Project Report

Word Frequency Counter

Jasen Ratnam 40094237

3rd December, 2019

Table Of Contents

Problem Description

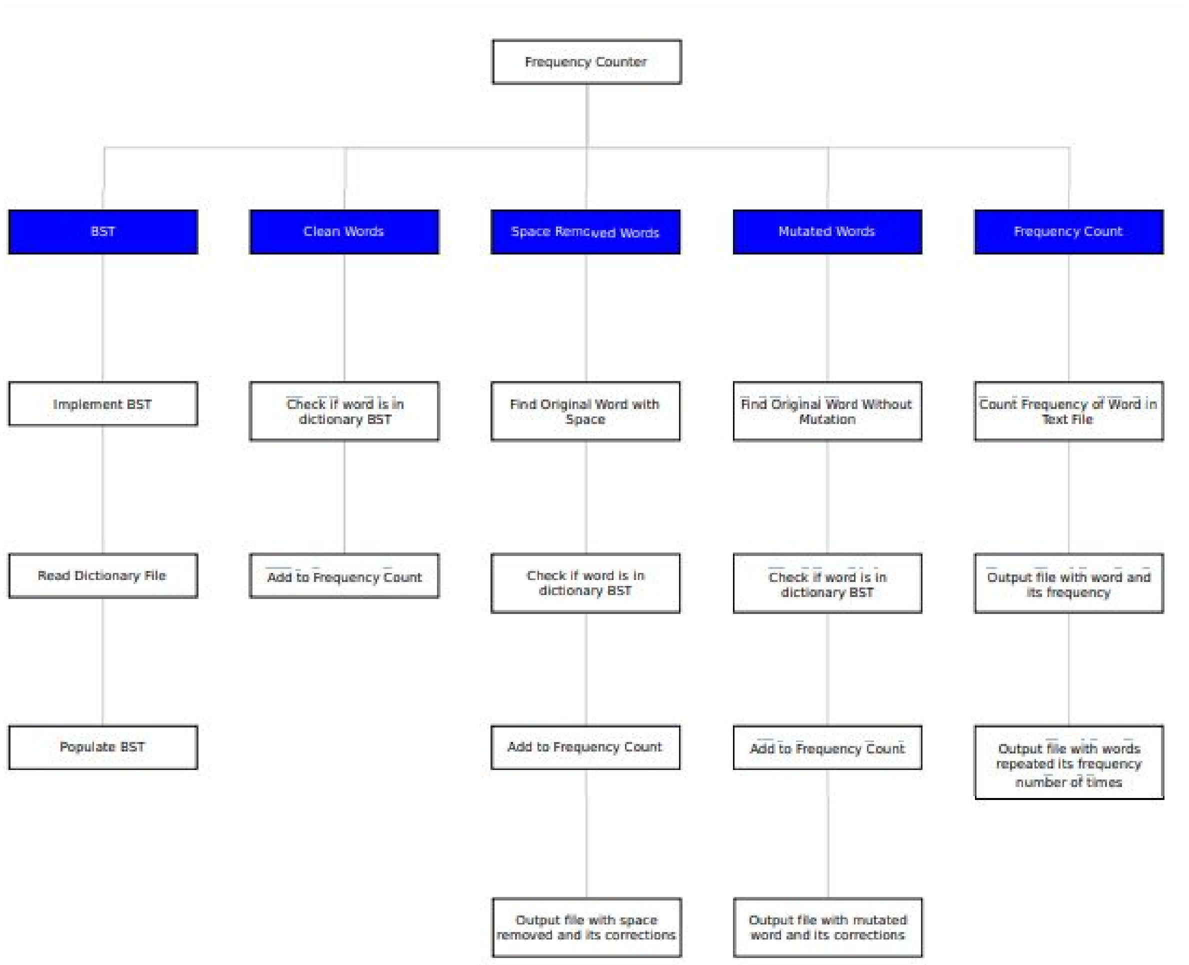
We are given the problem of finding the frequency of words in a given text file that exists in a given dictionary of legal words. The dictionary text file is made of a distinct set of words separated by spaces while the input file is a chapter of a book. To solve this problem, we must implement our own binary search tree (BST) and add the words from the dictionary file to it. We must use this BST to find the frequency of legal words in the input text file. The frequency of a word is the number of times the word is repeated in the text file. We will only find the frequency of words that are also in the BST. If a word from the input file is not in the BST, then the word should be ignored during the first part or verified for mutations in the other parts. The program solving this problem should output a text file with legal words found and their frequencies, sorted by their frequencies and then alphabetically in ascending order for both. We should get a word cloud image and a bar plot for this result.

In the second part of the problem, we get a text file with randomly missing spaces making two words into one. We must verify if we can get a legal word from the words not detected as being in the BST above, assuming that it may be two words with missing spaces. If the unknown word is made of a legal word, add this newly found word to the frequency count. For example, if the unknown word is “goodbut”, our solution should find the words “good” & “but” and add them to the frequency count.

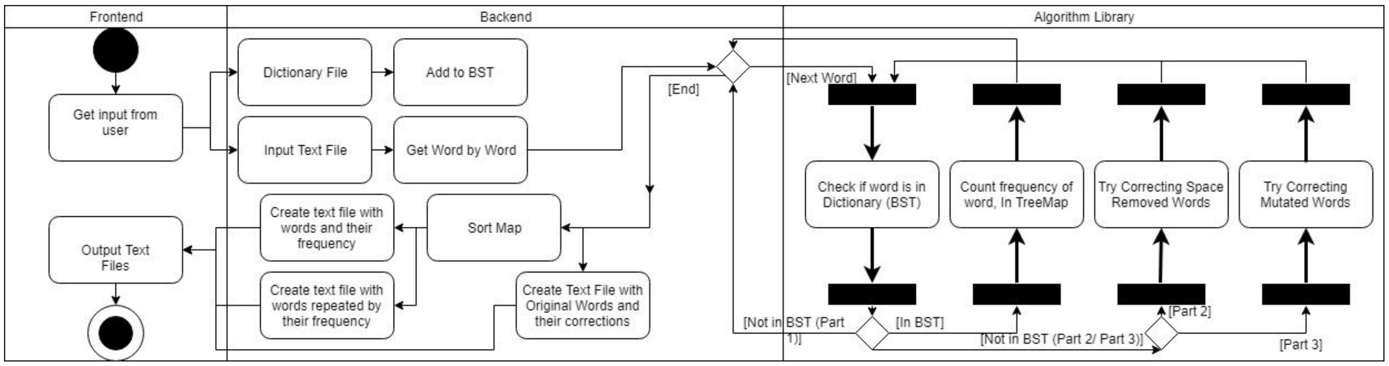
In the third part of the problem, we are given a text file with randomly mutated characters. In the text file, a character in a word is randomly substituted with another random character (any characters including spaces, newline, tabs, etc.). We should verify if this is a word from our BST with a mutated character and add the corrected word to the frequency count, if it is. For example, if the unknown word read is “g=od” our solution should find “good” and add it to the frequency counter.

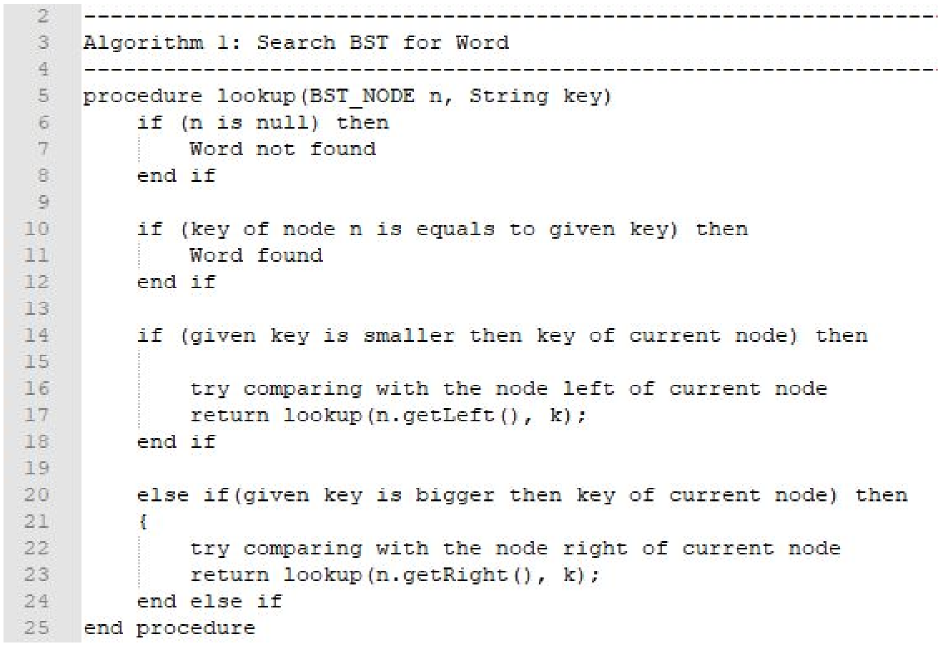
In brief, we are provided a set of input files for every part and a common dictionary. The input files for the second and third part are the same as the input file for the first part, but with the space being removed for part 2 and characters being mutated for part 3. We are then required to correct the space removed words and the mutated word, then find the frequency of legal words from the dictionary. If our solution to this problem is perfect, all 3 parts should result in the same exact frequencies.txt, repeated.txt, word cloud diagram and bar plot but it is impossible to get a perfect solution in real life.

Problem Breakdown


Frequency Counter
BST
Clean Words
Space Removed Words
Mutated Words
Frequency Count
Implement BST
Read Dictionary File
Populate BST
Check if word is in dictionary BST
Add to Frequency Count
Find Original Word with Space
Check if word is in dictionary BST
Add to Frequency Count
Output file with space removed and its corrections
Find Original Word Without Mutation
Check if word is in dictionary BST
Add to Frequency Count
Output file with mutated word and its corrections
Count Frequency of Word in Text File
Output file with word and its frequency
Output file with words repeated its frequency number of times

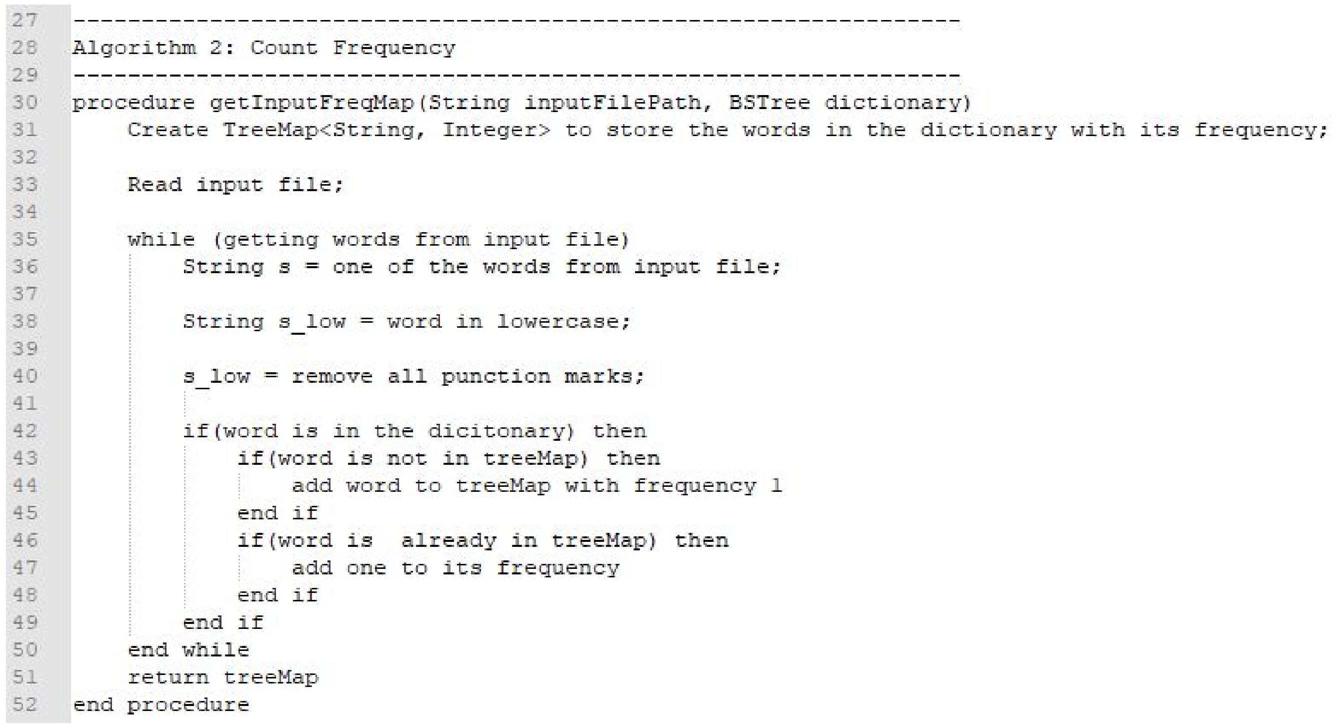
Solution Design


Frontend
Backend
Algorithm Library
Get input from user
Dictionary File
Add to BST
Input Text File
Get Word by Word
[End]
[Next Word]
Create text file with words and their frequency
Sort Map
Output Text Files
Create text file with words repeated by their frequency
Create Text File with Original Words and their corrections
Check if word is in Dictionary (BST)
Count frequency of word, In TreeMap
Try Correcting Space Removed Words
Try Correcting Mutated Words
[Not in BST (Part 1)]
[In BST]
[Not in BST (Part 2/ Part 3)]
[Part 2]
[Part 3]



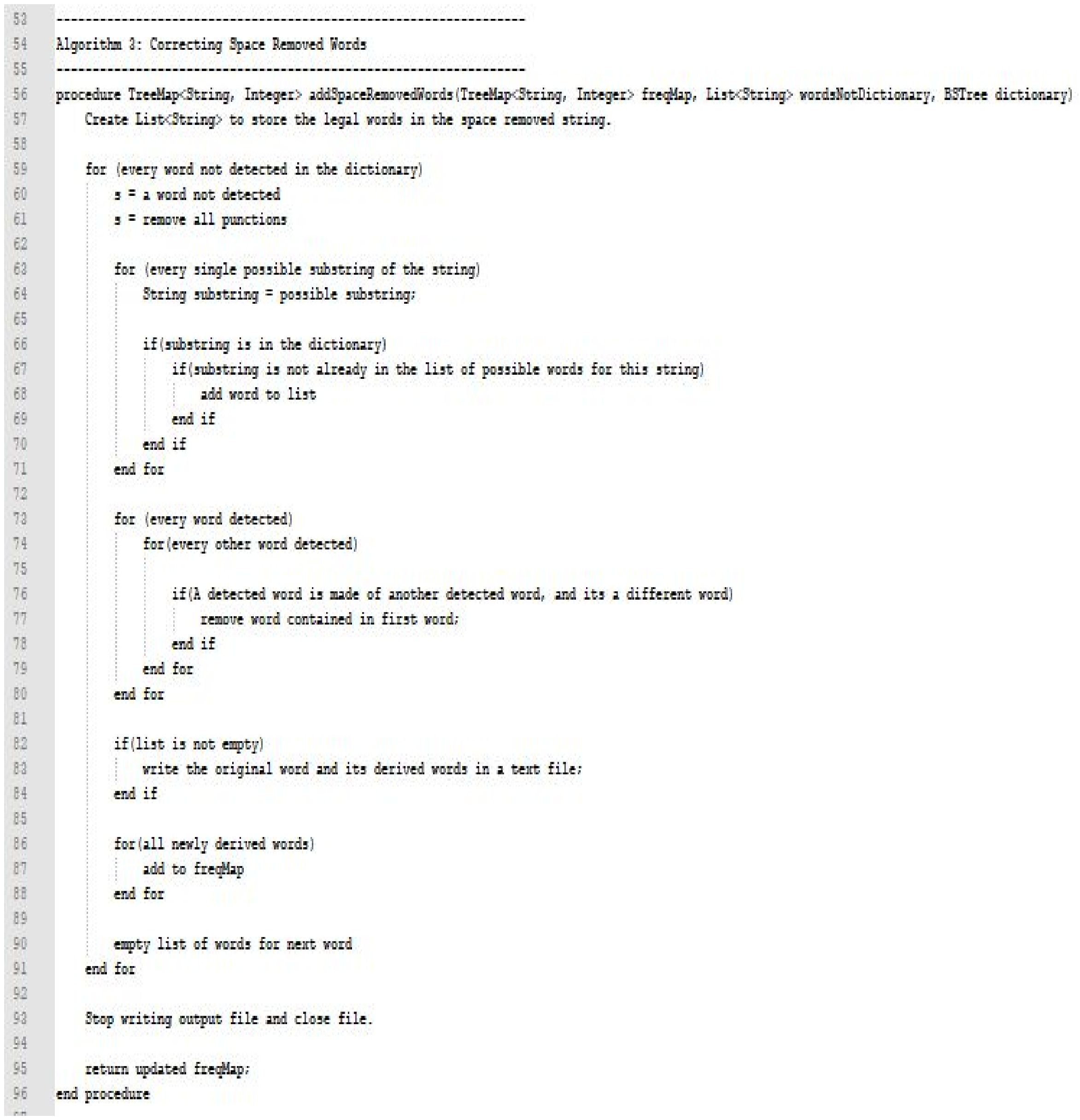
```
-----------------------------------------------------------------
Algorithm 1: Search BST for Word
-----------------------------------------------------------------
procedure lookup(BST_NODE n, String key)
    if (n is null) then
        Word not found
    end if

    if (key of node n is equals to given key) then
        Word found
    end if

    if (given key is smaller then key of current node) then

        try comparing with the node left of current node
        return lookup(n.getLeft(), k);
    end if

    else if(given key is bigger then key of current node) then
    {
        try comparing with the node right of current node
        return lookup(n.getRight(), k);
    end else if
end procedure
```

This algorithm is used to search if an element is inside a BST that we implemented. This is a recursive function that checks if the key of a node is the same key of the given key. If it's bigger go on the left branch, if it's smaller go on the right branch. Return true if found, false if not found. Here, a normal BST is used instead of other types like red-black trees, since it's the most simple one that does what is required. Since the dictionary is already sorted, we just split it and balance it in the tree. A red-black tree would be more efficient if the given dictionary is not sorted. Hence, this implementation may be a problem if the provided dictionary is not sorted. This type of BST is also simpler to implement with less chance of error. In retrospect, using hash table would of been a better idea since it is faster to search it. But the use of a BST was a requirement for this project.



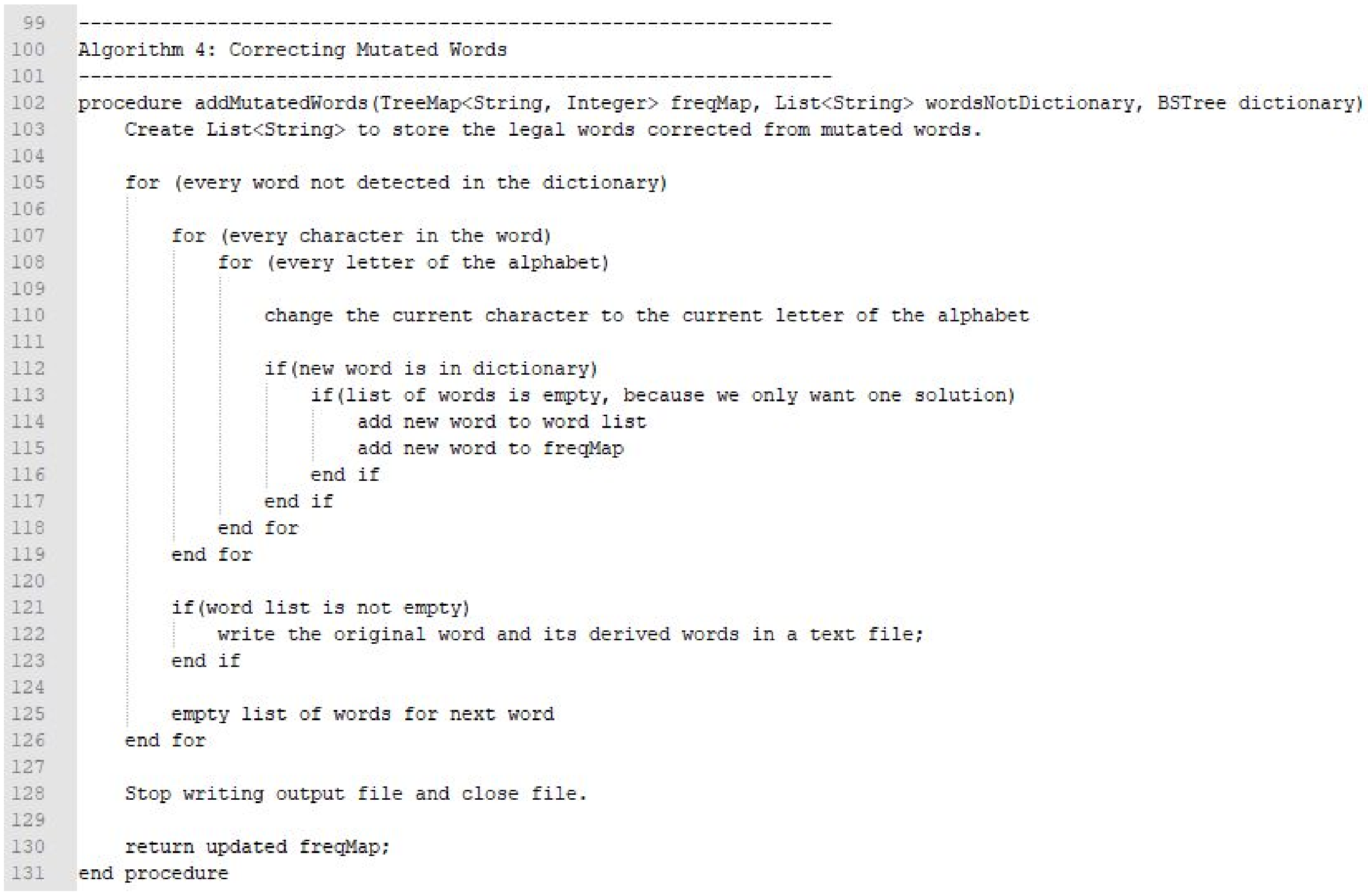
```
-----------------------------------------------------------------
Algorithm 2: Count Frequency
-----------------------------------------------------------------
procedure getInputFreqMap(String inputFilePath, BSTree dictionary)
    Create TreeMap<String, Integer> to store the words in the dictionary with its frequency;

    Read input file;

    while (getting words from input file)
        String s = one of the words from input file;

        String s_low = word in lowercase;

        s_low = remove all punction marks;

        if(word is in the dicitonary) then
            if(word is not in treeMap) then
                add word to treeMap with frequency 1
            end if
            if(word is  already in treeMap) then
                add one to its frequency
            end if
        end if
    end while
    return treeMap
end procedure
```

This algorithm is used to count the frequency of every word from the input text file that is in the dictionary BST. This algorithm gets a word from the text file, make it in lowercase and remove punctuation marks. Then use Algorithm 1 to verify if it's in the BST. If the word is in the BST, the program will add the word to a treeMap with adding 1 to its frequency if its already in the map, make the frequency 1 if it's the first-time word is added to the map. Here a treeMap is used to hold the words and their frequency since it holds its values in alphabetical order which will be useful when outputting the required files. The strings not found to be in the BST are added to a list that will be analysed to verify if they are space removed words or mutated words below.



```
-----------------------------------------------------------------
Algorithm 3: Correcting Space Removed Words
-----------------------------------------------------------------
procedure TreeMap<String, Integer> addSpaceRemovedWords(TreeMap<String, Integer> freqMap, List<String> wordsNotDictionary, BSTree dictionary)
    Create List<String> to store the legal words in the space removed string.

    for (every word not detected in the dictionary)
        s = a word not detected
        s = remove all punctions

        for (every single possible substring of the string)
            String substring = possible substring;

            if(substring is in the dictionary)
                if(substring is not already in the list of possible words for this string)
                    add word to list
                end if
            end if
        end for

        for (every word detected)
            for(every other word detected)

                if(A detected word is made of another detected word, and its a different word)
                    remove word contained in first word;
                end if
            end for
        end for

        if(list is not empty)
            write the original word and its derived words in a text file;
        end if

        for(all newly derived words)
            add to freqMap
        end for

        empty list of words for next word
    end for

    Stop writing output file and close file.

    return updated freqMap;
end procedure
```

This algorithm is used to correct space removed words. For example, if the string ‘goodbut’ is inputted, the words ‘good’ and ‘but’ should be detected by this algorithm (Assuming their both in the dictionary). The algorithm gets a string that may be two or more space removed words and checks every possible substring of that string to see if it’s a legal word from the dictionary. The legal words found are added to the treeMap with its frequency. This method also makes sure that the longer word is chosen when we have multiple choices with the same letters. The substrings are gotten using a double for loop. This method also writes an output file with the corrected words and its original form.

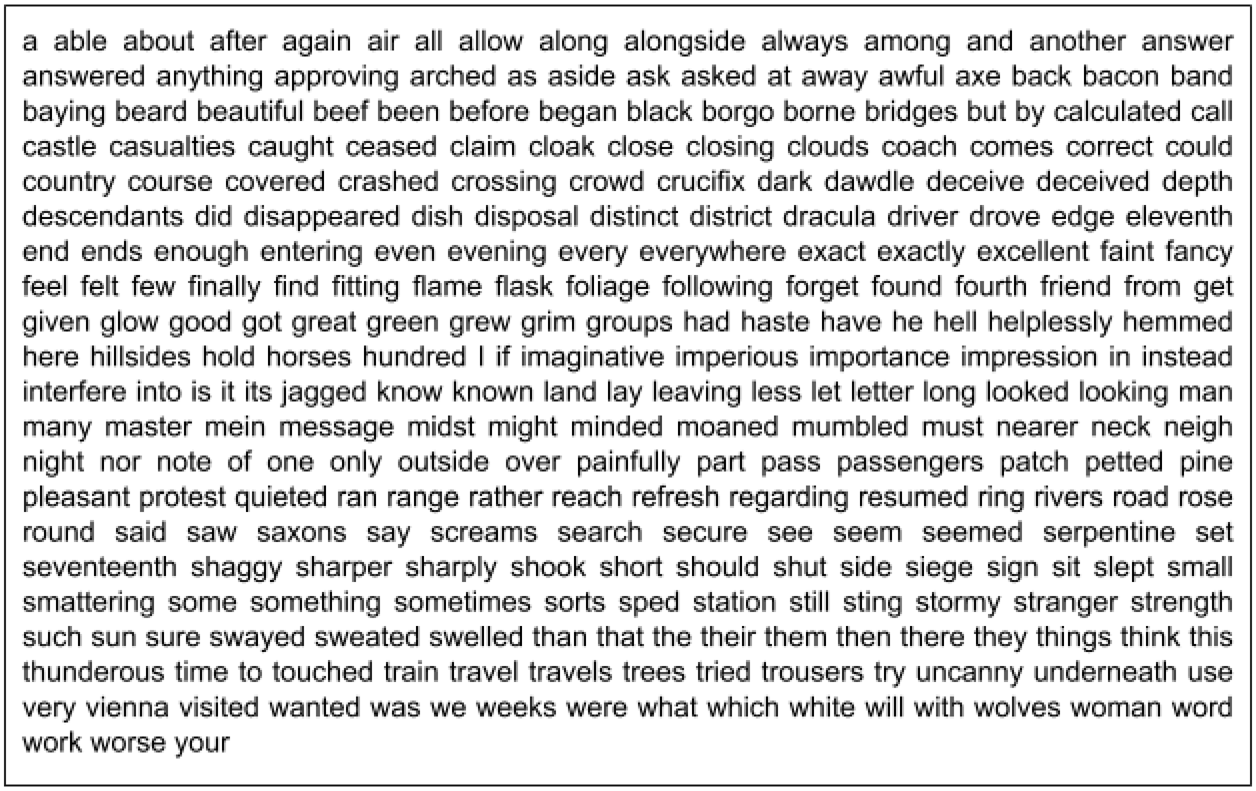


```
-----------------------------------------------------------------
Algorithm 4: Correcting Mutated Words
-----------------------------------------------------------------
procedure addMutatedWords(TreeMap<String, Integer> freqMap, List<String> wordsNotDictionary, BSTree dictionary)
    Create List<String> to store the legal words corrected from mutated words.

    for (every word not detected in the dictionary)

        for (every character in the word)
            for (every letter of the alphabet)

                change the current character to the current letter of the alphabet

                if(new word is in dictionary)
                    if(list of words is empty, because we only want one solution)
                        add new word to word list
                        add new word to freqMap
                    end if
                end if
            end for
        end for

        if(word list is not empty)
            write the original word and its derived words in a text file;
        end if

        empty list of words for next word
    end for

    Stop writing output file and close file.

    return updated freqMap;
end procedure
```

This algorithm is used to correct mutated words. For example, if the string ‘g=od’ is inputted, the word ‘good’ should be detected by this algorithm. The algorithm gets a string that may be a mutated word and goes through every character of the word, replacing it by every letter of the alphabet until the string matches a word of the dictionary. This solution is highly inefficient but it works and solves the problem in a reasonable amount of time for mutated words that are small in length. If the length of a word increases significantly this solution will be worse since it needs to go through every single character of the word even if it is not a mutated character. One way to make this solution more efficient would be to verify if a character is indeed mutated before trying every single letter of the alphabet on it. But we will need to find a way to know if a character is indeed mutated which i can’t find with the time limit given. Hence, this is the best solution i can implement which as low error.

Results & Analysis

For all three parts of the problem, the following text file is used to get the legal words, in order to count their frequencies. This is the dictionary used for the results and analysis done in this report.

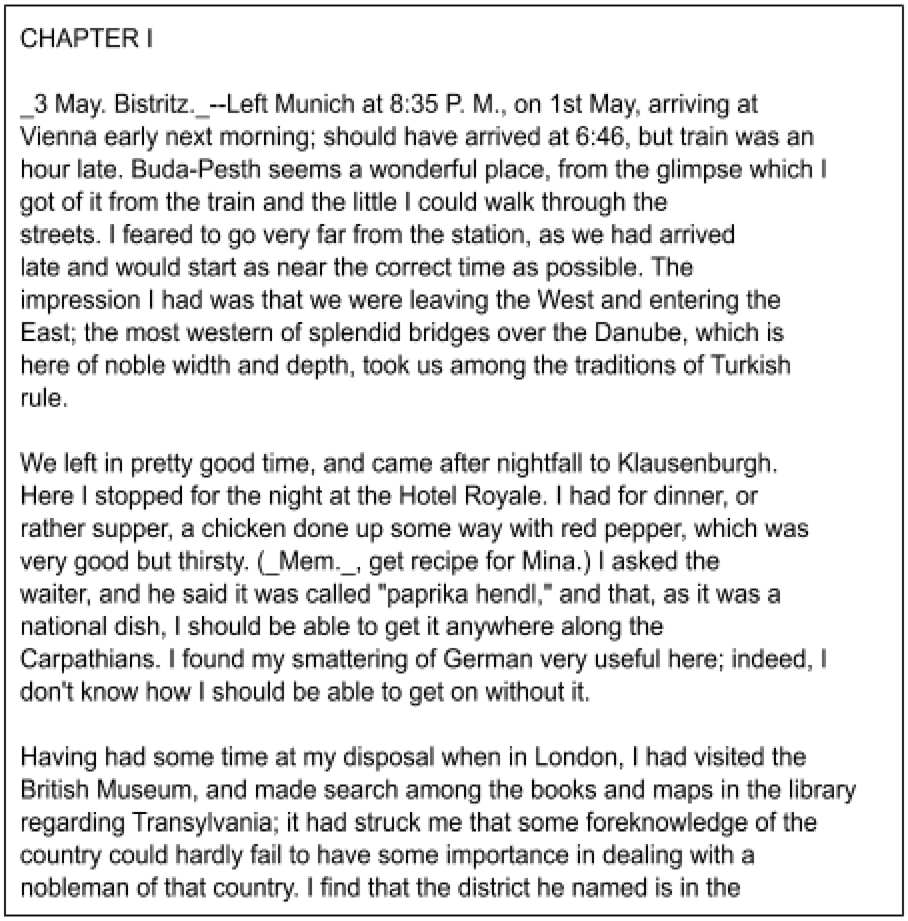


a able about after again air all allow along alongside always among and another answer answered anything approving arched as aside ask asked at away awful axe back bacon band baying beard beautiful beef been before began black borgo borne bridges but by calculated call castle casualties caught ceased claim cloak close closing clouds coach comes correct could country course covered crashed crossing crowd crucifix dark dawdle deceive deceived depth descendants did disappeared dish disposal distinct district dracula driver drove edge eleventh end ends enough entering even evening every everywhere exact exactly excellent faint fancy feel felt few finally find fitting flame flask foliage following forget found fourth friend from get given glow good got great green grew grim groups had haste have he hell helplessly hemmed here hillsides hold horses hundred I if imaginative imperious importance impression in instead interfere into is it its jagged know known land lay leaving less let letter long looked looking man many master mein message midst might minded moaned mumbled must nearer neck neigh night nor note of one only outside over painfully part pass passengers patch petted pine pleasant protest quieted ran range rather reach refresh regarding resumed ring rivers road rose round said saw saxons say screams search secure see seem seemed serpentine set seventeenth shaggy sharper sharply shook short should shut side siege sign sit slept small smattering some something sometimes sorts sped station still sting stormy stranger strength such sun sure swayed sweated swelled than that the their them then there they things think this thunderous time to touched train travel travels trees tried trousers try uncanny underneath use very vienna visited wanted was we weeks were what which white will with wolves woman word work worse your

dictionary.txt

Part 1:

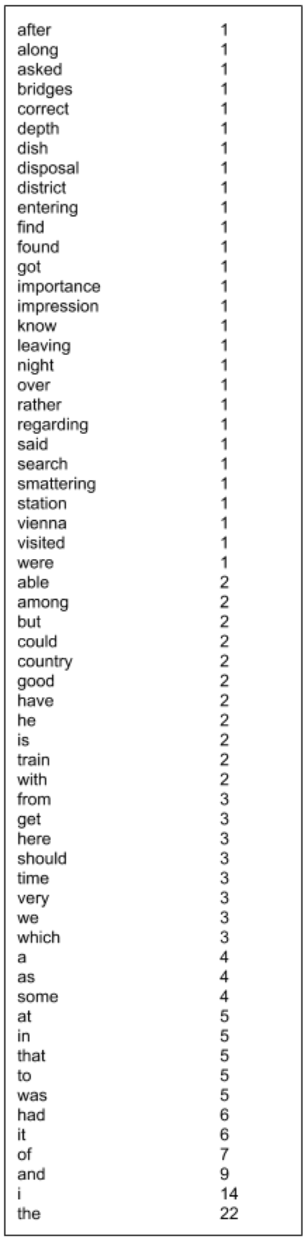
For this part, the program gets a clean input text file which is one-page with around 300 words. We can assume that this text file does not contain any errors such as space removed and mutated characters. The following is the input text used for the results and analysis done in this part.



```
CHAPTER I

_3 May. Bistritz._--Left Munich at 8:35 P. M., on 1st May, arriving at
Vienna early next morning; should have arrived at 6:46, but train was an
hour late. Buda-Pesth seems a wonderful place, from the glimpse which I
got of it from the train and the little I could walk through the
streets. I feared to go very far from the station, as we had arrived
late and would start as near the correct time as possible. The
impression I had was that we were leaving the West and entering the
East; the most western of splendid bridges over the Danube, which is
here of noble width and depth, took us among the traditions of Turkish
rule.

We left in pretty good time, and came after nightfall to Klausenburgh.
Here I stopped for the night at the Hotel Royale. I had for dinner, or
rather supper, a chicken done up some way with red pepper, which was
very good but thirsty. (_Mem._, get recipe for Mina.) I asked the
waiter, and he said it was called "paprika hendl," and that, as it was a
national dish, I should be able to get it anywhere along the
Carpathians. I found my smattering of German very useful here; indeed, I
don't know how I should be able to get on without it.

Having had some time at my disposal when in London, I had visited the
British Museum, and made search among the books and maps in the library
regarding Transylvania; it had struck me that some foreknowledge of the
country could hardly fail to have some importance in dealing with a
nobleman of that country. I find that the district he named is in the
```

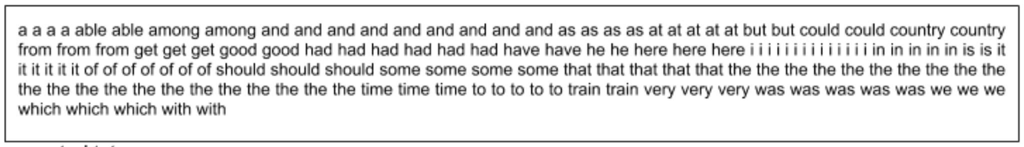
0_clean.txt

The solution of this part has the following outputs; a text file with the legal words found in the input file above and their frequency, and a text file with the legal words found repeated by its frequency count. Both these text files are shown below for the input above.



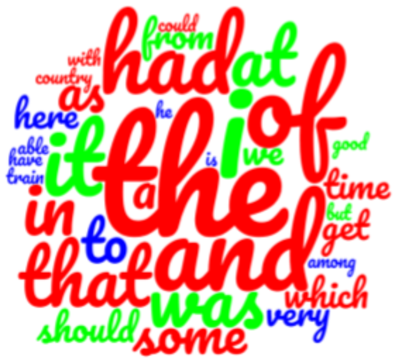
```
after           1
along           1
asked           1
bridges         1
correct         1
depth           1
dish            1
disposal        1
district        1
entering        1
find            1
found           1
got             1
importance      1
impression      1
know            1
leaving         1
night           1
over            1
rather          1
regarding       1
said            1
search          1
smattering      1
station         1
vienna          1
visited         1
were            1
able            2
among           2
but             2
could           2
country         2
good            2
have            2
he              2
is              2
train           2
with            2
from            3
get             3
here            3
should          3
time            3
very            3
we              3
which           3
a               4
as              4
some            4
at              5
in              5
that            5
to              5
was             5
had             6
it              6
of              7
and             9
i               14
the             22
```

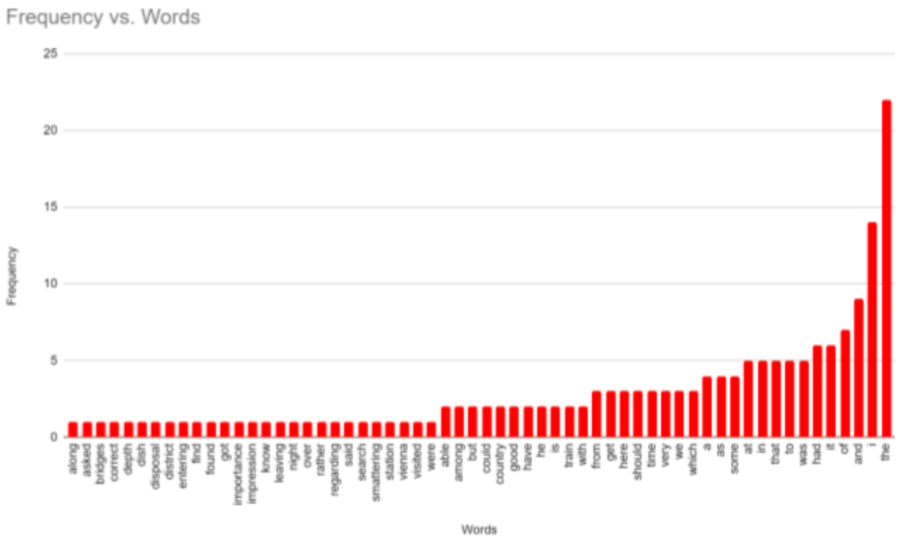
frequency.txt



```
a a a a able able among among and and and and and and and and and as as as as at at at at at but but could could country country
from from from get get get good good had had had had had had have have he he here here here i i i i i i i i i i i i i i in in in in in is is it
it it it it it of of of of of of of should should should some some some some that that that that that the the the the the the the the the the
the the the the the the the the the the the the time time time to to to to to train train very very very was was was was was we we we
which which which with with
```

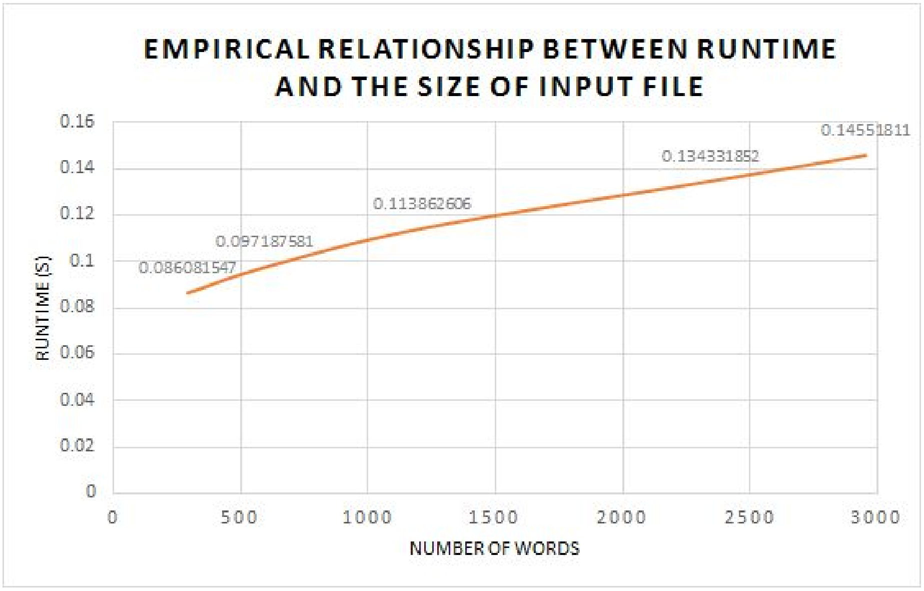
repeated.txt





These are the results for the input text above. By inspection, we can see that the result is accurate and that our program works as intended. The program efficiently reads the input file and counts the frequency of words that are also in the dictionary text file. The wordCloud and bar plot shows that the word 'the' is the most used in the input file. In a perfect world, these results would be the same for the following parts. But this is not the case, as we will see in the following sections of the report. The reasons for this error will be explained in their respective sections. Below is the time analysis for this part of the problem:

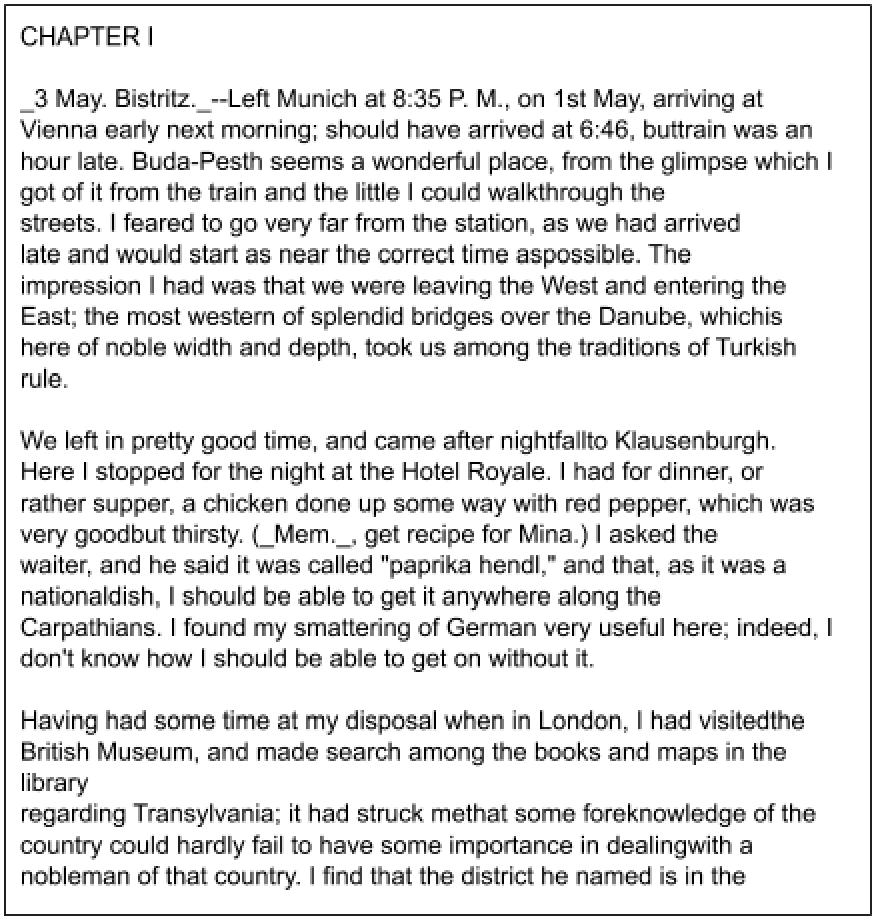
Part 1	RUNTIME 1	RUNTIME 2	RUNTIME 3	AVERAGE RUNTIME (NS)	AVERAGE RUNTIME IN SECONDS (S)
1 file (293 words)	86409159	86871881	84963602	86081547.33	0.08608154733
2 files (597 words)	94446403	98793887	98322452	97187580.67	0.09718758067
4 files (1,211 words)	113511758	111416288	116659771	113862605.7	0.1138626057
8 files (2,345 words)	136797500	133178952	133019103	134331851.7	0.1343318517
10 files (2,955 words)	146535998	144020773	145997559	145518110	0.14551811
Average for 5 files (NS)	115396339.1	Average for 5 files (S)	0.1153963391		
Standard Deviation for 5 files (NS)	22177208.83	Standard Deviation for 5 files (S)	0.02217720883		



From the results above, we can observe that the runtime and the size of the input file has a relationship where the runtime increases by about 0.01 to 0.02 seconds when the input file is doubled. As the word count grows higher, the runtime also grows higher. On average, the runtime for this part is of 0.1153963391 seconds with a standard deviation of 0.02217720883 seconds. From observation, we may make a conclusion that the relationship is linear since the run time linearly increases when the size of the file increases.

Part 2:

For this part, the program gets an input text file of one-page with around 300 words with randomly removed spaces. We will assume that the words in the input file that are not found in the dictionary above may be two words with a space removed and try to find a legal word from it. The following is the input text used for the results and analysis done in this part.



```
CHAPTER I

_3 May. Bistritz._--Left Munich at 8:35 P. M., on 1st May, arriving at
Vienna early next morning; should have arrived at 6:46, buttrain was an
hour late. Buda-Pesth seems a wonderful place, from the glimpse which I
got of it from the train and the little I could walkthrough the
streets. I feared to go very far from the station, as we had arrived
late and would start as near the correct time aspossible. The
impression I had was that we were leaving the West and entering the
East; the most western of splendid bridges over the Danube, whichis
here of noble width and depth, took us among the traditions of Turkish
rule.

We left in pretty good time, and came after nightfallto Klausenburgh.
Here I stopped for the night at the Hotel Royale. I had for dinner, or
rather supper, a chicken done up some way with red pepper, which was
very goodbut thirsty. (_Mem._, get recipe for Mina.) I asked the
waiter, and he said it was called "paprika hendl," and that, as it was a
nationaldish, I should be able to get it anywhere along the
Carpathians. I found my smattering of German very useful here; indeed, I
don't know how I should be able to get on without it.

Having had some time at my disposal when in London, I had visitedthe
British Museum, and made search among the books and maps in the
library
regarding Transylvania; it had struck methat some foreknowledge of the
country could hardly fail to have some importance in dealingwith a
nobleman of that country. I find that the district he named is in the
```

0_space.txt

The solution of this part has the following outputs; a text file with the legal words found in the input file above and their frequency, a text file with the legal words found repeated by its frequency count, and a text file with the corrected words and their original version. These text files are shown below for the input above, as well as a graph and a wordcloud to demonstrate the frequency of different words graphically.

after	1
along	1
asked	1
bridges	1
call	1
correct	1
depth	1
did	1
dish	1
disposal	1
district	1
edge	1
entering	1
find	1
found	1
got	1
importance	1
impression	1
know	1
leaving	1
one	1
over	1
rather	1
regarding	1
said	1
search	1
seem	1
smattering	1
station	1
use	1
vienna	1
visited	1
were	1
able	2
among	2
but	2
could	2
country	2
good	2
have	2
is	2
man	2
night	2
train	2
from	3
get	3
he	3
should	3
time	3
very	3
which	3
with	3
as	4
here	4
some	4
at	5
that	5
was	5
we	5
had	6
in	6
it	6
to	6
of	7
and	9
a	12
i	14
the	22

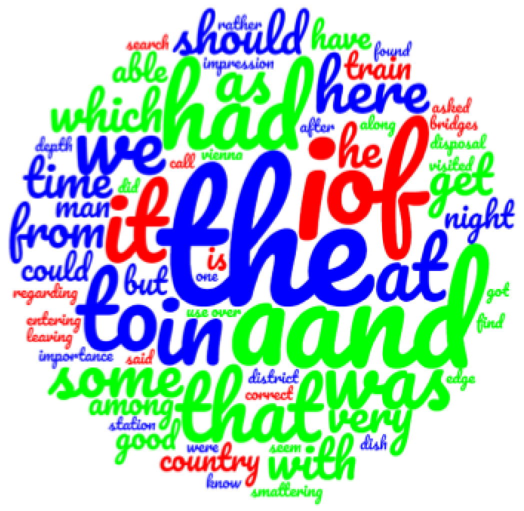
frequency.txt

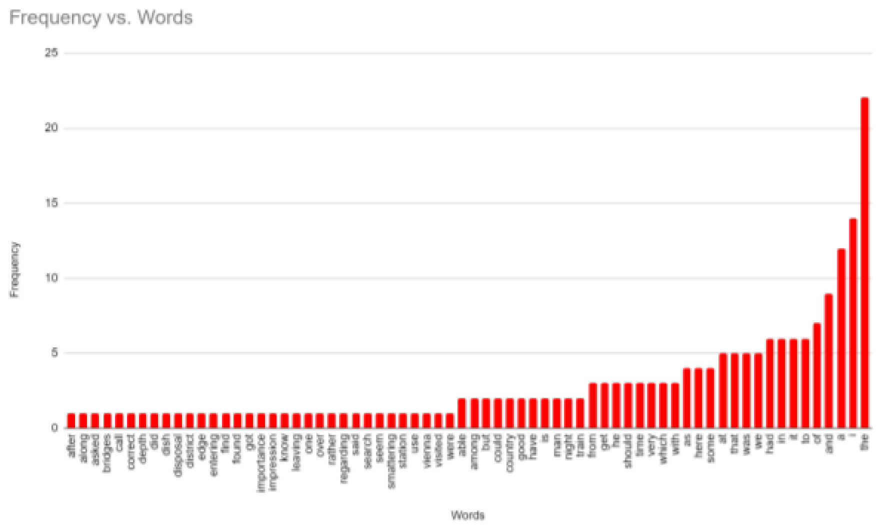
```
a a a a a a a a a a a a able able among among and and and and and and and and and as as as as at at at at at but but could could
country country from from from get get get good good had had had had had had have have he he he here here here here i i i i i i i i i
i i i i i in in in in in in is is it it it it it it man man night night of of of of of of of should should should some some some some that that
that that that the the the the the the the the the the the the the the the the the the the the the the time time time to to to to to to train
train very very very was was was was was we we we we we which which which with with with
```

repeated.txt

arriving	[a]
arrived	[a]
buttrain	[but, train]
an	[a]
seems	[seem]
arrived	[a]
aspossible	[as]
west	[we]
western	[we]
splendid	[did]
whichis	[which, is]
took	[to]
nightfallto	[night, to]
done	[one]
goodbut	[good, but]
mina	[a]
called	[call]
paprika	[a]
hendl	[he]
nationaldish	[dish]
anywhere	[a, here]
german	[man]
useful	[use]
indeed	[in]
without	[with]
visitedthe	[visited, the]
transylvania	[a]
methat	[that]
foreknowledge	[edge]
dealingwith	[with]
nobleman	[man]

corrected_words_detected.txt



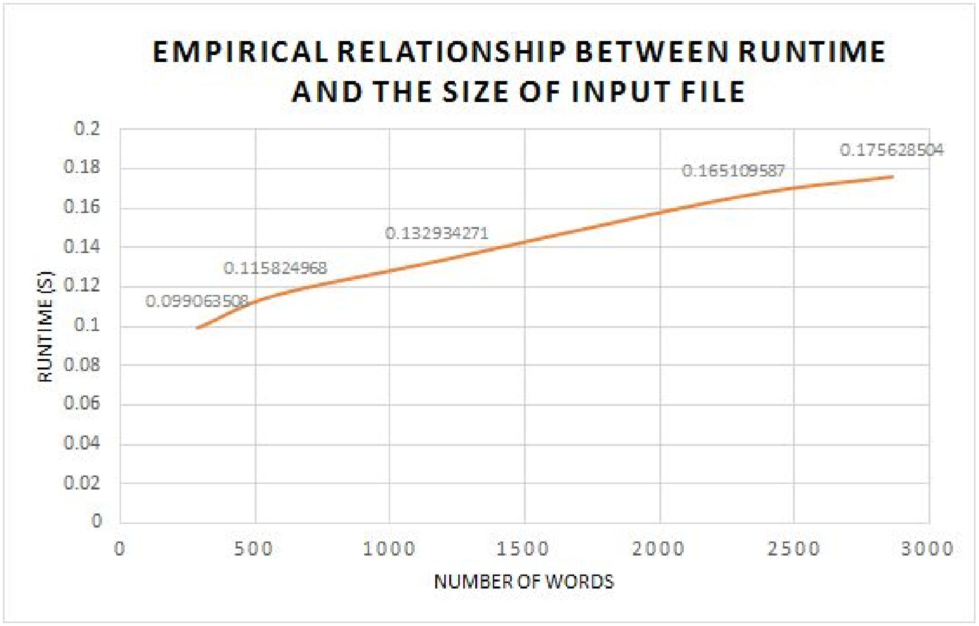


frequency.txt

These are the results for the input text above. By inspecting it against the results from Part 1, we can see that the result is not accurate. The program should output the exact same results. This is not the case because, as we can see from our “corrected_words_detected” text file, we have some errors. Since there's no way of knowing if a word is indeed a word or two words with space removed, the program assumes that all words not found in the dictionary are two words without space and tries to find legal words contained in them. Therefore, if a standalone word is composed of a legal word, it will detect it and output that legal word. Hence, the error we get.

For example, the input file contains the word “seems” which is clearly not a space removed word because there's no word “s” in English, but the computer has no way of knowing that so it assumes that “seem” and “s” are two different words with a space removed and outputs the word “seem” since it's in the dictionary. Although we have some errors, all the space removed words were found using this program. Hence, I would conclude this as a success. It is possible to improve this solution by reducing the number of errors but due to lack of time and knowledge this is not possible by me at the moment. One possible simple way to reduce the number of errors is to ask the user to verify the corrections being done by the computer. Although this will be inefficient for longer text files. Below is the time analysis for this part of the problem:

Part 2	RUNTIME 1	RUNTIME 2	RUNTIME 3	AVERAGE RUNTIME	AVERAGE RUNTIME IN SECONDS (S)
1 file (283 words)	100179951	99004216	98006358	99063508.33	0.09906350833
2 files (578 words)	116472278	117942774	113059852	115824968	0.115824968
4 files (1,174 words)	133401299	133518483	131883030	132934270.7	0.1329342707
8 files (2,276 words)	166393090	167552299	161383371	165109586.7	0.1651095867
10 files (2,863 words)	173036158	178929556	174919797	175628503.7	0.1756285037
Average for 5 files (NS)	137712167.5	**Average for 5 files (S)**	0.1377121675		
Standard Deviation for 5 files (NS)	28927024.68	**Standard Deviation for 5 files (S)**	0.02892702468		

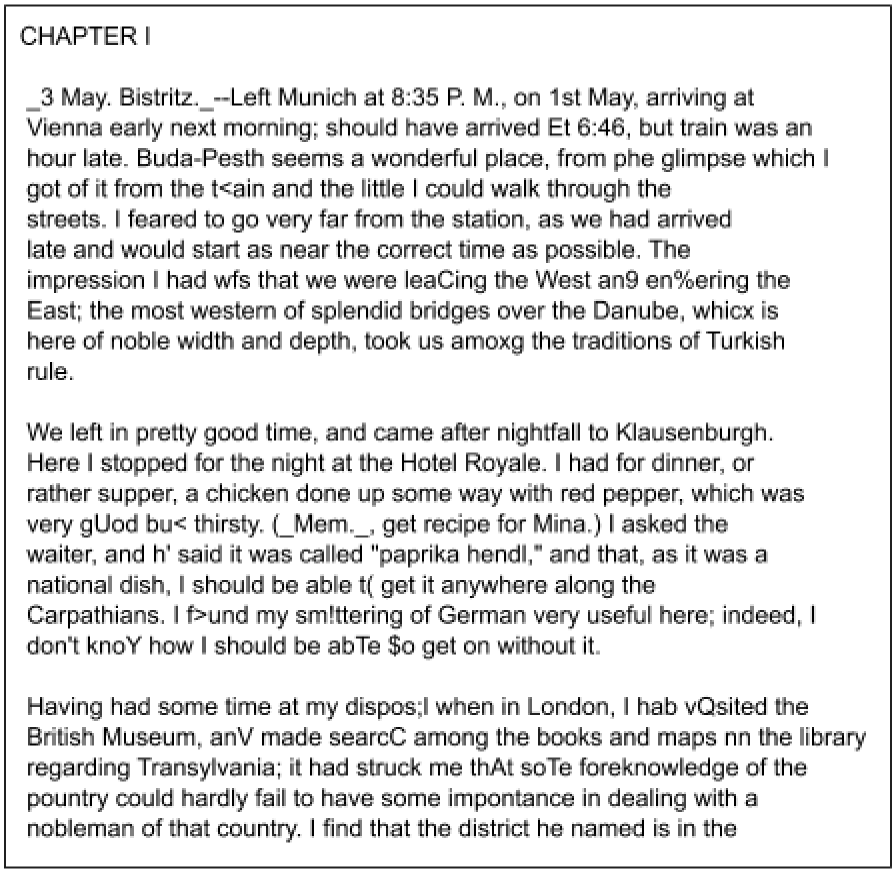


From the results above, we can observe that the runtime and the size of the input file has a relationship where the runtime increases by about 0.01 to 0.02 seconds when the input file is doubled. As the word count grows higher, the runtime also grows higher. On average, the runtime for this part is of 0.1377121675 seconds with a standard deviation of 0.02892702468 seconds. Compared to the results from part 1, we can see that the average runtime increased from 0.1153963391 to 0.1377121675 seconds. Although, the overall runtime went up for this part, the relation remains the same with runtime increase of about 0.01 to 0.02 seconds when the input file is doubled.

From observation, we may make a conclusion that the relationship is between logarithm and linear. Although this solution to the program as a runtime close to the runtime of part 1, which means that algorithm to correct the space removed words only add 0.02 seconds to the runtime. There are ways to make this solution even more efficient, for example the current solution goes through every possible substring of a word, even the ones that cannot possibly be a word such as only the first letter of a word. This also causes false-positives shown above. If we could find a way to fix this error, the program will be much more efficient.

Part 3:

For this part, the program gets an input text file of one-page with around 300 words with randomly mutated characters in a word. We will assume that the words in the input file that are not found in the dictionary above may be a word with a mutated character and try to find a legal word from it. The following is the input text used for the results and analysis done in this part.



```
CHAPTER I

_3 May. Bistritz._--Left Munich at 8:35 P. M., on 1st May, arriving at
Vienna early next morning; should have arrived Et 6:46, but train was an
hour late. Buda-Pesth seems a wonderful place, from phe glimpse which I
got of it from the t<ain and the little I could walk through the
streets. I feared to go very far from the station, as we had arrived
late and would start as near the correct time as possible. The
impression I had wfs that we were leaCing the West an9 en%ering the
East; the most western of splendid bridges over the Danube, whicx is
here of noble width and depth, took us amoxg the traditions of Turkish
rule.

We left in pretty good time, and came after nightfall to Klausenburgh.
Here I stopped for the night at the Hotel Royale. I had for dinner, or
rather supper, a chicken done up some way with red pepper, which was
very gUod bu< thirsty. (_Mem._, get recipe for Mina.) I asked the
waiter, and h' said it was called "paprika hendl," and that, as it was a
national dish, I should be able t( get it anywhere along the
Carpathians. I f>und my sm!ttering of German very useful here; indeed, I
don't knoY how I should be abTe $o get on without it.

Having had some time at my dispos;l when in London, I hab vQsited the
British Museum, anV made searcC among the books and maps nn the library
regarding Transylvania; it had struck me thAt soTe foreknowledge of the
pountry could hardly fail to have some impontance in dealing with a
nobleman of that country. I find that the district he named is in the
```

0_mutated.txt

The solution of this part has the following outputs; a text file with the legal words found in the input file above and their frequency, a text file with the legal words found repeated by its frequency count, and a text file with the corrected words and their original version. These text files are shown below for the input above, as well as a graph and a wordcloud to demonstrate the frequency of different words graphically. These outputs should be the same as the results of Part 1, but it is as we will see below it is not because of uncertainties from the computer.

after	1
along	1
asked	1
bridges	1
correct	1
depth	1
dish	1
disposal	1
district	1
entering	1
find	1
found	1
got	1
importance	1
impression	1
know	1
lay	1
leaving	1
must	1
night	1
note	1
over	1
rather	1
regarding	1
said	1
search	1
smattering	1
station	1
then	1
vienna	1
visited	1
were	1
your	1
able	2
among	2
but	2
by	2
country	2
good	2
have	2
is	2
train	2
with	2
could	3
from	3
get	3
here	3
nor	3
should	3
some	3
time	3
very	3
we	3
which	3
a	4
as	5
at	5
he	5
that	5
was	5
had	6
it	6
to	6
in	8
of	8
and	9
i	14
the	22

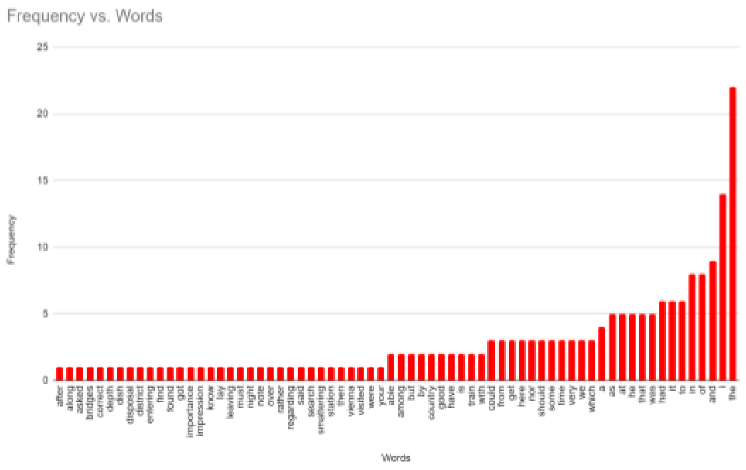
frequency.txt

```
a a a a able able among among and and and and and and and and and as as as as as at at at at at but but by by could could could
country country from from from get get get good good had had had had had had have have he he he he he here here here i i i i i i i i
i i i i i i in in in in in in in in is is it it it it it it nor nor nor of of of of of of of of should should should some some some that that that that
that the the the the the the the the the the the the the the the the the the the the the the time time time to to to to to to train train very
very very was was was was was we we we which which which with with
```

repeated.txt

on	[in]
et	[at]
an	[in]
hour	[your]
phe	[the]
t<ain	[train]
go	[to]
would	[could]
wfs	[was]
leacing	[leaving]
an9	[and]
en%ering	[entering]
most	[must]
whicx	[which]
us	[as]
amoxg	[among]
for	[nor]
for	[nor]
or	[of]
way	[lay]
guod	[good]
bu<	[but]
for	[nor]
h'	[he]
be	[he]
t(	[to]
f>und	[found]
my	[by]
sm!ttering	[smattering]
knoy	[know]
be	[he]
abte	[able]
$o	[to]
on	[in]
my	[by]
dispos;l	[disposal]
when	[then]
hab	[had]
vqsited	[visited]
anv	[and]
searcc	[search]
nn	[in]
me	[he]
sote	[note]
pountry	[country]
importance	[importance]

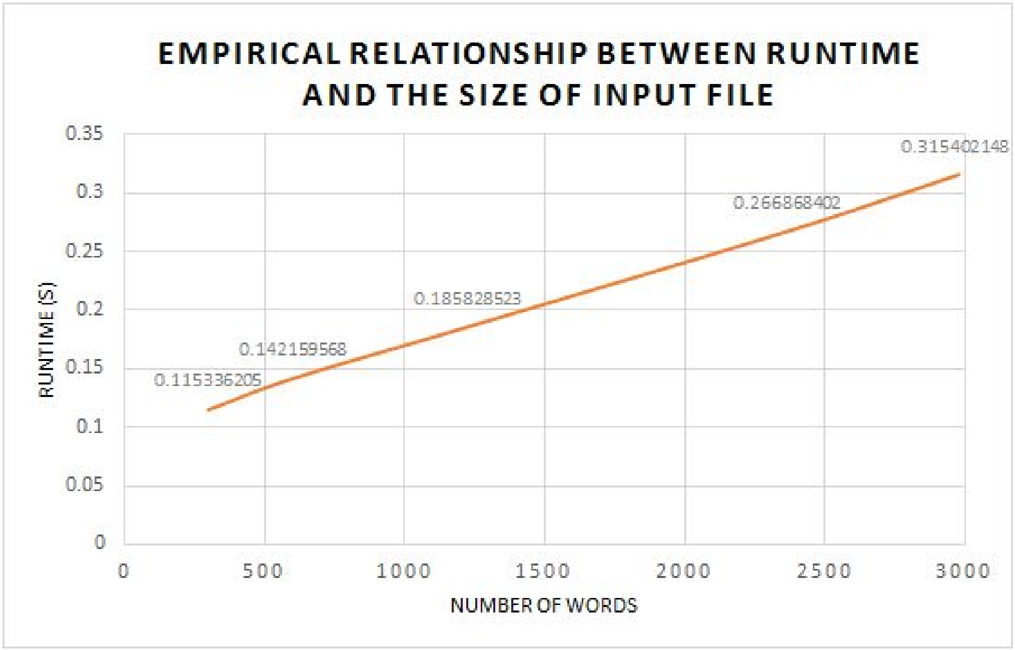
corrected_words_detected.txt



These are the results for the input text above. By inspecting it against the results from Part 1, we can see that the result is not accurate. The result has extra words and their frequencies are not accurate. This is the case because, as we can see from our “corrected_words_detected” text file, we have some errors. Since there's no way of knowing if a word is indeed a word that is just not in the dictionary or a mutated word with characters mutated, the program assumes that all words not found in the dictionary are mutated and tries to find a unmutated version that is a legal word. Therefore, if a standalone word can be mutated into a legal word, it will detect it and output that legal word. Hence, a perfectly fine word could be mutated to a legal word by error, which explains the errors seen.

For example, the input file contains the word “hour” which is clearly an English word without any mutations, but the computer has no way of knowing that since it’s not in our dictionary so it assumes that “hour” is a mutated word and outputs the word “your” since it's in the dictionary and the closest match to the word “hour”, assuming the character “h” is a mutation. This is an error, but with my knowledge, it is not possible to not get this error unless we put the word in the dictionary. Although we have some errors, all the mutated words in the text file were found using this program. Hence, I would conclude this as a success. It is possible to improve this solution by reducing the number of errors but due to lack of time and knowledge this is not possible by me at the moment. One possible simple way to reduce the number of errors is to have a bigger dictionary with every possible English words in it. This will be as simple as changing the dictionary text file since it is not hard coded to the program. This is the advantage of asking for a dictionary from the user instead of having one hard coded. Below is the time analysis for this part of the problem:

Part 3	RUNTIME 1	RUNTIME 2	RUNTIME 3	AVERAGE RUNTIME	AVERAGE RUNTIME IN SECONDS (S)
1 file (298 words)	114250310	116081368	115676937	115336205	0.115336205
2 files (603 words)	140614957	144012060	141851688	142159568.3	0.1421595683
4 files (1,224 words)	188569500	184198579	184717489	185828522.7	0.1858285227
8 files (2,364 words)	269338857	258921900	272344448	266868401.7	0.2668684017
10 files (2,981 words)	315675375	316957475	313573595	315402148.3	0.3154021483
Average for 5 files	205118969.2	**Average for 5 files (S)**	0.2051189692		
Standard Deviation for 5 files	75329043.86	**Standard Deviation for 5 files (S)**	0.07532904386		



From the results above, we can observe that the runtime and the size of the input file has a relationship where the runtime increases by about 0.03 to 0.04 seconds when the input file is doubled. As the word count grows higher, the runtime also grows higher. On average, the runtime for this part is of 0.2051189692 seconds with a standard deviation of 0.07532904386 seconds. Compared to the results from part 1, we can see that the average runtime increased from 0.1153963391 to 0.2051189692 seconds. The overall runtime increased significantly for this part. This is due to the fact that my design is not the most efficient possible due to the lack of time to spend on this project. Currently the program cause through every single character of a word and changes it to every letter of the alphabet until a legal word is found and it continues to find other possible words.

This is highly inefficient since it has gone through the n-length and 26 letters every single time. One possible way would be to start from the middle and do only half the word and then the other half if a legal word is not found. From observation, we may make a conclusion that the relationship is between logarithm and linear, tending more towards the linear side, since the graph looks more linear than log in this problem.

Mini Manual

1. Create input Files:
    a. Create a dictionary text file, contains dictionary words separated with spaces.
        i. Note down path of this file. (You may ignore ".txt")
    b. Create an input text file, contains any text of your choice.
        i. Note down path of this file. (You may ignore ".txt")
2. Running program:
    a. Run program on an IDE of your choice.
    b. In command line,
        i. Enter dictionary path noted above when prompted.
            1. If the dictionary is in the same folder as the project, you may enter only the file name. With or without ".txt" suffix.
        ii. Enter input text file path noted above when prompted.
            1. If the input is in the same folder as the project, you may enter only the file name. With or without ".txt" suffix.
        iii. Enter the part number of the problem you want to solve.
            1. '1' for Part 1 (Clean text file).
            2. '2' for Part 2 (Space randomly removed text file).
            3. '3' for Part 3 (Mutated words text file).
    c. Program will run and output files with the results.
3. Output:
    a. The program will output files with the results.
        i. A file with legal words found in the text file and their frequencies.
        ii. A file with legal words repeated by its frequency number.
        iii. If part 2 or part 3 was selected before running the program:
            1. A file containing the original words and their corrected version.
    b. All files are outputted in the same folder as the project.