Politechnika Śląska

Wydział Automatyki, Elektroniki i Informatyki

**Vigenere cipher decryption, analysis tool**

|  |  |
| --- | --- |
| Author | Wiktor Utracki |
| Instructor | Dr. inż. Anna Gorawska |
| Year | 2019/2020 |
| Lab group | Thursday, 10:00 – 11:30 |
| Deadline | 16.01.2020 |

1. **Project’s topic**

The main goal of he project is to help in analysis of a ciphertext ciphered using the Vigenere cipher. Program performs analysis of the given text, helping to find key length and then most probable key letters.

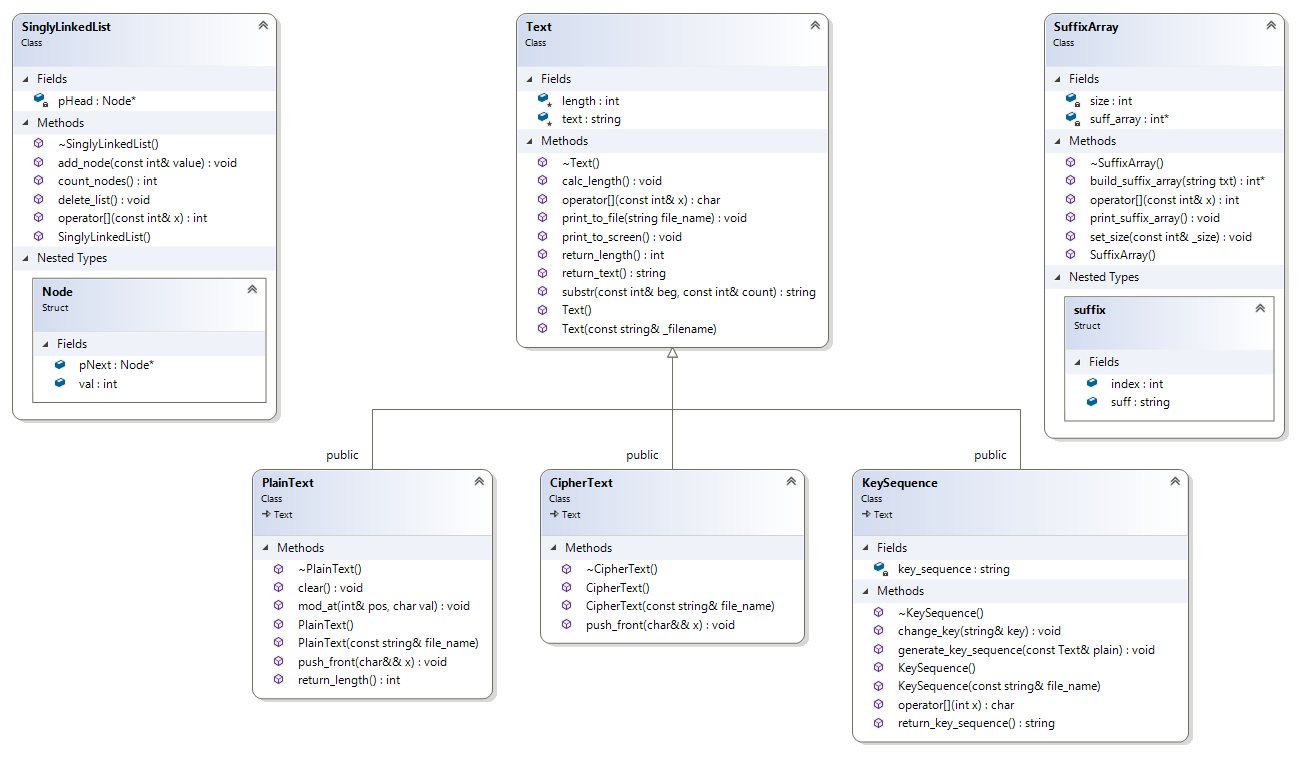
1. **Analysis of the task**
   1. **Data structures**

Figure 1 Class diagram.

Program uses 3 classes that are primarily used to store analyzed text: PlainText, CipherText and KeySequence. Names of those classes are self-explanatory. They all inherit from main class Text for they all represent text so they have a lot in common. The other two are SinglyLinkedList and SuffixArray. The last one is quite interesting. It stores suffix array built from ciphertext. Program utilizes also a UniqueList, which is a singly linked list but it stores only a single node of given value, and counts values that were put into the list.

* 1. **Algorithms**
     1. **Suffix array preparation**

The suffix array created in SuffixArray class is sorted but contains all

suffixes of the cipher and we can only use the patterns in text.



This function reduces the suffix array to patterns only and groups found patterns alphabetically. This function is quite straightforward: it takes first element as group symbol and second as current line, compares them and if they are a part of the same pattern it adds them to the new array and moves the current line to the next element if not it moves group symbol to the current symbol an current symbol the next one and the process is repeated until whole array is sorted.

There is a problem if ciphertext has some clustered letters for example:

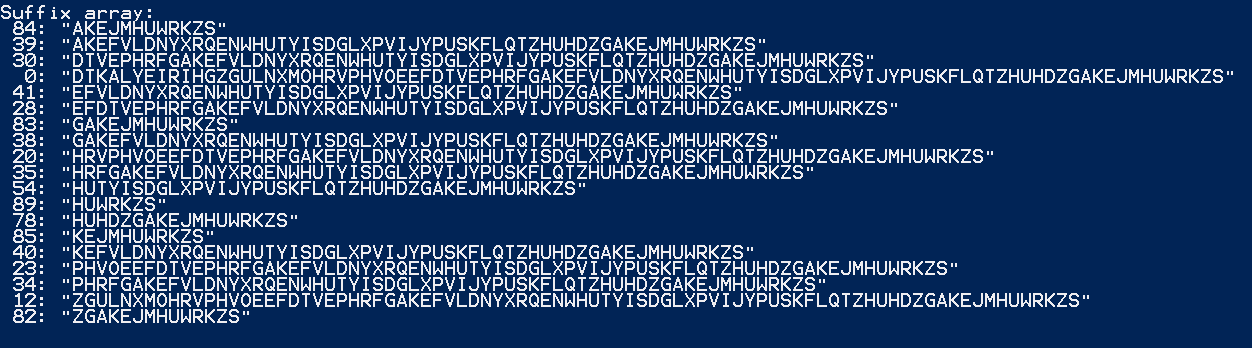
“AGPEKKKKM” because the suffix array would normally have them recognized as patterns (“KK(…)”, “KKK(…)”, “KKKK(…)”) and would have take them into account during Kasiski’s test. Letter grouped in ciphertext like that can mean that some cluster of the same plaintext letters has been encrypted using the same letter of the key (if plaintext would be “OOOO(…)” and encryption key “EEEELA” the ciphertext would look like “SSSS(…)” this can happen but is poorly probable. In most cases such clusters are just a coincidence since Vigenere cipher is polyalphabetic cipher so we would like to exclude them and my function makes that.

Figure 2 Example of a suffix array with patterns filtered out and grouped. Numbers on the left are positions in ciphertext.

* + 1. **Kasiski’s test**

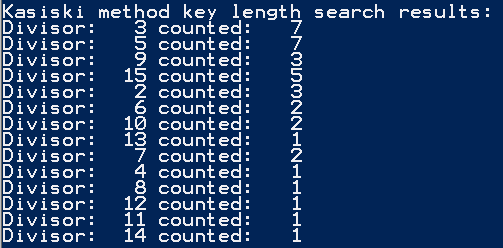
This is a way of guessing the key length. It analyses distances between patterns and finds it divisors. Then produces a table which contains counted divisors. Divisor with highest number is the most probable key length.

Figure 3 Kasiski method output. Since 3 and 5 are most probable w have to choose one that we think is the correct one.

* + 1. **method**

It is a method of finding the most probable key of given length.

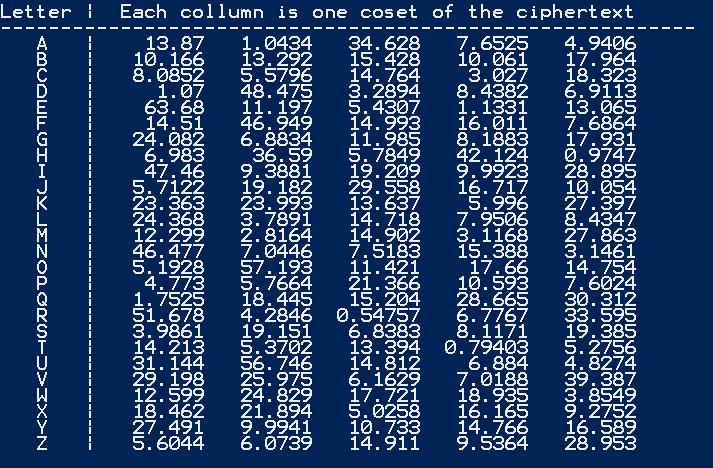
Since there is some frequency in which letters in English appears on average, we can determine how much the given test is close to being an average English text and chi squared method does exactly that. Vigenere cipher is polyalphabetic cipher so average frequency of each letter is going to be closer to 1/26 the better the key we produce will be. But we know that every *n-th* letter (*n* is the key length) is encrypted with the same letter and is just monoalphabetic cipher. We divide ciphertext in n cosets and then shift this coset left 26 times each time calculating the value. The shift witch the smallest value is most probably unencrypted (meaning the letter by which we shifted is the key letter).

Figure 4 method output. We can see that the most probable key is DARTH (which in this case was true)

* 1. **External specification**

Program takes following switches:

* -c <file\_name> is the input ciphertext-containing file
* -p <file\_name> is the input plain text
* -k <file\_name> is the input key
* -o <file\_name> is the output file for decryption
* -q <file\_name> is the output file for encryption

Combinations of plaintext, key [, output for encryption] or cipher [, output for decryption] are accepted.

* 1. **Conclusions**

Program runs smoothly, performs all required functions and has no memory leaks:

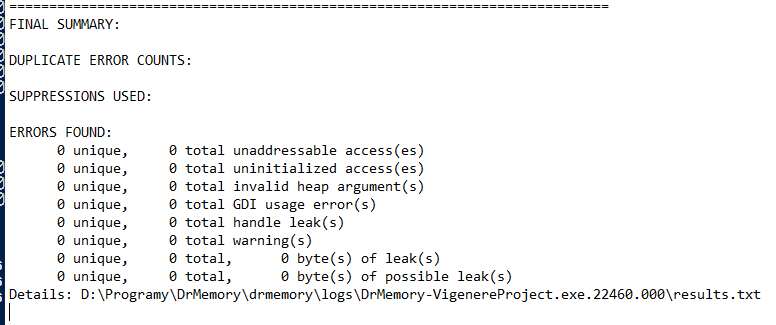


Figure 5 Raport from Dr.Memory laeaks detector.