МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

**ИНТЕЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ**

ОТЧЁТ

По лабораторной работе № \_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Выполнил:

Студент группы ИИ-22

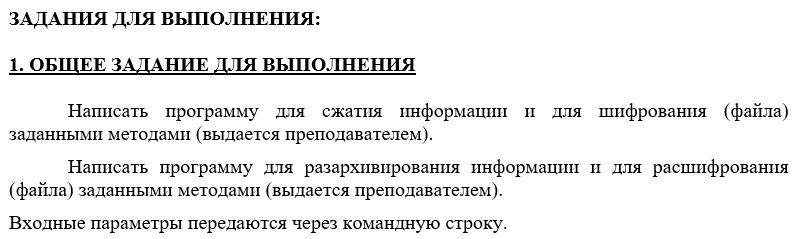
Копанчук Евгений Романович

Проверил\_\_:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Брест – 2023

**Ход работы**



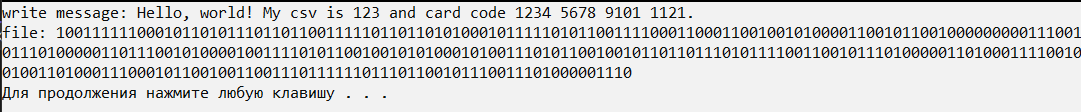
**HUFFMAN**

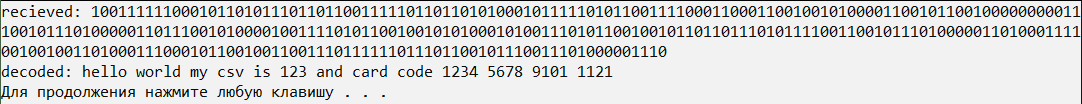
|  |
| --- |
| class Huffman {  private:  struct Node { char ch; int freq; Node\* left, \* right; };  struct comp { bool operator()(Node\* l, Node\* r) { return l->freq > r->freq; } };  unordered\_map<char, string> huffmanCode;  Node\* root;  public:  Huffman() {  string text = "abcdefghiklmnopqrstuvwxyz0123456789 ";  unordered\_map<char, int> freq;  for (char ch : text) freq[ch]++;  priority\_queue<Node\*, vector<Node\*>, comp> pq;  for (auto pair : freq) pq.push(GET\_NODE(pair.first, pair.second, nullptr, nullptr));  while (pq.size() != 1) {  Node\* left = pq.top(); pq.pop();  Node\* right = pq.top(); pq.pop();  int sum = left->freq + right->freq;  pq.push(GET\_NODE('\0', sum, left, right));  }  this->root = pq.top();  ENCODE(root, "", huffmanCode);  }  Node\* GET\_NODE(char ch, int freq, Node\* left, Node\* right) {  Node\* node = new Node();  node->ch = ch;  node->freq = freq;  node->left = left;  node->right = right;  return node;  }  void ENCODE(Node\* root, string str, unordered\_map<char, string>& huffmanCode) {  if (root == nullptr) return;  if (!root->left && !root->right) huffmanCode[root->ch] = str;  ENCODE(root->left, str + "0", huffmanCode);  ENCODE(root->right, str + "1", huffmanCode);  }  string COMPRESS(string message) {  string str = "";  for (char ch : message) str += huffmanCode[ch];  return str;  }  void DECODE(Node\* root, int& index, string str, char& ch) {  if (root == nullptr) { cout << "Tree is empty." << endl; return; }  if (!root->left && !root->right) { ch = root->ch; return; }  index++;  if (str[index] == '0') DECODE(root->left, index, str, ch);  else DECODE(root->right, index, str, ch);  }  string DECOMPRESS(string message) {  string str = ""; int index = -1;  while (index < (int)message.size() - 2) {  char next;  DECODE(root, index, message, next);  str += next;  }  return str;  }  }; |

**POLYBIUS**

|  |
| --- |
| class Polybius {  private:  vector<char> PolybiusSquare;  public:  Polybius() {  PolybiusSquare = {  'a', 'b', 'c', 'd', 'e', 'f',  'g', 'h', 'i', 'k', 'l', 'm',  'n', 'o', 'p', 'q', 'r', 's',  't', 'u', 'v', 'w', 'x', 'y',  'z', '0', '1', '2', '3', '4',  '5', '6', '7', '8', '9', ' '  };  }  int INDEX(char ch) {  for (int i = 0; i < 36; i++)  if (PolybiusSquare[i] == ch) return i;  return -1;  }  int REVERSE(int index) {  int row = index / 6, column = index % 6; return 6 \* column + row;  }  string ENCODE(string message) {  transform(message.begin(), message.end(), message.begin(),  [](unsigned char c) { return tolower(c); });  string str = "";  int size = message.size();  for (int i = 0; i < size; i++) {  int id = INDEX(message[i]);  if (id != -1) str += PolybiusSquare[REVERSE(id)];  }  return str;  }  string DECODE(string message) {  string str = ""; int size = message.size();  for (int i = 0; i < size; i++) {  int id = INDEX(message[i]);  if (id != -1) str += PolybiusSquare[REVERSE(id)];  }  return str;  }  }; |

**OUTPUT**

****

****