Jacob Chapman

11/19/2013

CS 301

Project 2 Report

I began with with the Caesar cipher, but I needed a few helper functions before I got started. I also plan to print out my data into text files. I will include any noteworthy files with this report in a separate folder. I have also included my .rkt program files in addition to placing it at the Appendix of this report.

**Histogram Generation:** For the file output, it takes a hash table and accumulates the total number of characters in the entire table. Then the program outputs the values of the 26 characters by their frequency. I chose to output the values as a series of asterisks. Each asterisk represents a single percentage of total frequency that character had in the hash table. For example, if A appeared 4 times, B appeared 20 times, and C appeared 0 times in the readable cipher out of a total of 200 letters. The equation I used for the resulting percentages had the form:

% share = number of characters / total number of characters.

In this example2% of the total characters were the letter A, 10% of the letters were B, and 0% of the characters were C. So the frequency representation would be printed as:

A: \*\*

B: \*\*\*\*\*\*\*\*\*\*

C:

However, I discovered the system of plots and vectors that scheme offers for displaying a table of data. This approach seemed vastly more efficient to what I was going for, so I scrapped my idea of an asterisk data output in favor a more sophisticated graphical interface. For this example, I called the values from the hash table and plotted them out on the histogram.

A previous problem I had with Project 1 was determining if a key in a hash table had been initialized yet. Much to my satisfaction there is the function (hash-has-key? *hash* *key*) that returns a Boolean dependent on if a given key is in a hash table.

As I had expected, having vectors named

(vector "A" (hash-ref ht "A"))

did not work. The key for A is actually #\A, so I have to make a slightchange to the plot function. Let’s see if this works:

(vector "A" (hash-ref ht #\A))

It did. I will test it on the Caesar hash table.

**Character Offsets:** To get the character offsets I initially planned to use two hash tables and trade values from each other to get shift values. However this did not work out because I could not get hash tables to take integers as keywords.

I decided on a string of characters “ABCDEFGHIJKLMNOPQRSTUVWXYZ” and utilized the (string-ref) procedure to get shifts. Suppose I have the letter G I need to offset by 5 characters. I loop through the string until I find G. Then I call the string-ref of the loop index plus the offset. So the output line equation would be:

-> (string-ref (+ 6 5))

-> L

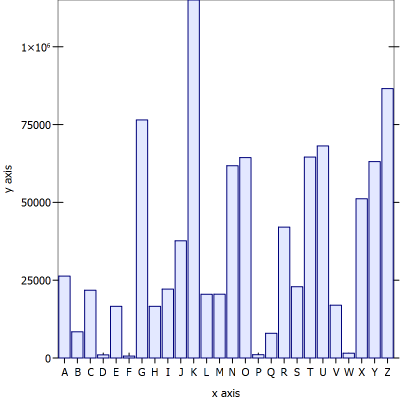
However this outputs #\ before every character, so I just went with a list of characters and referenced them with (list-ref). With a reference to (modulo n 25), a shift of the proper number of integers can be determined by referencing this list. I also moved this string into its own function and removed it as a global variable. Now it will get a character based on an integer value.

I was having issues with getting a character to return true when compared to a string until I discovered the (make-string) function. For a moment I thought that my offset was incorrect, but my assumption of modulo 25 was incorrect. Off by one, I changed it to modulo 26 and it worked.

**File Input (charcopy.rkt):** I wanted to create a function that removed the non-alphabetical characters from a given file. I would then use that new file to help decipher the Caesar and Vigenere ciphers. I used the function (char-alphabetic?) to filter out symbols, but this also filtered out spaces and newline commands. For my final output I needed to have these spaces, so I decided to add a conditional allowing the spaces and new lines to be outputted to the file as well. **ctest.txt** and **vtest.txt** are these output files from the Caesar and Vigenere ciphers respectively. [**myfiletest.txt** and **test\_0\_noformat.txt** are early examples of when I didn’t save the formatting.]

**Hash Tables (Caesar):** I decided to loop through the cleaned up file for the Caesar cipher and use the characters as keys for a hash table. I stored the number of times I would read that character under the key for that character, incrementing by one every time it was repeated. I ran into some trouble when I messed up my conditional statement, but a simple (not) boolean statement fixed things up. I reverse engineered a lot of my code from **filecopy.rkt**, which I talked about working on in the **charcopy.rkt** section of my report.

After that I printed the histogram. Here’s a picture of the output:

When referencing the usual character frequencies of the English language, it appeared that E was the most common letter, so I looked for the letter with the highest frequency. In this case it appears that K showed up the most often. Judging from its neighbor at G, I could tell that G must have been A from its relatively high frequency. So A has been encrypted into G, which is an offset of **6** characters. Now to begin the Caesar decipher.

**Caesar Decipher:** I essentially copy pasted my file input program and added

(write-char **(get\_char\_shift (make-string 1 next-char) n)** outfile))

And it worked the first time I ran it. The output file had correct spacing and word preservation. [**test2\_Success.txt** is that file]

**The Caesar Cipher is an offset of 6 characters and encrypted Herman Melville’s famous novel *Moby-Dick*.**

Done with the Caesar cipher, I began work on the Vigenere cipher. First, I wanted to figure out how long the keyword would be.

**Substring Hash Tables (Vigenere):** Hash tables for the Vigenere are utilized for finding the distances between repeated substrings. Finding a common denominator between the majorities of them has a high probability of being the keyword length of the cipher. I decided to store the characters from the cipher as a list and modify it as the program runs. Every time the list key references the hash table for the first time, the distance is recorded in the table and the loop continues. If data is already present, then I know this is another appearance of the substring and increment the current value by one. I decided to display the data at this point in the loop. It outputs a single number, the index value of the new substring subtracted by the old value already in the table. I then swap the new value for the old one and I continue until the file reaches the last character.

I thought I would need the helper function below, but I was able to achieve the results I needed within the procedure.

(define (get\_sl text n) 2)

;gets first n characters of the substring and puts them in a list

I created my alphabetical character only text file by running the Vigenere text through my charcopy.rkt program, yielding the file vtest.txt. This was what I used when testing this function.

(build\_v\_ht "vtest.txt" 0 4)

First output was inconclusive. And I believed it to be inaccurate.

17 17 17 14 14 14 14 14 14 105 30 30 30 80 80 80 80 23 23 23 23 16 188 8

These numbers seemed strange to me, so I displayed my hash table’s current list as I went along. Turns out I was saving spaces and saving values but still incrementing. Now my program would just ignore it and continue without iterating the numerical values of the loop.

After a round of bug squashing I tried running it again, this time using a substring length of 5 as suggested in the pdf. I also got rid of a parameter. I thought I wanted to increment by some measure but it proved to be unnecessary.

(build\_v\_ht "vtest.txt" 5)

I also output the current list. I realized I didn’t need to do this and I got rid of it later.

16(S H F D X)

16(H F D X Q)

12(M Z K J Z)

12(Z K J Z W)

12(K J Z W K)

68(G A Z F K)

68(A Z F K K)

20(Z Q J H G)

20(Q J H G M)

20(J H G M L)

20(H G M L G)

Although these results were cleaner, I still was unsure about the values. The largest common denominator of these numbers was 4, but I had noticed a problem from earlier. I was storing the return character for the end of lines. That would be shifting my i values in my loop more and more over time. I also realized that the equation for getting my distances,

(hash-set! v\_ht l i)

Is incorrect according to the pdf for this project. This gets the index of the last character, when I should be getting the index of the first instead.

(hash-set! v\_ht l (- i n))

Would be the correct equation.

Lets try again:

19(S H F D X)

19(H F D X Q)

17(M Z K J Z)

17(Z K J Z W)

17(K J Z W K)

72(G A Z F K)

72(A Z F K K)

24(Z Q J H G)

24(Q J H G M)

24(J H G M L)

24(H G M L G)

284(Z O S J X)

284(O S J X A)

284(S J X A Y)

93(V U V F F)

93(U V F F T)

93(V F F T A)

93(F F T A R)

17(G X L D W)

585(H G V X S)

585(G V X S J)

390(O K W W G)

390(K W W G K)

60(Y J F G H)

60(J F G H Y)

60(F G H Y K)

49(Y J F G H)

49(J F G H Y)

49(F G H Y K)

54(A A Y Z C)

54(A Y Z C D)

54(Y Z C D X)

24, 54, 60, 72, and 360 are all divisible by 3, 4, 6, and 12, but the results are hardly conclusive. Those prime numbers were very annoying. So I decided to print out my hash table and see if I could find anything wrong.

((#\H #\M #\X #\V #\T) . 13219)

((#\V #\M #\Z #\K #\J) . 146)

**((#\newline #\L #\H #\T #\K) . 4929)**

((#\E #\L #\G #\K #\I) . 4758)

((#\F #\Q #\B #\L #\K) . 5433)

((#\T #\E #\K #\A #\C) . 1733)

**((#\F #\Z #\S #\newline #\Y) . 1281)**

((#\K #\T #\L #\Z #\O) . 8587)

((#\T #\U #\B #\H #\O) . 5553)

((#\X #\K #\O #\F #\Z) . 13110)

((#\G #\U #\N #\S #\O) . 13873)

((#\R #\A #\V #\L #\K) . 4288)

((#\S #\R #\Z #\T #\V) . 2303)

((#\H #\H #\Q #\Y #\G) . 2475)

((#\A #\P #\D #\X #\O) . 2383)

((#\S #\K #\Y #\K #\Z) . 3950)

((#\P #\W #\X #\F #\J) . 8361)

**((#\V #\S #\W #\newline #\G) . 3537)**

**((#\newline #\P #\G #\M #\Z) . 9217)**

((#\N #\S #\K #\T #\A) . 6934)

((#\O #\B #\L #\N #\V) . 13710)

((#\Z #\B #\L #\K #\J) . 4357)

Newline was also modifying the values for my loops, so I added yet another conditional to ignore a #\newline character. I also realized I was still calculating distances wrong. I needed to calculate distance with the index of the last character of the old substring with the index of the first character of the new substring. With my new equation in hand:

(display (- (- (- i 1) n) (hash-ref v\_ht l)))

The program printed the following:

6(S H F D X)

6(H F D X Q)

6(M Z K J Z)

6(Z K J Z W)

6(K J Z W K)

60(G A Z F K)

60(A Z F K K)

12(Z Q J H G)

12(Q J H G M)

12(J H G M L)

12(H G M L G)

264(Z O S J X)

264(O S J X A)

264(S J X A Y)

78(T V U V F)

78(V U V F F)

78(U V F F T)

78(V F F T A)

78(F F T A R)

6(G X L D W)

It was obvious to me now. My keyword length was 6 characters long. A question still remained however: 6 characters of what?

I set off to make a frequency table of every n characters.

**Histogram for Vigenere**: After some displacement amount the program would obtain the frequencies of every n characters in the file.

First output was

#hash((V . 1) (D . 1) (X . 1) (Y . 1) (H . 1))

It seemed to terminating after some time. I realized that the way I had written the procedure:

(cond ((not (hash-has-key? c\_ht next-char))

(hash-set! c\_ht next-char 1)

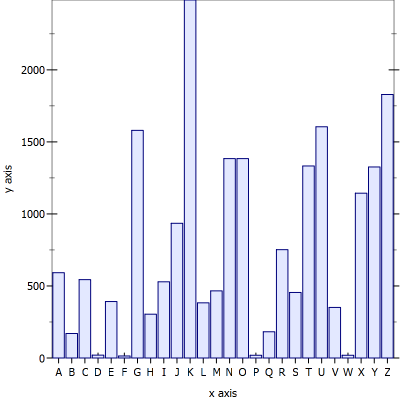
(loop n 0 (read-char infile)))))

had a flaw. The next time a repeated character showed up, I would not do anything. It needed to have an else statement included in this condition. I also moved the loop out of the conditional to avoid writing it twice.

I also realized my integer offset n was lagging behind by one character every time, so I changed the conditional to

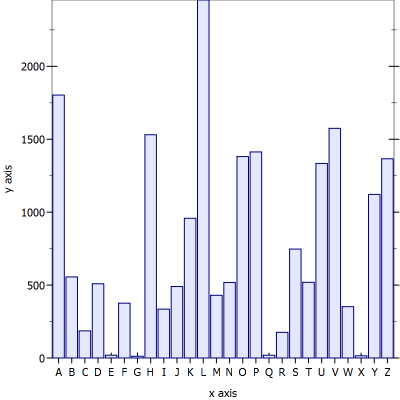
(= i 0) to (= i 1)

This histogram was my first result with an initial offset of 0.

 E is usually the largest value, so K would equal E in this instance. A is 4 characters before it, and conveniently also gives the character offset. G is 4 characters to the left of K, and gives us the first character of our keyword: “G”

Now I wanted to run the same test, but with a displacement of one character. That is, I wanted to take the frequencies of every n+1 character in the cipher.

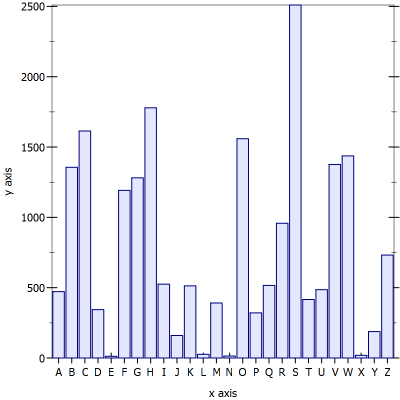
I’ll speed things up and just place the remaining histograms below:

L is most common character.

L = E

A= H

H is the 2nd character.

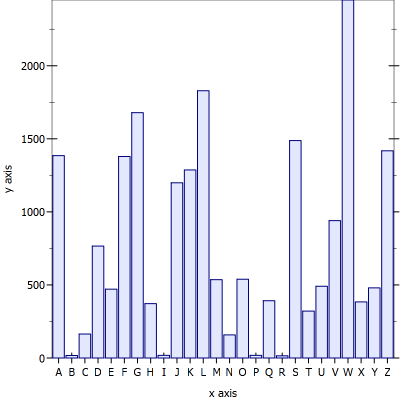


S is the most common character.

S = E

A = O

O is the 3rd character.

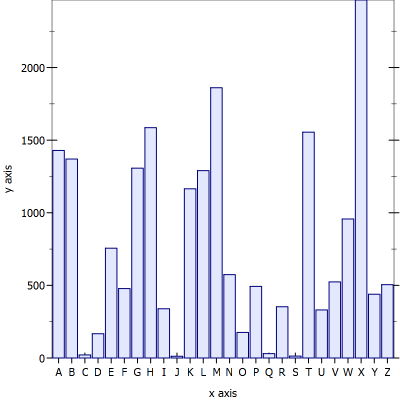


W is the most common character.

W = E

S = A

S is the 4th character.

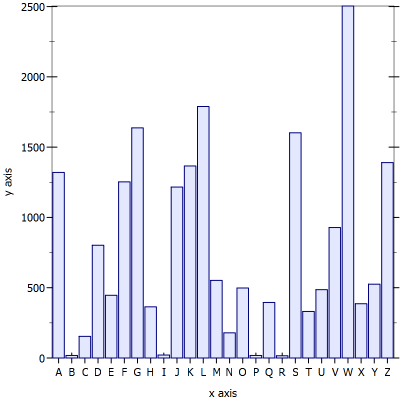


X is the most common character.

X = E

T = A

T is the 5th character.



W is the most common character.

W = E

S = A

S is the 6th and last character.

So the 6 character keyword is “GHOSTS”. Now time to decipher it.

**Vigenere Decipher:** First I needed a way to offset a character based on another character. I realized though that I would have a lot of repeated code, so I decided to just make a helper function (get\_offset l) which returns an integer value based on the input character. It’s essentially the same as (get\_char\_shift l n) but with an inverted list. Both require the character to be cast into a string.

I also just combined the functions into one to make it a bit easier to manipulate when I got to Vigenere Decipher.

I was initially worried that I would have to reset the “i” value every time I wanted to read the beginning of the keyword, but then I realized I could just use modulus to get the values reset like I wanted.

(write-char

(get\_v\_shift (make-string 1 next-char)

(make-string 1 (string-ref key i)))

outfile))

I ran it, and while I could get a bit of clarity from some words, it falls into incomprehensibility.

STAVE U AONXQCG VPZXT

EPMLRJ KOO OJAQ UV BEGIN SHUT LWZRR MG CW HPTPU

SGBFDOAQ NMGJO FTEH IPP SDUJZXFQ OF DHT MMGDAY AOG

I think this is due to me incrementing the loop every time I get a character. It would mess up the synchronicity with the cipher.

Lets try again.

(Vigenere\_Decipher "vtest.txt" "GHOSTS" "done.txt")

STAVE I MARLEYS GHOST

MARLEY WAS DEAD TO BEGIN WITH THERE IS NO DOUBT

WHATEVER ABOUT THAT THE REGISTER OF HIS BURIAL WAS

SIGNED BY THE CLERGYMAN THE CLERK THE UNDERTAKER

AND THE CHIEF MOURNER SCROOGE SIGNED IT AND

SCROOGES NAME WAS GOOD UPON CHANGE FOR ANYTHING HE

CHOSE TO PUT HIS HAND TO OLD MARLEY WAS AS DEAD AS A

DOORNAIL

**In conclusion, the keyword is: “GHOSTS” and the text is Charles Dickens’ “A Christmas Carol”.**

Appendix:

#lang racket

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; CSCI 301

;; Fall 2013

;;

;; Project #2

;;

;; Jacob Chapman

;; W01012392

;;

;; The purpose of this lab is to create a

;; program that aids in the deciphering of

;; Caesar and Vigenere ciphers.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(require plot)

(define (get\_char\_shift l n)

; l is a character

; n is the offset

(define z\_shift '("Z" "Y" "X" "W" "V" "U" "T" "S" "R" "Q" "P" "O" "N"

"M" "L" "K" "J" "I" "H" "G" "F" "E" "D" "C" "B" "A"))

(let loop ((i 0))

(cond

((equal? l (list-ref z\_shift i))

(string-ref (list-ref z\_shift (modulo (+ n i) 26)) 0))

(else (loop (+ i 1))))))

(define (get\_offset l)

; Returns a number based on the offset character

(define c\_shift '("A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M"

"N" "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"))

(let loop ((i 0))

(cond

((equal? l (list-ref c\_shift i))

i)

(else (loop (+ i 1))))))

(define (get\_v\_shift l v)

; Gives the offset of character l by character v

(get\_char\_shift l (get\_offset v))

)

(define (build\_ht\_C file)

; Sum the number of times the alphabetical characters were found

; Stores them in the c\_ht hash table

; If the number is already there, it increments

; Prints a histogram of the hash table

(define c\_ht (make-hash))

(let ((infile (open-input-file file)))

(let loop ((next-char (read-char infile)))

(when (not (eof-object? next-char))

(cond

((not (hash-has-key? c\_ht next-char))

(hash-set! c\_ht next-char 1))

(else

(hash-set! c\_ht next-char (+ 1 (hash-ref c\_ht next-char)))))

(loop (read-char infile))))

(close-input-port infile))

(print\_histo c\_ht))

(define (v\_histogram file n y)

; Sums every n characters after a given offset and prints to

; a histogram

(define c\_ht (make-hash))

(let ((infile (open-input-file file)))

(let loop ((i 0) (g y) (next-char (read-char infile)))

; i stores the n characters to go to before taking another value

; g tells us how many characters to ignore at the begining

(when (not (eof-object? next-char))

(cond

((equal? #\ next-char) (loop i g (read-char infile)))

((equal? #\return next-char) (loop i g (read-char infile)))

((equal? #\newline next-char) (loop i g (read-char infile)))

; Ignre the first g characters

((= g 0) ; Offset over, begin getting frequency of every n characters

(cond

((= i 0)

(cond

((not (hash-has-key? c\_ht next-char))

(hash-set! c\_ht next-char 1))

(else (hash-set! c\_ht next-char (+ 1 (hash-ref c\_ht next-char)))))

(loop (- n 1) 0 (read-char infile)))

(else (loop (- i 1) 0 (read-char infile)))))

(else (loop 0 (- g 1) (read-char infile))))))

(close-input-port infile))

(print\_histo c\_ht))

(define (build\_v\_ht file n)

(define v\_ht (make-hash))

; n is the length of the substring

(let ((infile (open-input-file file)))

(let loop ((i 0) (l '()) (next-char (read-char infile)))

(when (not (eof-object? next-char))

(cond

; Gets the first substring of n characters

; And skips the empty character

((equal? #\ next-char) (loop i l (read-char infile)))

((equal? #\return next-char) (loop i l (read-char infile)))

((equal? #\newline next-char) (loop i l (read-char infile)))

((< i n) (loop (+ i 1) (append l (cons next-char null))

(read-char infile)))

(else (cond

; Prints the distances between 2 iterations of a substring

((not (hash-has-key? v\_ht l)) (hash-set! v\_ht l i))

(else (display (- (- (- i 1) n) (hash-ref v\_ht l)))

(display l)

(newline)

(hash-set! v\_ht l i)))))

(loop (+ i 1) (cdr (append l (cons next-char null)))(read-char infile))))

(close-input-port infile)))

(define (Caesar\_Decipher file n ofile)

; This file operates nearly identically to charcopy.rkt, but performs

; a character shift before outputing to the file.

(let ((infile (open-input-file file))

(outfile (open-output-file ofile

#:mode 'text

#:exists 'replace)))

(let loop ((next-char (read-char infile)))

(when (not (eof-object? next-char))

(cond

((char-alphabetic? next-char)

; Shifts the characters here

(write-char

(get\_char\_shift (make-string 1 next-char) n)

outfile))

; Print out the following to retain formatting

((equal? #\newline next-char)

(write-char next-char outfile))

((equal? #\ next-char)

(write-char next-char outfile)))

(loop (read-char infile))))

(close-input-port infile)

(close-output-port outfile))

)

(define (Vigenere\_Decipher file key ofile)

; Uses the string key to decipher a file

(let ((infile (open-input-file file))

(outfile (open-output-file ofile

#:mode 'text

#:exists 'replace)))

(let loop ((i 0) (next-char (read-char infile)))

(when (not (eof-object? next-char))

(cond ; If the input character is alphabetical, space,

; or new line, it outputs to the file

((char-alphabetic? next-char)

; Shift the characters here

(write-char

(get\_v\_shift (make-string 1 next-char)

(make-string 1 (string-ref key

(modulo i (string-length key)))))

outfile)

(loop (+ i 1) (read-char infile)))

; Print out the following to retain formatting

((equal? #\newline next-char)

(write-char next-char outfile))

((equal? #\ next-char)

(write-char next-char outfile)))

(loop i (read-char infile))))

(close-input-port infile)

(close-output-port outfile))

)

(define (print\_histo ht)

; Prints a histogram of a hash table

(plot (discrete-histogram (list

(vector "A" (hash-ref ht #\A))

(vector "B" (hash-ref ht #\B))

(vector "C" (hash-ref ht #\C))

(vector "D" (hash-ref ht #\D))

(vector "E" (hash-ref ht #\E))

(vector "F" (hash-ref ht #\F))

(vector "G" (hash-ref ht #\G))

(vector "H" (hash-ref ht #\H))

(vector "I" (hash-ref ht #\I))

(vector "J" (hash-ref ht #\J))

(vector "K" (hash-ref ht #\K))

(vector "L" (hash-ref ht #\L))

(vector "M" (hash-ref ht #\M))

(vector "N" (hash-ref ht #\N))

(vector "O" (hash-ref ht #\O))

(vector "P" (hash-ref ht #\P))

(vector "Q" (hash-ref ht #\Q))

(vector "R" (hash-ref ht #\R))

(vector "S" (hash-ref ht #\S))

(vector "T" (hash-ref ht #\T))

(vector "U" (hash-ref ht #\U))

(vector "V" (hash-ref ht #\V))

(vector "W" (hash-ref ht #\W))

(vector "X" (hash-ref ht #\X))

(vector "Y" (hash-ref ht #\Y))

(vector "Z" (hash-ref ht #\Z))))))