МИНОБРНАУКИ РОССИИ

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)

ОТЧЁТ ПО ПРАКТИЧЕСКИМ РАБОТАМ

по дисциплине

«Структуры и алгоритмы обработки данных (часть 2/2)»

Выполнил студент группы ИКБО-01-22 Прокопчук Р.О.

Принял Ермаков С.Р.
Ассистент

Практические работы выполнены «___»_____2023 г. (подпись студента)

«Зачтено» «___»_____2023 г. (подпись преподавателя)

Москва 2023

СОДЕРЖАНИЕ

1 ПРАКТИЧЕСКАЯ РАБОТА №1 5
1.1 Цель работы 5
1.2 Задание 1 5
1.2.1 Формулировка задачи 5
1.2.2 Математическая модель решения 5
1.2.3 Код программы с комментариями 6
1.2.4 Результаты тестирования 6
1.3 Задание 2 7
1.3.1 Формулировка задачи 7
1.3.2 Математическая модель решения 7
1.3.3 Код программы с комментариями 8
1.3.4 Результаты тестирования 10
1.4 Задание 3 10
1.4.1 Формулировка задачи 10
1.4.2 Математическая модель решения 11
1.4.3 Код программы с комментариями 12
1.4.4 Результаты тестирования 13
1.5 Вывод 13
2 ПРАКТИЧЕСКАЯ РАБОТА №2 14
2.1 Цель работы 14
2.1 Задание 1 14
2.2.1 Формулировка задачи 14
2.2.2 Описание подхода к решению 14
2.2.3 Код программы с комментариями 14
2.2.4 Результаты тестирования 16
2.3 Задание 2 16
2.3.1 Формулировка задачи 16
2.3.2 Описание подхода к решению 16
2.3.3 Код программы с комментариями 16

2.3.4 Результаты тестирования 16
2.4 Задание 3 17
2.4.1 Формулировка задачи 17
2.4.2 Описание подхода к решению 17
2.4.3 Код программы с комментариями 17
2.4.4 Результаты тестирования 20
2.5 Результат работы программы 20
2.6 Выводы 20
3 ПРАКТИЧЕСКАЯ РАБОТА №3 21
3.1 Цель работы 21
3.2 Задание 21
3.2.1 Формулировка задачи 21
3.2.2 Код программы с комментариями 22
3.2.3 Результаты тестирования 25
3.3 Вывод 25
4 ПРАКТИЧЕСКАЯ РАБОТА №4 26
4.1 Цель работы 26
4.2 Задание 1 26
4.2.1 Формулировка задачи 26
4.2.2 Код программы с комментариями 27
4.2.3 Результаты тестирования 27
4.3 Задание 2 28
4.3.1 Формулировка задачи 28
4.3.2 Код программы с комментариями 28
4.3.3 Результаты тестирования 30
4.4 Вывод 30
5 ПРАКТИЧЕСКАЯ РАБОТА №5 31
5.1 Цель работы 31
5.2 Задание 31
5.2.1 Формулировка задачи 31

5.2.2 Код программы с комментариями 31
5.2.3 Результаты тестирования 33
5.3 Вывод 33
6 ПРАКТИЧЕСКАЯ РАБОТА №6 34
6.1 Цель работы 34
6.2 Задание 34
6.2.1 Формулировка задачи 34
6.2.2 Код программы с комментариями 34
6.2.3 Результаты тестирования 36
6.3 Вывод 36
7 ПРАКТИЧЕСКАЯ РАБОТА №7 37
7.1 Цель работы 37
7.2 Задание 1 37
7.2.1 Формулировка задачи 37
7.2.2 Код программы с комментариями 37
7.3 Задание 2 40
7.3.1 Формулировка задачи 40
7.3.2 Код программы с комментариями 40
7.3.3 Результаты тестирования 41
7.4 Вывод 41
8 ПРАКТИЧЕСКАЯ РАБОТА №8 42
8.1 Цель работы 42
8.2 Задание 42
8.2.1 Формулировка задачи 42
8.2.2 Код программы с комментариями 42
8.2.3 Результаты тестирования 43
8.3 Вывод 43

1 ПРАКТИЧЕСКАЯ РАБОТА №1

1.1 Цель работы

Освоить приёмы работы с битовым представлением беззнаковых целых чисел, реализовать эффективный алгоритм внешней сортировки на основе битового массива.

1.2 Задание 1

1.2.1 Формулировка задачи

1.а. Реализуйте вышеприведённый пример, проверьте правильность результата в том числе и на других значениях x.

1.б. Реализуйте по аналогии с предыдущим примером установку 7-го бита числа в единицу.

1.в. Реализуйте код листинга 1, объясните выводимый программой результат.

1.2.2 Математическая модель решения

В числе со знаком под модуль отведены все двоичные разряды, кроме старшего (рис. 2). Одно из значений старшего бита интерпретируется как знак «плюс», противоположное – как «минус». Т.к. разрядов под модуль числа на 1 меньше, то и наибольшее допустимое значение в типе со знаком вдвое меньше такового в беззнаковом типе.

Примечание: векторный способ организации числовых последовательностей (т.е. массивы чисел) в памяти компьютера формирует непрерывную последовательность бит от начального до конечного элемента этого массива.

При работе с битовыми представлениями чисел можно использовать ***битовые операции***, определённые в языке C++ (см. табл. 2).

Таблица 2. Битовые операции в C++.

x<<n	Сдвиг влево двоичного кода (умножение на 2^n)	int x=7; x=x<<2; // x=111<<2=11100
x>>n	Сдвиг вправо двоичного кода (деление на 2^n	100>>1=010

x & maska	Поразрядное И (запись в бит 0)	111 & 100 =100
		short int maska=0x1F;
		short int x=0xFFFFFFFF;
		x & maska (0x0000001F)

X | maska	Поразрядное ИЛИ (запись в бит 1)	111 | 100 =111
		short maska=0x1F;
		short int x=0xFFFFFF00;
		x & maska (0xFFFFFF1F)

X ^maska	Исключающее ИЛИ для поразрядных операций	unsigned int x=0xF, a=1; A=x^a; 1111 ^ 0001=1110
~	инверсия	x=0x0F; ~x (0xFFFFFFF0)

Пример – как установить 5-й бит произвольного целого числа в 0 и что получится в результате:

unsigned char x=255; *//8-разрядное двоичное число 11111111*

unsigned char maska = 1; *//1=00000001 – 8-разрядная маска*

x = x & (~ (maska<<4)); *//результат x=239*

1.2.3 Код программы с комментариями

Код программы представлен на рисунке 1.

```
void taskOneA(){
    unsigned char x = 255; // 8-ми разрядное число
    unsigned char maska = 1; // 1 = 00000001 - 8-разрядная маска
    x = x & (~(maska<<4)); // Устанавливаем 5-ый бит в 0
    cout << (int) x; // Вывод результата в виде числа
}

void taskOneB(){
    unsigned char x = 255; // 8-ми разрядное число
    unsigned char maska = 1; // 1 = 00000001 - 8-разрядная маска
    x = x & (~(maska<<6)); // Устанавливаем 7-ый бит в 0
    cout << (int) x; // Вывод результата в виде числа
}

void taskOneV(){
    unsigned int x = 25;
    const int n = sizeof(int)*8; //=32 - количество разрядов в числе типа int
    unsigned maska = (1 << (n - 1)); //1 в старшем бите 32-разрядной сетки
    cout << "Начальный вид маски: " << bitset<n> (maska) << endl;
    cout << "Результат: ";
    for(int i = 1; i <= n; i++){ //32 раза - по количеству разрядов:
        cout << ((x & maska) >> (n - i));
        maska = maska >> 1; //смещение 1 в маске на разряд вправо
    }
    cout << endl;
}
```

Рисунок 1 – Листинг кода программы

1.2.4 Результаты тестирования

Функции были успешно протестированы на разных числах.

1.3 Задание 2

1.3.1 Формулировка задачи

2.а. Реализуйте вышеописанный пример с вводом произвольного набора до 8-ми чисел (со значениями от 0 до 7) и его сортировкой битовым массивом в виде числа типа unsigned char. Проверьте работу программы.

Если количество чисел в исходной последовательности больше 8 и/или значения превосходят 7, можно подобрать тип беззнакового числа для битового массива с подходящим размером разрядной сетки – до 64 в типе unsigned long long (см. табл. 1).

2.б. Адаптируйте вышеприведённый пример для набора из 64-х чисел (со значениями от 0 до 63) с битовым массивом в виде числа типа unsigned long long.

Если количество чисел и/или их значения превосходят возможности разрядной сетки одного беззнакового целого числа, то можно организовать линейный массив (вектор) таких чисел, который в памяти ЭВМ будет представлен ***одной непрерывной битовой последовательностью***.

2.в. Исправьте программу задания 2.б, чтобы для сортировки набора из 64-х чисел использовалось не одно число типа unsigned long long, а линейный массив чисел типа unsigned char.

1.3.2 Математическая модель решения

Пусть даны не более 8 чисел со значениями от 0 до 7, например, {1, 0, 5, 7, 2, 4}.

Подобный набор чисел удобно отразить в виде 8-разрядной битовой последовательности **11101101**. В ней единичные биты показывают ***наличие*** в исходном наборе числа, равного номеру этого бита в последовательности (нумерация с 0 слева). Т.о. индексы единичных битов в битовом массиве – это и есть числа исходной последовательности.

Последовательное считывание бит этой последовательности и вывод индексов единичных битов позволит естественным образом получить исходный набор чисел ***в отсортированном виде*** – {0, 1, 2, 4, 5, 7}.

В качестве подобного битового массива удобно использовать беззнаковое однобайтовое число (его двоичное представление в памяти), например, типа unsigned char. Приёмы работы с отдельными битами числа были рассмотрены в предыдущем задании.

1.3.3 Код программы с комментариями

Код программы представлен на рисунках 2-4.

```
void taskTwoA(){
    unsigned char bitArray = 0; // Битовый массив
    unsigned char maska = 1;
    int inputNumber; // Переменная для ввода чисел
    const int sizeOfChar = 8; // Кол-во бит, которое приходится на unsigned
char

    vector<int> numbersArray;
    cout << "Введите числа (не больше 7) Для завершения ввода введите -1: "
<< endl;

    // Заполнение массива чисел
    while (true) {
        std::cin >> inputNumber;

        if (inputNumber == -1) {
            break;
        }
        numbersArray.push_back(inputNumber);
    }

    cout << "Изначальный массив: ";
    for(int el : numbersArray){
        cout << el << " ";
        bitArray = bitArray | (maska << el);
    }

    // Вывод набора чисел в битовой последовательности
    bitset<sizeOfChar> bitset(bitArray);
    cout << endl;
    cout << "Набор чисел в битовой последовательности: " << bitset << endl;

    // Вывод отсортированного массива
    cout << "Отсортированный массив: ";
    for(int i = 0; i < sizeOfChar; i++){
        if(bitset[i])
            cout << i << " ";
    }
}
```

Рисунок 2 – Листинг кода функции taskTwoA

```
void taskTwoB(){
    unsigned long long bitArray = 0; // Битовый массив
    unsigned long long maska = 1;
    int inputNumber; // Переменная для ввода чисел
    const int sizeOfUnsignedLongLong = 64; // Кол-во бит, которое приходится
на unsigned long long

    vector<int> numbersArray;
    cout << "Введите числа (не больше 63) Для завершения ввода введите -1: "
<< endl;

    // Заполнение массива чисел
    while (true) {
        std::cin >> inputNumber;

        if (inputNumber == -1) {
            break;
        }
        numbersArray.push_back(inputNumber);
    }

    cout << "Изначальный массив: ";
    for(int el : numbersArray){
        cout << el << " ";
        bitArray = bitArray | (maska << el);
    }

    // Вывод набора чисел в битовой последовательности
    bitset<sizeOfUnsignedLongLong> bitset(bitArray);
    cout << endl;
    cout << "Набор чисел в битовой последовательности: " << bitset << endl;

    // Вывод отсортированного массива
    cout << "Отсортированный массив: ";
    for(int i = 0; i < sizeOfUnsignedLongLong; i++){
        if(bitset[i])
            cout << i << " ";
    }
}
```

Рисунок 3 – Листинг кода функции taskTwoB

```
void taskTwoV(){
    const int bitArrayLength = 8; // Длинна массива из char значений
    unsigned char bitArray[bitArrayLength]; // Битовый массив
    unsigned char maska = 1;
    int inputNumber;
    const int sizeOfUnsignedChar = 8; // Кол-во бит, которое приходится на
unsigned char

    vector<int> numbersArray;
    cout << "Введите числа (не больше 63) Для завершения ввода введите -1: "
<< endl;

    // Заполнение массива чисел
    while (true) {
        std::cin >> inputNumber;

        if (inputNumber == -1) {
            break;
        }
        numbersArray.push_back(inputNumber);
    }

    cout << "Изначальный массив: ";
    for(int el : numbersArray){
        cout << el << " ";
        bitArray[el / sizeOfUnsignedChar] = bitArray[el / sizeOfUnsignedChar]
| (maska << (el % sizeOfUnsignedChar));
    }

    // Вывод набора чисел в битовой последовательности
    cout << endl;
    cout << "Набор чисел в битовой последовательности: ";
    for (int i = bitArrayLength - 1; i >= 0; i--) {
        cout << bitset<sizeOfUnsignedChar>(bitArray[i]);
    }
    cout << endl;

    // Вывод отсортированного массива
    cout << "Отсортированный массив: ";
    for (int i = 0; i < bitArrayLength; i++) {
        bitset<sizeOfUnsignedChar> bitset(bitArray[i]);
        for(int j = 0; j < sizeOfUnsignedChar; j++){
            if(bitset[j])
                cout << j + i * sizeOfUnsignedChar << " ";
        }
    }
}
```

Рисунок 4 – Листинг кода функции taskTwoV

1.3.4 Результаты тестирования

Функции были успешно протестированы на разных наборах входных данных.

1.4 Задание 3

1.4.1 Формулировка задачи

Входные данные: файл, содержащий не более $n=10^7$ неотрицательных целых чисел, среди них нет повторяющихся.

Результат: упорядоченная по возрастанию последовательность исходных чисел в выходном файле.

Время работы программы: ~10 с (до 1 мин. для систем малой вычислительной мощности).

Максимально допустимый объём ОЗУ для хранения данных: 1 МБ.

Очевидно, что размер входных данных гарантированно превысит 1МБ (это, к примеру, максимально допустимый объём стека вызовов, используемого для статических массивов).

Требование по времени накладывает ограничение на количество чтений исходного файла.

3.а. Реализуйте задачу сортировки числового файла с заданными условиями. Добавьте в код возможность определения времени работы программы.

Примечание: содержимое входного файла должно быть сформировано неповторяющимися значениями заранее, это время не должно учитываться при замере времени сортировки.

В отчёт внесите результаты тестирования для наибольшего количества входных чисел, соответствующего битовому массиву длиной 1МБ.

3.б. Определите программно объём оперативной памяти, занимаемый битовым массивом.

1.4.2 Математическая модель решения

Пусть даны не более 8 чисел со значениями от 0 до 7, например, {1, 0, 5, 7, 2, 4}.

Подобный набор чисел удобно отразить в виде 8-разрядной битовой последовательности **11101101**. В ней единичные биты показывают ***наличие*** в исходном наборе числа, равного номеру этого бита в последовательности (нумерация с 0 слева). Т.о. индексы единичных битов в битовом массиве – это и есть числа исходной последовательности.

Последовательное считывание бит этой последовательности и вывод индексов единичных битов позволит естественным образом получить

исходный набор чисел ***в отсортированном виде*** – {0, 1, 2, 4, 5, 7}.

В качестве подобного битового массива удобно использовать беззнаковое однобайтовое число (его двоичное представление в памяти), например, типа unsigned char. Приёмы работы с отдельными битами числа были рассмотрены в предыдущем задании.

1.4.3 Код программы с комментариями

Код программы представлен на рисунке 5.

```
void taskThree(){
    unsigned int start_time =  clock();
    // Открытие файла для чтения
    ifstream inputFile("input.txt");
    if(!inputFile){
        cout << "Не удается открыть файл для чтения!";
        return;
    }
    // Открытие файла для записи
    ofstream outputFile("output.txt");
    if(!outputFile){
        cout << "Не удается открыть файл для записи!";
        return;
    }
    outputFile.clear();
    const int bitArrayLength = 1048576; // Длинна массива из char значений
    unsigned char bitArray[bitArrayLength]; // Битовый массив
    const int sizeOfUnsignedChar = 8; // Кол-во бит, которое приходится на
unsigned char
    unsigned char maska = 1;
    std::memset(bitArray, 0, bitArrayLength);
    // Заполнение массива чисел
    vector<int> numbersArray;
    int num;
    while(inputFile >> num)
        numbersArray.push_back(num);
    inputFile.close();
    // Заполнение битового массива
    for(int el : numbersArray)
        bitArray[el / sizeOfUnsignedChar] = bitArray[el / sizeOfUnsignedChar]
| (maska << (el % sizeOfUnsignedChar));
    // Запись результатов в файл
    for (int i = 0; i < bitArrayLength; i++) {
        bitset<sizeOfUnsignedChar> bitset(bitArray[i]);
        for(int j = 0; j < sizeOfUnsignedChar; j++){
            if(bitset[j] == 1)
                outputFile << j + i * sizeOfUnsignedChar << " ";
        }
    }
    outputFile.close();
    // Вывод результатов работы программы
    unsigned int end_time = clock();
    cout << "Время сортировки: " << end_time - start_time << " мс" << endl;
    cout << "Размер битового массива: " << sizeof(bitArray) / 1024 / 1024 <<
" Мбайт";
}
```

Рисунок 5 – Листинг кода функции taskThree

1.4.4 Результаты тестирования

Функции были успешно протестированы на разных наборах входных данных.

1.5 Вывод

В ходе работы были Освоены приёмы работы с битовым представлением беззнаковых целых чисел, реализован эффективный алгоритм внешней сортировки на основе битового массива, изучены соответствующие конструкции языка C++.

Время роботы битовой сортировки составило 2с.

2 ПРАКТИЧЕСКАЯ РАБОТА №2

2.1 Цель работы

Поучить практический опыт по применению алгоритмов поиска в таблицах данных.

2.1 Задание 1

2.2.1 Формулировка задачи

Создать двоичный файл из записей (структура записи определена вариантом). Поле ключа записи в задании варианта подчеркнуто. Заполнить файл данными, используя для поля ключа датчик случайных чисел. Ключи записей в файле уникальны. Рекомендация: создайте сначала текстовый файл, а затем преобразуйте его в двоичный.

2.2.2 Описание подхода к решению

Записи в файле реализованы при помощи структуры с тремя полями: номер лицензии, название и учередитель.

Уникальные номера лицензий генерируются при помощи <random>. Уже использованные значения записываются в вектор. Затем при генерации нового числа программа проверяет, есть ли уже это число в векторе, если нет, то генерирует новое значение.

Структура записывается в файл в двоичном виде при помощи reinterpret_cast.

2.2.3 Код программы с комментариями

Код программы представлен на рисунке 6.

```
struct Registration{
    int licenseNumber = -1;
    char name[20]{};
    char founder[20]{};
};

bool isInVector(const vector<int>& vector, int n){ // Функция, проверяющая
есть ли число в векторе
    for(int el : vector)
        if(el == n)
            return true;
    return false;
}

void createBinaryFile(const string& fileName, int numRecords){
    // Инициализация всего необходимого для работы random
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dist(100000,999999);

    // Открытие текстового и бинарного файлов для записи
    ofstream file(fileName, ios::binary | ios::out);
    ofstream fileTxt(fileName + ".txt", ios::binary | ios::out);

    if(!file){
        cout << "Ошибка при открытии файла для бинарной записи!" << endl;
        return;
    }

    vector<int> licenseNumbers;
    for(int i = 0; i < numRecords; i++){ // Цикл, реализующий добавление
запсисей в файл
        int n;

        while (true){ // Цикл, генерирующий уникальные номера лицезий
            n = dist(gen);
            if(!isInVector(licenseNumbers, n))
                break;
        }
        licenseNumbers.push_back(n);

        Registration record;
        record.licenseNumber = n;
        // Запись в массив char строк
        snprintf(record.name, sizeof(record.name), "Company %d", i);
        snprintf(record.founder, sizeof(record.founder), "Founder %d", i);

        // добавление записей в файл
        file.write(reinterpret_cast<char*>(&record), sizeof(Registration));
        fileTxt << record.licenseNumber << " " << record.name << " " <<
record.founder << endl;
    }

    file.close();
    fileTxt.close();
}
```

Рисунок 6 – Листинг кода функции createBinaryFile

2.2.4 Результаты тестирования

Функции были успешно протестированы на разных наборах аргументов.

2.3 Задание 2

2.3.1 Формулировка задачи

Поиск в файле с применением линейного поиска

1. Разработать программу поиска записи по ключу в бинарном файле с применением алгоритма линейного поиска.

2. Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей.

3. Составить таблицу с указанием результатов замера времени

2.3.2 Описание подхода к решению

Для написания функции линейного поиска был использован стандартный алгоритм. Записи считываются из файла поочередно при помощи reinterpret_cast.

2.3.3 Код программы с комментариями

Код программы представлен на рисунке 7.

```
Registration linearSearch(const string& fileName, int key){ // Функция
линейного поиска
    ifstream file(fileName, ios::binary | ios::in);

    if (!file){
        cout << "Ошибка при открытии файла для линейного поиска!" << endl;
        return Registration{-2, "", ""};
    }

    Registration record;
    while (file.read(reinterpret_cast<char*>(&record), sizeof
(Registration))){
        if(record.licenseNumber == key){
            file.close();
            return record;
        }
    }

    file.close();
    return Registration{-1, "", ""};
}
```

Рисунок 7 – Листинг кода функции linearSearch

2.3.4 Результаты тестирования

Функции были успешно протестированы на разных наборах входных данных.

2.4 Задание 3

2.4.1 Формулировка задачи

Поиск записи в файле с применением дополнительной структуры данных, сформированной в оперативной памяти.

Для оптимизации поиска в файле создать в оперативной памяти структур данных – таблицу, содержащую ключ и ссылку (смещение) на запись в файле.

Разработать функцию, которая принимает на вход ключ и ищет в таблице элемент, содержащий ключ поиска, а возвращает ссылку на запись в файле. Алгоритм поиска определен в варианте.

Разработать функцию, которая принимает ссылку на запись в файле, считывает ее, применяя механизм прямого доступа к записям файла. Возвращает прочитанную запись как результат.

Провести практическую оценку времени выполнения поиска на файле объемом 100, 1000, 10 000 записей.

Составить таблицу с указанием результатов замера времени.

2.4.2 Описание подхода к решению

Бинарный поиск был реализован стандартным алгоритмом. Для хранения всех записей использовался vector. Данные были отсортированы по ключам при помощи функции sort из <algotrithm>.

2.4.3 Код программы с комментариями

Код программы представлен на рисунках 8-9.

```
Registration binarySearch(const string& fileName, int key){ // Функция
бинарного поиска
    ifstream file(fileName, ios::binary | ios::in);

    if(!file){
        cout << "Ошибка при открытии файла для бинарного поиска!";
        return Registration{-2, "", ""};
    }

    file.seekg(0, ios::end);
    int fileSize = file.tellg();
    int numRecords = fileSize / sizeof(Registration);

    vector<Registration> records;

    file.seekg(0, ios::beg);

    Registration record;
    while (file.read(reinterpret_cast<char*>(&record), sizeof(Registration)))
        records.push_back(record);

    file.close();

    sort(records.begin(), records.end(), [](const Registration& a, const
Registration& b){
        return a.licenseNumber < b.licenseNumber;
    });

    int left = 0;
    int right = numRecords - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (records[mid].licenseNumber == key){
            return records[mid];
        }

        if(records[mid].licenseNumber < key)
            left = mid + 1;
        else
            right = mid - 1;
    }

    return Registration{-1, "", ""};
}
```

Рисунок 8 – Листинг кода функции binarySearch

```
int main() {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dist(100000,999999);

    SetConsoleOutputCP(CP_UTF8);
    string fileName;
    int numberOfRecords[] = {100, 1000, 10000};

    for (int num : numberOfRecords) {
        fileName = "Registration_l" + to_string(num) + ".bin";
        createBinaryFile(fileName, num);

        int key = dist(gen);

        unsigned int start_time = clock();
        Registration result = linearSearch(fileName, key);
        unsigned int end_time = clock();

        cout << "Линейный поиск в файле из " << num << " элементов:" << endl;
        if (result.licenseNumber != -1){
            cout << "Запись найдена |Номер лицензии - " <<
result.licenseNumber
            << "|Учредитель - " << result.founder << "|Название - " <<
result.name
            << "|Время выполнение поиска - " << end_time - start_time << "мс"
<< endl;
        }
        else{
            cout << "Запись не найдена! Ключ - " << key << "|Время выполнение
поиска - "
            << end_time - start_time << "мс"  << endl;
        }
    }

    cout << endl << endl;

    for (int num : numberOfRecords) {
        fileName = "Registration_b" + to_string(num) + ".bin";
        createBinaryFile(fileName, num);
        int key = dist(gen);
        unsigned int start_time = clock();
        Registration result = binarySearch(fileName, key);
        unsigned int end_time = clock();
        cout << "Бинарный поиск в файле из " << num << " элементов:" << endl;
        if (result.licenseNumber != -1){
            cout << "Запись найдена |Номер лицензии - " <<
result.licenseNumber
                 << "|Учредитель - " << result.founder << "|Название - " <<
result.name
                 << "|Время выполнение поиска - " << end_time - start_time <<
"мс" << endl;
        }
        else{
            cout << "Запись не найдена! Ключ - " << key << "|Время выполнение
поиска - "
            << end_time - start_time << "мс" << endl;
        }
    }
    return 0;
}
```

Рисунок 9 – Листинг кода функции main

2.4.4 Результаты тестирования

Функции были успешно протестированы на разных наборах входных данных.

2.5 Результат работы программы

Таблица 1 – Время работы алгоритмов поиска с различным количеством тестовых записей.

Кол-во записей	Линейный поиск (время работы)	Бинарный поиск (время работы)
100	0 мс	0 мс
1000	0 мс	8 мс
10000	10 мс	16 мс

2.6 Выводы

В ходе работы был получен практический опыт по применению алгоритмов поиска в таблицах данных.

Линейный поиск показал лучшую эффективность за счет отсутствия сортировки, необходимой для работы бинарного поиска. Однако данный вывод нельзя сделать из набора чисел, заданного в задании, так как он слишком мал, чтобы сделать выводы о скорости работы программы, а погрешность замера скорости работы программы в этом случае слишком велика.

3 ПРАКТИЧЕСКАЯ РАБОТА №3

3.1 Цель работы

Освоить приёмы хеширования и эффективного поиска элементов множества.

3.2 Задание

3.2.1 Формулировка задачи

Разработайте приложение, которое использует ***хеш-таблицу*** (пары «ключ – хеш») для организации прямого доступа к элементам ***динамического множества*** полезных данных. Множество реализуйте на массиве, структура элементов (перечень полей) которого приведена в индивидуальном варианте (п.3).

Приложение должно содержать ***класс*** с ***базовыми операциями***: вставки, удаления, поиска по ключу, вывода. Включите в класс массив полезных данных и хеш-таблицу. Хеш-функцию подберите самостоятельно, используя правила выбора функции.

Реализуйте расширение размера таблицы и ***рехеширование***, когда это требуется, в соответствии с типом разрешения ***коллизий***.

Предусмотрите автоматическое заполнение таблицы 5-7 записями.

Реализуйте текстовый ***командный интерфейс*** пользователя для возможности вызова методов в любой произвольной последовательности, сопроводите вывод достаточными для понимания происходящего сторонним пользователем подсказками.

Проведите полное тестирование программы (все базовые операции, изменение размера и рехеширование), тест-примеры определите самостоятельно. Результаты тестирования включите в отчет по выполненной работе.

Примечание: тесты должны включать в себя случаи коллизий, проверке подлежит правильность вставки, поиска и удаления записей, вызвавших коллизию.

3.2.2 Код программы с комментариями

Код программы представлен на рисунках 10-12.

```
#ifndef SIAOD_PRACTISE_3_HASHTABLE_H
#define SIAOD_PRACTISE_3_HASHTABLE_H

#include <iostream>
#include <utility>

using namespace std;

class Entry{
public:
    string phoneNumber;
    string address;

    Entry(string  phone, string  addr) : phoneNumber(std::move(phone)),
address(std::move(addr)) {}
};


class HashTable {
private:
    static const int TABLE_SIZE = 11;
    Entry* table[TABLE_SIZE]{};

    static int hashFunction(const string& key);
    static int quadraticProbe(int index, int attempt);

public:
    HashTable();

    void insert(const string& phone, const string& addr);
    void remove(const string& phone);
    Entry* search(const string& phone);
    void display();
};

#endif //SIAOD_PRACTISE_3_HASHTABLE_H
```

Рисунок 10 – Листинг кода файла HashTable.h

```
HashTable::HashTable() {
    for(auto & el : table)
        el = nullptr;
}
int HashTable::hashFunction(const string& key) {
    int hash = 0;
    for(char ch : key)
        hash += static_cast<int>(ch);
    return hash % TABLE_SIZE;
}
int HashTable::quadraticProbe(int index, int attempt) {
    return (index + attempt * attempt) % TABLE_SIZE;
}

void HashTable::insert(const string& phone, const string& addr) {
    int index = hashFunction(phone);
    int attempt = 0;
    while (table[index] != nullptr){
        attempt++;
        index = quadraticProbe(index, attempt);
    }
    table[index] = new Entry(phone, addr);
    cout << "Запись добавлена!" << endl;
}

void HashTable::remove(const string& phone) {
    int index = hashFunction(phone);
    int attempt = 0;
    while (table[index] != nullptr){
        if(table[index]->phoneNumber == phone){
            delete table[index];
            table[index] = nullptr;
            cout << "Запись удалена!" << endl;
            return;
        }
        attempt++;
        index = quadraticProbe(index, attempt);
    }
    cout << "Запись не найдена!" << endl;
}
Entry *HashTable::search(const string& phone) {
    int index = hashFunction(phone);
    int attempt = 0;
    while (table[index] != nullptr){
        if(table[index]->phoneNumber == phone){
            cout << "Запись найдена! " << endl;
            return table[index];
        }
        attempt++;
        index = quadraticProbe(index, attempt);
    }
    cout << "Запись не найдена! " << endl;
    return nullptr;
}
void HashTable::display() {
    for(auto & el : table){
        if (el != nullptr)
            cout << "Номер телефона: " << el->phoneNumber << " | Адрес: " <<
el->address << endl;
    }
}
```

Рисунок 11 – Листинг кода файла HashTable.cpp

```
void printMenu(){
    cout << "----------------------------" << endl;
    cout << "Выберите действие:" << endl;
    cout << "1 - Добавить запись в таблицу;" << endl;
    cout << "2 - Удалить запись из таблицы;" << endl;
    cout << "3 - Найти запись в таблице;" << endl;
    cout << "4 - Вывести все записи;" << endl;
    cout << "0 - Выход." << endl;
    cout << "----------------------------" << endl;
}

int main() {
    SetConsoleOutputCP(CP_UTF8);

    HashTable phoneBook;
    phoneBook.insert("+79151981661", "Лазурная, 16к4");
    phoneBook.insert("+79163062984", "Проезд Одоевского, 7к3");
    phoneBook.insert("+74954225868", "Шоссейная, 5");

    int userChoice;
    string phoneNumber, address;
    bool isWorking = true;
    while (isWorking){
        printMenu();
        cin >> userChoice;
        switch (userChoice) {
            case 1:
                cout << "Введите номер телефона:";
                cin >> phoneNumber;
                cout << "Введите адрес:";
                cin.ignore();
                getline(cin, address);
                phoneBook.insert(phoneNumber, address);
                break;
            case 2:
                cout << "Введите номер телефона:";
                cin >> phoneNumber;
                phoneBook.remove(phoneNumber);
                break;
            case 3:
                cout << "Введите номер телефона:";
                cin >> phoneNumber;
                Entry *entry;
                entry = phoneBook.search(phoneNumber);
                if(entry != nullptr)
                    cout << "Номер телефона: " << entry->phoneNumber << " |
Адрес: " << entry->address << endl;
                break;
            case 4:
                phoneBook.display();
                break;
            case 0:
                isWorking = false;
                cout << "Завершение работы программы...";
                break;
            default:
                cout << "Такая команда не поддерживается!" << endl;
                break;
        }
    }
    return 0;
}
```

Рисунок 12 – Листинг кода файла main.cpp

3.2.3 Результаты тестирования

Программа была успешно протестирована на различных наборах данных.

3.3 Вывод

В ходе работы были освоены приёмы хеширования и эффективного поиска элементов множества.

4 ПРАКТИЧЕСКАЯ РАБОТА №4

4.1 Цель работы

Освоить приёмы реализации алгоритмов поиска образца в тексте.

4.2 Задание 1

4.2.1 Формулировка задачи

Пуст имеются некоторый ***текст*** T (haystack) длиной n и ***образец*** или шаблон W (needle) – тоже текст (подстрока) длиной m. Строки T и W можно рассматривать как массивы из n и m ***символов*** соответственно, причем $0<m\leq n$.

Элементы массивов T и W – это символы некоторого конечного ***алфавита***, к примеру: {0, 1}, или {a, …, z}, или {a, …, я}.

Задача поиска в простейшем случае сводится к нахождению первого слева вхождения этого образца в указанный текст; необходимо сообщить об успехе/неудаче и, возможно, вернуть индекс, начиная с которого образец присутствует в тексте.

Классический алгоритм решения этой задачи – последовательный (линейный) поиск. Он заключается в прикладывании образца к тексту, начиная с левого края, и посимвольного сравнения слева направо до конца образца (успех с возвратом индекса начала вхождения образца в текст) или до первого несоответствия символов (рис. 1). В последнем случае образец смещается на 1 символ вправо. Если несоответствие символов нашлось на последнем возможном смещении, то возвращается сообщение об отсутствии шаблона в тексте (неудача).

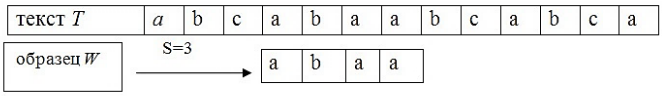

Рис. 1. Успех поиска в тексте T на третьем сдвиге образца W.

Примечание: «смещение» образца – это, конечно, не физический сдвиг в памяти, а приращение значения индексной переменной.

Код для реализации такой идеи будет включать в себя операции сравнения символов из T и W во вложенном цикле (внешний – сдвиг образца до успеха или достижения конца текста T, внутренний – собственно

посимвольное сравнение с начала до конца образца или до первого несоответствия).

Индивидуальный вариант задания: Дан текст, разделенных знаками препинания. Сформировать массив из слов, в которых заданная подстрока размещается с первой позиции.

4.2.2 Код программы с комментариями

Код программы представлен на рисунке 13.

```
bool isPunctuation(char c) {
    return std::ispunct(static_cast<unsigned char>(c));
}

vector<string> findWordsStartingWithSubstring(const string &text, const
string &substring) {
    vector<string> result;
    string word;
    bool substrInWord = false;
    for (char c: text) {
        if (!isPunctuation(c) && c != ' ') {
            word += c;
            if (word == substring)substrInWord = true;
        } else {
            if (substrInWord)
                result.push_back(word);
            word.clear();
            substrInWord = false;
        }
    }
    if (substrInWord)result.push_back(word);
    return result;
}

int main() {
    SetConsoleOutputCP(CP_UTF8);
    string text, substring;
    cout << "Введите текст: " << endl;
    getline(cin, text);
    cout << "Введите начало слова, которое вам необходимо найти: " << endl;
    getline(cin, substring);
    vector<string> words = findWordsStartingWithSubstring(text, substring);
    if (!words.empty()) {
        cout << "Слова тексте, начинающиеся с " << substring << ": " << endl;
        for (const auto &word: words)
            cout << word << endl;
    } else {
        cout << "Слов, начинающихся с " << substring << ", нет в тексте! " <<
endl;
    }
    return 0;
}
```

Рисунок 13 – Листинг кода программы

4.2.3 Результаты тестирования

Программа было успешно протестирована на различных наборах

данных.

4.3 Задание 2

4.3.1 Формулировка задачи

Разработайте приложения в соответствии с заданиями в индивидуальном варианте (п.3).

В отчёте в разделе «Математическая модель решения (описание алгоритма)» разобрать алгоритм поиска на примере. Подсчитать количество сравнений для успешного поиска первого вхождения образца в текст и безуспешного поиска.

Определить функцию (или несколько функций) для реализации алгоритма поиска. Определить предусловие и постусловие.

4.3.2 Код программы с комментариями

Код программы представлен на рисунке 14.

```
const int ALPHABET_SIZE = 256;

void precomputeBadCharacterShift(const string &pattern, vector<int>
&badCharacterShift) {
    int patternLength = pattern.length();
    for (int i = 0; i < ALPHABET_SIZE; i++)
        badCharacterShift[i] = patternLength;
    for (int i = 0; i < patternLength - 1; i++)
        badCharacterShift[static_cast<int>(pattern[i])] = patternLength-1-i;
}

void precomputeGoodSuffixShift(const string &pattern, vector<int>
&goodSuffixShift) {
    int patternLength = pattern.length();
    vector<int> suffixLength(patternLength, 0);

    for (int i = 0; i < patternLength - 1; i++) {
        int j = i;
        int k = 0;
        while (j >= 0 && pattern[j] == pattern[patternLength - 1 - k]) {
            j--;
            k++;
            suffixLength[k] = j + 1;
        }
    }

    for (int i = 0; i < patternLength; i++)
        goodSuffixShift[i] = patternLength;

    for (int i = patternLength - 1; i >= 0; i--)
        if (suffixLength[i] == i + 1)
            for (int j = 0; j < patternLength - 1 - i; j++)
                if (goodSuffixShift[j] == patternLength)
                    goodSuffixShift[j] = patternLength - 1 - i;

    for (int i = 0; i < patternLength - 1; i++)
        goodSuffixShift[patternLength - 1 - suffixLength[i]] = patternLength
- 1 - i;
}

void searchTurboBoyerMoore(const string &text, const string &pattern) {
    int textLength = text.length();
    int patternLength = pattern.length();
    vector<int> badCharacterShift(ALPHABET_SIZE, patternLength);
    vector<int> goodSuffixShift(patternLength, patternLength);

    precomputeBadCharacterShift(pattern, badCharacterShift);
    precomputeGoodSuffixShift(pattern, goodSuffixShift);
    int i = 0;
    while (i <= textLength - patternLength) {
        int j = patternLength - 1;
        while (j >= 0 && pattern[j] == text[i + j]) {
            j--;
        }
        if (j < 0) {
            cout << "Найдено вхождение строки. Индекс: " << i << endl;
            i += goodSuffixShift[0];
        } else
            i += max(badCharacterShift[static_cast<int>(text[i + j])] - j,
goodSuffixShift[j]);
    }
}
```

Рисунок 14 – Листинг кода программы

4.3.3 Результаты тестирования

Программа была успешно протестирована на различных наборах данных.

4.4 Вывод

В ходе работы были освоены приёмы реализации алгоритмов поиска образца в тексте.

5 ПРАКТИЧЕСКАЯ РАБОТА №5

5.1 Цель работы

Освоить структуру данных бинарное дерево поиска и алгоритмы работы с ней.

5.2 Задание

5.2.1 Формулировка задачи

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту.

Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню.

Провести полное тестирование программы на дереве размером n=10 элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Индивидуальный вариант: обратный обход, симметричный обход, найти сумму значений листьев, найти высоту дерева.

5.2.2 Код программы с комментариями

Код программы представлен на рисунках 15-16.

```
struct TreeNode { // Узел бинарного дерева
    int data;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

class BinarySearchTree {
private:
    TreeNode *root; // Корневой узел дерева
public:
    BinarySearchTree() : root(nullptr) {}

    void insert(int value) { // Метод для вставки нового элемента в бинарное
дерево поиска
        root = insertRecursive(root, value);
    }

    TreeNode *insertRecursive(TreeNode *current, int value) {
        // Рекурсивное добавление элемента в дерево с учетом порядка ключей
(меньшие значения слева, большие справа)
        if (current == nullptr)
            return new TreeNode(value);
        if (value < current->data)
            current->left = insertRecursive(current->left, value);
        else if (value > current->data)
            current->right = insertRecursive(current->right, value);
        return current;
    }
    . . .
}
```

Рисунок 15 – Листинг кода добавления узлов в дерево

```
void inorderTraversal(TreeNode *node) { // Симметричный обход дерева
    if (node != nullptr) {
        inorderTraversal(node->left);
        cout << node->data << " ";
        inorderTraversal(node->right);
    }
}

void postorderTraversal(TreeNode *node) { // Обратный обход дерева
    if (node != nullptr) {
        postorderTraversal(node->left);
        postorderTraversal(node->right);
        cout << node->data << " ";
    }
}

int sumLeafValues(TreeNode *node) { // Рекурсивное вычисление суммы значений
листьев дерева
    if (node == nullptr) return 0;
    if (node->left == nullptr && node->right == nullptr) return node->data;
    return sumLeafValues(node->left) + sumLeafValues(node->right);
}

int treeHeight(TreeNode *node) { // Рекурсивное нахождение высоты дерева
    if (node == nullptr) return 0;
    int leftHeight = treeHeight(node->left);
    int rightHeight = treeHeight(node->right);
    return 1 + max(leftHeight, rightHeight);
}
```

Рисунок 16 – Листинг кода алгоритмов из варианта

5.2.3 Результаты тестирования

Программа была успешно протестирована на различных наборах данных.

5.3 Вывод

В ходе работы была освоена структура данных бинарное дерево поиска и алгоритмы работы с ней.

6 ПРАКТИЧЕСКАЯ РАБОТА №6

6.1 Цель работы

Освоить способы представления графа в программе и изучить алгоритмы работы с ним.

6.2 Задание

6.2.1 Формулировка задачи

Составить программу создания графа и реализовать процедуру для работы с графом, определенную индивидуальным вариантом задания.

Самостоятельно выбрать и реализовать способ представления графа в памяти.

В программе предусмотреть ввод с клавиатуры произвольного графа. В вариантах построения остовного дерева также разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

Провести тестовый прогон программы на предложенном в индивидуальном варианте задания графе. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Сделать выводы о проделанной работе, основанные на полученных результатах.

Индивидуальный вариант: нахождение кратчайшего пути методом построения дерева решений.

6.2.2 Код программы с комментариями

Код программы представлен на рисунках 17-18.

```
struct Edge {
    int u, v, weight;
};

class Graph {
private:
    int V, E;
    vector<Edge> edges;
    vector<vector<int>> adjacencyMatrix;
public:
    Graph(int vertices, int edges) {
        V = vertices;
        E = edges;
        adjacencyMatrix.resize(V, vector<int>(E, 0));
    }

    void addEdge(int u, int v, int weight, int edgeIndex) {
        if (u == v) {
            adjacencyMatrix[u - 1][edgeIndex] = 2;
        } else {
            adjacencyMatrix[u - 1][edgeIndex] = 1;
            adjacencyMatrix[v - 1][edgeIndex] = -1;
        }
        edges.push_back({u, v, weight});
    }
    . . .
}
```

Рисунок 17 – Листинг кода добавления узлов граф

```
vector<int> dijkstra(int startVertex) {
    vector<int> distance(V, INT_MAX);
    vector<bool> visited(V, false);
    distance[startVertex - 1] = 0;
    for (int i = 0; i < V - 1; i++) {
        int minDistance = INT_MAX;
        int minVertex = -1;
        for (int v = 0; v < V; v++) {
            if (!visited[v] && distance[v] < minDistance) {
                minDistance = distance[v];
                minVertex = v;
            }
        }
        if (minVertex == -1) continue;
        visited[minVertex] = true;
        for (int j = 0; j < E; j++) {
            if (adjacencyMatrix[minVertex][j] == 1) {
                int neighbor = 0;
                for (int k = 0; k < V; k++) {
                    if (adjacencyMatrix[k][j] == -1) {
                        neighbor = k;
                        break;
                    }
                }
                if (!visited[neighbor] && distance[minVertex] != INT_MAX
                    && distance[minVertex] + edges[j].weight <
distance[neighbor]) distance[neighbor] = distance[minVertex] +
edges[j].weight;
            }
        }
    }
    return distance;
}
```

Рисунок 18 – Листинг кода алгоритма из варианта

6.2.3 Результаты тестирования

Программа была успешно протестирована на различных наборах данных.

6.3 Вывод

В ходе работы были освоены способы представления графа в программе и изучены алгоритмы работы с ним.

7 ПРАКТИЧЕСКАЯ РАБОТА №7

7.1 Цель работы

Освоить кодирование и сжатие данных методами без потерь.

7.2 Задание 1

7.2.1 Формулировка задачи

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.

7.2.2 Код программы с комментариями

Код программы представлен на рисунках 19-20.

```
struct Node {
    char character;
    double probability;
    string code;

    Node(int sym, double prob) : character(sym), probability(prob) {}
};

vector<Node *> nodes;
string inputStroke;
string codeStroke;

bool compareNodesProbabilities(const Node *n1, const Node *n2) {
    return n1->probability > n2->probability;
}

void createShannonFanoCodes(vector<Node *> &nodes, int start, int end) {
    if (start == end)
        return;
    int splitIndex = 0;
    double sumLeft;
    double sumRight;
    double minDifference = numeric_limits<double>::max();
    for (int i = start; i < end; ++i) {
        sumLeft = 0.0;
        sumRight = 0.0;
        for (int j = start; j <= i; ++j)
            sumLeft += nodes[j]->probability;
        for (int j = i + 1; j <= end; ++j)
            sumRight += nodes[j]->probability;
        double difference = abs(sumLeft - sumRight);
        if (difference == 0) {
            splitIndex = i;
            break;
        }
        if (difference < minDifference) {
            minDifference = difference;
            splitIndex = i;
        }
    }
    for (int i = start; i <= splitIndex; ++i)
        nodes[i]->code += "0";
    for (int i = splitIndex + 1; i <= end; ++i)
        nodes[i]->code += "1";
    createShannonFanoCodes(nodes, start, splitIndex);
    createShannonFanoCodes(nodes, splitIndex + 1, end);
}
```

Рисунок 19 – Листинг генерации кодов символов

```
void encodeShannonFano() {
    cout << "Enter a stroke:" << endl;
    getline(cin, inputStroke);

    map<char, int> characters;

    for (char ch: inputStroke)
        characters[ch] = characters[ch] + 1;

    for (const auto &pair: characters) {
        double b = double(pair.second) / inputStroke.size();
        nodes.push_back(new Node(pair.first, b));
        cout << pair.first << " : " << b << endl;
    }

    sort(nodes.begin(), nodes.end(), compareNodesProbabilities);

    createShannonFanoCodes(nodes, 0, nodes.size() - 1);

    for (char character: inputStroke) {
        for (Node *node: nodes) {
            if (node->character == character)
                codeStroke += node->code + " ";
        }
    }

    // Размер исходной строки и закодированной строки в байтах
    size_t inputSize = inputStroke.size();
    size_t encodedSize = codeStroke.size() / 8;  // Поскольку код Хаффмана
используует биты, делим на 8 для байтов

    // Вычисление коэффициента сжатия
    double compressionRatio = static_cast<double>(inputSize) /
static_cast<double>(encodedSize);

    cout << "Result: " << codeStroke << endl;
    cout << "Koef: " << compressionRatio << endl;
}

void decodeShannonFano() {
    string decodedResult;
    string currentCode;

    for (char c: codeStroke) {
        if (c == ' ')
            continue;

        currentCode += c;

        for (Node *node: nodes) {
            if (currentCode == node->code) {
                decodedResult += node->character;
                currentCode = "";
                break;
            }
        }
    }

    std::cout << "Decoded result: " << decodedResult << std::endl;
}
```

Рисунок 20 – Листинг кодирования и декодирования

7.3 Задание 2

7.3.1 Формулировка задачи

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.

7.3.2 Код программы с комментариями

Код программы представлен на рисунках 21-22.

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

    HuffmanNode(char data, int frequency) : data(data), frequency(frequency),
left(nullptr), right(nullptr) {}
};

// Функция для сравнения узлов при использовании очереди с приоритетом
struct CompareNodes {
    bool operator()(HuffmanNode *left, HuffmanNode *right) {
        return left->frequency > right->frequency;
    }
};

// Функция для построения дерева Хаффмана и возврата корня дерева
HuffmanNode *buildHuffmanTree(const string &text) {
    unordered_map<char, int> frequencyMap;
    for (char c: text)
        frequencyMap[c]++;

    priority_queue<HuffmanNode *, vector<HuffmanNode *>, CompareNodes>
minHeap;
    for (const auto &pair: frequencyMap)
        minHeap.push(new HuffmanNode(pair.first, pair.second));

    while (minHeap.size() > 1) {
        HuffmanNode *left = minHeap.top();
        minHeap.pop();
        HuffmanNode *right = minHeap.top();
        minHeap.pop();

        int combinedFrequency = left->frequency + right->frequency;
        auto *newNode = new HuffmanNode('\0', combinedFrequency);
        newNode->left = left;
        newNode->right = right;
        minHeap.push(newNode);
    }

    for (const auto &pair: frequencyMap)
        cout << pair.first << " : " << pair.second << endl;

    return minHeap.top();
}
```

Рисунок 21 – Листинг построения дерева Хаффмана

```
// Рекурсивная функция для создания кодов Хаффмана
void generateHuffmanCodes(HuffmanNode *root, const string &currentCode,
unordered_map<char, string> &codes) {
    if (!root)
        return;

    if (root->data != '\0')
        codes[root->data] = currentCode;

    generateHuffmanCodes(root->left, currentCode + "0", codes);
    generateHuffmanCodes(root->right, currentCode + "1", codes);
}

// Функция для кодирования строки с использованием кодов Хаффмана
string encodeHuffman(const string &text) {
    HuffmanNode *root = buildHuffmanTree(text);
    unordered_map<char, string> codes;
    generateHuffmanCodes(root, "", codes);

    string encodedText;
    for (char c: text)
        encodedText += codes[c];

    return encodedText;
}
```

Рисунок 22 – Листинг кодирования и декодирования

7.3.3 Результаты тестирования

Программа была успешно протестирована на различных наборах данных. Алгоритм Хаффмана показал лучший коэффициент сжатия по сравнению с zip-архивом.

7.4 Вывод

В ходе работы было освоено кодирование и сжатие данных методами без потерь.

8 ПРАКТИЧЕСКАЯ РАБОТА №8

8.1 Цель работы

Изучить работу различных методов программирования для решения задач.

8.2 Задание

8.2.1 Формулировка задачи

Разработать алгоритм решения задачи с применением метода, указанного в варианте и реализовать программу.

Индивидуальный вариант: Вычисление значения определенного интеграла с применением численных методов. «Вычислить значение определенного интеграла с заданной точностью определенным методом трапеции. Реализовать следующие подзадачи в виде функций:

- вычисление значения подинтегральной функции в заданной точке x;

- вычисление значения интеграла установленным методом на заданном отрезке интегрирования при n разбиениях;

- вычисление интеграла установленным методом с заданной точностью.

8.2.2 Код программы с комментариями

Код программы представлен на рисунке 23.

```
// Функция для вычисления значения подинтегральной функции в точке x
double f(double x) {
    return x * x;
}

// Функция для вычисления интеграла методом трапеции с n разбиениями на
отрезке [a, b]
double trapezoidalMethod(double a, double b, int n) {
    double h = (b - a) / n;
    double result = (f(a) + f(b)) / 2;

    for (int i = 0; i < n; i++) {
        double x = a + i * h;
        result += f(x);
    }
    return result * h;
}

// Функция для вычисления интеграла с заданной точностью
double computeIntegralWithPrecision(double a, double b, double precision) {
    int n = 2;
    double prevResult;
    double result = trapezoidalMethod(a, b, n);
    double currentPrecision = std::numeric_limits<double>::max();

    while (currentPrecision > precision) {
        n *= 2;
        prevResult = result;
        result = trapezoidalMethod(a, b, n);
        currentPrecision = abs(result - prevResult);
    }
    return result;
}
```

Рисунок 23 – Листинг кода добавления узлов граф

8.2.3 Результаты тестирования

Программа была успешно протестирована на различных наборах данных.

8.3 Вывод

В ходе работы была изучена работа различных методов программирования для решения задач.