HW1 Report

Task 1

Given the plain text and the key, simply loop over the entire plain text character by character and follow this logic

1. if it is not an alpha character, push it to the result string and continue to next character
2. if it is lowercase alpha character
    a. upper case it
    b. add it to the current index of the key
    c. modulo 26 to get the shift amount
    d. add it with ‘A’ so it shifted to the correct character
    e. lower case it back
    f. push it to result string
3. if it is uppercase alpha character
    a. add it to the current index of the key
    b. modulo 26 to get the shift amount
    c. add it with ‘A’ so it shifted to the correct character
    d. push it to result string
4. increment the current index of they key by 1, follow by modeling the length of the index such that it becomes a circular index
5. repeat step 1 - 4 until the entire plain text is parsed character by character

Task 2

Given the cipher text and the key, simply loop over the entire cipher text and follow this logic

1. if it is not an alpha character, push it to result string and continue to next character
2. if it is a lowercase alpha character
    1. uppercase it
    2. subtract the current index of the key and add 26
        - the add 26 is for the idea of wrapping around and allowing our modulo still works
    3. then take the modulo of 26 so it shifted to the correct character
    4. add it with 'A'
    5. lowercase it
    6. push it to result string
3. if it is uppercase alpha character,
    1. subtract the current index of the key and add 26
    2. then take the modulo of 26
    3. add it with 'A'
    4. lowercase it
    5. push it to result string
4. increment the current index of they key by 1, follow by modeling the length of the index such that it becomes a circular index
5. repeat step 1 - 4 until the entire cipher text is parsed character by character

Task 3

Given the cipher text and the length of the key

1. first simplify the string and assign it to a new variable we call simplify string
    - this string only consist of uppercase alphabet character, meaning all alphabet character from the original string is uppercased
2. then divide the simplify text such that each row represent the column index of the simplify text, there should be key_length of columns

   - for example if we’re given simplify string HELLOWORLD and a key_length of 2
     we would get a vector of string with a length of 2

     `[HLOOL, [ELWRD]`

   - Basically, the 0th, 2nd, 4th index .... become 1 string, and the 1st, 3rd, 5th index become the other string
   - the length of the vector is equivalent to the key length

3. After we obtain a vector of string of the column index of the simplify text, we will be conducting a chi squared goodness of fit test such that the column letter distribution matches the english letter distribution
4. for each column, conduct all 26 possible caesar shift
   a. for the caesar shift, loop over each character in the string, and compute the new value after it shift
      a. given the character, subtract it with the shift we’re doing
      b. subtract it from the the adjusted shift bound “A” so it is shifted to a 0-based index shift
      c. add 26 back so that the previous value won’t be negative
      d. take the modulo of 26
      e. add it with ‘A’ so it is represented with the ASCII representation again
      f. append to the result string
   b. once the caesar shift, compute the relative frequency of each characters relative to the length of each column
      a. we do that by just using a Map character, int pair and loop over all the character in the string
         a. if we see a character c, increment the value of the key c.
      b. then create another map which we will have character, double pair and loop over all the character

- c.
  - b.
    - a.
      - a. compute the double value of the relative frequency by dividing the count from the previous frequency Map with the length of the column
  - c. once we obtain the relative frequency map of all the character, we conduct the chi squared test
    - a. loop over all 26 characters using the exp_char from the map
      - a. if the key does not exist in the map, we set the observed expectation to 0.0
      - b. other wise use the value that we obtain from the relative frequency map
      - c. follow the formula $\frac{(\text{observed}-\text{expected})^2}{\text{expected}}$
      - d. add the value we obtain from previous line to a variable
    - b. return the sum of all the chi squared test we’ve done for all 26 characters
  - d. once we obtain a chi squared value, we compare to any previous computed chi squared value, if this value is lower then previous, we update the minimum to the current one and the shift amount to the current index of the loop
- 5. Once we obtain a shift for each column, add ‘A’ to each one of the shift and combine them into a result String
- f. then given that we find the best key string, plug the original cipher text and this new found key into our task 2 codes
- g. then print the key and the result of the decoded test

Task 4

Given the cipher text

1. Using the concept of index of coincidence (Wikipedia “Index of Coincidence”), we will attempt to figure the length of the key
   - a. first simplify the string and assign it to a new variable we call simplify string

    - this string only consist of uppercase alphabet character, meaning all alphabet character from the original string is uppercased

2. Compute the aggregate index of coincidence compare to the value 1.73 (Wikipedia “Index of Coincidence”)

    a. we will first assume the key_length to 1, and this will go up every time we iterate in here

    b. divide the simplify text such that each row represent the column index of the simplify text, there should be key_length of columns

    - for example if we’re given simplify string “HELLOWORLD” and a key_length of 2
we would get a vector of string with a length of 2

      `[HLOOL, [ELWRD]`

    - Basically, the 0th, 2nd, 4th index .... become 1 string, and the 1st, 3rd, 5th index become the other string

    - the length of the vector is equivalent to the key length

    c. we then iterate through all the column to compute the aggregate index of coincidence value

        a. for each column

            a. count the frequency of each character using a map again

            b. then we utilized this formula $\frac{\sum_{i=1}^{c} n_i*(n_i-1)}{N(N-1)/c}$ (Wikipedia “Index of Coincidence”), where c is the alphabet size which is 26, N is the length of the column and n_i is the frequency value of the ith index character in the alphabet

            c. add the value we got from the previous line into a variable such that it becomes the summation for all the columns

        b. we then divide the sum of their respective index of coincidence for each column by the key_length aka the amount of column

        c. finally, compare the value to 1.73 (Wikipedia “Index of Coincidence”)

        - In the code, I created a 10% tolerance. so anything above 1.73 - 1.73 * .1 will be taken as the key length
        - every other one that is NOT english will be returning an aggregate value closer to 1 as an indication that it is random

    d. increment the key_length

    e. repeat step b → e until we find a comparison higher then the lower bound or the aggregate index of coincidence became NAN (which we no longer can solve)

3. Once figured out the key length, plug this into task 3 and find the key and decode the text
4. then print both of them out.

Appendix

1. “Letter Frequency” *Wikipedia*, Wikimedia Foundation, 25 July 2024, en.wikipedia.org/wiki/Letter_frequency.
2. “Index of Coincidence” *Wikipedia*, Wikimedia Foundation, 2 Aug. 2024, en.wikipedia.org/wiki/Index_of_coincidence.