DECEMBER 3, 2019

TEXT CORRECTOR PROJECT REPORT

COEN 352

JEAN-PHILLIPPE DURAND
40052679
Concordia University

Table of Content

PROBLEM DESCRIPTION:

The goal of the project is to develop a simple text corrector. The final solution shall focus on typographical and orthography errors. The user shall provide a word bank using a plain text file (.txt), which will be used by the program as a dictionary.

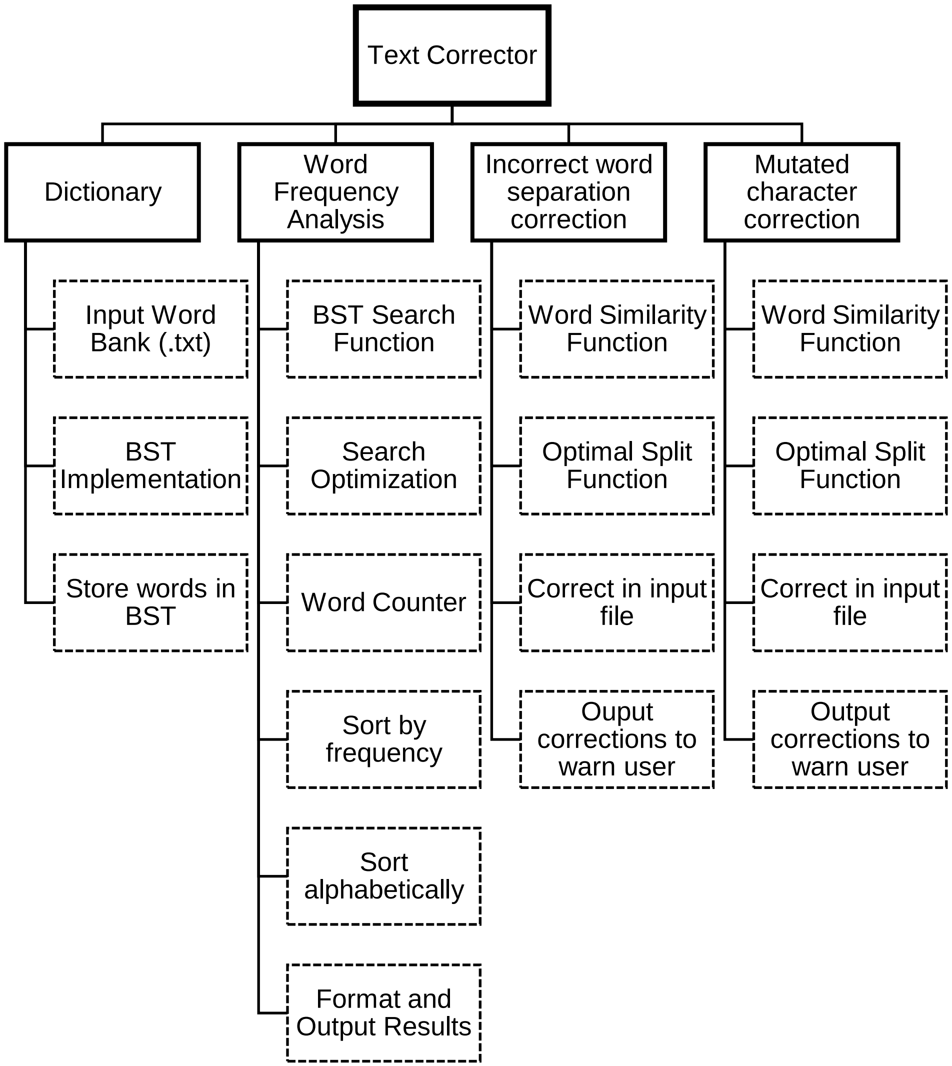
The input file used as a dictionary will contain an unknown number of words; each word (separated by a whitespace character). To prevent long computing times when analyzing the text to be corrected, the dictionary will be stored into a binary search tree (BST). Due to its low time complexity in the order of O(LogN), where N is the number of words stored in the BST, the BST will prevent the program from slowing down noticeably when inputting dictionaries containing a large number of words.

The BST created from the user inputted dictionary will constitute the foundation of the text corrector, as all correction operations will require searching in the dictionary for the validity of the word, or the closest match.

The corrector has three (3) functions. It will correct word separation in the case where the user forgot to add space between two know (to the dictionary) words. It will then correct words containing a mutated character (e.g. d=ctionary will be corrected to the dictionary) and warn the user of the correction by providing him with the initial word and its correction. For the last step, the corrector will analyze the frequency of known words in the text and output a list of the words found, sorted by their frequency and then alphabetically.

PROBLEM BREAKDOWN:

TEXT CORRECTOR WBS



SOLUTION & DESIGN:

This section will cover the design and solution for each part of the project; starting with the main function.

Solution Overview:

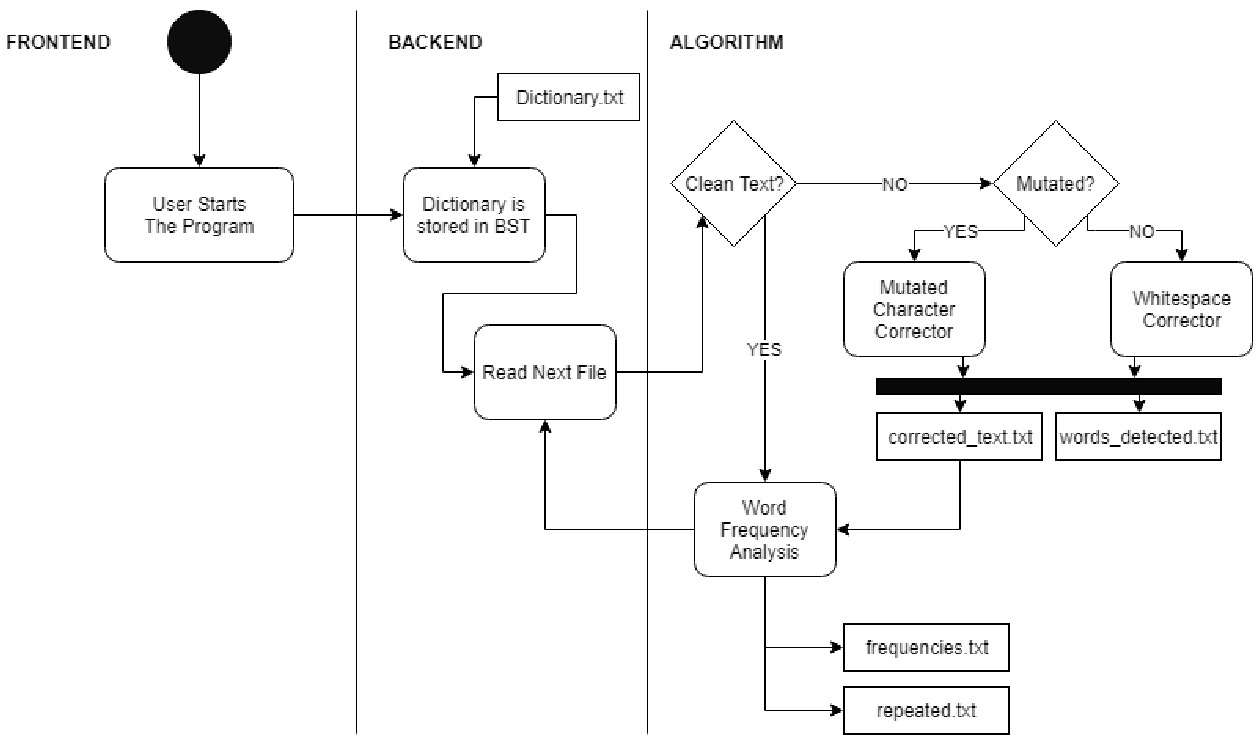


Figure 1: High-level Diagram of Information Processing Activities

The main function of the program takes care of the setup for each processing activity. The program starts by storing the words found in the dictionary.txt file into the Red-Black BST.

All three parts of the project are treated independently, and use their own algorithms, however, all three parts use the word frequency analysis algorithm. These three parts are run alphabetically and will be covered with more details in the next sections.

Word Frequency:

The word frequency was designed to meet the objectives of part A of the project but is also used in part B and C after the respective algorithms are run to correct the initial inputted text.

Pseudo-code:

```
procedure AnalyseFile()
    foreach (Word in InputText)
        if Dictionary.contains(Word) then
            if !WordHashMap.contains(Word) then
                WordHashMap.put(Word, 1)
            end if
            else then
                WordHashMap.put(Word, Value++)
            end else
        end if
    end foreach
    PrintOutputFile(WordHashMap, FileName)
end procedure


procedure PrintOutputFile(HashMap FoundWords, String FileName)
    FoundWords.sort()
    foreach(Pair in FoundWords)
        Outputfile.append(Pair.key + " " + Pair.value + "\n")
    end foreach
end procedure
procedure Sort()
    if a.value > b.value then return 1
    else if a.value < b.value then return -1
    else if a.key > b.key then return 1
    else return -1
end procedure
```

Whitespace Corrector

The whitespace corrector's algorithm searches each unknown word to the dictionary for a substring of a word contained in the dictionary. Using the Beyond Compare 4 software, the whitespace removed texts were compared with their clean text counterparts. This analysis concluded that all errors found in the customer's input texts were composed of two words merged. Therefore, the algorithm was designed to only allow one split; or, in other words, the addition of a single whitespace character. To achieve this, the algorithm splits the unknown word into two substrings. If there is no split resulting in two known words, the algorithm will weigh each split containing one known word and pick the one with the biggest weight. The weight of a split is determined by the length of the word found. If the unknown substring is smaller than 5, it will be tested against a second dictionary of common words in the English language; words like "the", "he", "she" and more. If the word is not found, points will be deducted, if it is found the weight will increase substantially. A one-letter word like "I" and "a" will be divided if and only if the second substring is also a known word. This is due to the high probability of creating a false correction. During testing, this design lead to a ~5% corrector inaccuracy, versus an approximate 20%-30% error when allowing the "a" and "I" to be split from a second unknown word.

Pseudo-code:

```
procedure CorrectWhiteSpaces()
        foreach (word in Input)
          if(!Dictionary.contains(word)) then
                  CorrectedWords = AnalyseWord(word)
                  Print(CorrectedWords)
          end if
        end foreach
end procedure
procedure String[] AnalyseWord(String iWord)
        String wWord = iWord.toLowerCase()
        HashMap PossibleSplits
        for(int i = wWord.length/2 i < wWord.length i++)
                leftSS = wWord.substring(0, i)
                rightSS = wWord.substring(i)
                leftHit = Dictionary.contains(leftSS)
```

```
            rightHit = Dictionary.contains(rightSS)
            if(leftHit && rightHit) then
                wOutput[0] = leftSS
                wOutput[1] = rightSS
                return wOutput
            end if
            else if (leftHit && i < wWord.length - 1) then
                PossibleSplits.put(leftSS, i)
            end else if
            else if (rightHit && i < wWord.length - 1) then
                PossibleSplits.put(rightSS, i)
            end else if
            if(j>1) then
                --j
                leftSS = wWord.substring(0, j)
                rightSS = wWord.substring(j)
                leftHit = Dictionary.contains(leftSS)
                rightHit = Dictionary.contains(rightSS)
            if(leftHit && rightHit) then
                wOutput[0] = leftSS
                wOutput[1] = rightSS
                return wOutput
            end if
            else if (leftHit && j > 1) then
                PossibleSplits.put(leftSS, j)
            end else if
            else if (rightHit && j > 1) then
                PossibleSplits.put(rightSS, j)
            end else if
        end if
    end forloop
    if(!PossibleSplits.isEmpty()) wOutput = PickBestSplit(PossibleSplits, wWord)
```

```
return wOutput
end procedure
    procedure String[] PickBestSplit(HashMap<String, Integer> iOptions, String iWord)
        String[] wOutput = new String[2]
        int MaxWeight = 0
        int tempWeight, tempSplit
        int SplitIndex = iWord.length()
        String SS1, SS2
        boolean SS1Hit
        for( Map.Entry<String, Integer> entry : iOptions.entrySet())
            tempSplit = entry.getValue()
            SS1 = iWord.substring(0, tempSplit)
            SS2 = iWord.substring(tempSplit)
            SS1Hit = (Dictionary.contains(SS1))
            if(SS1Hit) then
                tempWeight = SS1.length() - 2
                if (CommonWords.containsKey(SS1) && (!(SS2.length() < 6) || CommonWords.containsKey(SS2))) then
                    tempWeight += CommonWords.get(SS1)
                end if
                if (SS2.length() < 4 && !CommonWords.containsKey(SS2)) then
                    tempWeight -= 6
                end if
            end if
            else then
                tempWeight = SS2.length() - 2
            if(SS2.length() >=3) then
                if (SS2.substring(SS2.length() - 3).equalsIgnoreCase("ing")) then
                    tempWeight -= 6
                end if
            end if
```

```
                if (CommonWords.containsKey(SS2) && !((SS1.length() < 5) ||
CommonWords.containsKey(SS1))) then
                    tempWeight += CommonWords.get(SS2)
                end if
                if (SS1.length() < 4 && !CommonWords.containsKey(SS1)) then
                    tempWeight -= 6
                end if
                end else
              if(tempWeight > MaxWeight)
                MaxWeight = tempWeight
                SplitIndex = tempSplit
            end if
            end forloop
            wOutput[0] = iWord.substring(0, SplitIndex)
            wOutput[1] = iWord.substring(SplitIndex)
            if(MaxWeight <= 0 || wOutput[0].lastIndexOf('-') == (wOutput[0].length() - 1) ||
wOutput[1].indexOf('-') == 0) then
                wOutput[0] = iWord
                wOutput[1] = ""
            end if
            return wOutput
        end procedure
```

Mutated Character Corrector

The mutated character corrector utilizes the Levenshtein distance method to measure the distance between two strings. The Levenshtein distance algorithm finds the minimum amount of insertion, deletion or swaps required to modify the one input string into the reference string. By modifying this algorithm to only allow swaps, and comparing a word unknown to the dictionary to a word included in the dictionary, it can be assumed that the unknown word has been mutated from the reference word in the dictionary and thus, will be replaced in the text. By using the standard Levenshtein distance algorithm, the mutated character corrector could also be used to correct words containing an extra character, or a word with a missing character. For example, using the standard Levenshtein distance algorithm, the word “assumtion” and “aessumption" would both have a distance of 1 with the reference word “assumption” and therefore would be corrected.

Pseudo-code:

```
procedure CorrectMutatedChars()
    File file = new File(FilePath)
    String[] test = FilePath.split(Matcher.quoteReplacement(System.getProperty("file.separator")))
    String FileName = test[test.length-1]
    String sts
    String CorrectedWord = ""
    foreach( word in inputFile)
        String wWord = word.toLowerCase()
        if(!Dictionary.contains(word) && !Dictionary.contains(wWord) &&
!CommonWords.containsKey(wWord)) then
            Dictionary.BSTIterator()
            while(Dictionary.hasNext())
                String wKey = Dictionary.next()
                if(wKey.length() == wWord.length() && wKey.length() > 1) then
                    if(LevenshteinDistance.getLevenshteinDistance(wKey, wWord)
== 1) then
                        CorrectedWord = wKey
                        break
                    end if
```

```
                end if
            end while
            if(CorrectedWord != "") then
                outSuggestions.println(word +", " + CorrectedWord)
                outCorrectedText.append(CorrectedWord + " ")
            end if
            else then outCorrectedText.append(word + " ")
        end if
        else then outCorrectedText.append(word + " ")
        CorrectedWord = ""
    end foreach
  end procedure
procedure getLevenshteinDistance(String s, String t)
    int n = s.length()
    int m = t.length()
    if (n == 0) then
       return m
    end if
    else if (m == 0)
       return n
    end else if
    if (n > m)
       String tmp = s
       s = t
       t = tmp
       n = m
       m = t.length()
    end if
    int p[] = new int[n+1]
    int d[] = new int[n+1]
    int _d[]
    int i
```

```
 int j
 char t_j
 int cost
 for (i = 0 i<=n i++)
    p[i] = i
 end forloop
 for (j = 1 j<=m j++)
    t_j = t.charAt(j-1)
    d[0] = j
   for (i=1 i<=n i++)
       cost = s.charAt(i-1)==t_j ? 0 : 1
       d[i] = Math.min(Math.min(d[i-1]+1, p[i]+1),  p[i-1]+cost)
    end forloop
    _d = p
    p = d
    d = _d
 end forloop
 return p[n]
end procedure
```

RESULTS AND ANALYSIS

This section will cover the results and analysis for each part of the project. To analyze the accuracy of the whitespace and the mutated character corrector, I will compare the frequencies obtained with the clean text input (18.txt) to the frequencies obtained after correction of part B and C (18.txt).

Word Frequency

The results for word frequency of input text 18.txt are as follows (excluding prepositions):

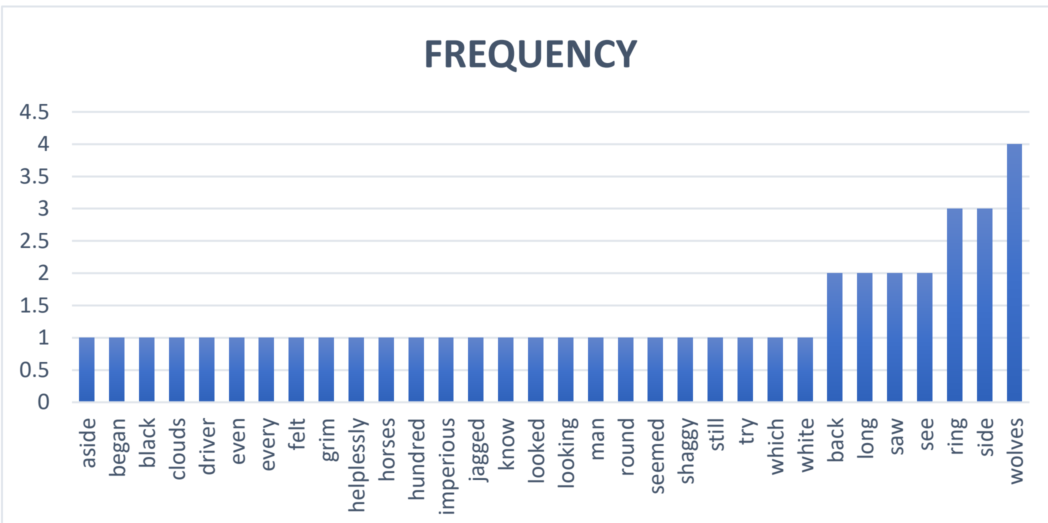


Figure 2: Results of Word Frequency Analysis for Text 18 of Part A

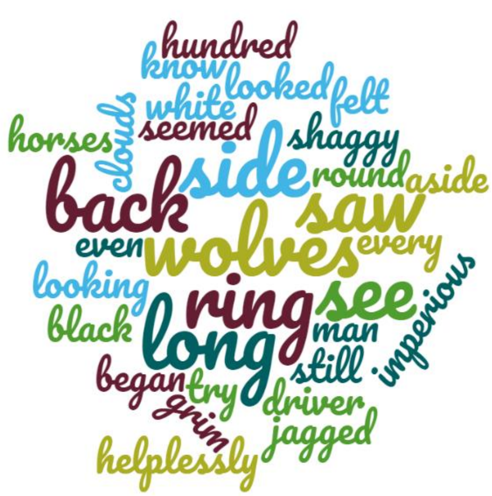


Figure 3: Word Cloud for Text 18 of Part A

Algorithm Runtime:

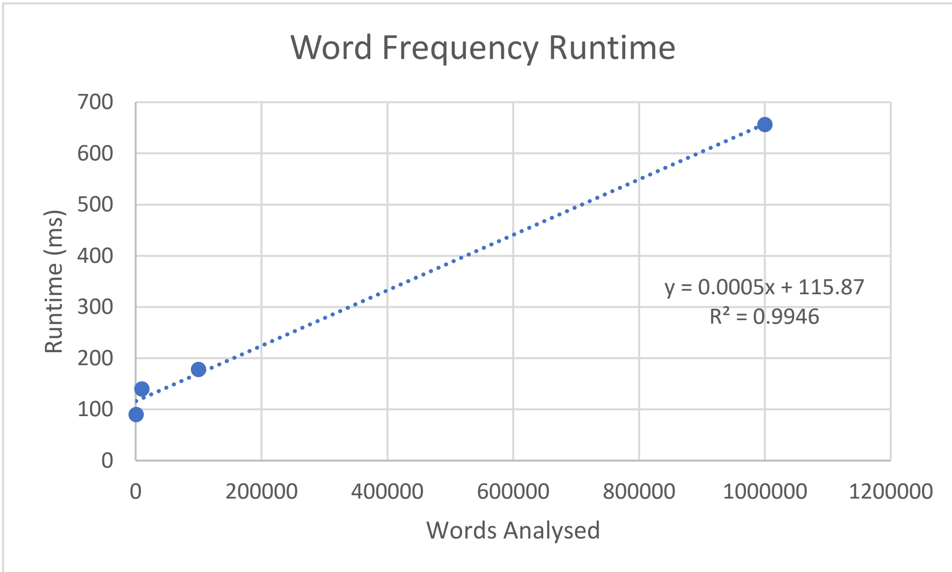


Figure 4: Word Frequency Analysis Runtime

The best fitting trend line for the results of the word frequency analysis points toward an algorithm with a time complexity of order O(N). The intercept at 115.87, indicates shows the overhead cost for the creation of the dictionary.

Whitespace Corrector

The results for word frequency of input text 18.txt of part B are as follows (excluding prepositions):

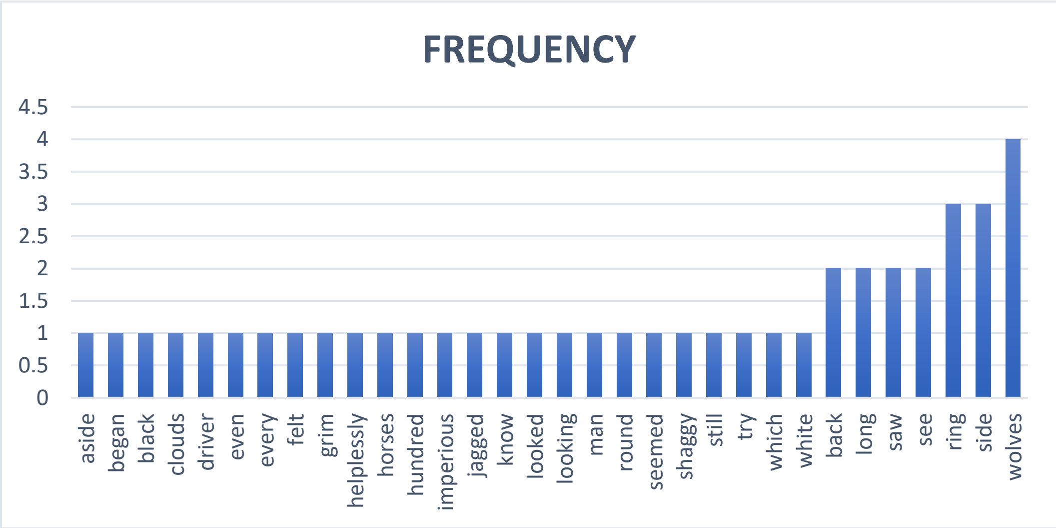
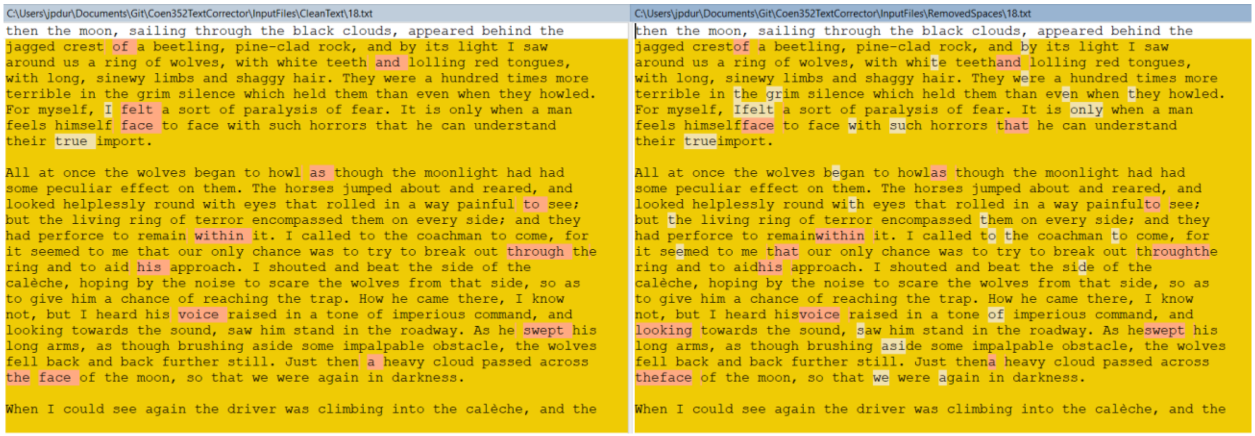


Figure 5: Results of Word Frequency Analysis for Text 18 of Part B

Figure 6: Word Cloud for Text 18 of Part B



C:\Users\jpdur\Documents\Git\Coen352TextCorrector\InputFiles\CleanText\18.txt

then the moon, sailing through the black clouds, appeared behind the
jagged crest of a beetling, pine-clad rock, and by its light I saw
around us a ring of wolves, with white teeth and lolling red tongues,
with long, sinewy limbs and shaggy hair. They were a hundred times more
terrible in the grim silence which held them than even when they howled.
For myself, I felt a sort of paralysis of fear. It is only when a man
feels himself face to face with such horrors that he can understand
their true import.

All at once the wolves began to howl as though the moonlight had had
some peculiar effect on them. The horses jumped about and reared, and
looked helplessly round with eyes that rolled in a way painful to see;
but the living ring of terror encompassed them on every side; and they
had perforce to remain within it. I called to the coachman to come, for
it seemed to me that our only chance was to try to break out through the
ring and to aid his approach. I shouted and beat the side of the
calèche, hoping by the noise to scare the wolves from that side, so as
to give him a chance of reaching the trap. How he came there, I know
not, but I heard his voice raised in a tone of imperious command, and
looking towards the sound, saw him stand in the roadway. As he swept his
long arms, as though brushing aside some impalpable obstacle, the wolves
fell back and back further still. Just then a heavy cloud passed across
the face of the moon, so that we were again in darkness.

When I could see again the driver was climbing into the calèche, and the

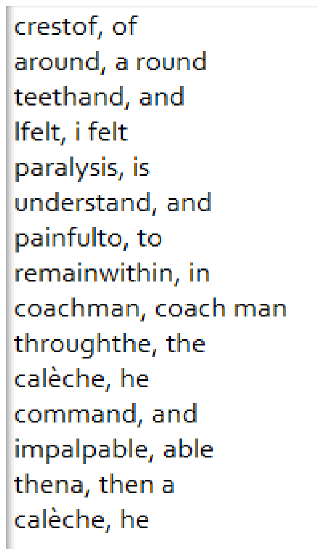
C:\Users\jpdur\Documents\Git\Coen352TextCorrector\InputFiles\RemovedSpaces\18.txt

then the moon, sailing through the black clouds, appeared behind the
jagged crestof a beetling, pine-clad rock, and by its light I saw
around us a ring of wolves, with white teethand lolling red tongues,
with long, sinewy limbs and shaggy hair. They were a hundred times more
terrible in the grim silence which held them than even when they howled.
For myself, Ifelt a sort of paralysis of fear. It is only when a man
feels himselfface to face with such horrors that he can understand
their trueimport.

All at once the wolves began to howlas though the moonlight had had
some peculiar effect on them. The horses jumped about and reared, and
looked helplessly round with eyes that rolled in a way painfulto see;
but the living ring of terror encompassed them on every side; and they
had perforce to remainwithin it. I called to the coachman to come, for
it seemed to me that our only chance was to try to break out throughthe
ring and to aidhis approach. I shouted and beat the side of the
calèche, hoping by the noise to scare the wolves from that side, so as
to give him a chance of reaching the trap. How he came there, I know
not, but I heard hisvoice raised in a tone of imperious command, and
looking towards the sound, saw him stand in the roadway. As heswept his
long arms, as though brushing aside some impalpable obstacle, the wolves
fell back and back further still. Just thena heavy cloud passed across
theface of the moon, so that we were again in darkness.

When I could see again the driver was climbing into the calèche, and the

Figure 7: Comparing both Inputs Using WinMerge



crestof, of
around, a round
teethand, and
Ifelt, i felt
paralysis, is
understand, and
painfulto, to
remainwithin, in
coachman, coach man
throughthe, the
calèche, he
command, and
impalpable, able
thena, then a
calèche, he

Figure 8: Corrections Found by Whitespace Corrector

At first look, it seems like the corrector missed multiple errors, however, when debugging, I found that the errors missed by the corrector are to be expected as these words do not appear in the dictionary. Notice also that the corrector split around into two words, as “around” does not appear in the dictionary, but “a” and “round” does.

Algorithm Runtime:

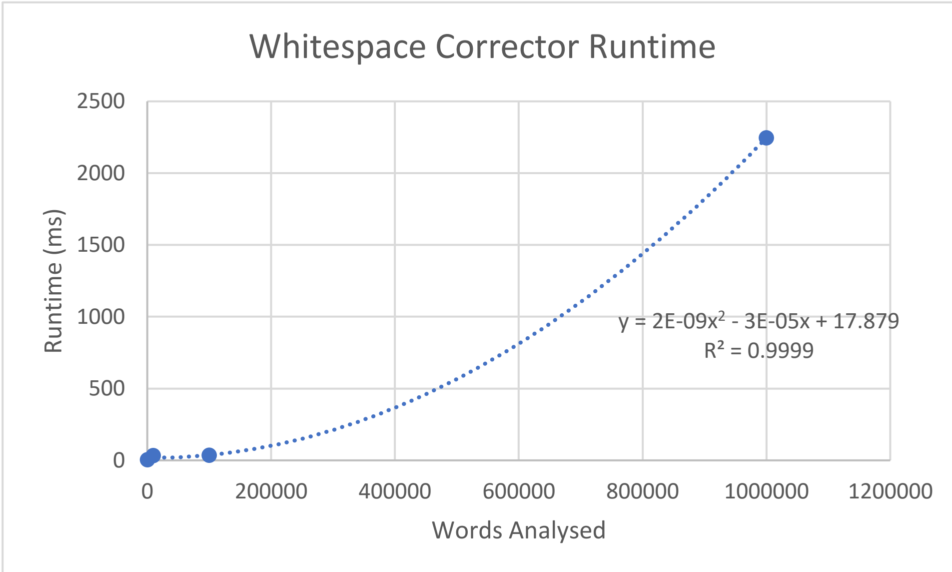


Figure 9: Whitespace Corrector Runtime

For the whitespace corrector, the overhead cost is low as the dictionary has already been created. The best fitting trend line for this algorithm is a polynomial of power 2; hinting towards a time complexity of $O(N^2)$

Mutated Character Corrector

The results for word frequency of input text 18.txt of part C are as follows (excluding prepositions):

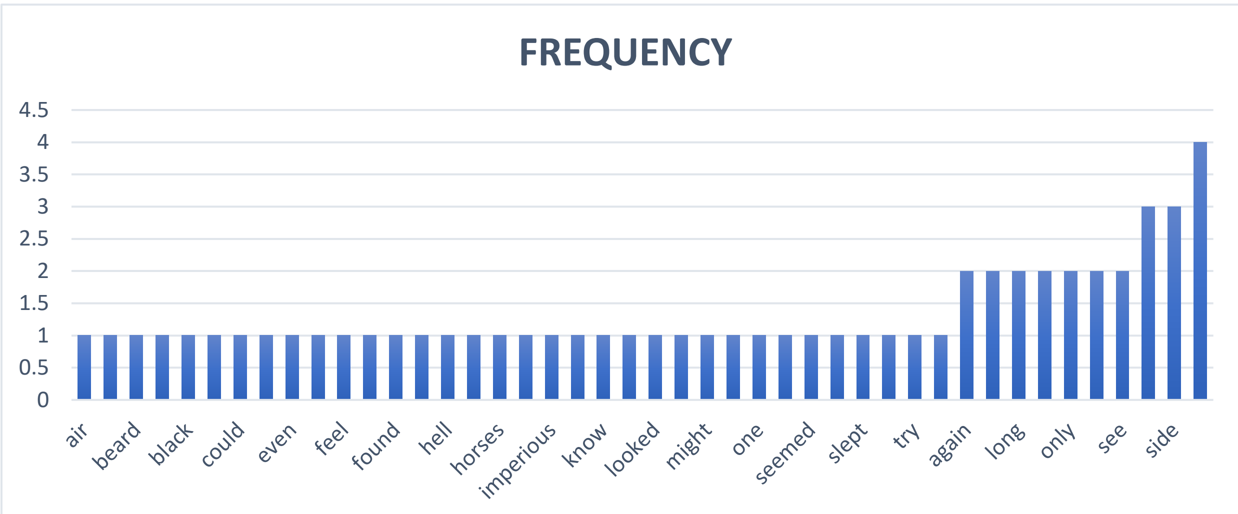
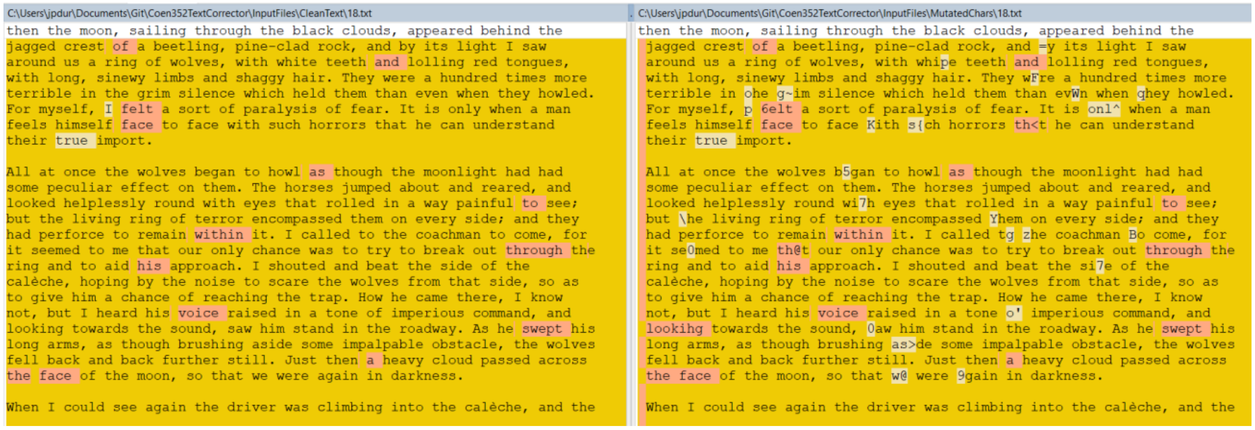


Figure 10: Results of Word Frequency Analysis for Text 18 of Part C

Figure 11: Word Cloud for Text 18 of Part C



```
C:\Users\jpdur\Documents\Git\Coen352TextCorrector\InputFiles\CleanText\18.txt
then the moon, sailing through the black clouds, appeared behind the
jagged crest of a beetling, pine-clad rock, and by its light I saw
around us a ring of wolves, with white teeth and lolling red tongues,
with long, sinewy limbs and shaggy hair. They were a hundred times more
terrible in the grim silence which held them than even when they howled.
For myself, I felt a sort of paralysis of fear. It is only when a man
feels himself face to face with such horrors that he can understand
their true import.

All at once the wolves began to howl as though the moonlight had had
some peculiar effect on them. The horses jumped about and reared, and
looked helplessly round with eyes that rolled in a way painful to see;
but the living ring of terror encompassed them on every side; and they
had perforce to remain within it. I called to the coachman to come, for
it seemed to me that our only chance was to try to break out through the
ring and to aid his approach. I shouted and beat the side of the
calèche, hoping by the noise to scare the wolves from that side, so as
to give him a chance of reaching the trap. How he came there, I know
not, but I heard his voice raised in a tone of imperious command, and
looking towards the sound, saw him stand in the roadway. As he swept his
long arms, as though brushing aside some impalpable obstacle, the wolves
fell back and back further still. Just then a heavy cloud passed across
the face of the moon, so that we were again in darkness.

When I could see again the driver was climbing into the calèche, and the
```

```
C:\Users\jpdur\Documents\Git\Coen352TextCorrector\InputFiles\MutatedChars\18.txt
then the moon, sailing through the black clouds, appeared behind the
jagged crest of a beetling, pine-clad rock, and =y its light I saw
around us a ring of wolves, with whipe teeth and lolling red tongues,
with long, sinewy limbs and shaggy hair. They wFre a hundred times more
terrible in ohe g~im silence which held them than evWn when qhey howled.
For myself, p 6elt a sort of paralysis of fear. It is onl^ when a man
feels himself face to face Kith s{ch horrors th<t he can understand
their true import.

All at once the wolves b5gan to howl as though the moonlight had had
some peculiar effect on them. The horses jumped about and reared, and
looked helplessly round wi7h eyes that rolled in a way painful to see;
but \he living ring of terror encompassed Yhem on every side; and they
had perforce to remain within it. I called tg zhe coachman Bo come, for
it se0med to me th@t our only chance was to try to break out through the
ring and to aid his approach. I shouted and beat the si7e of the
calèche, hoping by the noise to scare the wolves from that side, so as
to give him a chance of reaching the trap. How he came there, I know
not, but I heard his voice raised in a tone o' imperious command, and
lookihg towards the sound, 0aw him stand in the roadway. As he swept his
long arms, as though brushing as>de some impalpable obstacle, the wolves
fell back and back further still. Just then a heavy cloud passed across
the face of the moon, so that w@ were 9gain in darkness.

When I could see again the driver was climbing into the calèche, and the
```

Figure 12: Comparing both Inputs Using WinMerge

=y, by	tg, to
light, might	zhe, the
us, as	Bo, by
whipe, white	come, some
wFre, were	seomed, seemed
ohe, one	th@t, that
g~im, grim	out, but
held, hell	aid, air
evWn, even	si7e, side
when, then	not, got
qhey, they	heard, beard
6elt, felt	o', of
onl^, only	lookihg, looking
when, then	sound, found
Kith, with	oaw, saw
s{ch, such	swept, slept
th<t, that	as>de, aside
can, man	fell, feel
b5gan, began	Just, must
wi7h, with	w@, we
way, lay	9gain, again
\he, the	When, then
Yhem, them	

Figure 13: Mutated Character Corrected

The mutated character corrector was able to correct every error, and while it is to be expected, it also corrected words that do not appear in the dictionary but that had a distance of 1 with a word contained in the dictionary; e.g. way was corrected into lay.

USER MANUAL:

The Text Corrector has three functions; using a user provided dictionary, the program can analyze the usage frequency of the words contained in the dictionary in each input text. It can divide unknown words by adding a whitespace to possibly correct a forgotten whitespace between one or two words contained in the dictionary. Finally, the corrector can correct a mutated character within a word known to the dictionary.

SETUP:

1) In the source folder of the text corrector, locate the InputFiles folder.
2) Copy the dictionary you wish to use in the InputFiles folder. Make sure each word is separate by one (1) whitespace character.
3) Copy the texts you wish to analyze or correct in one of the three subfolders (CleanText, RemovedSpaces, MutatedChars). Each folder represents one of the functions described above the setup.
4) Rename all files from 0 to N. (N being the number of files to analyze – 1)
5) Using your preferred IDE, open the source files located in the SourceFiles folder.
6) In main.java, modify the FilesToCorrect integer to match the number of files you wish to correct.
7) Run main.java.
8) The results will be saved in the OutputFiles folder.