The ENIGMA Problem

Sabreen Jaber Felicity Lovelace Eram Mohammed
University of Illinois at Chicago

MSCS Undergraduate Research Laboratory

What is ENIGMA?

An "Enigma" machine is a type of machine that was used in World War II by the Germans to send encrypted messages. The machine itself is constructed using gears. Depending on the amount, orientation, labels, and sequence of those gears, there are potentially billions and billions of ways to encode a given message. One can see how that would make it incredibly difficult to crack.

Alan Turing, along with a team of researchers, were able to create a machine, the "Bombe" that helped decrypt German messages. Using information they intercepted, the Allied Powers were able to prevent a large number of attacks planned by the Nazis.

Motivation

Our goal is to build a program that decodes the Enigma V Encryption Machine in an efficient and timely manner utilizing the advancements of computer hardware and software made in the last 80 years. Our program will have the ability to emulate the decryption machine that was used during World War II, the Bombe.

We specifically are working on decrypting the following message:

0081 0061 0172 0165 0108 0174 0161

0123 0142 0079 0058 0108 0177 0129

ALQQ NXEN Z4QD SBQP ZS4N ZNUV OXIL

56ID N7FX ALGD N44F PMUW ALQT Z7XX

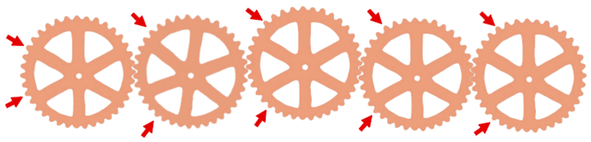
8Z6J Z7ZV 8A8N AN4G AXJL BJFL 64QQ

B9QJ 67XI 61FV 85VG FBQG OXZL 0NXL

R0KE HWK1 R.B

Our Machine

- Five gears
- Each gear is labelled on both sides
- Interchangeable arrangements
- Gears 1, 3, and 5 rotate clockwise
- Gears 2 and 4 rotate counterclockwise
- The indicators on Gear 1 are separated by five spokes
- The indicators on Gears 2, 3, 4, and 5 are separated by seven spokes



The picture above shows a simplified depiction of our Enigma machine.

Encryption Process

The encryption will require us to select an initial gear order and initial key that is 5 characters long. Each gear is set to begin at the corresponding letter from the key.

After selecting the message we'd like to encode, we set the top indicator of the first gear to the first letter of the encryption. The corresponding letter on the next gear is the first encrypted letter. As we continue encrypting the first 4 letters based on the top set of indicators, we must then shift to the bottom set of indicators for the next 4 letters of the message.

The process is repeated until the message is fully encoded.

Decryption Process

To begin our decryption, we define an initial gear order and the initial key that is 5-characters long. We must set each gear to begin at the corresponding letter from the key.

We then rotate the second gear so that the top indicator is now set at the first encrypted letter. We begin our decryption by reading the top indicator of the first gear, after the rotation. We then rotate the third gear so that the top indicator is pointed to the second encrypted letter.

This process is continued until the message is fully decoded. A challenge we are faced with involves the ability to define a new wheel order or a new key at any point during the encryption, thus our decryption must take this into account.

Some Prefatory Calculations

We have five, double-sided gears. That means the number of potential arrangements for the gears is given by:

$$5! * 2^5 = \mathbf{3{,}840}$$

Each gear has **37** spokes.

The characters on those spokes are **uniquely organized** on each gear.

Only the positions of the gears **relative to the starting arrows** matters to us.

So the number of possible keys is given by:

$$37^5 = \mathbf{69{,}343{,}957}$$

Therefore, the total number of settings for our machine is:

$$3{,}840 * 69{,}343{,}957 = \mathbf{266{,}280{,}794{,}880}$$

Even if a human being was able to test one setting every minute, 24 hours a day, every day of the year, it would take them **506,622.52 years** to try every single setting.

Theory

To analyze a given decrypted ciphertext we perform a two stage test. First, a quick, rough test to sort out those decrypts which are either too random or too structured. For example, if our ciphertext is 8 characters long, we wish to exclude decrypts like (FIBQWETV) and those like (AAAAAA11).

There is a method, described below, for performing this check with a constant number of simple arithmetic operations. After this quick check we perform a more intensive check using a dictionary of over 25,000 English words. From this we produce a ranking of decrypted ciphertext corresponding to the percentage of characters which appear in a word.

Our quick pass test using a metric popular in cryptanalysis called the *Index of Coincidence (IC)*. We will briefly describe the test here, but further materials can be found in [Wiki][Shene].

Essentially IC is a metric describing the probability of randomly pulling the same character twice from a given text. In particular the formula is given by:

$$IC = |A| \sum_{i=1}^{|A|} \frac{a_i(a_i - 1)}{N(N-1)}$$

where a^i is the number of occurrences of the i^{th} character in the alphabet, $|A|$ is the size of the alphabet, and N is the number of characters in the given message.

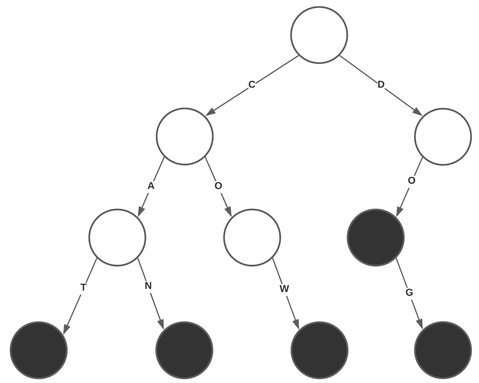
Given a complete random sequence, i.e. one with the number of occurrences of each character being approximately equal, we would have $IC = 1$. For a text consisting the same character repeated, we would have $IC = |A|$. Now, for a sufficiently large English text we expect to see an $IC = 1.7$. Using these bounds we can attempt to narrow in on "interesting" decrypts for further analysis.

However, having the same IC as English is not enough to say that a message is meaningful. For example, consider the message; "COME BACK SOON". We can simply rearrange the letters to obtain "ABCC EKMN OOOS", which has the same IC but is meaningless. So we must perform further analysis before we record a candidate decryption.

In order to check if a decrypt with a promising IC rank is meaningful, we test what percentage of characters belong in a word. For instance, rearranging our previous example again we can obtain "CABC KMEN OOSO", which contains "CAB", "ME", and "SO". Thus we have $\frac{7}{12} = 58\%$ of characters involved in words.

Testing every sequential substring against a dictionary is the most expensive computation we need to frequently perform. For that reason we had to select an efficient algorithm. We use a data structure known as a trie for this task, as it is very computationally efficient. An image of a trie is shown to the right.

Tries



Conclusions

Our Enigma machine came with a number of complications that made it difficult to write a program that emulated it. There were several iterations of code that we went through in order to make the code run faster. With billions of settings to try, we couldn't wait years to acquire the various outputs that would need to be filtered.

The execution of our thought process was inevitably faced with a fair amount of bumps in the road. Billions of potential outputs are a lot more results to comb through than we initially expected, and coming up with an efficient mechanism to filter these outputs turned out to be significantly more difficult than anticipated.

While we were not able to successfully construct a solution to the last part of this process in the given time frame, we are dedicated to continue constructing, testing, and modifying various algorithms that will allow us to filter through the outputs we are currently acquiring and narrow them down to those that contain sensible English.

References

K. Moore. "Enigma Machine" *Brilliant Math & Science Wiki.* https://brilliant.org/wiki/enigma-machine/.

"Index of Coincidence" *Wikipedia* Wikimedia Foundation, 9 Oct 2021, https://en.wikipedia.org/wiki/Index_of_coincidence/.

"Index of Coincidence" *MTU Pages* Michigan Technological University, https://pages.mtu.edu/ shene/NSF-4/Tutorial/VIG/Vig-IOC.html.