`

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;
}
```

hello world ->11041515224534222515 03

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;
}
```

```
Encoded text: 011000101011011111010110001101110
Decoded text: hello world
```

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