МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 8.1

по дисциплине
«Структуры и алгоритмы обработки данных»

Тема: «Алгоритмы кодирования и сжатия данных»

Выполнил студент: Лазаренко С.А.
Группа: ИКБО-10-23

Москва 2024

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

Исследовать и реализовать алгоритмы сжатия данных, а также сравнить их эффективности. В первой части задачи мы рассматриваем сжатие текста с использованием различных методов, таких как алгоритмы Хаффмана и Шеннона-Фано. В ходе выполнения задания будет проанализирован процесс сжатия данных на примерах, для чего каждый алгоритм будет представлен в виде таблицы с указанием результатов сжатия. Также будет описан процесс восстановления сжатого текста, что позволит понять, как происходит декодирование данных и восстановление исходной информации.

В рамках второй части работы планируется разработка программ для сжатия и восстановления текста с использованием методов Хаффмана и Шеннона-Фано.

17	Плыл по морю чемодан, В чемодане был диван, На диване ехал слон. Кто не верит – выйди вон!	0001000010101001101	webwerbweberweberweb

Оформить отчет в соответствии с требованиями документирования.

ЗАДАНИЕ 1

Формулировка задачи 1.1

Разработать решения задач с использованием заданных алгоритмов сжатия и кодирования.

Индивидуальный вариант работы – 17.

Описание подхода к решению

Алгоритм Шеннона-Фано основан на частоте символов, встречающихся в фразе. Более частым символам присваиваются более короткие коды, более редким — более длинные.

Ход работы

Частота	Символ	Код	бит
17		00	2
7	о	1101	4
7	е	1011	4
7	н	1100	4
6	а	1001	4
5	л	1000	4
5	д	0111	4
5	в	0110	4
4	и	11111	5
3	ы	10101	5
3	м	10100	5
2	,	111000	6
2	р	01000	5
2	ч	01011	5
2	т	01001	5
1	п	1111010	7
1	ю	010101	6
1	П	1110111	7
1	В	1110100	7
1	Н	1110101	7
1	К	1110011	7
1	.	1110110	7
1	-	1110010	7
1	б	1111000	7
1	х	010100	6
1	с	1111011	7
1	й	1111001	7

Рисунок 1 – Таблица решения задачи 1.1

Незакодированная фраза: 89 байта

Закодированная фраза: 392 бит

Для восстановления текста необходимо посимвольно сравнивать закодированную строку с кодами до нахождения совпадения. В случае совпадения обнулять буфер сравнения, а найденное совпадение сохранять.

Формулировка задачи 1.2

Сжатие данных по методу Лемпеля– Зива LZ77.

Ход работы

Таблица 1. - Решение задачи 1.2

Итерация	Словарь	Буфер	Совпадение	Код
1		0	(0, 0, 0)	(0, 0, 0)
2	0	0	(0, 0, 0)	(0, 0, 0)
3	0, 0	1	(0, 0, 1)	(0, 0, 1)
4	0, 0, 1	0	(2, 1, 0)	(2, 1, 0)
5	0, 0, 1, 0	0	(1, 0, 0)	(1, 0, 0)
6	0, 0, 1, 0, 0	1	(0, 0, 1)	(0, 0, 1)
7	0, 0, 1, 0, 0, 1	0	(3, 1, 0)	(3, 1, 0)
8	0, 0, 1, 0, 0, 1, 0,	10	(4, 1, 0)	(4, 1, 0)
9	0, 0, 1, 0, 0, 1, 0, 1	0	(5, 1, 0)	(5, 1, 0)
10	0, 0, 1, 0, 0, 1, 0, 1, 0	1	(6, 1, 1)	(6, 1, 1)
11	0, 0, 1, 0, 0, 1, 0, 1, 0,1	1	(0, 0, 1)	(0, 0, 1)

После завершения процесса кодирования, получаем следующие коды:

1. (0, 0, 0)
2. (0, 0, 0)
3. (0, 0, 1)
4. (2, 1, 0)
5. (1, 0, 0)
6. (0, 0, 1)
7. (3, 1, 0)
8. (4, 1, 0)
9. (5, 1, 0)
10. (6, 1, 1)
11. (0, 0, 1)

Формулировка задачи 1.3

Закодировать следующую фразу (webwerbweberweberweb), используя код LZ78.

Ход решения

Таблица 2. - Решение задачи 1.2

Индекс	Строка	Код
1	w	(0, w)
2	e	(0, e)
3	b	(0, b)
4	we	(1, e)
5	wer	(2, r)
6	r	(0, r)
7	be	(3, e)
8	we	(1, e)
9	web	(2, b)
10	weber	(6, w)
11	werw	(1, e)
12	webwer	(2, b)
13	webwerb	(2, e)

После завершения процесса кодирования, получаем следующие коды:

1. (0, 0, 0)
2. (0, 0, 1)
3. (2, 1, 0)
4. (1, 0, 0)
5. (0, 0, 1)
6. (3, 1, 0)
7. (4, 1, 0)
8. (5, 1, 0)
9. (6, 1, 1)
10. (0, 0, 1)

ЗАДАНИЕ 2

Разработать программы сжатия и восстановления текста методами Хаффмана и Шеннона – Фано.

1) Реализовать и отладить программы.

2) Сформировать отчет по разработке каждой программы в соответствии с требованиями.

- По методу Шеннона-Фано привести: постановку задачи, описать алгоритм формирования префиксного дерева и алгоритм кодирования, декодирования, код и результаты тестирования. Рассчитать коэффициент сжатия. Сравнить с результат сжатия вашим алгоритмом с результатом любого архиватора.
- По методу Хаффмана выполнить и отобразить результаты выполнения всех требований, предъявленных в задании и оформить разработку программы: постановка, подход к решению, код, результаты тестирования.

Формулировка задачи 2.1

Разработать алгоритм и реализовать программу сжатия текста алгоритмом Шеннона – Фано. Разработать алгоритм и программу восстановления сжатого текста. Выполнить тестирование программы на текстовом файле. Определить процент сжатия.

Математическая модель решения 2.1

Сжатие осуществляется за счет кодирования каждого символа исходного текста уникальным двоичным кодом, сформированным на основе частоты символов. Логика программы начинается с анализа частоты символов в исходном тексте. Сначала функция compressText загружает текст из файла inputFile, подсчитывает частоты каждого символа и сохраняет их в словаре frequencyMap, где ключом является символ, а значением — его частота. Затем все уникальные символы с их частотами добавляются в вектор symbols, где для каждого символа создается объект структуры Symbol, который хранит сам символ, его частоту и код, который будет присвоен в дальнейшем. Далее

вектор symbols сортируется по убыванию частот с помощью функции compareByFrequency, которая обеспечивает правильный порядок символов для кодирования.

Процесс кодирования выполняется рекурсивной функцией shannonFanoCoding, которая делит отсортированный массив символов на две группы с примерно равной суммарной частотой символов. Для этого в функции сначала подсчитывается общая частота всех символов в пределах индексов start и end, затем определяется точка разбиения splitPoint, где кумулятивная частота приближается к половине общей. Символам из первой группы присваивается код «0», а символам из второй — «1». Функция рекурсивно вызывает саму себя для каждой группы, пока не останется по одному символу в каждой группе, что завершает процесс кодирования для всех символов. В результате каждого символу в массиве symbols назначается уникальный код.

Затем программа сохраняет сжатый текст в новый файл outputFile. В этом процессе формируется строка compressedText, в которой каждый символ исходного текста заменяется его двоичным кодом из словаря encodingMap, сформированного по завершении shannonFanoCoding. Полученный сжатый текст сохраняется в outputFile в бинарном формате. После этого программа выводит процент сжатия, который вычисляется как отношение сжатого размера файла compressedSize к исходному originalSize, умноженное на 100, и вычтенное из 100%.

Для восстановления исходного текста из сжатого файла используется функция decompressText, которая считывает двоичный текст из файла outputFile и восстанавливает исходные символы, используя словарь decodingMap, построенный на основе encodingMap. Программа считывает сжатый текст, накапливая биты в строку codeBuffer до тех пор, пока эта последовательность битов не будет найдена в decodingMap как соответствующий код какого-либо символа. Когда совпадение найдено, соответствующий символ добавляется в decompressedText, и codeBuffer

очищается для накопления следующего кода. В конце восстановленный текст сохраняется в файл output2.txt, что завершает процесс восстановления.

Код программы 2.1

```cpp
#include <iostream>
#include <fstream>
#include <unordered_map>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

struct Symbol {
    char character;
    int frequency;
    string code;

    Symbol(char c, int freq, string cde) : character(c), frequency(freq), code(cde) {}
};

// Функция для сортировки символов по частотам
bool compareByFrequency(const Symbol& a, const Symbol& b) {
    return a.frequency > b.frequency;
}

// Рекурсивная функция для построения кодов Шеннона-Фано
void shannonFanoCoding(vector<Symbol>& symbols, int start, int end) {
    if (start >= end) return;

    int totalFrequency = 0;
    for (int i = start; i <= end; i++)
        totalFrequency += symbols[i].frequency;

    int splitPoint = start;
    int cumulativeFrequency = 0;
```

Рисунок 2 – Код программы 2.1 (1 часть)

```cpp
    for (int i = start; i <= end; i++) {
        cumulativeFrequency += symbols[i].frequency;
        if (cumulativeFrequency >= totalFrequency / 2) {
            splitPoint = i;
            break;
        }
    }

    // Разделяем и добавляем соответствующие префиксы
    for (int i = start; i <= splitPoint; i++)
        symbols[i].code += "0";
    for (int i = splitPoint + 1; i <= end; i++)
        symbols[i].code += "1";

    shannonFanoCoding(symbols, start, splitPoint);
    shannonFanoCoding(symbols, splitPoint + 1, end);
}

// Функция для сжатия текста
void compressText(const string& inputFile, const string& outputFile, unordered_map<char,
    ifstream inFile(inputFile);
    if (!inFile.is_open()) {
        cerr << "Ошибка открытия файла." << endl;
        return;
    }

    unordered_map<char, int> frequencyMap;
    string text((istreambuf_iterator<char>(inFile)), istreambuf_iterator<char>());
    inFile.close();

    // Подсчитываем частоту каждого символа
    for (char ch : text)
        frequencyMap[ch]++;
```

Рисунок 3 – Код программы 2.1 (2 часть)

```
// Функция для сжатия текста
id compressText(const string& inputFile, const string& outputFile, unordered_map<char, string>& encodingMap)
  ifstream inFile(inputFile);
  if (!inFile.is_open()) {
      cerr << "Ошибка открытия файла." << endl;
      return;
  }

  unordered_map<char, int> frequencyMap;
  string text((istreambuf_iterator<char>(inFile)), istreambuf_iterator<char>());
  inFile.close();

  // Подсчитываем частоту каждого символа
  for (char ch : text)
      frequencyMap[ch]++;

  vector<Symbol> symbols;
  for (auto& pair : frequencyMap)
      symbols.push_back(Symbol(pair.first, pair.second, ""));  // Исправлена инициализация

  // Сортируем символы по частоте
  sort(symbols.begin(), symbols.end(), compareByFrequency);

  // Строим кодировку Шеннона-Фано
  shannonFanoCoding(symbols, 0, symbols.size() - 1);

  // Заполняем карту кодирования
  for (auto& symbol : symbols)
      encodingMap[symbol.character] = symbol.code;

  // Сжимаем текст
  ofstream outFile(outputFile, ios::binary);
  string compressedText;
  for (char ch : text)
      compressedText += encodingMap[ch];
```

Рисунок 4 – Код программы 2.1 (3 часть)

```
    // Декодируем текст
    string codeBuffer;
    string decompressedText;
    for (char bit : compressedText) {
        codeBuffer += bit;
        if (decodingMap.find(codeBuffer) != decodingMap.end()) {
            decompressedText += decodingMap[codeBuffer];
            codeBuffer.clear();
        }
    }

    // Записываем декодированный текст в файл
    ofstream outFile(outputFile);
    outFile << decompressedText;
    outFile.close();
}

int main() {
    // Устанавливаем локаль для корректной работы с текстом на русском и других символах
    setlocale(LC_ALL, "en_US.UTF-8");

    // Карта для хранения кодов символов
    unordered_map<char, string> encodingMap;

    // Пример работы с сжатием и распаковкой текста
    compressText("input.txt", "output.txt", encodingMap);
    decompressText("output.txt", "output2.txt", encodingMap);

    return 0;
}
```

Рисунок 5 – Код программы 2.1 (4 часть)

Тестирование программы 2.1

input.txt

```
1    Hello from Sergey Lazarenko <3!
```

Рисунок 6 – Входной файл с текстом

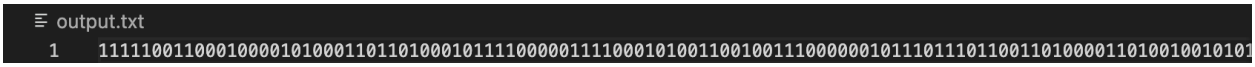

Рисунок 7 – Выходной файл с сжатым текстом

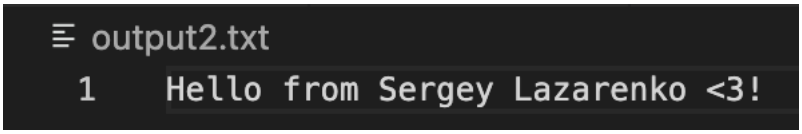

Рисунок 8 – Файл с восстановленным сжатым текстом

Рисунок 9 – Процент сжатия

Формулировка задачи 2.2

Провести кодирование(сжатие) исходной строки символов «Лазаренко Сергей Александрович» с использованием алгоритма Хаффмана. Исходная строка символов, таким образом, определяет индивидуальный вариант задания для каждого студента.

Код программы 2.2

```
struct HuffmanNode {
    char symbol;
    int frequency;
    HuffmanNode* left;
    HuffmanNode* right;

    // Конструктор для листовых узлов
    HuffmanNode(char symbol, int frequency) {
        this->symbol = symbol;
        this->frequency = frequency;
        left = right = nullptr;
    }

    // Конструктор для внутренних узлов, объединяющий два поддерева
    HuffmanNode(int frequency, HuffmanNode* left, HuffmanNode* right) {
        this->symbol = '\0';
        this->frequency = frequency;
        this->left = left;
        this->right = right;
    }
};

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

// Функция для построения кодов символов
void buildHuffmanCodes(HuffmanNode* root, string str, unordered_map<char, string>& huffmanCode) {
    if (!root) return;
    if (!root->left && !root->right) {
        huffmanCode[root->symbol] = str;
    }
    buildHuffmanCodes(root->left, str + "0", huffmanCode);
    buildHuffmanCodes(root->right, str + "1", huffmanCode);
}
```

Рисунок 10 – Код программы 2.2 (1 часть)

```
// Функция для освобождения памяти
void freeTree(HuffmanNode* root) {
    if (!root) return;
    freeTree(root->left);
    freeTree(root->right);
    delete root;
}

// Основная функция для кодирования
void huffmanEncode(const string& text) {
    unordered_map<char, int> frequency;
    for (char ch : text) {
        frequency[ch]++;
    }

    priority_queue<HuffmanNode*, vector<HuffmanNode*>, Compare> pq;
    for (auto pair : frequency) {
        pq.push(new HuffmanNode(pair.first, pair.second));
    }

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

        int sum = left->frequency + right->frequency;
        pq.push(new HuffmanNode(sum, left, right));
    }

    HuffmanNode* root = pq.top();

    unordered_map<char, string> huffmanCode;
    buildHuffmanCodes(root, "", huffmanCode);

    // Вывод кодов и закодированной строки
    cout << "Символы с кодами Хаффмана:\n";
    for (auto pair : huffmanCode) {
        cout << pair.first << ": " << pair.second << "\n";
    }
```

Рисунок 11 – Код программы 2.2 (2 часть)

```
    cout << "\nИсходная строка: " << text << "\nЗакодированная строка: ";
    string encodedString;
    int encodedSize = 0;
    for (char ch : text) {
        encodedString += huffmanCode[ch];
        encodedSize += huffmanCode[ch].length();
    }
    cout << encodedString << endl;

    // Расчет процента сжатия
    int originalSize = text.length() * 8;
    double compressionRatio = (1 - (double)encodedSize / originalSize) * 100;

    cout << "Исходный размер (в битах): " << originalSize << "\n";
    cout << "Закодированный размер (в битах): " << encodedSize << "\n";
    cout << "Процент сжатия: " << compressionRatio << "%\n";

    freeTree(root);
}

int main() {
    setlocale(LC_ALL, "RU");
    string text = "Лазаренко Сергей Александрович";
    huffmanEncode(text);
    return 0;
}
```

Рисунок 12 – Код программы 2.2 (3 часть)

Тестирование программы 2.2

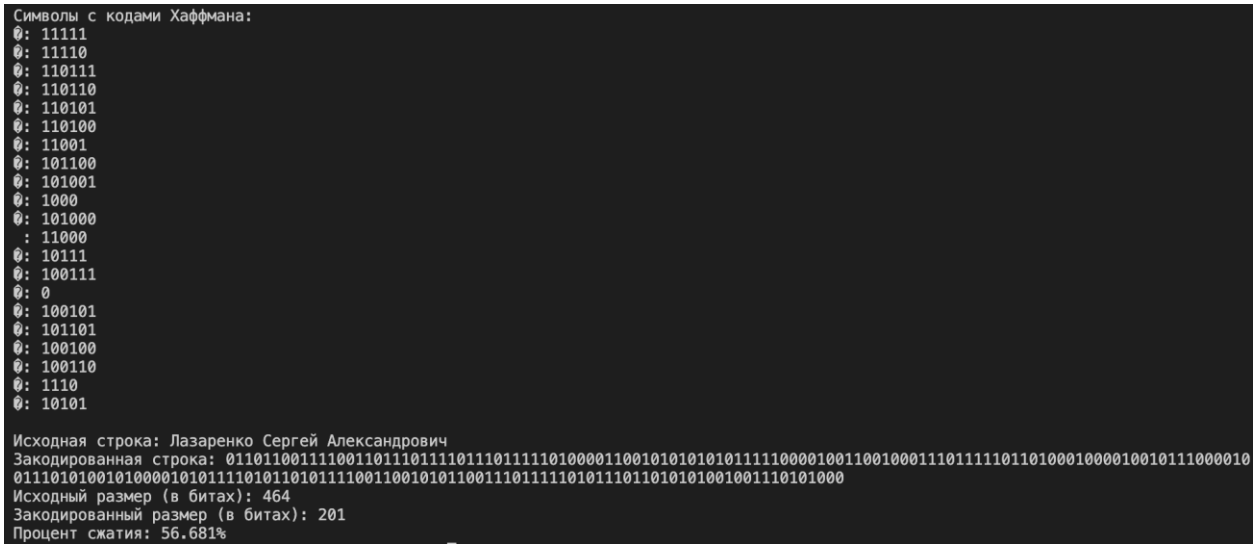

Рисунок 13 – Результат тестирования

ВЫВОД

В результате выполнения работы были освоены навыки по реализации алгоритмов сжатия и восстановления текста методами Хаффмана и Шеннона – Фано, сжатие данных по методу Лемпеля– Зива LZ77.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.

2. Документация по языку C++ [Электронный ресурс]. URL: https://docs.microsoft.com/ru-ru/cpp/cpp/ (дата обращения 08.09.2024).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 04.09.2024)