Programming Project Evaluation Form

Student Name:	**Project:**	**Date:**
Correctness Criteria: • Reads normal input data • Handles incorrect input data • Calculates correct results • Outputs results properly • Prints appropriate error messages		**Score:_____/ 50**
Design Criteria: • Problem decomposition • Choice of data structures • Choice of algorithms • Program efficiency (space/time)		**Score:_____/ 20**
Documentation Criteria: • Comments for classes and methods • Comments describing algorithms • Comments describing data structures • Description of program design • Description of test results • Description of known problems		**Score:_____/ 10**
Style Criteria: • Method and variable names • Program indenting • Use of white space • Ordering of methods • Easy to read code		**Score:_____/ 10**
Testing Criteria: • Normal input data • Incorrect input data • Special cases for data structures • Special cases for algorithms		**Score:_____/ 10**
Grader Comments: **Total:_____/ 100**		

Installation

Programming language: Python

Source code location: /home/sbillah/nlp/

Documentation:

```
sbillah@turing:~/nlp$ ./NLPEngine.py -h
usage: NLPEngine.py [-h] [-o DST_DIR] [-s SRC_DIR] [-nlp BIGRAMS]


This is Syed's NLP program for HW01
optional arguments:
  -h, --help            show this help message and exit
  -o DST_DIR, --output DST_DIR
                        The directory where output goes
  -s SRC_DIR, --source SRC_DIR
                        The directory where raw files reside
  -nlp BIGRAMS, --count_bigrams BIGRAMS
                    count the bigrams in src directory and write to
                    the dst folder in descending order
```

Here is a complete command line example:

```
sbillah@turing:~/nlp$ ./NLPEngine.py -s
/home/sgauch/public_html/5013IR/files/ -o parsed/ -nlp bigram_counts
Done!
```

Algorithm

I use an in-memory Hash-table to count all bigrams. The pseudo code is given bellow:

```
initialize hash-table ht<(tuple), int>
foreach file f in input_directory:
      pain_text = html_parser(f.read())
      tokens = tokenizer(plain_text)

      i = 0
      foreach token t in tokens:
            if i>0:
                  ht[(t[i-1],t[i])] += 1
            i++
sort ht
write ht to file
```

Time Complexity:
N = num_files
M= avg num_of_words_per_file
Bigram generation complexity: O(N*M)
Hash-table sorting complexity: O(N*M*log(N*M))
Total complexity: O(N*M) + O(N*M*log(N*M)) = **O(N*M*log(N*M))**

Parser Configuration

Here is the configuration of my html parser and tokenizer:

str_src_dir	/home/sgauch/public_html/5013IR/files/
str_dst_dir	parsed/
str_doc_id_file_name	bigram.txt
min_token_freq	3
max_token_freq	1000
min_token_len	3
max_token_len	12
str_stop_list	Stoplist from this link: http://www.csce.uark.edu/~sgauch/5013IR/S12/index.html

Runtime & Memory usage

Input size (# files)	Run time (sec)	Memory size (MB)	Total bigrams
100	5.60	110	55,013
200	19.21	210	116,697
300	30.31	277	166,179
505	54.64	440	272,646

Top 50 bigrams

risks	jul	607
net	alter	344
alter	dynip	340
health	care	215
paper	title	208
com	interramp	204
net	sunbelt	189
edu	psu	177
net	mci	152
edu	nodak	144
edu	uiuc	142
critical	analysis	142
mass	media	141
edu	umich	137
net	idt	133
edu	umn	132
mil	navy	130
political	science	129
rights	reserved	127
edu	arizona	120
human	rights	117
edu	indiana	115
social	security	111
hogy	nem	111
edu	utexas	111

nemzet	magyar	110
los	angeles	109
horn	gyula	109
send	comments	108
mci	campus	108
black	studies	107
world	war	106
home	page	106
book	report	106
term	papers	105
http	www	105
written	price	104
urban	studies	104
termpaper	com	104
term	paper	104
subject	index	104
sports	recreation	104
specific	paper	104
paper	written	104
paper	click	104
description	paper	104
copyright	asm	104
comments	termpaper	104
comments	comments	104
cold	surges	104

Bottom 20 bigrams

abacs	kiskun	1
ababa	response	1
aau	zoo	1
aau	psy	1
aau	hum	1
aas	nearly	1
aarp	national	1
aarp	american	1
aaron	word	1
aaron	netland	1

aaron	moshiashwili	1
aaron	jon	1
aaron	happened	1
aaron	comparison	1
aalen	image	1
aaemassago	sem	1
aaeliberalis	demokraciato	1
aachen	rad	1
aachen	oph	1
aaa	passed	1

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

'''
Created on Feb 23, 2012

@author: Masum
'''

from myparser import BiGram
from myparser import src_dir, dst_dir
from mycollection import argparse

class NLPEngine():
    config= {}

    def __init__(self):
        self.read_config()

    def read_config(self, name="config.txt"):

    def build_bigram_index(self):
        _ = BiGram(self.config)

#import argparse
if __name__=='__main__':
    args = argparse.ArgumentParser(description="This is Syed's NLP program")
    args.add_argument("-o", "--output", dest="dst_dir",
                      help="The directory where output goes", default="")
    args.add_argument("-s", "--source", dest="src_dir",
                      help="The directory where raw files reside", default="")

    args.add_argument("-nlp", "--count_bigrams", dest="bigrams",
                      help="Re-index of all files - and overwrite existing dict and post files,
  i.e. -i all", default="")


    args = args.parse_args()
    src_dir = args.src_dir
    dst_dir = args.dst_dir

    if not args.bigrams:
        print 'Invalid or No arguments. Try with -h for help!!'
    else:
        nlp = NLPEngine()
        if args.bigrams: nlp.build_bigram_index()
        print 'Done!'

#    nlp = SearchEngine()
#    nlp.search_query('.8 susan .1 uark .1 edu', True )
#    print 'Done!'


```

```
'''
Created on Mar 11, 2012

@author: Masum
'''

import re
from collections import defaultdict

#============ global regex ==============================
is_word = re.compile("[a-zA-Z_]+$")
delim = re.compile(" |\t|\.|-|:|\@|\\|/|,|;|\"|!|\*]+")

#================path ==============================
dst_dir = ''
src_dir = ''

#============ global hash variables ====================
doc_id= defaultdict(int)
term_count= defaultdict(int)

def parse(line, config):
    line =line.strip()
    if not line: return []

    tokens= []
    for word in delim.split(line):
        word = word.strip("~`$%^&()[]|<>=+-_/").lower()
        if word in config['str_stop_list']: continue
        if config['min_token_len']<= len(word) <= config['max_token_len'] and
is_word.match(word):
            tokens.append(word)
    return tokens
```

```
'''
Created on Feb 4, 2013

@author: Masum
'''
import os

from BiGramHTMLParser import BiGramHTMLParser
from collections import defaultdict

class BiGram():
    def __init__(self, config):
        self.config = config
        self.ht = defaultdict(int)
        self.htmlparser = BiGramHTMLParser(self.config, self.ht)
        self.start_batch_processing()
        self.write_file_map()

    def start_batch_processing(self):
        file_id=0

        for in_file in os.listdir(self.config['str_src_dir']):
            #if in_file not in ['medium.html','simple.html']: continue #for testing
            with open(self.config['str_src_dir']+ in_file, 'r') as f:
                self.htmlparser.feed(f.read(),file_id)
                file_id += 1

    def write_file_map(self):
        #writing bigram file to a file named under document id
        with open(self.config['str_dst_dir']+  self.config['str_doc_id_file_name'],'wb+') as f:
            for words, count in sorted(self.ht.iteritems(), key=lambda (k,v): (v,k), reverse=
  True):
                f.write(words[0]+" "+words[1]+" "+str(count)+"\n")

```

```
'''
Created on Mar 11, 2012

@author: Masum
'''
import sys
from HTMLParser import HTMLParser
from shared import parse


class BiGramHTMLParser(HTMLParser):
    text, N, newline ='', 0,{'br':'\n','BR':'\n'}

    def __init__(self, config, ht):
        HTMLParser.__init__(self)
        self.config = config
        self.ht = ht


    def handle_data(self, raw):
        self.text = self.text+ raw+ self.newline.get(self.lasttag,'')

    #format is (token =>value, here val)
    def feed(self,data, did):
        #extracting html text
        try:
            HTMLParser.feed(self,data)
        except:
            sys.exc_clear()

        #tokenizing extracted data
        for line in self.text.splitlines():
            tokens= parse(line, self.config)
            if not tokens: continue
            for i in xrange(len(tokens)):
                if i > 0 :
                    self.ht[(tokens[i], tokens[i-1])] +=1;
                    self.N +=1

        #cleaning for next feed
        self.text  = ''; self.N=0
        HTMLParser.reset(self)


```