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-21

Литвинюк Т. В.

**Проверил:**

Хацкевич М. В.

Брест 2023

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

**Ход работы:**

Шифрование

#include <iostream>

#include <fstream>

using namespace std;

void get\_index(char \*\*tabl, int lenI, int lenJ, char symb, int &i, int &j){

for (i = 0; i < lenI; i++)

for (j = 0; j < lenJ; j++)

if (tabl[i][j] == symb)

return;

i = -1; j = -1;

}

string encrypt(string path, char \*\*alph){

ifstream file(path);

string text, encryptedText;

int i, j;

getline(file, text);

for (int x = 0; x < text.length(); x++){

get\_index(alph, 5, 6, text[x], i, j);

if (i != -1 and j != -1)

encryptedText += to\_string(i) + to\_string(j);

}

file.close();

return encryptedText;

}

string decrypt(string cipher, char \*\*alph){

string decryptedText;

for (int x = 0; x < cipher.length(); x += 2){

int index\_i = cipher[x] - '0';

int index\_j = cipher[x+1] - '0';

decryptedText += alph[index\_i][index\_j];

}

return decryptedText;

}

int main(){

string alphabet="abcdefghijklmnopqrstuvwxyz.,? ";

char \*\*alph;

alph = new char \*[5];

for (int i = 0; i < 5; i++)

alph[i] = new char[6];

for (int i = 0; i < 30; i++){

alph[i / 6][i % 6] = alphabet[i];

}

string path;

cout << "Введите имя файла: "; getline(cin, path);

ofstream file(path+"\_encr");

file << encrypt(path, alph);

cout << "Успешно!";

file.close();

for (int i = 0; i < 5; i++)

delete [] alph[i];

delete [] alph;

}

Сжатие

#include <iostream>

#include <fstream>

#include <queue>

#include <unordered\_map>

using namespace std;

// Definition of a node in the Huffman tree

struct Node {

char c;

int freq;

Node\* left;

Node\* right;

Node(char c, int freq, Node\* left = nullptr, Node\* right = nullptr) {

this->c = c;

this->freq = freq;

this->left = left;

this->right = right;

}

~Node() {

if (left != nullptr) delete left;

if (right != nullptr) delete right;

}

};

// Build the Huffman tree based on the frequency table

Node\* build\_huffman\_tree(const unordered\_map<char, int>& freq\_table) {

// Build a priority queue of nodes

auto cmp = [](Node\* a, Node\* b) { return a->freq > b->freq; };

priority\_queue<Node\*, vector<Node\*>, decltype(cmp)> pq(cmp);

for (auto& p : freq\_table) {

pq.push(new Node(p.first, p.second));

}

// Build the Huffman tree

while (pq.size() > 1) {

Node\* min1 = pq.top();

pq.pop();

Node\* min2 = pq.top();

pq.pop();

Node\* internal\_node = new Node('\0', min1->freq + min2->freq, min1, min2);

pq.push(internal\_node);

}

// Return the root of the Huffman tree

return pq.top();

}

void generate\_huffman\_codes\_helper(Node\* node, string& code, unordered\_map<char, string>& huffman\_codes) {

if (node->left == nullptr && node->right == nullptr) {

huffman\_codes[node->c] = code;

return;

}

code.push\_back('0');

generate\_huffman\_codes\_helper(node->left, code, huffman\_codes);

code.pop\_back();

code.push\_back('1');

generate\_huffman\_codes\_helper(node->right, code, huffman\_codes);

code.pop\_back();

}

// Generate the Huffman codes for each symbol in the Huffman tree

unordered\_map<char, string> generate\_huffman\_codes(Node\* root) {

unordered\_map<char, string> huffman\_codes;

string code = "";

generate\_huffman\_codes\_helper(root, code, huffman\_codes);

return huffman\_codes;

}

// Encode the input text using the Huffman codes

string encode(const string& text, const unordered\_map<char, string>& huffman\_codes) {

string encoded\_text = "";

for (char c : text) {

encoded\_text += huffman\_codes.at(c);

}

return encoded\_text;

}

// Decode the encoded text using the Huffman tree

string decode(const string& encoded\_text, Node\* root) {

string decoded\_text = "";

Node\* node = root;

for (char c : encoded\_text) {

if (c == '0') {

node = node->left;

} else if (c == '1') {

node = node->right;

}

if (node->left == nullptr && node->right == nullptr) {

decoded\_text += node->c;

node = root;

}

}

return decoded\_text;

}

int main() {

// Read the input text

string path, text;

cout << "Введите имя файла: "; getline(cin, path);

ifstream file(path);

getline(file, text);

file.close();

// Compute the frequency table

unordered\_map<char, int> freq\_table;

for (char c : text) {

freq\_table[c]++;

}

// Build the Huffman tree and generate the Huffman codes

Node\* root = build\_huffman\_tree(freq\_table);

unordered\_map<char, string> huffman\_codes = generate\_huffman\_codes(root);

// Encode the input text using the Huffman codes

string encoded\_text = encode(text, huffman\_codes);

ofstream out(path + "\_comp");

out << endl << encoded\_text;

out.close();

cout << "Encoded text: " << encoded\_text << endl;

// Decode the encoded text using the Huffman tree

string decoded\_text = decode(encoded\_text, root);

cout << "Decoded text: " << decoded\_text << endl;

// Clean up the memory

delete root;

}

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