COMP122 - Assessment 2

Information

Name:	Oliver Legg
Email:	O.Legg@student.liverpool.ac.uk
Student ID:	201244658

Requirements

Part 1

For part 1, I am required to create a program that will crack a given Caesar cipher text. I will have to assume that the cipher text has been produced using a Caesar cipher by some (unknown) shift. This means I will have to search through all the possible shifts and find the one which makes sense.

I will need to do the following tasks in my requirements

- Display the original plain text on the screen
- Spaces, any punctuation is left untouched (don't shift them)
- Don't change lower and upper-case letters in the shifts. For example, 'Ifmmp Uifsf' -> 'Hello There'
- I will have to know which texts are closest to English. So, when I brute force the shifts, the shift which is most similar to English, is the one that I keep.

Part 2

For part 2, I will be computing TFIDF values for a collection of documents.

For this part, I will have to call my program **"TFIDF.java"**. When I use the program, instead of running it and then inputting which files I want using the `Scanner` class, I am required to input it like so:

```
java TFIDF Adv-3.txt Dunwich.txt Chapter-4-frank.txt time.txt
```

From doing this, my program should see I entered 4 text files and my program should interpret what that means.

When my TFIDF calculates, for each document, it will have to output the highest TFIDF value. If there are several such words, print out any one of those words with the highest TFIDF value. For example, if I ran:

```
java TFIDF Adv-3.txt when.txt fear.txt
```

in when.txt there are words like

```
necessary=0.012542916485999216
```

and

`when=0.012542916485999216`

which share the same value. It doesn’t matter if it’s either one, as long as it’s the lowest value, it meets the requirements.

```
==========
when.txt
==========
necessary  0.012542916485999216

{which=0.0, necessary=0.012542916485999216, in=0.012542916485999216,
one=0.012542916485999216, another=0.012542916485999216,
for=0.012542916485999216, political=0.012542916485999216,
them=0.012542916485999216, it=0.012542916485999216,
bands=0.012542916485999216, when=0.012542916485999216,
people=0.012542916485999216, becomes=0.012542916485999216, the=0.0,
connected=0.012542916485999216, with=0.012542916485999216, of=0.0,
dissolve=0.012542916485999216, have=0.0, course=0.012542916485999216, to=0.0,
human=0.012542916485999216, events=0.012542916485999216}

==========
fear.txt
==========
fear  0.0177076468037636

{needed=0.0088538234018818, unreasoning=0.0088538234018818,
convert=0.0088538234018818, unjustified=0.0088538234018818, we=0.0088538234018818,
advance=0.0088538234018818, firm=0.0088538234018818, that=0.0088538234018818,
into=0.0088538234018818, assert=0.0088538234018818, of=0.0,
me=0.0088538234018818, only=0.0088538234018818, have=0.0, let=0.0088538234018818,
nameless=0.0088538234018818, so=0.0088538234018818, belief=0.0088538234018818,
fear=0.0177076468037636, all=0.0088538234018818, which=0.0,
terror=0.0088538234018818, paralyzes=0.0088538234018818, is=0.0088538234018818,
my=0.0088538234018818, the=0.0, retreat=0.0088538234018818,
itself=0.0088538234018818, efforts=0.0088538234018818, to=0.0,
thing=0.0088538234018818, first=0.0088538234018818}
```

Analysis and Design

Part 1

Part 1 is creating a program that will crack a given Caesar cipher text. I will have to assume that the cipher text has been produced using a Caesar cipher by some (unknown) shift. This means I will have to search through all the possible shifts and find the one which makes sense.

The closeness algorithm is done like so:

$$x^2 = \sum_{\propto=\alpha}^{z} \frac{(\text{freq} \propto\; - English \propto)^2}{English\; \propto}$$

The iteration of this algorithm is done on every letter of a string.
Frequency is the number of times that letter appears in the string divided by the length of the string.
English is the known frequency of a letter.
This then loops to the next letter index of the string and repeats until the string has no more letters left.

For example:
On the string "aaabbc".
I start at letter 'a'. freq = **(3/6)** = **0.5**.
I know that the known frequency for a is predefined and is **0.0855**.
I apply the two numbers together. **(0.5** - **0.0855)** ^{2} = **0.17181025**.
(**0.17181025**/**0.0855**) = **2.00947661**
Afterwards I'd apply this to the next letter of the string. Then sum up the result for each letter.
The lowest number produced for each string, is closest to English.

I plan to put into the information on the program by entering it when you run it. For example:

```
java BreakCaesar "Wsccsccszzs novdk lveoc"
```

Then the information should output like so:

```
=================
Wsccsccszzs novdk lveoc
=================
Mississippi delta blues
```

With my program, I want it to split it up into different functions. For example, the main class will handle the user input when the program starts and will be the place that calls all of the functions. I intend my main functions to be called names like shift() which takes a message and shifts it in a certain direction. Another function called by getValue which takes a message and returns its value based on it's closeness to English. I will present my full design in UML on the next page.

Class diagram

BreakCaesar
+knownFreq: double[] +alphabetlower: Char[] +alphabetupper: Char[]
+main(args: String[]): void +shifttext(shiftnum: int, text: String): String +getStringValue(text: String): double +frequency(text: String, letter: char): int +decrypt(text: String): String

Class attributes

The `knownFrequency`, `alphabetlower`, `alphabetupper` is a public variable because all the functions use this. There's no point in passing these as parameters in functions as they're constants.

Main Function

I plan the main functions will host the decrypt function which does all the maths and hosts the rest of the functions. The main function will handle the user input and print all the information.

Shift Text

I plan this to return the string value entered but every letter's index in the alphabet is changed by the amount that is entered the function.

Get String Value

This gets the value of the string. I plan this to take any string and measure its closeness to English. The closer it is, the lower the value.

Decrypt

The decrypt function will return the string which has the closest value to 0. It will loop through all 26 shifts and measure the value by using the shift text and get string value.

Pseudocode

```
FUNCTION main(args):
        GLOBAL alphabetlower
        GLOBAL alphabetupper
        IF args.LENGTH < 1 THEN
                EXIT()
        FOR i IN "a","z" DO
                        LIST alphabetlower.add(i)
                LIST alphabetupper.add(i.UPPERCASE())
        OUTPUT("==============================")
        OUTPUT(args[i])
        OUTPUT("==============================")
        OUTPUT(decrypt(args[i]))

FUNCTION shifttext(shift_num, text):
        stringLENGTH <- text.LENGTH()
        finalstring <- ""
        character_in_alphabet <- FALSE
        FOR i IN (0, text.LENGTH()) DO
                c <- text[i] //gets the first letter of the string and puts it in
                FOR v IN (0,26) DO
                        IF (c == alphabetlower[v] OR c == alphabetupper[v]) THEN
                                character_in_alphabet <- TRUE
                                IF (c.ISUPPER()) THEN
                                        finalstring <- finalstring + alphabetupper[(v + shift_num MOD 26)]
                                ELSE
                                        finalstring <- finalstring + alphabetlower[(v + shift_num MOD 26)]
                                BREAK
                IF (!character_in_alphabet) THEN
                        finalstring <- finalstring + c
                character_in_alphabet <- FALSE
        RETURN finalstring


FUNCTION getStringValue(text):
```

```
        finalnumber <- 0
        power <- 0
        FOR i IN (0, text.LENGTH()) DO
                FOR v IN (0,26) DO
                        c <- i[v]
                        IF (c == alphabetupper[v] OR c == alphabetlower[v]) THEN
                                power <- (frequency(text, c) - knownFreq[v])^2
                                finalnumber <- finalnumber + (power / (knownFreq[v]))
        RETURN finalnumber

FUNCTION frequency(text, letter):
        count <- 0
        FOR i IN (0, text.LENGTH()) DO
                IF (text[i] == letter) THEN
                        count+=1
        RETURN count


FUNCTION decrypt(text):
        value <- getStringValue(text)
        newtext <- text
        FOR i IN (0,26) DO
                IF (getStringValue(shifttext(i,text)) < value) THEN
                        value <- getStringValue(shifttext(i,text))
                        newtext <- shifttext(i,text)
        RETURN newtext
```

Name: Oliver Legg
Student ID: 201244658

Part 2

For part 2, I will be calculating the TFIDF values for a collection of documents. TFIDF stands for term frequency inverse document frequency. The point of this is to find important words for document to find a point of interest. More specifically, the calculation is denoted like so:

$$tfidf(t, d_{\mathrm{i}}, \mathrm{D}) = tf(t, d_{\mathrm{i}}) \cdot idf(t, D)$$

t = term.

d_{i} = a specific document

D = all the documents

tf = returns the number of occurrences of t in d_{i} divided by the total number of words in d_{i}

idf = returns $\log\left(\frac{n}{number\ of\ documents\ containing\ t}\right)$ with n being the total number of documents

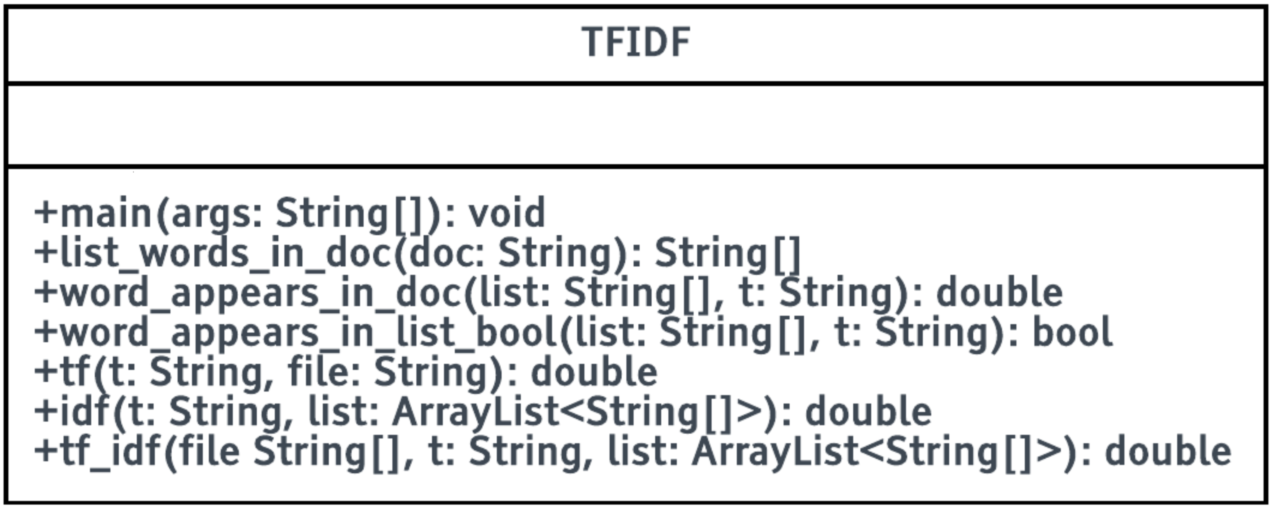
$tf \cdot idf$ = returns $tf(t, d_{\mathrm{i}}) \cdot idf(t, D)$

Using these values, I will be able to find the most important word of a document. The number with the highest value is considered the most important.

To help make this program, I intend to do each part of the TFIDF function by function. I know that I will have to get all the words from a document at some point because I am comparing find how many times x appears in a document therefore, I will create a function that returns words to a list. From this I can use this list to reference other documents. I can create other functions that can get the number of times x appears in a document as well as getting all the words into a list. However, when I test my program in the future, it is most important to test the maths because there is more chance of error appearing there as the error can go unnoticed as it is a logical error.

O.Legg@student.liverpool.ac.uk

Class diagram



Main class

The main class will be the section which gets all the words from the list words in document function and puts it in the TFIDF function. I will also check which words produce the largest value in the main class.

List words in document

This gets all the words out of the document and returns it in a string[] of words. This is called once in the main class and is passed into the tf_idf.

Word appears in document

This returns the number of times a given word appears in a document. This is used by the TF function.

Word appears in list Boolean

This returns a Boolean if a given word is in a document. Returns true if it's in, false if it's not.

TF

Term frequency $tf(t, d_i)$ returns the number of words in a document divided by the number of times a given word appears in a document

IDF

The idf $idf(t, D)$ returns the number of occurrences of t in d_i divided by the total number of words in d_i

TFIDF

Multiplies the TF with IDF results back.

Pseudocode

```
FUNCTION list_words_in_doc(String doc):
	strlist <- LIST[STRING]
	line = READLINE(doc)
	WHILE (line /= NULL)
		//splits up all of the words. so that words words with '.' or ',' aren't considered part of the word
		words <- line.REMOVEPUNC() //this puts all of the words in the line into a list too
		line <- FILEREAD(doc)
		FOR i IN (0,words.LENGTH) DO
			IF (words[i] == NULL OR words[i] == "") THEN
				PASS
			ELSE
				strlist.ADD(words[i])
	RETURN strarray


//returns the number of times a given word in the document appears.
FUNCTION word_appears_in_doc(l, String t):
	counter <- 0
	FOR i IN (0, l.LENGTH) DO
		IF (l[i] == t) THEN
			counter++
	RETURN counter

//returns TRUE of FALSE whether t is in a document or not
FUNCTION word_in_list_bool(String[] list, String t):
	FOR i IN (0, list.LENGTH) DO
		IF (list[i] == t) THEN
			RETURN TRUE
	RETURN FALSE




//Calculate term frequency
```

```
FUNCTION tf(t, list):
        RETURN word_appears_in_doc(list, t)/list.LENGTH
FUNCTION idf(t, list):
        tc <- 0 //TC is the term count. Term count is the number of documents that contain the term
        FOR x IN (0,list.LENGTH) DO //lists over every file
                IF (word_in_list_bool(list.GET(x), t))
                        tc++
        RETURN LOG(list.SIZE() / tc)


FUNCTION tf_idf(file, t, list):
        IF (list.SIZE() > 1 THEN)
                RETURN (tf(t, file)*idf(t, list))
        ELSE:
                RETURN tf(t, file)

//main part of the program takes the arguments of the txt file names and locations when the program executes.
FUNCTION main(String[] args):
        IF (args.LENGTH < 1) THEN
                OUTPUT("Oops, you haven't given enough parameters! Usage: java TFIDF \"filename.txt\" ...")
                EXIT(1)
        IF (args.LENGTH == 1) THEN
                OUTPUT ("\nMax TFIDF value FOR this file.\n")
        ELSE
                OUTPUT ("\nMax TFIDF value FOR each file.\n")

        //This is where all of the inverse term frequency data is being held.
        list <- LIST[STRING[]]
        tfidflist <- HASHMAP
        FOR i IN (0, args.LENGTH) DO
                list.ADD(list_words_in_doc(args[i]))


        //FOR loop that runs FOR the number of txt files entered to the program.
        FOR i IN (0, args.LENGTH) DO
```

```
        FOR v IN (0, list.GET(i).LENGTH) DO
            IF (NOT tfidflist.CONTAINS(list.GET(i)[v]))
                tfidflist.PUT(list.GET(i)[v], tf_idf(list.GET(i), list.GET(i)[v], list))

    //FOR loop that runs FOR the number of txt files entered to the progrma.
    FOR i IN (0, args.LENGTH) DO
        highest_value <- 0.0 //This is defined FOR the next section
        highest_word <- "no data" //this data will also appear IF no important words are found.
        OUTPUT ("==========")
        OUTPUT (args[i])
        OUTPUT ("==========")
        FOR v IN (0, list.GET(i).LENGTH) DO
            IF (tfidflist.GET(list.GET(i)[v]) >= highest_value) THEN
                highest_value <- tfidflist.GET(list.GET(i)[v])
                highest_word <- list.GET(i)[v]
        OUTPUT(highest_word)
        OUTPUT(highest_value)
```

Testing

Part 1

Test table

Test number	Description	Expected Results	Actual results	Remedial Action
1	Non-alphanumeric characters	Entering in punctuation must not shift them	Didn’t shift non-alphanumeric characters	-
2	No parameters	Entering nothing as a parameter should create a helpful error	Entering nothing made a useful error	-
3	htcs aplntgh, vjch, pcs bdctn	send lawyers, guns, and money	send lawyers, guns, and money	-
4	aqw?nn pgxgt wpfgtuvcpf vjku	you?ll never understand this	you?ll never understand this	-
5	Wsccsccszzs novdk lveoc	Mississippi delta blues	Mississippi delta blues	-
6	Vlcha siol mqilx. Qy unnuwe un xuqh.	Bring your sword. We attack at dawn.	Bring your sword. We attack at dawn.	-
7	Hvs eiwqy pfckb tcl xiadsr cjsf hvs zonm rcug.	The quick brown fox jumped over the lazy dogs.	The quick brown fox jumped over the lazy dogs.	-
8	Captital letters must not be shifted test data: Hvs eiwqy pfckb tcl xiadsr cjsf hvs zonm rcug	The quick brown fox jumped over the lazy dogs.	The quick brown fox jumped over the lazy dogs.	-

Name: Oliver Legg
Student ID: 201244658

Evidence table

Test no	Description	Evidence
1	Non-alphanumeric characters	λ java TFIDF when.txt fear.txt Max TFIDF value for each file. ========= when.txt ========= another 0.012542916485999216 ========= fear.txt ========= fear 0.0177076468037636
2	No parameters	λ java BreakCaesar Oops, you haven't given enough parameters! Usage: java BreakCaesar "string"
3	htcs aplntgh, vjch, pcs bdctn	λ java BreakCaesar "htcs aplntgh, vjch, pcs bdctn" ============================== htcs aplntgh, vjch, pcs bdctn ============================== send lawyers, guns, and money
4	aqw?nn pgxgt wpfgtuvcpf vjku	λ java BreakCaesar "aqw'nn pgxgt wpfgtuvcpf vjku" ============================== aqw?nn pgxgt wpfgtuvcpf vjku ============================== you?ll never understand this
5	Wsccsccszzs novdk lveoc	λ java BreakCaesar "Wsccsccszzs novdk lveoc" ============================== Wsccsccszzs novdk lveoc ============================== Mississippi delta blues
6	Vlcha siol mqilx. Qy unnuwe un xuqh.	λ java BreakCaesar "Vlcha siol mqilx. Qy unnuwe un xuqh." ============================== Vlcha siol mqilx. Qy unnuwe un xuqh. ============================== Bring your sword. We attack at dawn.

7	Hvs eiwqy pfckb tcl xiadsr cjsf hvs zonm rcug.	`λ java BreakCaesar "Hvs eiwqy pfckb tcl xiadsr cjsf hvs zonm rcug."`
		`==============================`
		`Hvs eiwqy pfckb tcl xiadsr cjsf hvs zonm rcug.`
		`==============================`
		`The quick brown fox jumped over the lazy dogs.`

8	Captital letters must not be shifted
	test data:
	
	Hvs eiwqy pfckb tcl xiadsr cjsf hvs zonm rcug
		`λ java BreakCaesar "Hvs eiwqy pfckb tcl xiadsr cjsf hvs zonm rcug."`
		`==============================`
		`Hvs eiwqy pfckb tcl xiadsr cjsf hvs zonm rcug.`
		`==============================`
		`The quick brown fox jumped over the lazy dogs.`

Part 2

Test table

Test no	**Description**	**Expected Results**	**Actual results**	**Remedial Action**
1	User input in program	File reads the input as expected and the program interprets the rest	Yes	-
2	Maths behind TF function When.txt Fear.txt looking for the term fear. Specifically searching in fear.txt	2 / 34 = 0.0588235294	Yes 0.0588235294	-
3	IDF function total number of documents / no documents containing t $idf(t, D) = log\left(\frac{no\ docs}{t\ in\ no\ docs}\right)$ when.txt fear.txt looking for the term fear. Specifically searching in fear.txt	log(2/1) = 0.30102999566	Yes 0.30102999566	-
4	TFIDF function when.txt fear.txt looking for the term fear. Specifically searching in fear.txt	0.0588235294*0.30102999566 Ans = 0.0177076468	Yes 0.0177076468	-
5	Passing no parameters when user starts the program	There should be an error message saying “Oops, you haven't given enough parameters! Usage: java TFIDF "filename.txt" ...”	There was an error message saying: “Oops, you haven't given enough parameters! Usage: java TFIDF "filename.txt" ...”	-
6	Passing incorrect parameters when user starts the program	There should be an error message saying “no data”	There was an error message saying: “no data”	-
7	Passing one file as a parameter	The highest value should be 0	The highest value was 0	-

8	Comparing test data with assignment requirement sheet.
	
	java TFIDF Adv-3.txt Dunwich.txt Chapter-4-Frank.txt
		`==========`
		`Adv-3.txt`
		`==========`
		`holmes  0.0031003782620574196`
		
		`==========`
		`Dunwich.txt`
		`==========`
		`whateley  0.00199966996948726`
		
		`==========`
		`Chapter-4-Frank.txt`
		`==========`
		`pursuit  0.001313349895020699`
			Yes	-

9	java TFIDF Adv-3.txt Dunwich.txt Chapter-4-Frank.txt time.txt	========== Adv-3.txt ========== holmes 0.003912241785716382 ========== Dunwich.txt ========== whateley 0.002523302562145693 ========== Chapter-4-Frank.txt ========== feelings 0.0014205111867745869 ========== time.txt ========== weena 9.886042550541253E-4	========== Adv-3.txt ========== holmes 0.00391556052609731 ========== Dunwich.txt ========== whateley 0.0025282474439864045 ========== Chapter-4-Frank.txt ========== feelings 0.0014205111867745869 ========== time.txt ========== weena 9.883337750937822E-4	
10	java TFIDF Chapter-4-Frank.txt	========== Chapter-4-Frank.txt ========== the 0.053480141565080616	λ java TFIDF Chapter Max TFIDF value for ========== Chapter-4-Frank.txt ========== the 0.0534801415650	-
11	java TFIDF when.txt Chapter-4-Frank.txt	$ java TFIDF when.txt Chapter-4-Frank.txt Max TFIDF value for each file. ========== when.txt ========== necessary 0.012542916485999216 ========== Chapter-4-Frank.txt ========== i 0.012192721019815203	========== when.txt ========== another 0.0125429164 ========== Chapter-4-Frank.txt ========== i 0.012192721019815	-

Evidence Table

Test no	Description	Evidence
1	User input in program	λ java TFIDF when.txt fear.txt Max TFIDF value for each file. ========== when.txt ========== another 0.012542916485999216 ========== fear.txt ========== fear 0.0177076468037636
2	Maths behind TF function When.txt Fear.txt looking for the term fear. Specifically searching in fear.txt	λ java TFIDF when.txt fear.txt Max TFIDF value for each file. ========== when.txt ========== another 0.012542916485999216 TF fear = 0.058823529411764705 IDF fear = 0.3010299956639812 ========== fear.txt ========== fear 0.0177076468037636 TF fear = 0.058823529411764705
3	IDF function Total number of documents / no documents containing t $idf(t,D) = log\left(\frac{no\ docs}{t\ in\ no\ docs}\right)$ when.txt fear.txt looking for the term fear. Specifically searching in fear.txt	λ java TFIDF when.txt fear.txt Max TFIDF value for each file. ========== when.txt ========== another 0.012542916485999216 TF fear = 0.058823529411764705 IDF fear = 0.3010299956639812 ========== fear.txt ========== fear 0.0177076468037636 TF fear = 0.058823529411764705 IDF fear = 0.3010299956639812
4	TFIDF function when.txt fear.txt looking for the term fear. Specifically searching in fear.txt	λ java TFIDF when.txt fear.txt Max TFIDF value for each file. ========== when.txt ========== another 0.012542916485999216 TF fear = 0.058823529411764705 IDF fear = 0.3010299956639812 ========== fear.txt ========== fear 0.0177076468037636 TF fear = 0.058823529411764705
5	Passing no parameters when user starts the program	λ java TFIDF Oops, you haven't given enough parameters! Usage: java TFIDF "filename.txt" ...

6	Passing incorrect parameters when user starts the program	λ java TFIDF whin.txt feare.tx Max TFIDF value for each file. ========== whin.txt ========== no data 0.0 ========== feare.tx ========== no data 0.0 Notice that when and fear are spelt wrong
7	Passing one file as a parameter	λ java TFIDF time.txt Max TFIDF value for each file. ========== time.txt ========== man 0.0
8	Passing correct parameters	λ java TFIDF Adv-3.txt Dunwich.txt Chapter-4-Frank.txt Max TFIDF value for each file. ========== Adv-3.txt ========== holmes 0.0031030083015841188 ========== Dunwich.txt ========== whateley 0.0020035886956312815 ========== Chapter-4-Frank.txt ========== pursuit 0.001313349895020699
9	java TFIDF Adv-3.txt Dunwich.txt Chapter-4-Frank.txt time.txt	========== Adv-3.txt ========== holmes 0.00391556052609731 ========== Dunwich.txt ========== whateley 0.0025282474439864045 ========== Chapter-4-Frank.txt ========== feelings 0.0014205111867745869 ========== time.txt ========== weena 9.883337750937822E-4
10	java TFIDF Chapter-4-Frank.txt	λ java TFIDF Chapter-4-Frank.txt Max TFIDF value for each file. ========== Chapter-4-Frank.txt ========== the 0.053480141565080616

11	java TFIDF when.txt Chapter-4-Frank.txt	========== when.txt ========== another 0.012542916485999216 ========== Chapter-4-Frank.txt ========== i 0.012192721019815203	

Extra questions

Part 1

1. What would we do differently if we know the language we're examining isn't English but some other language (e.g. suppose we know the people communicating via this Caesar cipher usually writes/speaks in Polish)?

    To calculate English, we know the letters frequency is

```
public static double[] knownFreq = {0.0855, 0.0160, 0.0316, 0.0387, 0.1210,
        0.0218, 0.0209, 0.0496, 0.0733, 0.0022,
        0.0081, 0.0421, 0.0253, 0.0717, 0.0747,
        0.0207, 0.0010, 0.0633, 0.0673, 0.0894,
        0.0268, 0.0106, 0.0183, 0.0019, 0.0172,
        0.0011};
```

    Frequencies for English will be different to polish. For example, the most used letter in English is E. The most used letter in polish is A. Therefore the knownFreq will have to have different values.

2. Suppose we (somehow) know that the person doing the encryption uses one shift value for lowercase letters, and a different shift value for uppercase letters. What would we have to do differently? How would that affect our calculations, or how would we have to alter our program/calculations to account for this?

    To implement the permutations over upper and lowercase shifts, you would have to edit your program by adding another loop and which shifts the uppercase characters and lowercase characters of the string. You would need a system that makes only a specific letter case shift in a direction. For this we would have to try all possible shift values again. I would use 2 functions. ShiftUpper(c, n) and shiftLower(c, n). I would pass 2 parameters in each function. The character to shift would be c. the amount to shift would be an integer number n. If there was the encrypted string, "Pg", you would have to loop like so:

    X=Uppercase characters
    Y=Lowercase characters

x=0	y=0	x=1	y=0	…	…	x=25	y=0	x=0	y=1	x=1	y=1
Pe		Qe		…		Og		Of		Pf	

    Implementing this would affect the calculation despite the closeness of the string still calculating the closeness to English. Unfortunately, there are far more permutations (26*26 = 676 instead of just 26). This means it's likely to get a word like with letters that are frequent. Especially with names which capital letters are frequent. Most sentences have very few capital letters. If there is only one capital letter in the string. It would always go to the most frequent letter (like 'e'). Instead of using the previous calculation, you could compare the shifts with the most common English phrases like 'the'. You might even go further to get the shift that produces the most words in the English dictionary. This would take a lot longer to computer as not only are you doing 676 permutations, you also checking to see what words in the shift are in the English dictionary.

Part 2

1. If we are considering only a single document (or file in our case), as stated the idf value doesn't make any sense, because it will always be 0 for any word in the document. (There is a single document, so n = 1, and any word in that document obviously appears in the document so idf(t, d1, D) = log(1/1) = log(1) = 0.) So how could we proceed in this case? Instead of tf idf values, I suggested considering tf values alone, but then words with high tf values are likely to be non-interesting words like "and", "the", and "or". (See the above example where I had only one file, and the word "the" had the highest tf value.) Would you have any suggestions how we might change our approach?

   The best method I would use to solve this problem is split the document into 2 or maybe even more. If it's a book, I could split the document up into chapters so that the IDF could be used in the calculation.