Министерство науки и высшего образования Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Фундаментальные Науки»

КАФЕДРА ФН-12 «Математическое моделирование»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ НА ТЕМУ:

Сжатие по Хаффману

Студент: _____ Мациевский И. М.

дата, подпись Ф.И.О.

Преподаватель: _____ Волкова Л. Л.

дата, подпись Ф.И.О.

Москва, 2023

Содержание

Введение

Цель лабораторной работы: реализовать алгоритмы сжатия по Хаффману.
Для достижения поставленной цели требуется решить следующие **задачи**.

1. Описать сжатие по Хаффману.
2. Реализовать сжатие файла.
3. Реализовать возможность считывать данные из файла.
4. Реализовать возможность записать ответ в бинарный файл.
5. Провести небольшое исследование для 10 файлов с одинаковым характером наполнения (с увеличением размера на каждом шаге не менее, чем на 20 %) с оценкой эффекта от сжатия, визуализировать результаты в виде графика.

Согласно варианту, требуется разработать сжатие файла по Хаффману для нотной записи формата инструмент-темп-скорость.

1 Аналитическая часть

Сжатие по Хаффману — это метод сжатия данных, при котором используется оптимальное префиксное бинарное дерево, которое называется деревом Хаффмана. Основная идея заключается в том, чтобы присвоить более короткие бинарные коды (битовые последовательности) более часто встречающимся символам или фрагментам данных. Процесс построения дерева Хаффмана включает следующие шаги.

1. **Вычисление частот символов** — подсчитываются частоты встречаемости каждого символа или символьного фрагмента в исходных данных.

2. **Построение дерева.** Строится бинарное дерево, где каждый лист представляет собой символ или фрагмент данных, а расстояние от корня до листа соответствует длине кода. Чем ближе к корню, тем короче код.

3. **Присвоение кодов.** Каждому символу присваивается уникальный бинарный код, который представляет собой последовательность на пути от корня до листа в дереве.

4. **Сжатие данных.** Заменяются исходные данные соответствующими бинарными кодами. Более часто встречающиеся символы получают более короткие коды, что обеспечивает общее уменьшение объема данных.

5. **Декодирование.** При восстановлении данных используется тот же дерево Хаффмана для раскодирования бинарных данных и восстановления исходных символов или фрагментов.

2 Конструкторская часть

Ниже представлены структуры и методы, реализованные в работе.

1. Структура $Node$: узел дерева Хаффмана. Содержит информацию об идентификаторе, частоте, а также указатели на левого и правого потомков.

2. Метод $getNode$ создает новый узел с указанным идентификатором, частотой и потомками.

3. Структура $Compare$ оператор сравнения, необходимый для приоритетной очереди при построении дерева Хаффмана. Он используется для сравнения частот узлов. Для его работы перегружается оператор (). Сравнивается значение $freq$ (частота) для двух узлов. Если частота узла l больше частоты узла r, то метод возвращает $true$, что говорит о том, что узел l имеет более высокий приоритет. Иначе возвращает $false$.

4. $encode$ рекурсивно проходит по дереву и строит Хаффман-коды для каждого узла. Код для каждого листа (конечного узла) записывается в $huffmanCode$.

5. $writeHuffmanCodesToFile$ записывает Хаффман-коды (идентификаторы, частоты, длины кодов) и закодированный текст в бинарный файл.

6. $gain$ рассчитывает выигрыш от сжатия по Хаффману: в $bit1$ хранится количество бит в исходном тексте (без сжатия), в $bit2$ хранится количество бит в сжатом тексте после применения Хаффман-кодов.
   $dataSize$ представляет собой размер метаданных, которые записываются в бинарный файл. Для каждого идентификатора записывается сам идентификатор (строка), частота (int) и длина Хаффман-кода ($char$).
   $compressedSize$ представляет собой размер закодированного текста после применения Хаффман-кодов.
   win это отношение количества бит в исходном тексте к общему размеру сжатых данных и метаданных.

7. $buildHuffmanTree$ используется в построении кодов Хаффмана для уникальных идентификаторов (в данном случае, нот) в тексте. Ниже представлен алгоритм работы.

   (a) Проход по тексту и подсчет частоты каждого уникального идентификатора (ноты).

   (b) Создание приоритетной очереди для узлов дерева с учетом частоты. На этом этапе каждый уникальный идентификатор (нота) представляется в виде узла дерева.

(c) Используя приоритетную очередь, строится дерево Хаффмана. На каждом шаге извлекаются два узла с наименьшей частотой, создается новый узел с их суммарной частотой, и этот новый узел добавляется обратно в приоритетную очередь. Процесс повторяется до тех пор, пока в приоритетной очереди не останется только один узел - корень дерева Хаффмана.

(d) Рекурсивно обходится дерево Хаффмана, начиная с корня, и присваиваются уникальные коды (строки из '0' и '1') каждому узлу.
Коды сохраняются в $huffmanCode$.

(e) Создается закодированный текст, при этом каждый уникальный идентификатор заменяется его соответствующим кодом Хаффмана.

(f) Коды и метаданные записываются в бинарный файл $Huffman_codes.bin$. В файле каждый уникальный идентификатор представлен своим строковым представлением, за которым следует его частота и длина кода.

(g) Выводится информация о том, во сколько раз удалось сжать исходный текст с использованием алгоритма Хаффмана.

3 Технологическая часть

Для реализации выбран язык C++. На листинге 1 представлена реализация программы (Реализация 1)

Листинг 1 – Исходный код

```
#include <iostream>
#include <fstream>
#include <queue>
#include <map>
#include <vector>
#include <string>

using namespace std;

struct Node {
    string id;
    int freq;
    Node* left, *right;
};

struct Compare {
    bool operator()(Node* l, Node* r) {
        return (l->freq > r->freq);
    }
};

Node* getNode(string id, int freq, Node* left, Node* right) {
    Node* node = new Node();
    node->id = id;
    node->freq = freq;
    node->left = left;
    node->right = right;
    return node;
}

void encode(Node* root, string str, map<string, string>& huffmanCode) {
    if (root == nullptr) return;
    if (!root->left && !root->right) {
        huffmanCode[root->id] = str;
    }
    encode(root->left, str + "0", huffmanCode);
```

```
    encode(root->right, str + "1", huffmanCode);
}

void writeHuffmanCodesToFile(const map<string, string>& huffmanCode,
   const string& encodedText, const map<string, int>& freq) {
    ofstream outputFile("/Users/ilya/DownloadsТипы/ и структуры данных 2
   курс, 1 семестр/laba_7/laba_7/Huffman_codes.bin", ios::binary);

    if (outputFile.is_open()) {
        for (const auto& pair : huffmanCode) {
            // Записываем уникальный строковый идентификатор ноты
            outputFile.write(pair.first.c_str(), pair.first.size());
            outputFile.put('\0');  // Разделитель между идентификатором и
   частотой

            int frequency = freq.at(pair.first);
            outputFile.write(reinterpret_cast<const char*>(&frequency),
    sizeof(frequency));

            char codeLength = static_cast<char>(pair.second.length());
            outputFile.put(codeLength);
        }

        outputFile.put('\0'); // Разделитель между метаданными и
   закодированным текстом

        for (char ch : encodedText) {
            outputFile.put(ch);
        }

        outputFile.close();
        cout << "Данные успешно записаны в бинарный файл 'huffman_codes.bin'
   " << endl;
    }
    else {
        cout << "Невозможно открыть файл для записи" << endl;
    }
}

void gain(string text, const map<string, string>& huffmanCode, const
   map<string, int>& freq) {
```

```
    int bit_1 = text.size();
    int bit_2 = 0;

    for (const auto& pair : huffmanCode) {
        bit_2 += pair.second.size() * freq.at(pair.first) + pair.first.
   size() + pair.second.size();
    }

    int compressedSize = bit_2;  // размер закодированного текста в байтах
    double win = (bit_1 + 0.0) / (compressedSize);

    cout << "Сжатие в " << bit_1 << " байт / (" << compressedSize << "
   байт + " << " байт) = " << win << " раз" << endl;
}



void buildHuffmanTree(string text) {
    map<string, int> freq;
    size_t pos = 0;
    size_t nextPos = text.find('\n', pos);

    while (nextPos != string::npos) {
        string id = text.substr(pos, nextPos - pos);
        freq[id]++;
        pos = nextPos + 1;
        nextPos = text.find('\n', pos);
    }

    priority_queue<Node*, vector<Node*>, Compare> pq;

    for (const auto& pair : freq) {
        pq.push(getNode(pair.first, pair.second, nullptr, nullptr));
    }

    while (pq.size() != 1) {
        Node* left = pq.top(); pq.pop();
        Node* right = pq.top(); pq.pop();

        int sumFreq = left->freq + right->freq;
        pq.push(getNode("", sumFreq, left, right));
```

```
    }

    Node* root = pq.top();

    map<string, string> huffmanCode;
    encode(root, "", huffmanCode);

    string encodedText;
    pos = 0;
    nextPos = text.find('\n', pos);

    while (nextPos != string::npos) {
        string id = text.substr(pos, nextPos - pos);
        encodedText += huffmanCode[id];
        pos = nextPos + 1;
        nextPos = text.find('\n', pos);
    }

    cout << "Частоты идентификаторов нот: " << endl;
    for (const auto& pair : freq) {
        cout << pair.first << " : " << pair.second << endl;
    }
    cout << endl;

    cout << "Код идентификаторов нот: " << endl;
    for (const auto& pair : huffmanCode) {
        cout << pair.first << " : " << pair.second << endl;
    }
    cout << endl;

    cout << "Закодированный текст по Хаффману: " << encodedText << endl;

    writeHuffmanCodesToFile(huffmanCode, encodedText, freq);
    cout << endl;
    gain(text, huffmanCode, freq);
}

int main() {
    setlocale(0, "");
    ifstream file("/Users/ilya/DownloadsТипы/ и структуры данных 2 курс, 1
   семестр/laba_7/input.txt");  // Замените путь на путь к вашему файлу
```

```
    string text = "";

    if (file.is_open()) {
        string line;
        while (getline(file, line)) {
            text += line + '\n';
        }
        file.close();
    }
    else {
        cout << "Невозможно открыть файл";
        return 0;
    }

    cout << "Прочитанный текст: " << endl << text << endl;
    buildHuffmanTree(text);

    return 0;
}
```

Примеры работы.

Далее на рисунках 1-10 представлен результат работы программы. Размер входного файла с каждым тестом уменьшается на 20 %. Входной файл — уникальный строковый идентификатор ноты в формате инструмент-темп-скорость.

1. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 1.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65

Частоты идентификаторов нот:
piano-C4-80 : 1
piano-E5-75 : 1
violin-A3-60 : 1
violin-G4-65 : 1

Код идентификаторов нот:
piano-C4-80 : 00
piano-E5-75 : 01
violin-A3-60 : 11
violin-G4-65 : 10

Закодированный текст по Хаффману: 00110110
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 50 байт / 62байт = 0.806452 раз
```

Рис. 1 – Пример работы 1

2. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 2.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65
piano-D4-90

Частоты идентификаторов нот:
piano-C4-80 : 1
piano-D4-90 : 1
piano-E5-75 : 1
violin-A3-60 : 1
violin-G4-65 : 1

Код идентификаторов нот:
piano-C4-80 : 110
piano-D4-90 : 111
piano-E5-75 : 10
violin-A3-60 : 00
violin-G4-65 : 01

Закодированный текст по Хаффману: 110001001111
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 62 байт / 81байт = 0.765432 раз
```

Рис. 2 – Пример работы 2

3. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 3.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65
piano-D4-90
piano-E5-75


Частоты идентификаторов нот:
 : 1
piano-C4-80 : 1
piano-D4-90 : 1
piano-E5-75 : 2
violin-A3-60 : 1
violin-G4-65 : 1

Код идентификаторов нот:
 : 110
piano-C4-80 : 111
piano-D4-90 : 00
piano-E5-75 : 01
violin-A3-60 : 100
violin-G4-65 : 101

Закодированный текст по Хаффману: 111100011010001110
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 75 байт / 91байт = 0.824176 раз
```

Рис. 3 – Пример работы 3

4. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 4.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65
piano-D4-90
piano-E5-75
piano-E5-75


Частоты идентификаторов нот:
 : 1
piano-C4-80 : 1
piano-D4-90 : 1
piano-E5-75 : 3
violin-A3-60 : 1
violin-G4-65 : 1

Код идентификаторов нот:
 : 010
piano-C4-80 : 011
piano-D4-90 : 00
piano-E5-75 : 11
violin-A3-60 : 100
violin-G4-65 : 101

Закодированный текст по Хаффману: 01110011101001111010
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 87 байт / 93байт = 0.935484 раз
```

Рис. 4 – Пример работы 4

5. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 5.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65
piano-D4-90
piano-E5-75
piano-E5-75
piano-E5-75


Частоты идентификаторов нот:
 : 1
piano-C4-80 : 1
piano-D4-90 : 1
piano-E5-75 : 4
violin-A3-60 : 1
violin-G4-65 : 1

Код идентификаторов нот:
 : 1110
piano-C4-80 : 1111
piano-D4-90 : 110
piano-E5-75 : 0
violin-A3-60 : 100
violin-G4-65 : 101

Закодированный текст по Хаффману: 111110001011100001110
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 99 байт / 96байт = 1.03125 раз
```

Рис. 5 – Пример работы 5

6. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 6.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65
piano-D4-90
piano-E5-75
piano-E5-75
piano-E5-75
piano-D4-90
piano-D4-90

Частоты идентификаторов нот:
piano-C4-80 : 1
piano-D4-90 : 3
piano-E5-75 : 4
violin-A3-60 : 1
violin-G4-65 : 1

Код идентификаторов нот:
piano-C4-80 : 1110
piano-D4-90 : 10
piano-E5-75 : 0
violin-A3-60 : 1111
violin-G4-65 : 110

Закодированный текст по Хаффману: 111011110110100001010
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 122 байт / 92байт = 1.32609 раз
```

Рис. 6 – Пример работы 6

7. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 7.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65
piano-D4-90
piano-E5-75
piano-E5-75
piano-E5-75
piano-D4-90
piano-D4-90
violin-A3-60
violin-G4-65

Частоты идентификаторов нот:
piano-C4-80 : 1
piano-D4-90 : 3
piano-E5-75 : 4
violin-A3-60 : 2
violin-G4-65 : 2

Код идентификаторов нот:
piano-C4-80 : 100
piano-D4-90 : 01
piano-E5-75 : 11
violin-A3-60 : 101
violin-G4-65 : 00

Закодированный текст по Хаффману: 100101110001111111010110100
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 148 байт / 96байт = 1.54167 раз
```

Рис. 7 – Пример работы 7

8. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 8.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65
piano-D4-90
piano-E5-75
piano-E5-75
piano-E5-75
piano-D4-90
piano-D4-90
violin-A3-60
violin-G4-65
piano-E5-75
piano-C4-80

Частоты идентификаторов нот:
piano-C4-80 : 2
piano-D4-90 : 3
piano-E5-75 : 5
violin-A3-60 : 2
violin-G4-65 : 2

Код идентификаторов нот:
piano-C4-80 : 100
piano-D4-90 : 01
piano-E5-75 : 11
violin-A3-60 : 101
violin-G4-65 : 00

Закодированный текст по Хаффману: 10010111000111111101011010011100
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 172 байт / 101байт = 1.70297 раз
```

Рис. 8 – Пример работы 8

9. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 9.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65
piano-D4-90
piano-E5-75
piano-E5-75
piano-E5-75
piano-D4-90
piano-D4-90
violin-A3-60
violin-G4-65
piano-E5-75
piano-C4-80
sexafon-E1-78
guitar-W6-17
piano-D4-90

Частоты идентификаторов нот:
guitar-W6-17 : 1
piano-C4-80 : 2
piano-D4-90 : 4
piano-E5-75 : 5
sexafon-E1-78 : 1
violin-A3-60 : 2
violin-G4-65 : 2

Код идентификаторов нот:
guitar-W6-17 : 0100
piano-C4-80 : 100
piano-D4-90 : 00
piano-E5-75 : 11
sexafon-E1-78 : 0101
violin-A3-60 : 011
violin-G4-65 : 101

Закодированный текст по Хаффману:
10001111101001111110000011101111000101010000
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 211 байт / 147байт = 1.43537 раз
```

Рис. 9 – Пример работы 9

10. Входной файл — уникальный строковый идентификатор ноты. Результат приведён на рис. 10.

```
Прочитанный текст:
piano-C4-80
violin-A3-60
piano-E5-75
violin-G4-65
piano-D4-90
piano-E5-75
piano-E5-75
piano-E5-75
piano-D4-90
piano-D4-90
violin-A3-60
violin-G4-65
piano-E5-75
piano-C4-80
sexafon-E1-78
guitar-W6-17
piano-D4-90
piano-C4-80
sexafon-E1-78
guitar-W6-17
piano-D4-90

Частоты идентификаторов нот:
guitar-W6-17 : 2
piano-C4-80 : 3
piano-D4-90 : 5
piano-E5-75 : 5
sexafon-E1-78 : 2
violin-A3-60 : 2
violin-G4-65 : 2

Код идентификаторов нот:
guitar-W6-17 : 000
piano-C4-80 : 110
piano-D4-90 : 01
piano-E5-75 : 10
sexafon-E1-78 : 001
violin-A3-60 : 1111
violin-G4-65 : 1110

Закодированный текст по Хаффману:
1101111101110011010100101111111101011000100001111000100001
Данные успешно записаны в бинарный файл 'huffman_codes.bin'

Сжатие в 262 байт / 160байт = 1.6375 раз
```

Рис. 10 – Пример работы 10

4 Исследовательская часть

Анализируя выполненные тесты, можно отметить, что при увеличении объема данных алгоритм Хаффмана становится более эффективным. Это происходит благодаря созданию оптимальных кодов для представления данных: наиболее часто встречающиеся символы кодируются более короткими кодами, что позволяет значительно сократить общий размер данных.

Однако нотная запись очень разнообразна, а значит есть вероятность того, что большое количество нот будет встречаться один раз, то есть сжатие по Хаффману будет невыгодно из-за того, что при данном типе сжатия хранится не только сжатая запись, но и данные для декодирования.

Визуализация исследования представлена на рис. 11.

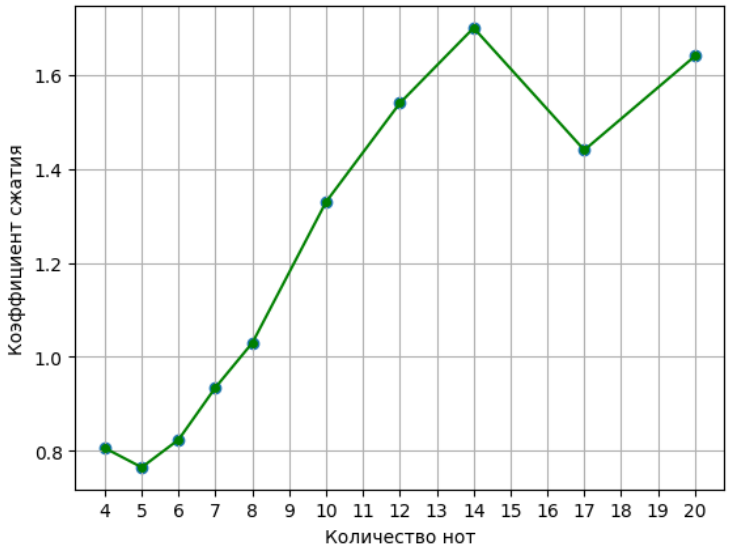

Рис. 11 – Зависимость коэффициента сжатия от размера входых данных

Таким образом, при увеличении объема данных сжатие по Хаффману обычно становится эффективнее, но для данных формата нотной записи он может быть неэффективен как на больших, так и на маленьких типах данных.

Заключение

Цель достигнута: реализованы алгоритмы сжатия по Хаффману. В результате были выполнены все задачи:

1. Описано сжатие по Хаффману.
2. Реализовано сжатие файла.
3. Реализована возможность считывать данные из файла.
4. Реализована возможность записать ответ в бинарный файл.
5. Проведено исследование для 10 файлов с одинаковым характером наполнения (с увеличением размера на каждом шаге не менее, чем на 20 %) с оценкой эффекта от сжатия, результаты визуализированы в виде графика.

Список литературы

1. Котиева, Х. М. Сжатие данных без потерь. Использование алгоритма Хаффмана / Х. М. Котиева. — Текст : непосредственный // Молодой ученый. — 2020. — № 35 (325 с.).