Szukanie klucza a s d
0 15 3
0,15,3, 0,15,3, 0,15,3
Kot lubi psy

Klucz długość 3

1. Plik zaszyfrowana wiadomość index of coincidence
    a. Dla wszysykich liter
    b. Adgsbdgf
        1: A g b g ¼ 2/4 ¼
        2: D s d f 2/4 ¼ ¼

Index of coincidence (¼)^2 + (2/4)^2 + (¼)^2 = wynik
Index of coincidence (¼)^2 + (2/4)^2 + (¼)^2 = wynik 2

(Wynik + wynik 2) / 2 = index of coincidence

1 2 3 4
index of coincidence,index of coincidence,index of coincidence,index of coincidence,index of coincidence
5

0,4285 0,48458, 0,78 40,207 0,452 0,79

0,4285 0,48458,0,4285 0,48458,0,4285 0,48458,0,4285 0,48458,0,4285 0,48458,0,4285 0,48458,0,4285 0,48458,0,4285 0,48458,0,4285 0,48458,0,4285 0,48458,0,4285 0,48458,

Wyliczasz średnie występowanie litery w wiadomości z tekstem jawnym
std::deque
[0,78 , 035 …….. Z występowanie]

[0,223 , 035 …….. Z występowanie]

[zapisuje różnice]

[0,58 , 035 …….. Z występowanie]

[0,768 , 035 …….. Z występowanie]

Funkcja liczenie liter w pliku z tekstem jawnym:

```
std::deque<double> count_Letters(const std::string& file_name)
{
	const int SIZE = 'z' - 'a' + 1;
	std::deque<double> Letters_Counted;
	std::ifstream file(file_name);
	int file_size = 0;
	Letters_Counted.assign(26,0);
	bool exists = false;

	if (file)
	{

		char letterread = ' ', letter = ' ';
		while (file >> letterread)
		{
			letter = tolower(letterread);
			exists = is_Alphabetic_Character(letter);

			if (exists)
			{
				file_size++;
				Letters_Counted[letter - 'a']++;
			}
		}
		for (int i = 0; i < Letters_Counted.size(); i++)
		{
			Letters_Counted[i] = Letters_Counted[i] / file_size;
		}

	}
	return Letters_Counted;
}


bool is_Alphabetic_Character(char& letter)
{
	char allowed_Letters[] = {
'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z' };

char letterevaluated = ' ';
```

```
	bool exists = std::find(std::begin(allowed_Letters), std::end(allowed_Letters), letter) !=
std::end(allowed_Letters);

	return exists;
}
```

Długść klucza

```
int findkeyLength(const std::string& file_name)
{
	std::ifstream file(file_name);
	int file_size = 0;
	char letterread = ' ', letter = ' ';
	std::deque<std::deque<double>> coincedence_indexes;
	std::deque<double> coincedence_indexes_of_spacing_i;
	const double proportion = 0.40;

	while (file >> letterread)
	{
		letter = tolower(letterread);
		bool exists = is_Alphabetic_Character(letter);

		if (exists)
		{
			file_size++;
		}
	}




	for (int i = 1; i < (file_size / 2) + 1; i++)
	{
		std::deque<std::deque<char>> letters_separated =
create_deque_Sepparated_letters(file_name, i);

		std::deque<std::deque<double>> averages_counted;

		for (int j = 0; j < letters_separated.size(); j++)
		{

averages_counted.push_back(count_Letters(letters_separated[j]));
```

```
		}

		double average_of_index = 0;

		for (int k = 0; k < averages_counted.size(); k++)
		{
			average_of_index +=
calculate_index_of_coincedence(averages_counted[k]);
		}

		average_of_index = (average_of_index / averages_counted.size());
		coincedence_indexes_of_spacing_i.push_back(average_of_index);

		if (coincedence_indexes_of_spacing_i.size() > 2)
		{
			for (int k = 1; k < coincedence_indexes_of_spacing_i.size(); k++)
			{
				double compare = (coincedence_indexes_of_spacing_i[k - 1] +=
(coincedence_indexes_of_spacing_i[k - 1] * proportion));

				if (compare < coincedence_indexes_of_spacing_i[k])
				{
					int key = k + 1;

					//std::cout <<"Key: "<< key << std::endl;

					return key;
				}
```

klucz = 2

c. Adgsbdgf Adgsbdgf Adgsbdgf

A g b g a g bg ag bg = > ilość liter a / przez ilośc wszystkich liter w tym dequ \\\ tyo sam,o dla b c de … z

D s d f d s d s D s d f d s d s = > ilość liter a / przez ilośc wszystkich liter w tym dequ \\\ tyo sam,o dla b c de … z