Assignment 1 External Documentation

Overview

The Program includes substitution cipher functionality. In one option, the User will provide an encryption key and cipher text, after which the plaintext can be obtained by matching the plaintext and cipher text characters in the given Key. In the second option, the User will enter the original language and Ciphertext into the Program. The Encryption key will be generated using the frequency table of the given languages and Ciphertext.

Files and external data

The Program has 6 files:

- A1.java is the main file from which the execution begins, and the User will interact with the Program.
- SubstituitionCipher.java – This file contains all of the method definitions and the Program's core logic, such as ciphertext decryption and key generation from the frequency tables of the language and Ciphertext.
- Ciphertext.txt - The encrypted Ciphertext in this text file will be used to generate plaintext and the encryption key.
- Language_A.txt, Language_B.txt, Language_C.txt – Three language files will be created as a container for the various original languages, a frequency table for these languages will be generated, and language matching with cipher text will be performed.

Data Structures and Main Algorithms

The Program primarily employs four data structures: HashMap, LinkedHashMap, ArrayList, and Queue.

All data structures are primarily used and maintained in the substitution cipher Class, which contains the core logic. Hashmaps and LinkedHashMaps are commonly used in tables to store the character and its frequencies as a key-value pair. These hashmaps will store the character as a key and their frequencies as values in a hashtable.

The advantage of using a hashmap is that the User can obtain the values of the characters in 0(1) time complexity, which is very useful if the User intends to use the search functionality frequently.

The LinkedHashMap will function similarly to a regular hashmap, but it will remember the Key's insertion order, making retrieving data in that order easier.

In order to access the list items by their index number, which is used to travel through each element in a for loop, an ArrayList has been used to store the keyset or values of the map.

The Program also uses a queue data structure to retrieve list items based on their insertion order; the queue operates on a first-in-first-out basis.

The sorting algorithm is used multiple times in the Program because the frequency distribution table must be sorted in descending order. For this purpose, the sorting algorithm was manually implemented using a loop.

Classes Co-relations

The Program consists of two classes: A1 and SubstitutionCipher.

The A1 class is the main class, where the project execution begins, where the substitution cipher's objects are created, and that Object will call the methods of the substitution cipher.

The Core logic of the decryption and generation of the encryption key is done in the SubstitutionCipher.

These 2 scenarios in the Program.

1. If the User provides Ciphertext with an encryption key and User will get the decrypted plaintext
2. Suppose the User provides the original language and the Ciphertext. In that case, Program will return the encryption key based on mapping the frequency distribution table of the original languages and cipher text.

A typical order of the method calls in both scenarios

Scenario 1: The User has a key

- SubstitutionCipher's Object will be created with Key.
- Key validation will be done.
- The Ciphertext () method will be called, which will read the ciphertext file and create the frequency table with the help of Linkedhashmap.
- The decode text() method will be called and will return the string file containing the decrypted/decoded text.

Scenario 2: Generating a key from the original language and ciphertext frequency table

- SubstitutionCipher's Object will be created without a key being passed.
- The Ciphertext () method will be called, which will read the ciphertext file and create the frequency table with the help of Linkedhashmap.

- orginialLaguage() method will be called with the name of the language and filename as an argument, which creates the frequency table of the language. This method can be called multiple times.
- matchLanguage() will be called, and it will return the name of the language whose letter frequency distribution most closely matches that of the Ciphertext
- guessKeyFromFrequencies() will be called next, and it will generate the Key from the language and cipher distribution table and return the true if it can derive the Key.
- Key validation will be done.
- The Program will print the derived Key.

Assumptions

- The letters a-z are used in all encrypted text (possibility upper case). All other characters, such as punctuation and accented characters, will be preserved in the encrypted file.
- No line in a file will contain more than 120 characters.

Limitations

- The derived Key will not be accurate as it depends on many factors such as the original language, the original language's characters length, and the Cipher text's characters length.
- The decryption of the Ciphertext will not be accurate if the Ciphertext's length is small; as the program efficiency is highly dependent on the statistics, the more the data, the higher the accuracy.
- If the original languages are similar, then language matching would become difficult as the frequency distribution table's values would be close to each other. Example…English and Scots languages are close, so the matching would be challenging to guess.