`**`ay`**`,be,because,been,but,by,can,cannot,could,dear,did,do,does,`**`dost`**`,`**`doth`**`,either,else,ever,every,for,from,get,got,had,has,`**`hath`**`,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,`**`o`**`,of,off,often,on,only,or,other,our,own,rather,said,say,says,she,should,since,so,some,than,that,the,`**`thee`**`,their,them,then,there,these,they,`**`thine`**`,this,`**`thou`**`,`**`thy`**`,tis,to,too,twas,`**`upon`**`,us,wants,was,we,were,what,when,where,which,while,who,whom,why,will,with,would,yet,you,your`

The bold words were added. You may add additional words if you like.

The first part of this assignment is to modify your n-gram extractor to ignore n-grams where any of the words are in the stop word list. We don't want to ignore the stop words when we are tokenizing the input because we don't want something like “once upon a time” to be read as “once time”. Instead, we want to find the n-grams exactly as in programming assignment, but only store them if they don't contain any stop words.

My n-gram program modified to ignore the stop words above gives the following results with *N* = 4 for the shakespeare.txt file:

```
There are 26852 1-grams.
The 10 most common ones are:
  ("shall") [3603]
  ("good") [2817]
  ("now") [2798]
  ("lord") [2717]
  ("come") [2565]
  ("sir") [2541]
  ("well") [2519]
  ("more") [2291]
  ("here") [2149]
  ("ill") [2002]

There are 96636 2-grams.
The 10 most common ones are:
  ("c" "sar") [346]
  ("here" "comes") [153]
  ("good" "lord") [153]
  ("sir" "john") [152]
  ("come" "come") [116]
  ("good" "morrow") [113]
  ("come" "hither") [107]
  ("mine" "eyes") [104]
  ("mine" "honour") [79]
  ("c" "sars") [73]

There are 35876 3-grams.
The 10 most common ones are:
  ("sir" "john" "falstaff") [19]
  ("three" "thousand" "ducats") [13]
  ("one" "word" "more") [12]
  ("mistress" "anne" "page") [11]
  ("fie" "fie" "fie") [11]
  ("julius" "c" "sar") [11]
  ("fare" "ye" "well") [10]
  ("ha" "ha" "ha") [10]
  ("good" "old" "man") [9]
  ("here" "comes" "one") [8]

There are 11377 4-grams.
The 10 most common ones are:
  ("kill" "kill" "kill" "kill") [5]
  ("come" "hither" "come" "hither") [5]
  ("sing" "willow" "willow" "willow") [4]
  ("william" "de" "la" "pole") [4]
  ("well" "thats" "set" "down") [3]
  ("good" "time" "here" "comes") [3]
  ("go" "play" "boy" "play") [2]
  ("good" "captain" "blunt" "bear") [2]
  ("being" "dead" "many" "years") [2]
  ("john" "de" "la" "car") [2]
```

Part 2

The 2-grams can be view as a graph. In particular, we will use each of the (two) words in the 2-gram as nodes in the graph with an edge between them. We will look at visualizing this graph using two different visualization programs. By using external programs, we don't have to worry about programming languages or application program interfaces (API). Instead, we will produce text files that describe the graph and these will be be processed by the graph visualization programs.

We will look at two different graph visualization programs: Graphviz and Gephi. Graphviz simply renders graphs – although it is very flexible. Gephi is an interactive program for analyzing graphs. Fortunately, both of these can use the same text file format.

Graphviz

Graphviz is open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. It has important applications in networking, bioinformatics, software engineering, database and web design, machine learning, and in visual interfaces for other technical domains.

You can download Graphviz from http://www.graphviz.org/.

We will be using the dot format for describing graphs. Here is a small example for the dot User's Manual, which is installed with Graphviz or available online.

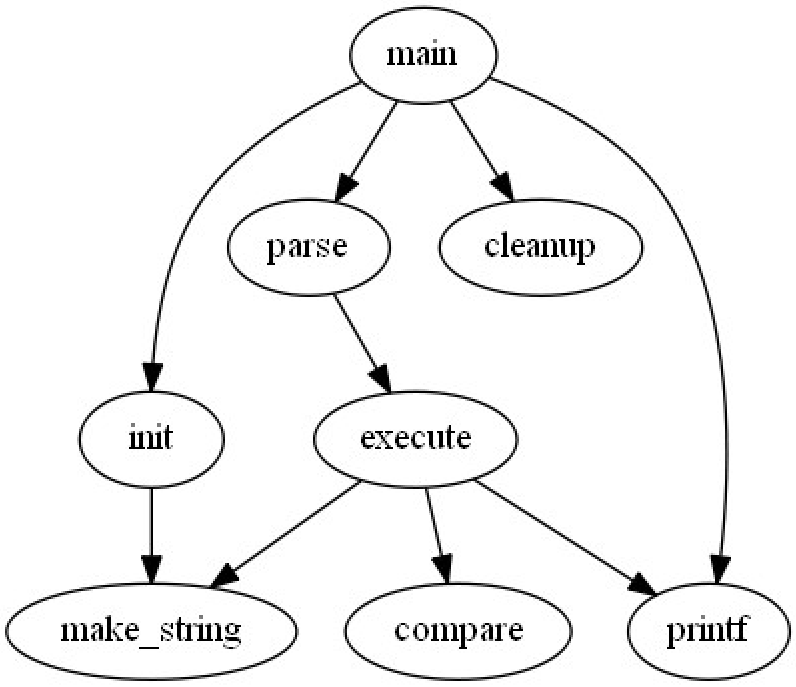
```
digraph G {
      main -> parse -> execute;
      main -> init;
      main -> cleanup;
      execute -> make_string;
      execute -> printf
      init -> make_string;
      main -> printf;
      execute -> compare;
}
```

This defines a directed graph, G, with eight nodes – main, parse, cleanup, init, execute, make_string, compare, and printf – and nine (directed) edges between them.

This can be rendered as a png file using the following command:

```
dot -Tpng -O small-graph.dot
```

The resulting graph is shown below.



We will generate a file in dot format for the 2-grams. Here is the first and last ten lines from my output for the program.

```
digraph words {
    "go" -> "home" [weight=16];
    "mine" -> "enemy" [weight=19];
    "good" -> "old" [weight=21];
    "come" -> "again" [weight=25];
    "fathers" -> "house" [weight=15];
    "old" -> "sir" [weight=14];
    "good" -> "sir" [weight=73];
    "well" -> "go" [weight=17];
    "gentle" -> "lady" [weight=12];
    . . .
    "shalt" -> "see" [weight=12];
    "humbly" -> "thank" [weight=17];
    "go" -> "back" [weight=13];
    "wast" -> "born" [weight=13];
    "good" -> "night" [weight=46];
    "farewell" -> "good" [weight=14];
    "here" -> "take" [weight=14];
    "much" -> "better" [weight=12];
    "give" -> "up" [weight=13];
}
```

The weight parameter is the count for the corresponding 2-gram. In order to keep the graph reasonable size, I only write a record for 2-grams whose count is greater than 10. This gives 425 records (edges) in the graph.

This size graph does not render will using the dot renderer. However, the sfdp renderer is designed to handle larger graphs and works fine on this size graph. The graph can be rendered with:

```
sfdp -Tpdf -O n-grams.dot
```

This will create a pdf file with the graph rendering. It is a rather cluttered graph, but you can play with additional parameters to make it better. Try the following examples:

```
sfdp -Goverlap=scale -Tpdf -O n-grams.dot
sfdp -Goverlap=prism -Tpdf -O n-grams.dot
```

You can also play with changing the edges to be displayed with curved lines – i.e., splines.

```
sfdp -Goverlap=prism -Gsplines=true -Tpdf -O n-grams.dot
```

The last may may take some time on a smaller computer.

Gephi

Gephi is an interactive visualization and exploration platform for all kinds of networks and complex systems, dynamic and hierarchical graphs.

Gephi can be downloaded from https://gephi.org/.

It can read the same dot file format as Graphviz. The main thing is just to download it and play with it using your graphs.

References:

Gephi, https://gephi.org/.

Graphiz – Graph Visualization Software, http://www.graphviz.org/.

N-gram. (2013, January 10). In Wikipedia, The Free Encyclopedia. Retrieved 15:08, January 30, 2013, from http://en.wikipedia.org/w/index.php?title=N-gram&oldid=532444843.

Norvig, Peter, Natural Language Corpus Data: Beautiful Data, http://norvig.com/ngrams/.