Priya Singh
Professor Boctor
COMP 424
10/10/2020

Assignment 1 Report

Methods Used

The utilized code was found from various sources and used directly or modified into to the files Main.java and Utility.java to understand how the code works. The program uses brute force methods to decrypt the ciphertext. Main.java calls the methods from Utility.java in this order:

1. `static Map countFrequency(String cipherText)`
   Since the given ciphertext is in English and only contains letters, I can use frequency analysis to get a grasp on which characters in the ciphertext are which possible English characters by counting the frequency of each symbol in the cipher text to see how often the occur[1].

```
----jGRASP exec: java Main
Frequency Map: {B=6, D=7, E=2, F=1, G=1, H=6, I=3, K=6, L=5, N=1, O=5, P=5, Q=4, R=6, S=2, U=3, V=7, W=5, X=2}

----jGRASP: operation complete.
```

2. `static ArrayList sortFrequencyList(Map frequencyMap)`
   The generated frequency map from `countFrequency()` is then sorted in the order of most frequent to least frequent[2][3]:

```
----jGRASP exec: java Main
Sorted List: [D, V, B, H, K, R, L, O, P, W, Q, I, U, E, S, X, F, G, N]

----jGRASP: operation complete.
```

3. `static Set analyzeFrequencies(List<Character> sortedList)`
   After the list is sorted, I determine the number of possible shifts it would take for the ciphertext symbol to reach a common English character, going from Z to A[4]. According to Wikipedia, the common characters in the English are: ‘E’,’T’,’A’,’O’,’I’,’N’,’S’,’H’,’R’,’D’,’L’, and ‘U’[5]. For example, the ciphertext symbol ‘D’ will take 3 shifts to get to the English letter ‘A’ (D→C→B→A = 3 shifts). The list of possible shifts are the possible number of shifts to brute-force the shift substitution decryption:

```
----jGRASP exec: java Main
Possible Shifts: [1, 2, 3]

----jGRASP: operation complete.
```

4. `static Set shiftSubstitute(Set<Integer> possibleShifts, String ciphertext)`
   Given the possible shifts from `analyzeFrequencies()` I then perform the shift substitution on the ciphertext, in the direction of A to Z, using all possible shifts to generate a set of all possible strings that contain the plaintext[4]:

```
Possible strings: [JTGOUHAPJUNRGVGWAONETWGQOUKKCCVUNRJVOQDCCUUHCVPQAGAFKKAAOJPTMQUNGPDHQKVNJJQCC,

HREMSFYNHSLPETEUYMLCRUEOMSIIAATSLPHTMOBAASSFATNOYEYDIIYYMHNRKOSLENBFOITLHHOAA,

ISFNTGZOITMQFUFVZNMDSVFPNTJJBBUTMQIUNPCBBTTGBUOPZFZEJJZZNIOSLPTMFOCGPJUMIIPBB]
```

5. `static List<String> generateKeys()`
   Next, I need to generate the key to perform columnar transposition. The key is brute forced by first generating all possible keys bases using the given key length, from 1 to 10, based on the English Alphabet. For example, for the key bases for the given key lengths are: 1 = 'A', 2 = 'AB', 3 = 'ABC', …, 10 = 'ABCDEFGHIJ'[4].
   `generateKeys` () then uses `permutation()` to get all possible keys using the key bases:
   a. `static Set<String> permutation(String input)`
      The permutations are performed recursively on the given string from `generateKeys()` to find all possible keys by rearranging the characters of the key base that was passed in[6]. For example, for the key base 'ABC' the possible keys generated are:
      `Key List: [ACB, BCA, ABC, CBA, BAC, CAB]`

   This process is repeated for each key base, until I have generated all possible keys for our given range of key bases:

   `Key List: [A, AB, BA, ACB, BCA, ABC, CBA, BAC, CAB, BADC, BCDA, DCBA, DABC, …`

6. `static void columnarTranspose(Set<String> possibleStrings, List<String> keyList)`
   Now that I can generate all possible keys using `generateKeys()`, I can perform columnar transposition on the possible strings generated by `shiftSubstitute()`.

   First, I need to convert the chosen alphabetic key into the numeric equivalent[4][7]. For example, key = BA = 10, where A = 0, B = 1, C = 2, … , J = 9.

   Then, I arrange the chosen string into the grid[4]:
   For each column, starting from the last, check if there will empty cells that will require padding at the end. Padding is required when the length of the given string is not a multiple of the given key, so there will be remaining space.
   If padding is not required, take the portion of the string for the column from the initial string and add it to the column, updating the initial string to account for the portion taken.

   Finally, get the deciphered message by going through each row of the grid and pulling out each character in the row. The order of the characters is arranged based on the numeric key.

   Now to see our work, I print out the deciphered text, along with the given string and key:

```
Candidate: [JTGOUHAPJUNRGVGWAONETWGQOUKKCCVUNRJVOQDCCUUHCVPQAGAFKKAAOJPTMQUNGPDHQKVNJJQCC] key = EFDGCAB
Deciphered: #1 [DGJRVOJHQTGPJVQOGVQPOKUOGATQVKUWGMDNKHAAQCJCAOFUCJCPNKNUQVJEKGUCUUTAPHCNNW]

Candidate: [JTGOUHAPJUNRGVGWAONETWGQOUKKCCVUNRJVOQDCCUUHCVPQAGAFKKAAOJPTMQUNGPDHQKVNJJQCC] key = EFDGCBA
Deciphered: #2 [DGRJVOJHQGTPJVQOVGQPOKUGOATQVKWUGMDNKAHAQCJCOAFUCJCNPKNUQVEJKGUCUTUAPHCNWN]

Candidate: [JTGOUHAPJUNRGVGWAONETWGQOUKKCCVUNRJVOQDCCUUHCVPQAGAFKKAAOJPTMQUNGPDHQKVNJJQCC] key = EFBDGCA
Deciphered: #3 [RDGJVOGJHQTPJVVQOGQPGOKUOATWQVKUGMADNKHAQOCJCAFUNCJCPKNEUQVJKGTUCUUAPWHCNN]

Candidate: [JTGOUHAPJUNRGVGWAONETWGQOUKKCCVUNRJVOQDCCUUHCVPQAGAFKKAAOJPTMQUNGPDHQKVNJJQCC] key = EFDABGC
Deciphered: #4 [JRDGVOJTGHQPJVGVQOQPOOGKUATQUWVKGMDHANKAQCAOJCFUCPNJCKNUJEQVKGUUTCUAPHNWCN]

Candidate: [JTGOUHAPJUNRGVGWAONETWGQOUKKCCVUNRJVOQDCCUUHCVPQAGAFKKAAOJPTMQUNGPDHQKVNJJQCC] key = EFBDGAC
Deciphered: #5 [RDJGVOGJHTQPJVVQGOQPGOKOUATWQVUKGMADNHKAQOCJACFUNCJPCKNEUQJVKGTUCUUAPWHCNN]
```

To make it easier to find the decrypted text, I used the `segmentString` () method to add spaces after English words found in the Dictionary.txt file[4]:

a. `private static String segmentString(String newString, Dictionary dictionary)`
   Given the deciphered message, I recursively look through all possible substrings of the given string to find English words using `wordExists()`. If English words are found, a space is added after the word[4].
b. `public boolean wordExists(String wordToLookup)`
   Takes a string to check if it exists in the dictionary file from `Dictionary()[4]`.
c. `public Dictionary()`
   Scans the words from Dictionary.txt into the dictionary file. If Dictionary.txt does not exist, it terminates the program[4].

Using the Find function in the IDE, I started looking for words with spaces after them and eventually found the deciphered message:

```
Candidate: [HREMSFYNHSLPETEUYMLCRUEOMSIIAATSLPHTMOBAASSFATNOYEYDIIYYMHNRKOSLENBFOITLHHOAA] key = GCAEDBF
Deciphered: #414 [BE HAPPY FOR THE MOMENT THIS MOMENT IS YOUR LIFE BY KHAY YAMOH AND ALSO THIS CLASS IS FUN]
```

Works Cited

[1] SINGH, RISHABH. “Using Hash Maps in Java to Find Frequency of Characters in a String.” *Stack Overflow*, 6 Feb. 2020, https://stackoverflow.com/a/60094876.

[2] Alam, Rais. “Sorting HashMap by values [duplicate].” *Stack Overflow*, 17 Dec. 2012, https://stackoverflow.com/a/13913206.

[3] Matteo, Shashank Agrawal. “Get keys from HashMap in Java.” *Stack Overflow*, 3 Mar. 2016, https://stackoverflow.com/a/10462838.

[4] Tapia, Eric. “CipherText.” *GitHub*, 22 Jan. 2017, https://github.com/eat41899/CipherText.

[5] Wikipedia Contributors. “Frequency analysis.” *Wikipedia, The Free Encyclopedia*, 24 Jul. 2020, https://en.wikipedia.org/wiki/Frequency_analysis.

[6] devastator. ”Generating all permutations of a given string” *StackOverflow*, 16 Dec. 2013, https://stackoverflow.com/a/20614037.

[7] Iqmal, Muhammad H. “Columnar Transposition Cipher With Key”, *Blogger*, 23 Sept. 2015, https://programmingcode4life.blogspot.com/2015/09/columnar-transposition-cipher.html.