Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №1**

По дисциплине «Криптографические методы защиты информации»

Тема: «Основные принципы криптографии»

**Выполнил:**

Студент 2 курса

Группы ИИ-23

Макаревич Н.Р.

**Проверил:**

Хацкевич М. В.

Брест 2024

**Цель:** научиться применять алгоритмы шифрования и сжатия информации.

**Ход работы:**

Шифрование методом Хила:

int Hill::getAlphabetPosition(char letter) {

if (letter == '.')

return 25;

if (letter == ',')

return 26;

if (letter == '-')

return 27;

if (letter == ' ')

return 28;

letter = std::toupper(letter);

int asciiCode = static\_cast<int>(letter);

int alphabetPosition = asciiCode - 65;

return alphabetPosition;

}

char Hill::getAlphabetLetter(int num) {

if (num < 0) {

num += 29;

}

if (num > -1 && num < 25)

return 'A' + num;

if (num == 25)

return '.';

if (num == 26)

return ',';

if (num == 27)

return '-';

if (num == 28)

return ' ';

return '\*';

}

Hill::Hill() {

size = 2;

numOfLetters = 29;

fillerLetter = '.';

textLimiter = ';';

enkey = std::vector<std::vector<int>>(size, std::vector<int>(size, 0));

/\*for (int i = 0; i < size; i++) {

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

enkey[i][j] = getRandomNumber();

}

}\*/

enkey[0][0] = 4;

enkey[0][1] = 5;

enkey[1][0] = 3;

enkey[1][1] = 4;

dekey = enkey;

dekey[0][1] \*= -1;

dekey[1][0] \*= -1;

}

void Hill::encryptText(std::string& path) {

cryptText(path, true);

}

std::vector<int> Hill::transformToNumbers(std::string str) {

std::vector <int> res(str.size());

for (int i = 0; i < str.size(); i++) {

res[i] = getAlphabetPosition(str[i]);

}

return res;

}

std::vector<int> Hill::encrypt(std::vector<int> origText, bool mode) {

//mode true = encrypt

//mode false = decrypt

std::vector <int> res(origText.size());

if (mode) {

for (int i = 0; i < origText.size(); i++) {

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

res[i] += enkey[i][j] \* origText[j];

}

res[i] = res[i] % numOfLetters;

}

}

else {

for (int i = 0; i < origText.size(); i++) {

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

res[i] += dekey[i][j] \* origText[j];

}

res[i] = res[i] % numOfLetters;

}

}

return res;

}

void Hill::saveEncryptedText(std::string path, std::vector<int> encryptedText) {

std::ofstream fout;

fout.open(path, std::ios::app);

for (int i = 0; i < encryptedText.size(); i++) {

fout << getAlphabetLetter(encryptedText[i]);

}

fout.close();

}

void Hill::decryptText(std::string& path) {

cryptText(path, false);

}

void Hill::cryptText(std::string& path, bool mode) {

//true = encrypt

//false = decrypt

std::string foutPath;

bool subMode;

if (mode) {

foutPath = "encrypted.txt";

subMode = true;

}

else {

foutPath = "decrypted.txt";

subMode = false;

}

std::ifstream fin;

fin.open(path);

if (!fin.is\_open()) {

std::cout << "Failed to open file: " << path << std::endl;

return;

}

std::string line;

cleanFile(foutPath);

while (std::getline(fin, line, textLimiter)) {

if (line.length() >= size) {

for (size\_t i = 0; i < line.length(); i += size) {

std::string substr = line.substr(i, size);

if (substr.length() == size)

saveEncryptedText(foutPath, encrypt(transformToNumbers(substr), subMode));

else {

substr.append(size - substr.length(), fillerLetter);

saveEncryptedText(foutPath, encrypt(transformToNumbers(substr), subMode));

}

}

}

else {

line.append(63 - line.length(), fillerLetter);

saveEncryptedText(foutPath, encrypt(transformToNumbers(line), subMode));

}

}

std::ofstream fout(foutPath, std::ios::app);

fout << ";";

fin.close();

}

**Сжатие методом Хаффмена:**

Huffman::Huffman() {

alphabetSize = 26;

alphabet = std::vector <node\*>();

for (int i = 0; i < alphabetSize; i++) {

node\* newNode = new node(i);

alphabet.push\_back(newNode);

}

alphabet[0]->chance = 817;

alphabet[1]->chance = 149;

alphabet[2]->chance = 278;

alphabet[3]->chance = 425;

alphabet[4]->chance = 1270;

alphabet[5]->chance = 223;

alphabet[6]->chance = 202;

alphabet[7]->chance = 609;

alphabet[8]->chance = 697;

alphabet[9]->chance = 15;

alphabet[10]->chance = 77;

alphabet[11]->chance = 403;

alphabet[12]->chance = 241;

alphabet[13]->chance = 675;

alphabet[14]->chance = 751;

alphabet[15]->chance = 193;

alphabet[16]->chance = 10;

alphabet[17]->chance = 599;

alphabet[18]->chance = 633;

alphabet[19]->chance = 906;

alphabet[20]->chance = 276;

alphabet[21]->chance = 98;

alphabet[22]->chance = 236;

alphabet[23]->chance = 15;

alphabet[24]->chance = 197;

alphabet[25]->chance = 7;

encodedLetters = std::vector<std::string>(alphabet.size(), "");

encodeLetters();

}

std::pair<int, int> Huffman::find2minId(std::vector<node\*>& vec) {

std::pair <int, int> minId = { -1, -1 };

std::pair <int, int> minChance = { 99999, 99999 };

int temp = -1;

for (int i = 0; i < vec.size(); i++) {

if (!vec[i]->isUsed && vec[i]->chance < minChance.first && i != temp) {

minChance.first = vec[i]->chance;

minId.first = i;

temp = i;

}

}

for (int i = 0; i < vec.size(); i++) {

if (!vec[i]->isUsed && vec[i]->chance < minChance.second && i != temp) {

minChance.second = vec[i]->chance;

minId.second = i;

}

}

return minId;

}

void Huffman::encodeLetters() {

std::vector <node\*> tempVector;

tempVector = alphabet;

std::pair <int, int> minId = { -1, -1 };

while(true){

minId = find2minId(tempVector);

if (minId.first == -1 || minId.second == -1) {

break;

}

tempVector[minId.first]->isUsed = true;

tempVector[minId.second]->isUsed = true;

node\* newNode = new node(tempVector.size());

tempVector[minId.first]->next = newNode;

tempVector[minId.second]->next = newNode;

newNode->prev1 = tempVector[minId.first];

newNode->prev2 = tempVector[minId.second];

newNode->chance = tempVector[minId.first]->chance + tempVector[minId.second]->chance;

tempVector.push\_back(newNode);

minId = { -1, -1 };

}

for (int i = 0; i < alphabet.size(); i++) {

encodeHelper(tempVector[i], encodedLetters[i]);

//encodedLetters[i] = encodedLetters[i].substr(0, encodedLetters[i].length() / 2);

std::reverse(encodedLetters[i].begin(), encodedLetters[i].end());

std::cout << tempVector[i]->letter << " " << encodedLetters[i] << "\n";

}

}

void Huffman::encodeHelper(node\* cur, std::string& res) {

if (cur->next == nullptr)

return;

node\* nextNode = cur->next;

if (nextNode->prev1 >= nextNode->prev2) {

if (nextNode->prev1->letter == cur->letter)

res += '1';

else if (nextNode->prev2->letter == cur->letter)

res += '0';

}

else if (nextNode->prev1 < nextNode->prev2) {

if (nextNode->prev1->letter == cur->letter)

res += '0';

else if (nextNode->prev2->letter == cur->letter)

res += '1';

}

encodeHelper(nextNode, res);

}

std::string Huffman::encodeLetter(char letter) {

for (int i = 0; i < alphabetSize; i++) {

if (alphabet[i]->letter == letter)

return encodedLetters[i];

}

return "\*-\*";

}

void Huffman::compressData(std::string path) {

std::ifstream fin(path);

std::ofstream fout("compressed.txt");

if (!fin.is\_open()) {

std::cout << "File don't exist\n";

fin.close();

return;

}

char letter;

while (fin.get(letter)) {

fout << encodeLetter(letter);

}

fin.close();

fout.close();

}

char Huffman::decodeLetter(std::string str) {

for (int i = 0; i < alphabetSize; i++) {

if (encodedLetters[i] == str)

return alphabet[i]->letter;

}

return '\*';

}

void Huffman::decompressData(std::string path) {

std::ifstream fin(path);

std::ofstream fout("decompressed.txt");

if (!fin.is\_open()) {

std::cout << "File don't exist\n";

fin.close();

return;

}

char number;

std::string tempstr = "";

while (fin.get(number)) {

tempstr += number;

char foundLetter = decodeLetter(tempstr);

if (foundLetter != '\*') {

fout << foundLetter;

tempstr = "";

}

}

fin.close();

fout.close();

}

**Вывод:** в ходе лабораторной работы я научился шифровать и сжимать информацию.