Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Фізико-технічний інститут

КРИПТОГРАФІЯ

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №3

Криптоаналіз афінної біграмної підстановки

Виконали
Студенти гр.Фб-22 Пунько Артем
гр.Фб-22 Руденко Поліна

Київ – 2024

Мета роботи:

Набуття навичок частотного аналізу на прикладі розкриття моноалфавітної підстановки; опанування прийомами роботи в модулярній арифметиці.

Хід роботи:

0. Уважно прочитати методичні вказівки до виконання комп’ютерного практикуму.

1. Реалізувати підпрограми із необхідними математичними операціями: обчисленням оберненого елементу за модулем із використанням розширеного алгоритму

Евкліда, розв’язуванням лінійних порівнянь. При розв’язуванні порівнянь потрібно коректно обробляти випадок із декількома розв’язками, повертаючи їх усі.

2. За допомогою програми обчислення частот біграм, яка написана в ході виконання комп’ютерного практикуму No1, знайти 5 найчастіших біграм запропонованого шифртексту (за варіантом).

3. Перебрати можливі варіанти співставлення частих біграм мови та частих біграм шифртексту (розглядаючи пари біграм із п’яти найчастіших). Для кожного співставлення знайти можливі кандидати на ключ (a,b) шляхом розв’язання системи (1).

4. Для кожного кандидата на ключ дешифрувати шифртекст. Якщо шифртекст не є змістовним текстом російською мовою, відкинути цього кандидата.

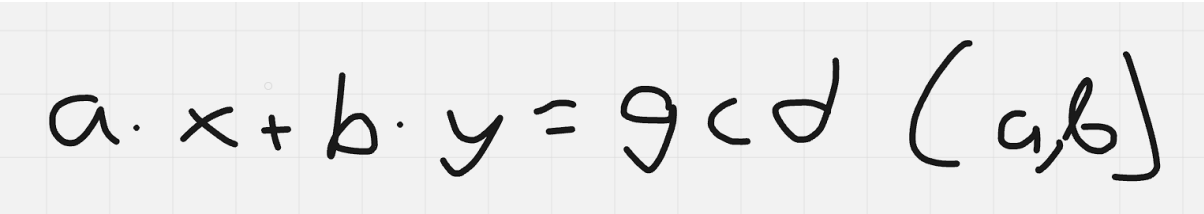
5. Повторювати дії 3-4 доти, доки дешифрований текст не буде змістовним.

Хедер файл з усіма прототипами функцій і глобальними змінними по типу алфавіту або найчастіших літер мови

```cpp
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <Windows.h>
#include <map>
#include <algorithm>
#include <cmath>
#include <fstream>

//alphabet without ё and ь
static const std::string alphabet = "абвгдежзийклмнопрстуфхцчшщьыэюя";
static const int alphabetLength = static_cast<int>(alphabet.size());
static const int powedAlphabetLength = static_cast<int>(pow(alphabetLength, 2));

//most frequent bigramms of russian language
static const std::vector<std::string> mostFreqLanguageBigramms = { "ст", "но", "то", "на", "ен" };

//most and least frequent letters of russian language
static const std::string mostFreqLettersToCheck = "оеаин", leastFreqLettersToCheck = "фэщцю";

//read_file.cpp
std::string readFile(const wchar_t* filePath);

//math.cpp
int gcd(int a, int b);
int inverseElement(int a, int alphLen);
std::vector<int> solveLinearCongruence(int a, int b, int n);

//letters_and_bigramms.cpp
std::map<std::string, uint64_t> countBigramms(std::string& text);
std::vector<std::pair<std::string, uint64_t>> bigrammsSortedByFrequency(std::string& text);
std::map<char, uint64_t> countLetters(std::string& text);
std::vector<std::pair<unsigned int, unsigned int>> formPairs(const std::vector<std::string>& bigramms);

//text_analizer.cpp
bool naturalLanguageAnalizer(std::string& text);

//decryption.cpp
std::pair<int, int> guessKey(std::pair<unsigned int, unsigned int> textPairBigramm, std::pair<unsigned int, unsigned int> langPairBigramm);
std::vector<std::pair<int, int>> guessAllKeys(const std::vector<std::string>& topTextBigramms);
std::string decrypt(const std::string& cryptedText, const std::pair<int, int>& key, const std::string& alphabet);
void decryptAndSave(const std::vector<std::pair<int, int>>& potentialKeys, std::string& text, int& written);
```

1

Для початку реалізуємо функцію найбільшого спільного дільника, а також обчислення зворотного елемента за модулем з використанням розширеного алгоритму Евкліда.



Використовуємо розширений алгоритм Евклиду. Якщо числа a і модуль m взаємно прості, функція повертає зворотний елемент, інакше - повідомляє, що зворотного елемента не існує.

```c
#include "headers.h"

int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int inverseElement(int a, int alphLen) {
    if (alphLen <= 0) return -1;
    a = ((a % alphLen) + alphLen) % alphLen;
    if (gcd(a, alphLen) != 1) return -1;

    int prevCoef = 0, currCoef = 1;
    int prevRemainder = alphLen, currRemainder = a;

    while (currRemainder != 0) {
        int quotient = prevRemainder / currRemainder;

        int tempCoef = currCoef;
        currCoef = prevCoef - quotient * currCoef;
        prevCoef = tempCoef;

        int tempRemainder = currRemainder;
        currRemainder = prevRemainder - quotient * currRemainder;
        prevRemainder = tempRemainder;
    }

    if (prevCoef < 0) prevCoef += alphLen;
    return prevCoef;
}
```

Далі функція винокує рішення линійних порівнянь

$$a \cdot x \equiv b \pmod{n}$$

Перевіряємо чи b mod(НОД) не дорівнює 0. Повертаємо список усіх можливих рішень порівнянь

```cpp
std::vector<int> solveLinearCongruence(int a, int b, int n) {
    int GCD = gcd(a, n);

    if (GCD == 1) {
        int inv = inverseElement(a, n);
        return { (inv * b) % n };
    }

    if (b % GCD != 0) {
        return {};
    }
    a /= GCD;
    b /= GCD;
    n /= GCD;

    int x0 = (inverseElement(a, n) * b) % n;

    std::vector<int> solutions;
    for (int i = 0; i < GCD; ++i) {
        solutions.push_back((x0 + i * n) % (n * GCD));
    }

    return solutions;
}
```

2

Беремо программу для обчислення частот біграм з Лаб.роботи №1

Розбиваємо текст на биграми та підрахуємо частоту їх появлення

```cpp
#include "headers.h"

std::map<std::string, uint64_t> countBigramms(std::string& text) {
    std::map<std::string, uint64_t> dictionary_bigramm;
    for (char first : alphabet) {
        for (char second : alphabet) {
            std::string bigram = { first, second };
            dictionary_bigramm[bigram] = 0;
        }
    }

    uint64_t bigramm_count = 0;
    bigramm_count = text.size() / 2;
    for (uint64_t i = 0; i < text.size() - 1; i += 2) {
        std::string bigramm = text.substr(i, 2);
        dictionary_bigramm[bigramm]++;
    }

    return dictionary_bigramm;
}

std::vector<std::pair<std::string, uint64_t>> bigrammsSortedByFrequency(std::string& text) {
    std::map<std::string, uint64_t> bigramms = countBigramms(text);
    std::vector<std::pair<std::string, uint64_t>> bigrammsVector(bigramms.begin(), bigramms.end());

    std::sort(bigrammsVector.begin(), bigrammsVector.end(),
        [](const std::pair<std::string, uint64_t>& a, const std::pair<std::string, uint64_t>& b) {
            return a.second > b.second;
        });

    return bigrammsVector;
}
```

В нас вишло:

```
PS C:\Users\Артем\source\repos\crypta3\x64\Release> .\crypta3.exe .\03.txt
5 most frequent bigramms:
тд:     77
рб:     53
во:     52
що:     45
кд:     42

Text has been decrypted: with key (199, 700) and been written in: "decrypted_texts.txt"
Possible texts: 1
```

3

Для підбору можливих ключей
Беремо пару біграм шифртексту та відкритого текста для рівняння

$$\begin{cases} Y^{*} \equiv aX^{*} + b \pmod{m^2} \\ Y^{**} \equiv aX^{**} + b \pmod{m^2} \end{cases},$$

Для того щоб знайти a и b

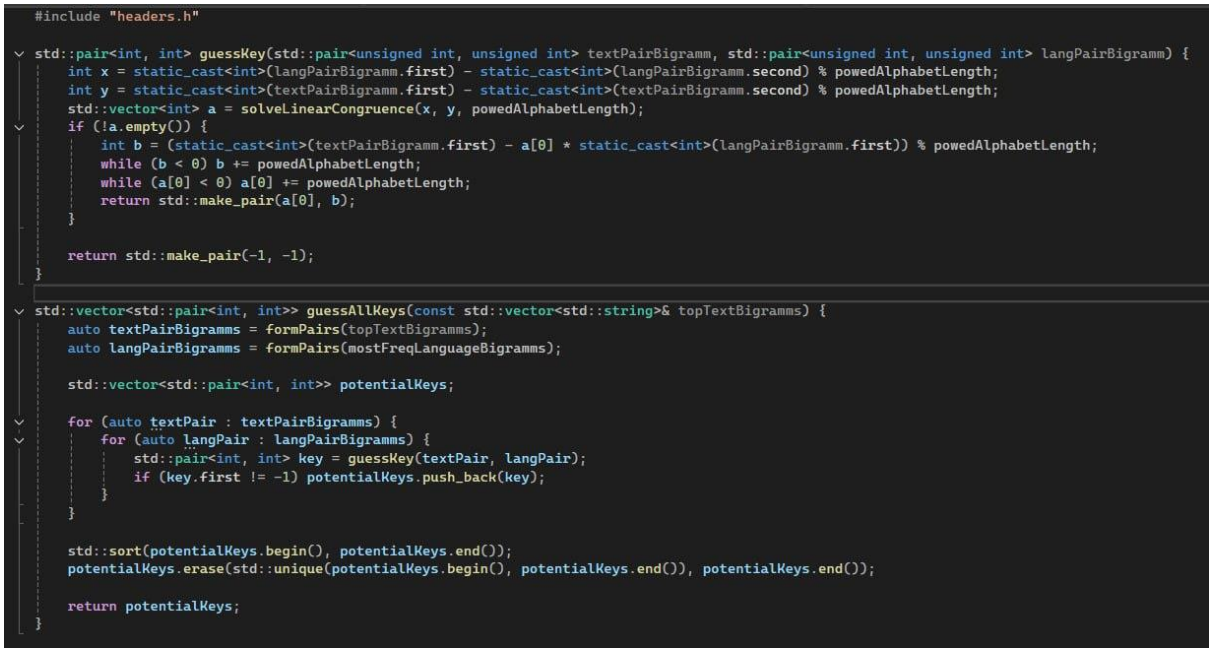
Рахуємо a:

$$Y^{*} - Y^{**} \equiv a(X^{*} - X^{**}) \pmod{m^2}.$$

Рахуємо b:

$$b = (Y^{*} - aX^{*}) \bmod m^2.$$

```cpp
std::map<char, uint64_t> countLetters(std::string& text) {
    std::map<char, uint64_t> letters;
    for (char ch : text) {
        letters[ch]++;
    }
    return letters;
}

std::vector<std::pair<unsigned int, unsigned int>> formPairs(const std::vector<std::string>& bigramms) {
    std::vector<std::pair<unsigned int, unsigned int>> pairs;
    for (int i = 0; i < bigramms.size(); i++) {
        for (int j = i + 1; j < bigramms.size(); j++) {

            unsigned int firstCharIndex = alphabet.find(bigramms[i][0]);
            unsigned int secondCharIndex = alphabet.find(bigramms[i][1]);
            unsigned int firstBigramm = (firstCharIndex * alphabet.length()) + secondCharIndex;

            unsigned int thirdCharIndex = alphabet.find(bigramms[j][0]);
            unsigned int fourthCharIndex = alphabet.find(bigramms[j][1]);
            unsigned int secondBigramm = (thirdCharIndex * alphabet.length()) + fourthCharIndex;

            pairs.push_back({ firstBigramm, secondBigramm });
        }
    }
    return pairs;
}
```



```cpp
#include "headers.h"

std::pair<int, int> guessKey(std::pair<unsigned int, unsigned int> textPairBigramm, std::pair<unsigned int, unsigned int> langPairBigramm) {
    int x = static_cast<int>(langPairBigramm.first) - static_cast<int>(langPairBigramm.second) % powedAlphabetLength;
    int y = static_cast<int>(textPairBigramm.first) - static_cast<int>(textPairBigramm.second) % powedAlphabetLength;
    std::vector<int> a = solveLinearCongruence(x, y, powedAlphabetLength);
    if (!a.empty()) {
        int b = (static_cast<int>(textPairBigramm.first) - a[0] * static_cast<int>(langPairBigramm.first)) % powedAlphabetLength;
        while (b < 0) b += powedAlphabetLength;
        while (a[0] < 0) a[0] += powedAlphabetLength;
        return std::make_pair(a[0], b);
    }

    return std::make_pair(-1, -1);
}

std::vector<std::pair<int, int>> guessAllKeys(const std::vector<std::string>& topTextBigramms) {
    auto textPairBigramms = formPairs(topTextBigramms);
    auto langPairBigramms = formPairs(mostFreqLanguageBigramms);

    std::vector<std::pair<int, int>> potentialKeys;

    for (auto textPair : textPairBigramms) {
        for (auto langPair : langPairBigramms) {
            std::pair<int, int> key = guessKey(textPair, langPair);
            if (key.first != -1) potentialKeys.push_back(key);
        }
    }

    std::sort(potentialKeys.begin(), potentialKeys.end());
    potentialKeys.erase(std::unique(potentialKeys.begin(), potentialKeys.end()), potentialKeys.end());

    return potentialKeys;
}
```

Спочатку переводимо біграми у числа і намагаємося з них сформувати ключі. Додаємо ключ в масив тільки у випадку можливості розв’язку лінійного рівняння, інакше функція поверне пару {-1, -1}, що значить «некоректний ключ», а значить він не буде повернутий з функції вгадування ключів.

4,5

Для кожної пари ключів дешифруємо за подомогою формули:

$$X_i = a^{-1}(Y_i - b) \bmod m^2$$

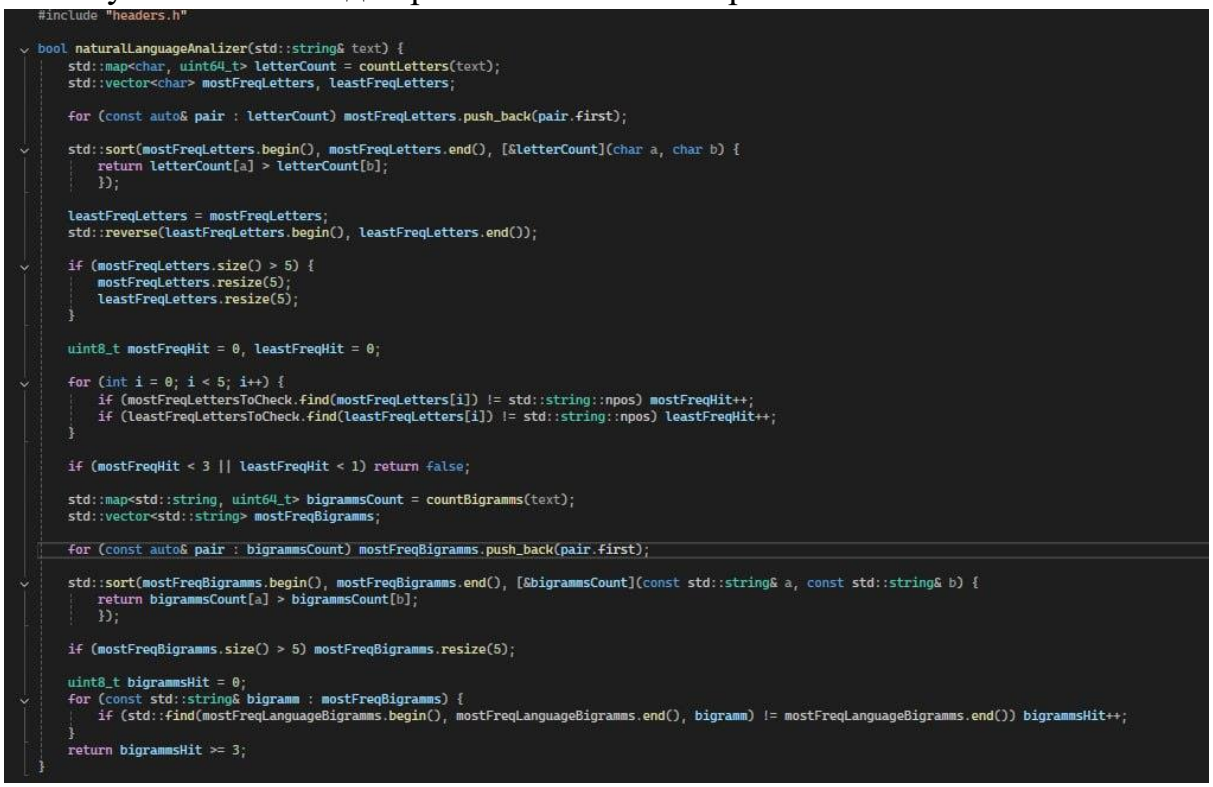
Де a^{-1} це наш зворотний елемент для а по модулю

```cpp
std::string decrypt(const std::string& cryptedText, const std::pair<int, int>& key, const std::string& alphabet) {
    int aInverted = inverseElement(key.first, powedAlphabetLength);

    std::vector<int> textAsNums(cryptedText.size() / 2, 0);

    for (size_t i = 0; i < cryptedText.size() / 2; i++) {
        int firstChar = alphabet.find(cryptedText[i * 2]);
        int secondChar = alphabet.find(cryptedText[i * 2 + 1]);
        textAsNums[i] = firstChar * alphabetLength + secondChar;
    }

    std::vector<int> decryptedNums(textAsNums.size());
    for (size_t i = 0; i < textAsNums.size(); i++) {
        int decryptedNum = (aInverted * (textAsNums[i] - key.second)) % powedAlphabetLength;
        while (decryptedNum < 0) decryptedNum += powedAlphabetLength;
        decryptedNums[i] = decryptedNum;
    }

    std::string decrypted;
    for (int num : decryptedNums) {
        int firstChar = num / alphabetLength;
        int secondChar = num % alphabetLength;
        decrypted += alphabet[firstChar];
        decrypted += alphabet[secondChar];
    }

    return decrypted;
}
```

Перетворюємо біграми у числа, далі розшифровуємо їх ключем і конвертуємо числа у біграми назад і віддаємо це на перевірку «аналізатору натуральної мови»

```cpp
void decryptAndSave(const std::vector<std::pair<int, int>>& potentialKeys, std::string& text, int& written) {
    std::ofstream outFile(L"decrypted_texts.txt");
    if (!outFile) {
        std::wcerr << L"Cannot create file \"decrypted_texts.txt\" for write.\n";
        return;
    }

    for (const auto& key : potentialKeys) {
        int aInverted = inverseElement(key.first, static_cast<int>(std::pow(alphabet.size(), 2)));
        if (aInverted == -1) continue;

        std::string decryptedText = decrypt(text, key, alphabet);
        if (naturalLanguageAnalizer(decryptedText)) {
            outFile << "Key: (" << key.first << ", " << key.second << ")\n";
            outFile << decryptedText << "\n\n";
            std::wcout << L"Text has been decrypted: with key (" << key.first << L", " << key.second
                << L") and been written in: \"decrypted_texts.txt\"\n";
            written++;
        }
    }

    outFile.close();
}
```

Намагаємося з усього масиву отриманих текстів знайти ті, які відповідають критеріям "натуральної мови". Тобто ми перевіряємо частоту найчастіших і найрідших літер російської мови в тексті, а далі перевіряємо частоту найчастіших для російської мови біграм



```cpp
#include "headers.h"

bool naturalLanguageAnalizer(std::string& text) {
    std::map<char, uint64_t> letterCount = countLetters(text);
    std::vector<char> mostFreqLetters, leastFreqLetters;

    for (const auto& pair : letterCount) mostFreqLetters.push_back(pair.first);

    std::sort(mostFreqLetters.begin(), mostFreqLetters.end(), [&letterCount](char a, char b) {
        return letterCount[a] > letterCount[b];
        });

    leastFreqLetters = mostFreqLetters;
    std::reverse(leastFreqLetters.begin(), leastFreqLetters.end());

    if (mostFreqLetters.size() > 5) {
        mostFreqLetters.resize(5);
        leastFreqLetters.resize(5);
    }

    uint8_t mostFreqHit = 0, leastFreqHit = 0;

    for (int i = 0; i < 5; i++) {
        if (mostFreqLettersToCheck.find(mostFreqLetters[i]) != std::string::npos) mostFreqHit++;
        if (leastFreqLettersToCheck.find(leastFreqLetters[i]) != std::string::npos) leastFreqHit++;
    }

    if (mostFreqHit < 3 || leastFreqHit < 1) return false;

    std::map<std::string, uint64_t> bigrammsCount = countBigramms(text);
    std::vector<std::string> mostFreqBigramms;

    for (const auto& pair : bigrammsCount) mostFreqBigramms.push_back(pair.first);

    std::sort(mostFreqBigramms.begin(), mostFreqBigramms.end(), [&bigrammsCount](const std::string& a, const std::string& b) {
        return bigrammsCount[a] > bigrammsCount[b];
        });

    if (mostFreqBigramms.size() > 5) mostFreqBigramms.resize(5);

    uint8_t bigrammsHit = 0;
    for (const std::string& bigramm : mostFreqBigramms) {
        if (std::find(mostFreqLanguageBigramms.begin(), mostFreqLanguageBigramms.end(), bigramm) != mostFreqLanguageBigramms.end()) bigrammsHit++;
    }
    return bigrammsHit >= 3;
}
```

Один з наших ключів, а саме **(a = 199; b = 700)** дав змістовний текст

```
Text has been decrypted: with key (199, 700) and been written in: "decrypted_texts.txt"
Possible texts: 1
```

Знаходимо цей текст у відкритому джерелі

Зигмунд Фрейд Достоевский и отцеубийство

Отцеубийство, как известно, основное и изначальное преступление человечества и отдельного человека. Во всяком случае, оно — главный источник чувства вины, неизвестно, единственный ли; исследованиям не удалось еще установить душевное происхождение вины и потребности искупления. Но отнюдь не существенно — единственный ли это источник. Психологическое положение сложно и нуждается в объяснениях. Отношение мальчика к отцу, как мы говорим, амбивалентно. Помимо ненависти, из-за которой хотелось бы отца, как соперника, устранить, существует обычно некоторая доля нежности к нему. Оба отношения сливаются в идентификацию с отцом, хотелось бы занять место отца, потому что он вызывает восхищение, хотелось бы быть, как он, и потому, что хочется его устранить. Все это наталкивается на крупное препятствие. В определенный момент ребенок начинает понимать, что попытка устранить отца как соперника, встретила бы со стороны отца наказание через кастрацию. Из страха кастрации, то есть в интересах сохранения своей мужественности, ребенок отказывается от желания обладать матерью и от устранения отца. Поскольку это желание остается в области бессознательного, оно является основой для образования чувства вины. Нам кажется, что мы описали нормальные процессы, обычную судьбу так называемого Эдипова комплекса; следует, однако, внести важное дополнение.

Висновки

Під час виконання лабороторної роботи ми навчились виявленню криптографічних недоліки за моноалфавітної підставновки через аналіз частотний біграм. Таким чином цей підхід є оптимальним щоб дешифрувати зашифровані тексти, знаходячи різницу між вживаними біграмами в шифртексти та їх походженням. Це дуже вплинуло на наш розвиток та спринянню криптографії. Ми засвоїли методи лінійних рівнянь та навчили на практиці використовувати ці знання.