Федеральное государственное автономное образовательное учреждение высшего образования
«Российский университет дружбы народов имени Патриса Лумумбы»
Инженерная академия
Кафедра механики и процессов управления

ОТЧЕТ

По **Информатике и программированию**

Направление: **Прикладная математика и информатика**
(код направления / название направления)

Профиль: **Математические методы механики космического полета и анализ геоинформационных данных**
(название профиля)

Тема: **Алгоритмы сжатия данных: реализация кодирования Хаффмана.**
(название лабораторной / курсовой)

Выполнено: **Юнгова Анастасия Сергеевна**
(ФИО)

Группа: **ИПМбд-02-23**
№ студенческого: 1132233510

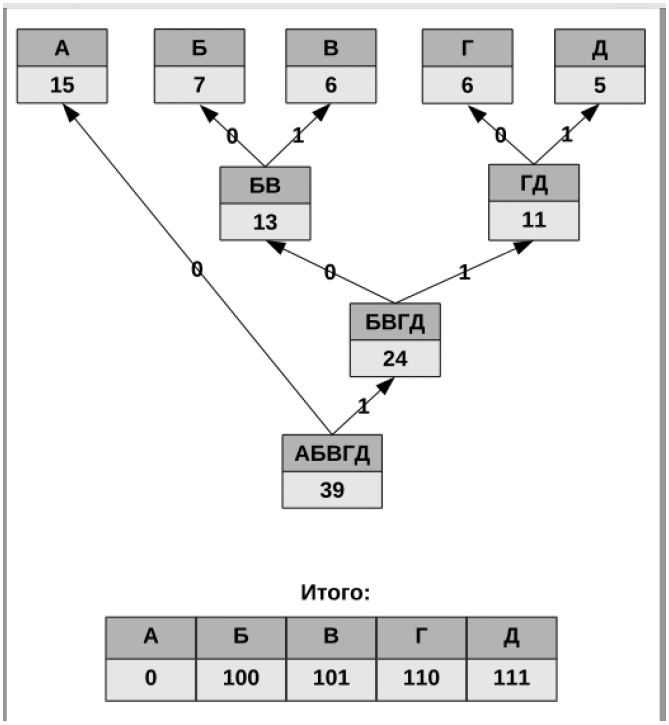
Москва, 2025

Теоретическая сводка: алгоритм Хаффмана

Кодирование Хаффмана — это алгоритм сжатия данных без потерь, основанный на частоте появления символов.
Он был предложен Дэвидом Хаффманом в 1952 году. Основная идея заключается в использовании переменной длины кодов для представления символов: чем чаще символ встречается, тем короче его код.

Основные шаги алгоритма:

1. Подсчитать частоту каждого символа во входном сообщении.
2. Построить приоритетную очередь (очередь с минимальным приоритетом), где каждый узел — это символ и его частота.
3. Извлекать два узла с наименьшими частотами и объединять их в новый узел, сумма частот которых равна частоте нового узла.
4. Повторять шаг 3, пока в очереди не останется один узел — это и будет корень дерева Хаффмана.
5. Обход дерева позволяет задать бинарный код для каждого символа.



Преимущества:

- Эффективное сжатие для текстов с неравномерным распределением символов.
- Простота реализации.

Недостатки:

- Неэффективно при равномерном распределении символов.
- Требует дополнительной информации (например, дерево или таблицу кодов) для декодирования.

Алгоритм включает два этапа:

- Кодирование: преобразование текста в битовую последовательность.
- Декодирование: восстановление исходного текста из битовой последовательности.

Приложение

```cpp
#include <iostream>
#include <queue>
#include <unordered_map>
#include <vector>
#include <string>
#include <memory>
#include <iomanip>

using namespace std;




class HuffmanEncoder {
    private:

        struct Node {
            char ch;
            int freq;
            shared_ptr<Node> left, right;

            Node(char c, int f) : ch(c), freq(f), left(nullptr), right(nullptr) {}
            Node(shared_ptr<Node> l, shared_ptr<Node> r) : ch('\0'), freq(l->freq + r->freq), left(l), right(r) {}
        };


        struct Compare {
            bool operator()(shared_ptr<Node> a, shared_ptr<Node> b) {
                return a->freq > b->freq;
            }
        };


        shared_ptr<Node> root;
        unordered_map<char, string> huffmanCode;


        void buildCodeTable(shared_ptr<Node> node, const string& str) {
            if (!node) return;
            if (!node->left && !node->right) huffmanCode[node->ch] = str;
            buildCodeTable(node->left, str + "0");
            buildCodeTable(node->right, str + "1");
        }

    public:
```

```
92          void printCodeTable() {
93              cout << "\nHuffman Code Table:\n";
94              cout << "Char | Code\n";
95              cout << "-------------\n";
96              for (const auto& pair : huffmanCode) {
97                  if (pair.first == ' ') cout << "' '  | " << pair.second << endl;
98                  else cout << " " << pair.first << "   | " << pair.second << endl;
99              }
100         }
101
102
103         void evaluateCompression(const string& originalText, const string& encodedText) {
104             int originalBits = originalText.length() * 8;
105             int compressedBits = encodedText.length();
106             double ratio = (double)compressedBits / originalBits * 100;
107
108             cout << fixed << setprecision(2);
109             cout << "\nCompression Analysis:\n";
110             cout << "Original size:   " << originalBits << " bits\n";
111             cout << "Compressed size: " << compressedBits << " bits\n";
112             cout << "Compression rate: " << ratio << " %\n";
113         }
114 };
115
116
117
118
119 int main() {
120     while (true) {
121     cout << "Huffman Compression Menu\n";
122     cout << "  1. Encode text\n";
123     cout << "  2. Exit\n";
124     cout << "\nChoose an option: ";
125
126     int choice;
127     cin >> choice;
128
129     if (cin.fail()) {
130         cin.clear();
131         cin.ignore();
132         cout << "Invalid input. Please enter a number.\n";
133         continue;
134     }
135
```

```
45  public:
46
47      void build(const string& text) {
48          unordered_map<char, int> freq;
49          for (char ch : text) freq[ch]++;
50
51          priority_queue<shared_ptr<Node>, vector<shared_ptr<Node>>, Compare> pq;
52          for (auto pair : freq) {
53              pq.push(make_shared<Node>(pair.first, pair.second));
54          }
55
56          while (pq.size() > 1) {
57              auto left = pq.top(); pq.pop();
58              auto right = pq.top(); pq.pop();
59              pq.push(make_shared<Node>(left, right));
60          }
61
62          root = pq.top();
63          buildCodeTable(root, "");
64      }
65
66
67      string encode(const string& text) {
68          string encoded = "";
69          for (char ch : text) {
70              encoded += huffmanCode[ch];
71          }
72          return encoded;
73      }
74
75
76      string decode(const string& encodedText) {
77          string decoded = "";
78          shared_ptr<Node> current = root;
79          for (char bit : encodedText) {
80              if (bit == '0') current = current->left;
81              else current = current->right;
82
83              if (!current->left && !current->right) {
84                  decoded += current->ch;
85                  current = root;
86              }
87          }
88          return decoded;
89      }
```

```
129     if (cin.fail()) {
132         cout << "Invalid input. Please enter a number.\n";
133         continue;
134     }
135
136     cin.ignore();
137
138
139     switch (choice) {
140         case 1: {
141             string text;
142             cout << "\nEnter text to encode: ";
143             getline(cin, text);
144
145             HuffmanEncoder encoder;
146             encoder.build(text);
147             encoder.printCodeTable();
148
149             string encoded = encoder.encode(text);
150             cout << "\nEncoded text: " << encoded << endl;
151
152             string decoded = encoder.decode(encoded);
153             cout << "Decoded text: " << decoded << endl;
154
155             encoder.evaluateCompression(text, encoded);
156
157             cout << "\n----------------------------------------\n";
158             break;
159         }
160
161         case 2:
162             cout << "Exiting...\n";
163             return 0;
164
165         default:
166             cout << "Invalid choice. Please try again.\n";
167             break;
168     }
169 }
170
171     return 0;
172 }
```

Ссылка на гитхаб