

Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №4
По дисциплине « **Алгоритмы и структуры данных** »
Тема: «**Построение кодов Хаффмана.**»

Выполнил:
Студент 2 курса
Группы ПО-11(2)
Сымоник И.А
Проверила:
Глущенко Т.А

Цель работы: изучить алгоритм построения кодов Хаффмана.

Ход работы

Задание 1. Дан текстовый файл размером не менее 5 *кбайт*. Построить для данного текста *коды Хаффмана* (см. материал лекции). Написать программу для кодировки и раскодировки заданного файла. Указать размеры файла до и после сжатия алгоритмом *Хаффмана*.

Исходный код:

```
#include <iostream>
#include <fstream>
#include <queue>

#include <iostream>
#include <string>
#include <queue>
#include <sstream>
#include <unordered_map>

class BitWriter
{
public:
    void Write(const std::string& str, const std::string& file)
    {
        std::fstream fl;
        fl.open("C:\\rep\\code_dec.txt", std::ios::out | std::ios::binary);

        int buf = 0;
        for(int i = 0 ; i< str.size(); i++)
        {
            if (bitCount == 32)
            {
                fl.write((char*)&buf, 4);
                buf = 0;
                bitCount = 0;
            }

            buf |= (static_cast<int>(str[i] - 48) << bitCount);
            bitCount++;
        }
    }

private:
    int bitCount = 0;
};
```

```

class BitReader
{
public:

    void Read(std::string& str, const std::string& file)
    {
        std::fstream fl;
        fl.open(file, std::ios::in | std::ios::binary);

        std::stringstream ss;
        fl >> ss.rdbuf();
        std::string buf = ss.str();

        int j = 0;
        for(size_t i = 0; i < buf.size(); i++)
        {
            j++;
            if (j > 7)
                j = 0;

            str += static_cast<char>((((int)buf[i] & (1 << j))>>j) + 48));

        }
    }
private:

    int bitCount = 0;

};

```

```

struct Node
{
    char ch;
    int freq;
    Node* left, * right;
};

```

```

Node* getNode(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;
}

```

```

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

```

```

void encode(Node* root, std::string str,
            std::unordered_map<char, std::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);
}

```

```

void decode(Node* root, long long& index, const std::string& str, std::string& decodeStr)
{
    if (root == nullptr) {
        return;
    }

    if (!root->left && !root->right)
    {
        decodeStr = root->ch;
        std::cout << decodeStr;
        return;
    }

    index++;

    if (str[index] == '0')
        decode(root->left, index, str, decodeStr);
    else
        decode(root->right, index, str, decodeStr);
}

```

```

void buildHuffmanTree(const std::string& text)
{
    std::unordered_map<char, int> freq;
    for (char ch : text) {
        freq[ch]++;
    }

    std::priority_queue<Node*, std::vector<Node*>, comp> 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 sum = left->freq + right->freq;
        pq.push(getNode('\0', sum, left, right));
    }
}

```

```

Node* root = pq.top();

std::unordered_map<char, std::string> huffmanCode;
encode(root, "", huffmanCode);

std::cout << "Коды Хаффмана:\n" << std::endl;
for (const auto& pair : huffmanCode) {
    std::cout << pair.first << " " << pair.second << std::endl;
}

std::string str = "";
for (char ch : text) {
    str += huffmanCode[ch];
}
int size1 = (str.size());

BitWriter bw;
bw.Write(str, "C:\\rep\\code_dec.txt");

std::cout << "\Закодированная строка :\n" << str << '\n';

std::string encodedString;
BitReader br;
br.Read(encodedString, "C:\\rep\\code_dec.txt");
std::string decodedString;

long long index = -1;
while (index < (long long)str.size() - 2) {
    decode(root, index, str, decodedString);
}

std::cout << decodedString;

}

int main()
{
    std::string text;

    std::fstream inFile;
    inFile.open("C:\\rep\\code.txt", std::ios::in);
    if (!inFile.is_open())
    {
        std::cout << "Failed to open file" << std::endl;
        return -1;
    }

    std::stringstream buffer;
    buffer << inFile.rdbuf();

    buildHuffmanTree(buffer.str());

    return 0;
}

```

Построенные коды Хаффмана

A 000000

r 0101

2 0010000000

b 1011101

т 000001

n 0111

. 0000100

O 0010000001

? 0000101

y 000011

3 1011111110010

s 0001

Щ 10111111101

Y 0010000010

G 0010000011

0 0010000100

N 0010000101

P 001000011

+ 10111001

k 00100010

- 00100011

7 101111111110

w 0010010

_ 00100110000

U 00100110001

R 0010011001

A 001001101

8 1011111111000

x 00100111

d 00101

! 101111111000
a 0011
) 1110000000
i 0100
5 1011111111011
u 01100
c 01101
4 1011111111010
t 1000
g 100100
f 100101

100110
h 100111
o 1010
l 10110
M 10111000000
B 10111000001
F 1011100001
j 1011100010
H 10111000110
q 10111000111
v 1011110
C 10111110
l 101111110
{ 1110000001001
; 1011111110011
Q 1011111111001
= 1110000001000
} 1011111111110
V 1011111111111

110
D 111000000101
Ь 111000000110
Э 111000000111
L 1110000010
(1110000011
, 11100001
: 1110001
р 111001
У 1110100
S 111010100
E 1110101010
z 1110101011
1 1110101100
W 1110101101
T 111010111
m 111011
e 1111

Размер файла до сжатия : 6 804 байт

Размер файла после сжатия : 3 948 байт

Задание 2. Решить задачу **3. Longest Substring Without Repeating Characters** на ресурсе *LeetCode*.

Профиль: <https://leetcode.com/DOXECEES/>

Исходный код:

```
class Solution {  
public:  
    int lengthOfLongestSubstring(string s)  
    {  
        if(s.size() == 0)  
            return 0;
```



```

std::bitset<128> bs;
int left = 0;
int right = 1;
bs[static_cast<int>(s[left])] = true;
int res = 1;
int size = s.size();

while(right < size)
{
    if(bs[static_cast<int>(s[right])] == true)
    {
        bs[static_cast<int>(s[left])] = false;
        left++;
        continue;
    }
    bs[static_cast<int>(s[right])] = true;
    int temp = right - left + 1;
    if(res < temp) {
        res = temp;
    }
    right++;
}

return res;
}
};

```

Runtime

Details

3ms

Beats 96.63% of users with C++

Memory

Details

7.15MB

Beats 98.26% of users with C++

Контрольные вопросы

1. Что такое *сжатие без потерь*?

Сжатие без потерь - это метод сжатия данных, при котором исходная информация восстанавливается точно и без потери качества после распаковки.

2. Опишите алгоритм построения кодов *Шеннона-Фано* для сжатия данных.

А) Упорядочиваем вероятности символов в порядке не возрастания.

Б) Не меняя порядка символов делим их на 2 группы, так чтобы суммарные вероятности были примерно равны.

В) Записываем к группе слева 0, а к группе справа 1.

Г) Если число элементов в группе больше 1, то переходим к шагу Б, если равно 1, то построение кода для этого элемента завершено.

3. Что такое *префиксные коды*, являются ли в данных алгоритмах коды *префиксными* и для чего они используются?

Префиксный код – это код, в котором ни одна из его комбинаций не является префиксом другой комбинации того же кода.

В обоих алгоритмах коды являются префиксными. Префиксные коды используются чтобы однозначно декодировать закодированный текст.

4. Благодаря каким принципам происходит *сжатие* данных в указанных алгоритмах?

Сжатие происходит благодаря тому, что наиболее часто встречаемым символам присваиваются коды, размер которых меньше 8 бит.

5. Укажите недостатки указанных кодов, средние коэффициенты сжатия для указанных алгоритмов.

Коды Хаффмана:

А) Недостаток эффективности при работе с неравномерными вероятностями: Коды Хаффмана лучше работают с данными, где вероятности символов близки и равномерно распределены. В случае, когда вероятности символов сильно отличаются, эффективность сжатия может снижаться.

Б) Избыточность для данных с небольшим количеством символов: Коды Хаффмана требуют создания таблицы кодирования для каждого символа, что может быть избыточным при работе с небольшим количеством символов.

В) Затраты на хранение таблицы кодирования: Для распаковки данных, необходимо хранить таблицу кодирования, которая может занимать дополнительное место.

Средняя степень сжатия для кодов Хаффмана - 1.48

Коды Шеннона-Фано:

А) Не всегда обеспечивает оптимальное сжатие: В отличие от кодов Хаффмана, коды Шеннона-Фано не всегда обеспечивают оптимальное сжатие. Они могут приводить к неравномерным длинам кодов и потере эффективности при работе с неравномерными вероятностями символов.

Б) Сложность построения: Алгоритм построения кодов Шеннона-Фано требует вычисления и сортировки вероятностей символов, что может быть более сложным и затратным процессом, особенно для больших наборов данных.

В) Затраты на хранение таблицы кодирования: Как и в случае с кодами Хаффмана, для распаковки данных, необходимо хранить таблицу кодирования, что может занимать дополнительное место.

Средняя степень сжатия для кодов Шеннона-Фано – 1.43

Вопросы по обработке алгоритма «Скользящего окна»

1. В каких еще задачах применяется этот алгоритм?

А) Сжатие данных: В задачах сжатия данных, алгоритм "скользящего окна" может использоваться для идентификации повторяющихся фрагментов данных.

Б) Обнаружение аномалий: Алгоритм "скользящего окна" может использоваться для обнаружения аномалий или изменений в последовательных данных.

В) Обработка текстовых данных: В задачах обработки текстовых данных, алгоритм "скользящего окна" может использоваться для анализа последовательностей символов или слов.

2. Чему равна временная сложность алгоритма? Чему равна временная сложность при решении задачи «в лоб»?

Сложность алгоритма «Скользящего окна» - $O(n)$

Сложность при решении в «лоб» - $O(n^2)$

3. Чему равна емкостная сложность алгоритма?

Ёмкостная сложность алгоритма – $O(1)$

Вывод: изучили алгоритм построения и декодирования кодов Хаффмана.