ADFGVX

How to run the program

❗ **C++17 and CMAKE are required to run the code**

To build and run the program, run the following commands inside the project folder:

```
cmake .
make
./ADFGVXCrackerV2
```

What we found

Transposition key: `{ 0 4 3 6 2 1 5}`

Permutations the program found

`GF - E, AG - T, FV - A, XX - 0, FG - I, DD - N, GX - S, GV - R, AF - H, FA - D, GA - L, VX - C, AA - U, DA - M, GG - F, AV - P, FX - G, AX - W, GD - Y, XF - B, XV - V, DX - K, AD - X, XD - J, VV - Q, XG - Z`

Permutations after manual changes

`GF - E, AG - T, FV - 0, XX - A, FG - I, DD - M, GX - H, GV - R, AF - S, FA - L, GA - D, VX - W, AA - C, DA - U, GG - F, AV - G, AX - M, FX - P, GD - Y, XF - B, XV - V, DX - K, AD - Z, XD - Q, W - X, XG - J`

Deciphered text:

> `IT WAS A BRIGHT COLD DAY IN APRIL AND THE CLOCKS WERE STRIKING THIRTEEN WINSTON SMITH HIS CHIN NUZZLED INTO HIS BREAST IN AN EFFORT TO ESCAPE THE VILE WIND SLIPPED QUICKLY THROUGH THE GLASS DOORS OF VICTORY MANSIONS THOUGH NOT QUICKLY ENOUGH TO PREVENT A SWIRL OF …`

Approach used

To crack the ADGVX code we first have to **brute force** all possible column permutations. Then we'll have to find the right substitution of the bigrams.

To solve the problem more efficiently we use threads, each one will try to find the right column transposition with a different size for the key.

The program offers the possibility to start working on the manual changes required to obtain the perfectly cracked code even if some of threads are still running.

How it works

1. for each possible column transposition key rearrange the text according to the key
2. after rearranging the text calculate the IC (index of coincidence) of that text, if the IC is sufficiently similar to the one of the English language (0.0667 in the non normalized form) we add the pair `<text, IC>` to a list which is shared between all threads and is accessible in the main method. We chose 0.05 as a minimum value

After the program has been running for some time, there are several elements in the list, and the user can select one of the possible permutations. This occurs while the threads continue to add possible permutations to the list.

After choosing a permutation, the program will associate each bigram with a letter of the alphabet based on its frequency in the text and the frequency of that letter in the English language.

```
std::vector<std::pair<std::string, double>> Cracker::calc_bifreq(const std::string& text) {
    std::map<std::string, double> frequency_map;

    for (size_t i = 0; i < text.length() - 1; i += 2) {
        std::string bigram = text.substr(i, 2);
        frequency_map[bigram]++;
    }

    auto size = text.length() / 2;
    for (auto& pair : frequency_map) {
        pair.second = pair.second / size;
    }

    std::vector<std::pair<std::string, double>> sorted_frequencies(frequency_map.begin(), frequency_map.end());

    std::sort(
        sorted_frequencies.begin(),
        sorted_frequencies.end(),
        [](const std::pair<std::string, double>& a, const std::pair<std::string, double>& b) {
            return a.second > b.second;
        }
    );

    return sorted_frequencies;
}
```

This is achieved by ordering the bigrams and the letters based on their frequencies and making the association between the two in the same position of the ordered sequences.

The program will then apply the substitution based on these associations, yielding a text that is almost the deciphered text.

```
std::cout << "[+] Applying substitution based on frequency" << std::endl;

auto ciphered_freq = Cracker::calc_bifreq(ics[selected].second.text);
auto words_freq = Cracker::get_words_freq();

std::vector<std::pair<std::string, char>> substitution;
substitution.reserve(ciphered_freq.size());

for (int i = 0; i < ciphered_freq.size(); i++) {
    std::pair<std::string, char> pair;

    if (i < words_freq.size()) {
        pair.first = ciphered_freq[i].first;
        pair.second = words_freq[i].first[0];
    } else {
        pair.first = ciphered_freq[i].first;
        pair.second = '!';
    }

    substitution.push_back(pair);
}
```

The user can then manually change some of the associations to obtain the actual deciphered text.