МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №7

Тема: Алгоритмы кодирования и сжатия данных

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент Хан А.А.

группа ИКБО-04-20

Москва 2021

Оглавление

Тема

Кодирование и сжатие данных методами без потерь.

Цель

Получение практических навыков и знаний по выполнению сжатия данных рассматриваемыми методами.

Персональный вариант

Вариант №9

Отчет по заданию №1

Постановка задачи

Применение алгоритма группового сжатия текста.

Сжать текст, используя метод RLE (run length ecoding/кодирование длин серий/групповое кодирование).

1) Описать процесс сжатия алгоритмом RLE.

2) Придумать текст, в котором есть длинные (в разумных пределах) серии из повторяющихся символов. Выполнить сжатие текста. Рассчитать коэффициент сжатия.

3) Придумать текст, в котором много неповторяющихся символов и между ними могут быть серии. Выполнить групповое сжатие, показать коэффициент сжатия. Применить алгоритм разделения текста при групповом кодировании, позволяющий повысить эффективность сжатия этого текста. Рассчитать коэффициент сжатия после применения алгоритма.

4) В отчете представьте ответы на вопросы пунктов задания с 1 по 3.

Выполнение работы

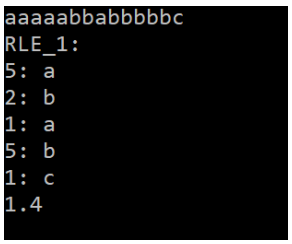
RLE-это алгоритм сжатия данных, заменяющий повторяющиеся символы (серии) на один символ и число его повторов.

Пример: WWWWWWWWWBBBWWWWWWWBWWWWWWWWWWWWWW -> 9W3B7W1B14W

Тестирование

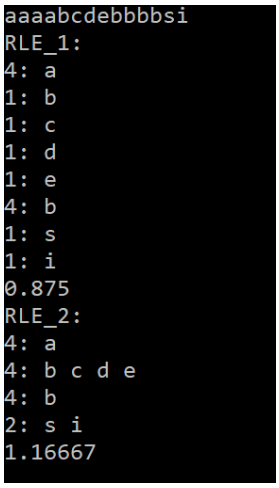
RLE без группового сжатия.

Тест	Результат	Коэффициент сжатия
aaaaabbabbbbbc	5a2b1a5b1c	1.4



RLE с групповым сжатием.

Тест	Тип RLE	Результат	Коэффициент сжатия
aaaabcdebbbbsi	Без группового	4a1b1c1d1e4b1s1i	0.875
aaaabcdebbbbsi	С групповым	4a4bcde4b2si	1.167



Отчет по заданию №2

Постановка задачи

Исследование алгоритмов сжатия Лемпеля –Зива (LZ77), LZ78 на примерах.
Тексты для сжатия по вариантам в таб1. Столбцы 2 и 3.
1) Выполнить каждую задачу варианта, представив алгоритм решения в виде таблицы и указав результат сжатия. Примеры оформления решения представлены в Приложении1 этого документа.
2) Описать процесс восстановления сжатого текста.
3) Сформировать отчет, включив задание, вариант задания, результаты выполнения задания варианта.

Выполнение работы

LZ77 – алгоритм сжатия, основанный на замене повторений на ссылки на позиции в тексте, где такие подстроки уже встречались.

Информацию о повторении можно закодировать парой чисел — смещением назад от текущей позиции (offset) и длиной совпадающей подстроки (length).

Алгоритм LZ77 кодирует ссылки блоками из трёх элементов - ⟨offset, length, next⟩. Параметр next означает первый символ после найденного совпадающего фрагмента. Если LZ77 не удалось найти совпадение, то считается, что offset = length = 0.

Для декодирования LZ77 необходимо пройти по уже раскодированной строке назад, вывести необходимую последовательность, затем следующий символ.

Тестирование

Тест: 000100101100100010001

Буфер	Совпадение	Сдвиг	Длина	Следующий символ
-	-	0	0	0
0	00	1	2	1
0001	0010	3	4	1
00101	10	3	2	0

01100	100	3	3	0
01000	10001	4	5	-

```
0001001011001000 1000
encode LZ77:
0 0 0
1 2 1
3 4 1
3 2 0
3 3 0
4 4
decode LZ77:
0001001011001000 1000
```

LZ78 генерирует временный словарь во время кодирования и декодирования.

Изначально словарь пуст, а алгоритм пытается закодировать первый символ. На каждой итерации мы пытаемся увеличить кодируемый префикс, пока такой префикс есть в словаре. Кодовые слова такого алгоритма будут состоять из двух частей — номера в словаре самого длинного найденного префикса (pos) и символа, который идет за этим префиксом (next). При этом после кодирования такой пары префикс с приписанным символом добавляется в словарь, а алгоритм продолжает кодирование со следующего символа.

Декодирование происходит аналогично кодированию, на основе декодируемой информации строим словарь и берем из него значения.

Тестирование

Тест: kloklonkolonklonkl

Словарь	Осталось обработать	pos	next
-	kloklonkolonklonkl	0	k
k	loklonkolonklonkl	0	l
k, l	oklonkolonklonkl	0	o
k, l, o	klonkolonklonkl	1	l
k, l, o, kl	onkolonklonkl	3	n
k, l, o, kl, on	kolonklonkl	1	o
k, l, o, kl, on, ko	lonklonkl	2	o

k, l, o, kl, on, ko, lo	nklonkl	0	n
k, l, o, kl, on, ko, lo, n	klonkl	4	o
k, l, o, kl, on, ko, lo, n, klo	nkl	8	k
k, l, o, kl, on, ko, lo, n, klo, nk	l	0	l

```
kloklonkolonklonkl
encode LZ78:
0 k
0 l
0 o
1 l
3 n
1 o
2 o
0 n
4 o
8 k
0 l
decode LZ78:
kloklonkolonklonkl
```

Отчет по заданию №3

Постановка задачи

Разработать программы (или только алгоритмы на псевлокоде или словесно) сжатия и восстановления текста методами Шеннона-Фано и Хаффмана.

1. Сформировать отчет по разработке каждого алгоритма в соответствии с требованиями.

1.1. По методу Шеннона-Фано. Данные для выполнения задания: таб.1, ваш вариант, текст столбца 1.

1) Привести постановку задачи, описать алгоритм формирования префиксного дерева и алгоритм кодирования, декодирования.

2) Представить таблицу формирования кода.

3) Изобразить префиксное дерево.

4) Рассчитать коэффициент сжатия.

1.2. По методу Хаффмана Данные для выполнения задания: ваша фамилия имя отчество.

1) Привести постановку задачи, описать алгоритм формирования префиксного дерева и алгоритм кодирования, декодирования.

2) Построить таблицу частот встречаемости символов в исходной строке для чего сформировать алфавит исходной строки и посчитать количество вхождений (частот) символов и их вероятности появления.

3) Изобразить префиксное дерево Хаффмана.

4) Упорядочить построенное дерево слева-направо (при необходимости) и изобразить его.

5) 2.8 Провести кодирование исходной строки по аналогии с примером:

6) Рассчитать коэффициенты сжатия относительно кодировки ASCII и относительно равномерного кода.

7) Рассчитать среднюю длину полученного кода и его дисперсию.

8) По результатам выполненной работы сделать выводы и сформировать отчет. Отобразить результаты выполнения всех требований, предъявленных в задании и оформить разработку программы: постановка, подход к решению, код, результаты тестирования.

1.3. Реализовать и отладить программу. Применить алгоритм Хаффмана для архивации данных текстового файла. Выполнить практическую оценку сложности алгоритма Хаффмана. Провести архивацию этого же файла любым архиватором. Сравнить коэффициенты сжатия разработанного алгоритма и архиватора.

Часть 1

Метод Шеннона-Фано — метод сжатия, который использует коды переменной длины: часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся — кодом большей длины. Коды Шеннона-Фано — префиксные, что позволяет однозначно декодировать любую последовательность кодовых слов.

Алгоритм:

- Символы первичного алфавита выписывают по убыванию вероятностей.
- Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.
- В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».
- Полученные части рекурсивно делятся, и их частям назначаются соответствующие двоичные цифры в префиксном коде.

Построение дерева:

Код Шеннона-Фано строится с помощью дерева. Построение этого дерева начинается от корня. Всё множество кодируемых элементов соответствует корню дерева (вершине первого уровня). Оно разбивается на два подмножества с приблизительно одинаковыми суммарными вероятностями. Эти подмножества соответствуют двум вершинам второго уровня, которые соединяются с корнем. Далее каждое из этих подмножеств разбивается на два подмножества с примерно одинаковыми суммарными вероятностями. Им соответствуют вершины третьего уровня. Если подмножество содержит единственный элемент, то ему соответствует концевая вершина кодового дерева; такое подмножество разбиению не подлежит. Подобным образом поступаем до тех пор, пока не получим все концевые вершины.

Кодирование:

Для получения кода символа необходимо пройти от корня дерева до листа, содержащего нужны символ. Начиная с кода, равного пустой строке, приписываем к нему 0 при переходе в «левое» поддерево (1 – в «правое»). Таким образом, за один обход дерева получим коды всехсимволов.

Декодирование:

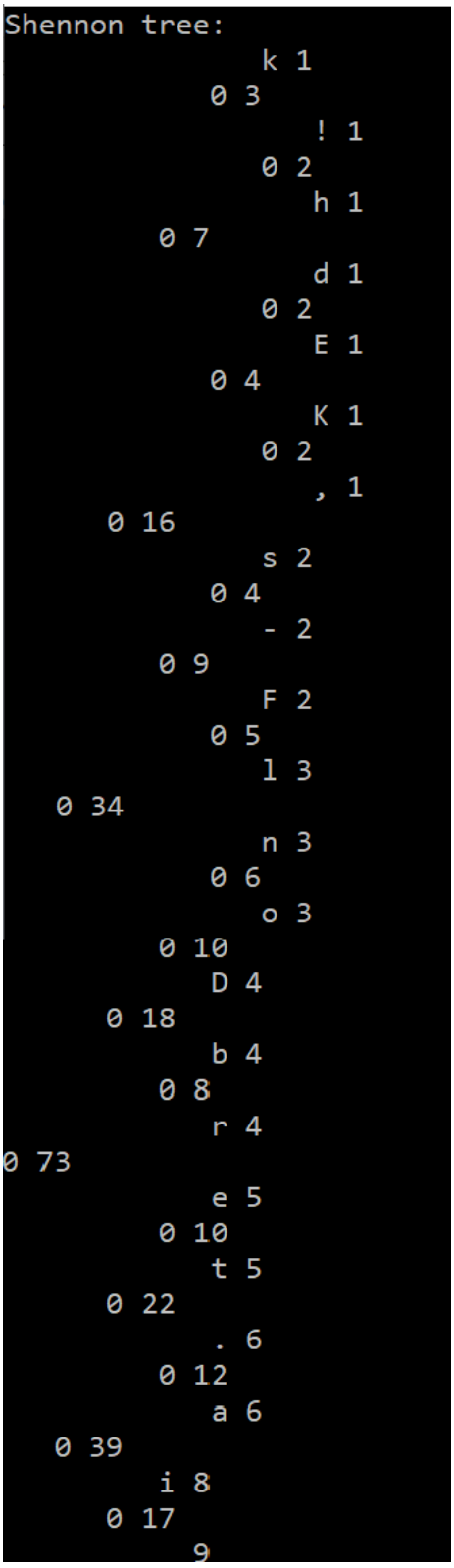
Декодирование происходит так же, как и в случае любого префиксного кода с использованием построенного соответствия двоичного кода символу: необходимо выбрать максимальный префикс строки, совпадающий с кодом какого-то символа, дописать к строке ответа соответствующий символ и повторять эти действия, пока закодированная строка не кончится.

Тестирование

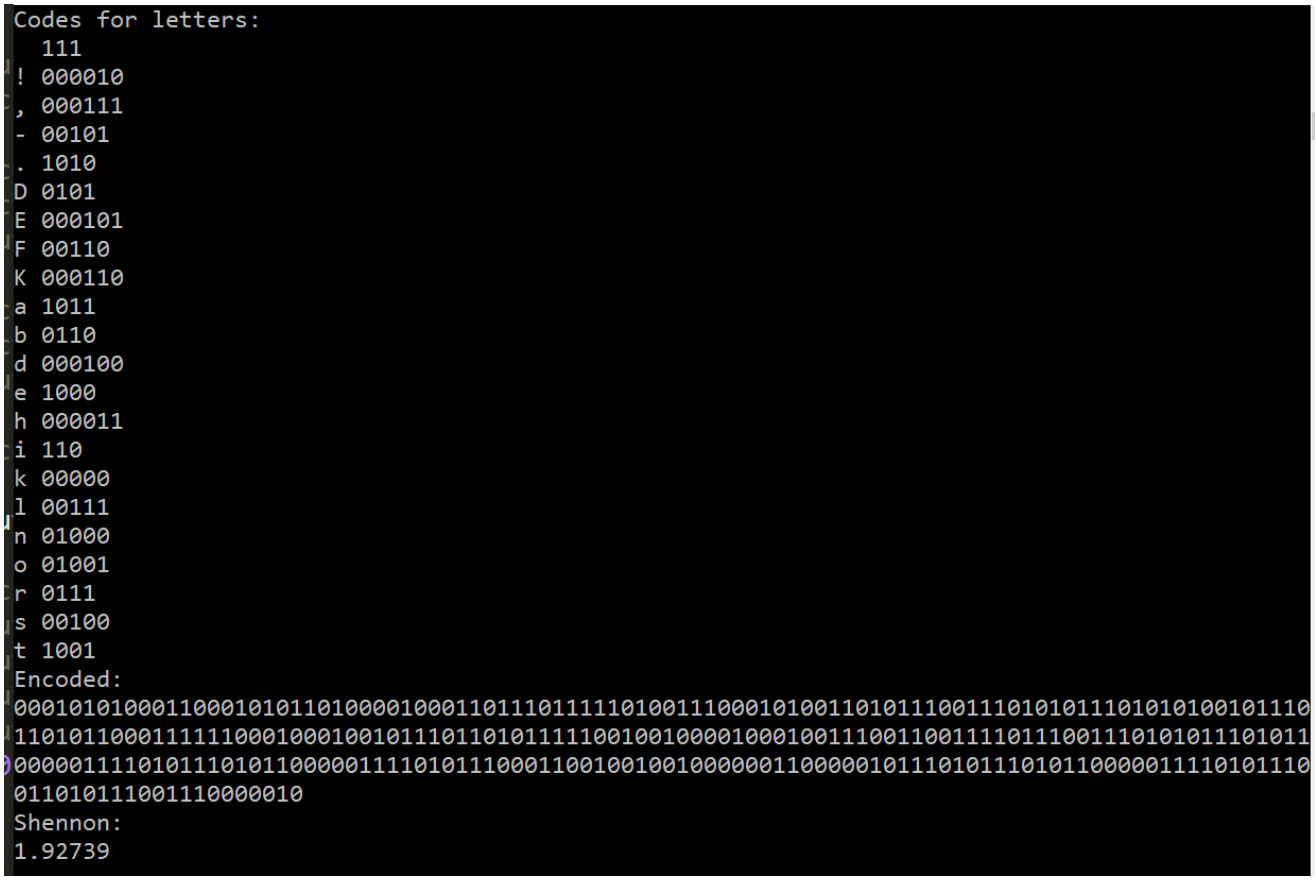
Тест: «Eni-beni riti-Fati. Dorba, dorba sentibrati. Del. Del. Koshka. Del. Fati!»

<table>
<tr><td>k</td><td>1</td><td rowspan="16">0</td><td rowspan="10">00</td><td rowspan="6">000</td><td rowspan="2">0000</td><td>00000</td><td></td></tr>
<tr><td>!</td><td>1</td><td rowspan="2">00001</td><td>000010</td></tr>
<tr><td>h</td><td>1</td><td>000011</td></tr>
<tr><td>d</td><td>1</td><td rowspan="4">0001</td><td rowspan="2">00010</td><td>000100</td></tr>
<tr><td>E</td><td>1</td><td>000101</td></tr>
<tr><td>K</td><td>1</td><td rowspan="2">00011</td><td>000110</td></tr>
<tr><td>,</td><td>1</td><td>000111</td></tr>
<tr><td>s</td><td>2</td><td rowspan="4">001</td><td rowspan="2">0010</td><td>00100</td></tr>
<tr><td>-</td><td>2</td><td>00101</td></tr>
<tr><td>F</td><td>2</td><td rowspan="2">0011</td><td>00110</td></tr>
<tr><td>l</td><td>3</td><td>00111</td></tr>
<tr><td>n</td><td>3</td><td rowspan="6">01</td><td rowspan="4">010</td><td rowspan="2">0100</td><td>01000</td></tr>
<tr><td>o</td><td>3</td><td>01001</td></tr>
<tr><td>D</td><td>4</td><td>0101</td></tr>
<tr><td>b</td><td>4</td><td rowspan="2">011</td><td>0110</td></tr>
<tr><td>r</td><td>4</td><td>0111</td></tr>
<tr><td>e</td><td>5</td><td rowspan="5">1</td><td rowspan="4">10</td><td rowspan="2">100</td><td>1000</td></tr>
<tr><td>t</td><td>5</td><td>1001</td></tr>
<tr><td>.</td><td>6</td><td rowspan="2">101</td><td>1010</td></tr>
<tr><td>a</td><td>6</td><td>1011</td></tr>
<tr><td>i</td><td>8</td><td rowspan="2">11</td><td>110</td></tr>
<tr><td></td><td>9</td><td>111</td></tr>
</table>

Построенное дерево (вершина: символ (или 0, если вершина не лист) и частота встречаемости в тексте):



Коды символов и закодированное сообщение:



Коэффициент сжатия: 1.927

Часть 2

Метод Хаффмана – жадный алгоритм оптимального префиксного кодирования алфавита сминимальной избыточностью.

Алгоритм:

- Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
- Выбираются два свободных узла дерева с наименьшими весами.
- Создается их родитель с весом, равным их суммарному весу.
- Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
- Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.
- Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

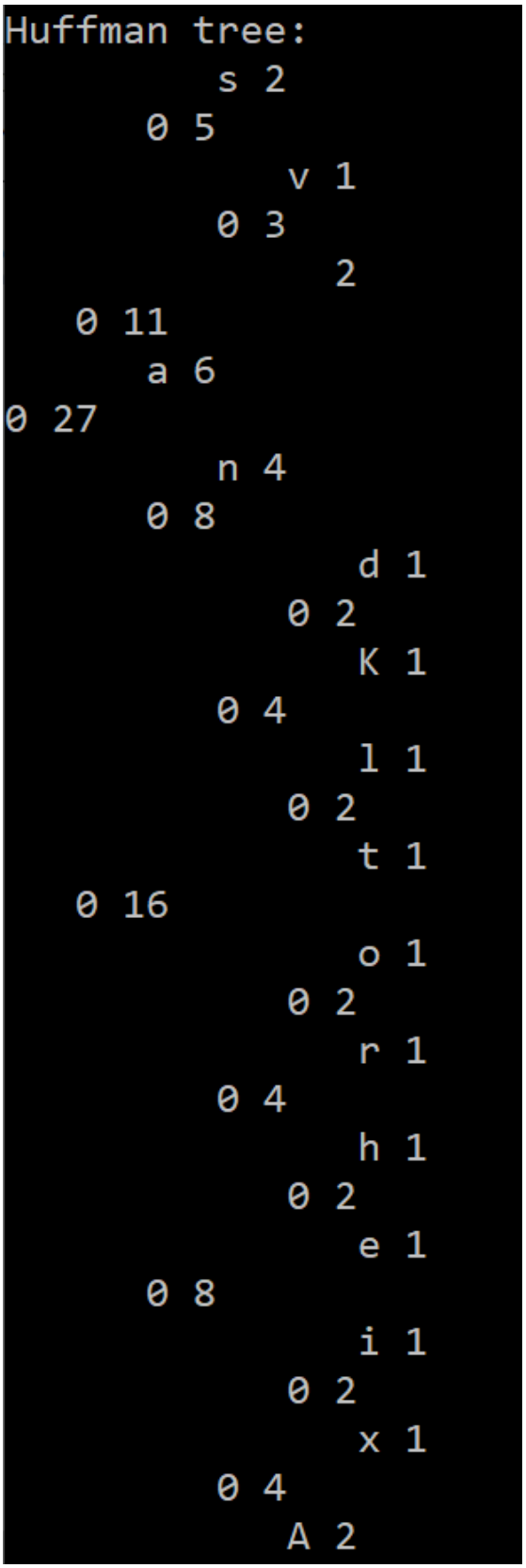
Кодирование и декодирование проводится с использованием дерева так же, как в методе Шеннона-Фано.

Тестирование

Тест: Khan Anastasia Alexandrovna

Символ	Частота
i	1
h	1
x	1
l	1
e	1
K	1
d	1
v	1
t	1
r	1
o	1
s	2
A	2
	2
n	4
a	6

Префиксное дерево Хаффмана (вершина: символ (или 0, если вершина не лист) и частота встречаемости в тексте):



Коды символов:

```
Codes for letters:
  0011
A 1111
K 10101
a 01
d 10100
e 11011
h 11010
i 11100
l 10110
n 100
o 11000
r 11001
s 000
t 10111
v 0010
x 11101
Encoded:
10101110100110000111111100010001011101000111000100111111101101011110101100101001100111000001010001
Huffman:
2.16
```

Коэффициент сжатия: 2.16

Часть 3

Метод Хаффмана для кодирования файла.

Оценка сложности

Пусть N – количество символов во входном файле.

Подсчет количества символов – O(N). Добавление пар (символ, количество) в очередь с приоритетами O(Nlog(N)). Создание дерева – O(Nlog(N)). Получение кодов символов – проход по всему дереву – O(Nlog(N)). Итоговая сложность: – O(Nlog(N)).

Тестирование

Входной файл	Сжатый по Хаффману	Сжатый win.rar
2,73 КБ	460 байт	172 байт

Коэффициенты сжатия: 6.08 по Хаффману, 16.25 с помощью win.rar

Код приложения

В файле huf.cpp – сжатие текстового файла in.txt по Хаффману (вывод в бинарный файл out.bin). В файле enc.cpp – все остальные части задания.

huf.cpp

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

struct node_H {
    char c;
    int p;
    node_H * left;
    node_H * right;

    node_H(char _c = 0, int _p = 0) {
        c = _c;
        p = _p;
        left = NULL;
        right = NULL;
    }

    bool operator<(const node_H *a) {
        return p > a->p;
    }

};

class Compare_H {
public:
    bool operator() (node_H * a, node_H * b) {
        return a->p > b->p;
    }
};

bool comp_node(const node_H * a, const node_H *b) {
    return a->p < b->p;
}

bool comp_H(std::pair <int,int> p1, std::pair <int,int> p2) {
```

```
    if (p1.second == 0) {
        return 0;
    }
    if (p2.second == 0) {
        return 1;
    }
    return p1.second <= p2.second;
}

void print_node(node_H *v, int h = 0) {
    if (v == NULL) {
        return;
    }
    print_node(v->left, h + 1);

    for (int i = 0; i < h; ++i) {
        std::cout << "    ";
    }
    if (v->c == 0) {
        std::cout << "0 ";
    } else {
        std::cout << (char)v->c << " ";
    }
    std::cout << v->p << "\n";

    print_node(v->right, h + 1);

}

void get_code_H(node_H * v, std::string cur, std::map <char, std::string> &d) {
    if (v == NULL) {
        return;
    }
    if (v->c != 0) {
        d[v->c] = cur;
        return;
    }
    get_code_H(v->left, cur + "0", d);
    get_code_H(v->right, cur + "1", d);
}

unsigned int from_str(int L, int R, std::string &s) {
    unsigned int ans = 0;
    for (int i = L; i <= R; ++i) {
        ans <<= 1;
        ans += s[i] - '0';
    }
```

```
    return ans;
}

void encode_H(std::vector <std::string> &s, std::vector <std::pair <int,int>> &cnt,
std::ofstream &f) {
    std::sort(cnt.begin(), cnt.end(), comp_H);
    std::priority_queue <node_H *, std::vector<node_H *>, Compare_H> q;

    for (auto i : cnt) {
        if (i.second == 0) {
            break;
        }
        q.push(new node_H((char)i.first, i.second));
    }

    while (q.size() > 1) {
        node_H *v1 = q.top();
        q.pop();
        node_H *v2 = q.top();
        q.pop();

        node_H *v = new node_H(0, v1->p + v2->p);
        v->left = v1;
        v->right = v2;

        q.push(v);
    }

    node_H *root = q.top();

    std::map <char, std::string> d;
    get_code_H(root, "", d);

    for (auto str : s) {
        std::string s_ans = "";
        for (auto c : str) {
            s_ans += d[c];
        }
        for (int i = 0; i < s_ans.size(); i += 32) {
            unsigned int tmp = from_str(i, std::min(i + 32, (int)s_ans.size()) - 1, s_ans);
            f.write((char *)&tmp, sizeof(int));
        }
    }
}

int main() {
    std::ofstream f("out.bin", std::ios::out | std::ios::binary);
```

```
    std::ifstream in("in.txt");
    char buf[1024];

    std::vector <std::string> s;

    std::vector <std::pair <int,int>> cnt;
    for (int c = 0; c < 256; ++c) {
        cnt.push_back({c, 0});
    }

    while (1) {
        in.read(buf, 1024);
        int sz = in.gcount();

        if (sz == 0) {
            break;
        }

        std::string new_s = "";

        for (int i = 0; i < sz; ++i) {
            new_s += buf[i];
            cnt[buf[i]].second += 1;
        }

        s.push_back(new_s);
        std::cout << sz << "\n";
    }

    encode_H(s, cnt, f);

    f.close();
    in.close();


    return 0;
}
```

enc.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <algorithm>
```

```
#include <queue>

double RLE_1(std::string &s) {
    std::vector <unsigned char> ans;
    size_t i = 0;
    while (i < s.size()) {
        size_t j = i;
        while (j < s.size() && s[j] == s[i]) {
            ++j;
            if (j - i == 255) {
                break;
            }
        }
        unsigned char cnt = j - i;
        ans.push_back(cnt);
        ans.push_back(s[i]);

        i = j;
    }

    for (int i = 0; i < ans.size(); i += 2) {
        std::cout << (unsigned int)ans[i] << ": " << ans[i + 1] << "\n";
    }

    return (double)s.size() / (double)ans.size();
}


double RLE_2(std::string &s) {
    std::vector <unsigned char> ans;
    size_t i = 0;
    while (i < s.size()) {
        size_t j = i;
        while (j < s.size() && s[j] == s[i]) {
            ++j;
            if (j - i == 127) {
                break;
            }
        }
        unsigned char cnt = j - i;

        if (cnt > 1) {
            unsigned char res = (1 << 7) | cnt;
            ans.push_back(res);
            ans.push_back(s[i]);
        }
        else {
```

```
            while (j < s.size() && s[j] != s[j - 1]) {
                ++j;
                if (j - i == 127) {
                    break;
                }
            }
            if (j > 1 && s[j] == s[j - 1]) {
                j -= 1;
            }

            cnt = j - i;
            ans.push_back(cnt);

            for (int u = i; u < j; ++u) {
                ans.push_back(s[u]);
            }
        }

        i = j;
    }
    i = 0;
    while (i < ans.size()) {
        unsigned int cnt = ans[i] & 127;
        if (ans[i] & (1 << 7)) {
            std::cout << cnt << ": " << ans[i + 1] << "\n";
            i += 2;
        } else {
            std::cout << cnt << ": ";
            for (int j = i + 1; j < i + cnt + 1; ++j) {
                std::cout << ans[j] << " ";
            }
            std::cout << "\n";
            i += cnt + 1;
        }
    }

    return (double)s.size() / (double)ans.size();
}

struct node_77 {
    int offset;
    int length;
    char c;

    node_77(int _o, int _l, char _c) {
        offset = _o;
        length = _l;
```

```
        c = _c;
    }
};

std::vector <node_77> encode_LZ77(std::string &s) {
    int i = 0, L = 0, R = -1, SZ = 5;
    std::vector <node_77> ans;

    while (i < s.size()) {
        if (R < L) {
            ans.push_back(node_77(0, 0, s[i]));
            R += 1;
            L = std::max(L, R - SZ + 1);
            i += 1;
        } else {
            int ind = i, max_len = 0;
            for (int j = L; j <= R; ++j) {
                int u = 0;
                while (i + u < s.size() && s[j + u] == s[i + u]) {
                    u += 1;
                }
                if (u > max_len) {
                    ind = j;
                    max_len = u;
                }
            }
            ans.push_back(node_77(i - ind, max_len, s[i + max_len]));
            i += max_len + 1;

            R += max_len + 1;
            L = std::max(L, R - SZ + 1);
        }
    }

    for (auto j : ans) {
        std::cout << j.offset << " " << j.length << " " << j.c << "\n";
    }
    return ans;
}

std::string decode_LZ77(std::vector <node_77> &ans) {
    std::string s = "";
    for (auto elem : ans) {
        int start = s.size() - elem.offset;
        for (int i = 0; i < elem.length; ++i) {
            s += s[start + i];
        }
```

```
        s += elem.c;
    }
    return s;
}

struct node_78 {
    int ind;
    char c;
    node_78(int _i, char _c) {
        ind = _i;
        c = _c;
    }
};

std::vector <node_78> encode_LZ78(std::string &s) {
    std::vector <node_78> ans;
    std::string buf = "";
    std::map <std::string, int> d;
    d[""] = 0;

    for (int i = 0; i < s.size(); ++i) {
        if (d.find(buf + s[i]) == d.end()) {
            std::cout << "dict\n";
            for (auto u : d) {
                std::cout << u.first << " " << u.second << "\n";
            }
            std::cout << "\n";
            std::cout << "push " << d[buf] << " " << s[i] << "\n\n";
            ans.push_back(node_78(d[buf], s[i]));
            d[buf + s[i]] = d.size() - 1;
            buf = "";
        } else {
            buf += s[i];
        }
    }

    if (buf.size()) {
        char c = buf[buf.size() - 1];
        buf.pop_back();
        ans.push_back(node_78(d[buf], c));
    }

    for (auto i : ans) {
        std::cout << i.ind << " " << i.c << "\n";
    }
    return ans;
}
```

```
std::string decode_LZ78(std::vector <node_78> &ans) {
    std::string s = "";
    std::vector <std::string> d;
    d.push_back("");

    for (auto elem : ans) {
        std::string cur = d[elem.ind] + elem.c;
        s += cur;
        d.push_back(cur);
    }
    return s;
}

struct node_H {
    char c;
    int p;
    node_H * left;
    node_H * right;

    node_H(char _c = 0, int _p = 0) {
        c = _c;
        p = _p;
        left = NULL;
        right = NULL;
    }

    bool operator<(const node_H *a) {
        return p > a->p;
    }

};

class Compare_H {
public:
    bool operator() (node_H * a, node_H * b) {
        return a->p > b->p;
    }
};

bool comp_node(const node_H * a, const node_H *b) {
    return a->p < b->p;
}

bool comp_H(std::pair <int,int> p1, std::pair <int,int> p2) {
    if (p1.second == 0) {
        return 0;
```

```
    }
    if (p2.second == 0) {
        return 1;
    }
    return p1.second <= p2.second;
}

void print_node(node_H *v, int h = 0) {
    if (v == NULL) {
        return;
    }
    print_node(v->left, h + 1);

    for (int i = 0; i < h; ++i) {
        std::cout << "    ";
    }
    if (v->c == 0) {
        std::cout << "0 ";
    } else {
        std::cout << (char)v->c << " ";
    }
    std::cout << v->p << "\n";

    print_node(v->right, h + 1);

}

void get_code_H(node_H * v, std::string cur, std::map <char, std::string> &d) {
    if (v == NULL) {
        return;
    }
    if (v->c != 0) {
        d[v->c] = cur;
        return;
    }
    get_code_H(v->left, cur + "0", d);
    get_code_H(v->right, cur + "1", d);
}

double encode_H(std::string &s) {
    std::vector <std::pair <int,int>> cnt;
    for (int c = 0; c < 256; ++c) {
        cnt.push_back({c, 0});
    }
    for (auto c : s) {
        cnt[c].second += 1;
    }
```

```
    std::sort(cnt.begin(), cnt.end(), comp_H);
    std::priority_queue <node_H *, std::vector<node_H *>, Compare_H> q;
    std::cout << "Frequences:\n";

    for (auto i : cnt) {
        if (i.second == 0) {
            break;
        }
        std::cout << (char)i.first << " " << i.second << "\n";
        q.push(new node_H((char)i.first, i.second));
    }

    while (q.size() > 1) {
        node_H *v1 = q.top();
        q.pop();
        node_H *v2 = q.top();
        q.pop();

        node_H *v = new node_H(0, v1->p + v2->p);
        v->left = v1;
        v->right = v2;

        q.push(v);
    }

    std::cout << "Huffman tree:\n";
    node_H *root = q.top();
    print_node(root);

    std::map <char, std::string> d;
    get_code_H(root, "", d);

    std::cout << "Codes for letters:\n";
    for (auto i : d) {
        std::cout << i.first << " " << i.second << "\n";
    }

    std::string s_ans = "";
    for (auto c : s) {
        s_ans += d[c];
    }

    std::cout << "Encoded:\n" << s_ans << "\n";

    return (double)(s.size() * 8) / (double)(s_ans.size());
}
```

```
node_H * div_node_SH(node_H *v, int L, int R, int sum, std::vector <node_H *> &a) {
    if (L == R) {
        return a[L];
    }
    std::cout << L << " " << R  << "\n";
    int ind = L, cur_sum = a[L]->p, div = std::abs(sum - 2 * cur_sum), sum_L = cur_sum;
    for (int i = L + 1; i < R; ++i) {
        cur_sum += a[i]->p;
        int cur_div = std::abs(sum - 2 * cur_sum);

        if (cur_div < div) {
            div = cur_div;
            ind = i;
            sum_L = cur_sum;
        }
    }

    v->left = new node_H(0, sum_L);
    v->right = new node_H(0, sum - sum_L);

    v->left = div_node_SH(v->left, L, ind, sum_L, a);
    v->right = div_node_SH(v->right, ind + 1, R, sum - sum_L, a);
    return v;
}

double encode_SH(std::string &s) {
    std::vector <std::pair <int,int>> cnt;
    for (int c = 0; c < 256; ++c) {
        cnt.push_back({c, 0});
    }
    for (auto c : s) {
        cnt[c].second += 1;
    }

    std::sort(cnt.begin(), cnt.end(), comp_H);
    std::cout << "Frequences:\n";
    std::vector <node_H *> a;

    for (auto i : cnt) {
        if (i.second == 0) {
            break;
        }
        std::cout << (char)i.first << " " << i.second << "\n";
        a.push_back(new node_H((char)i.first, i.second));
    }
```

```
    node_H * root = new node_H(0, s.size());
    root = div_node_SH(root, 0, a.size() - 1, s.size(), a);

    std::cout << "Shennon tree:\n";
    print_node(root);

    std::map <char, std::string> d;
    get_code_H(root, "", d);

    std::cout << "Codes for letters:\n";
    for (auto i : d) {
        std::cout << i.first << " " << i.second << "\n";
    }

    std::string s_ans = "";
    for (auto c : s) {
        s_ans += d[c];
    }

    std::cout << "Encoded:\n" << s_ans << "\n";

    return (double)(s.size() * 8) / (double)(s_ans.size());
}

int main() {
    std::string s = "";
    std::cin >> s;

    /*std::cout << "encode LZ77:\n";
    std::vector <node_77> ans = encode_LZ77(s);
    std::cout << "decode LZ77:\n" << decode_LZ77(ans) << "\n";*/

    /*std::cout << "encode LZ78:\n";
    std::vector <node_78> ans = encode_LZ78(s);
    std::cout << "decode LZ78:\n" << decode_LZ78(ans) << "\n";*/

    //std::cout << "Huffman:\n" << encode_H(s);

    //std::cout << "Shennon:\n" << encode_SH(s);

    /*std::cin >> s;
    std::cout << "RLE_1:\n";
    std::cout << RLE_1(s) << "\n";
    std::cout << "RLE_2:\n";
    std::cout << RLE_2(s) << "\n";*/
    return 0;
```

Выводы

В результате проделанной работы, я получила практические навыки и знания по выполнению сжатия данных рассматриваемыми методами. А также научилась разрабатывать и реализовывать задачи с применением методов RLE, LZ77, LZ78, Шеннона-Фано и Хаффмана.

Список информационных источников

1. Лекционный материал по структурам и алгоритмам обработки данных Сартакова М. В. (дата обращения 3.12.2021)

2. Дополнительный материал к практическим работам по структурам и алгоритмам обработки данных Сорокина А. В. (дата обращения 3.12.2021)