****

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 5.1

**Тема: РАБОТА С ДАННЫМИ ИЗ ФАЙЛА**

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Королихин В.Н.

группа ИКБО-21-23

**Москва 2024**

**Цель работы:** освоить приёмы работы с битовым представлением беззнаковых целых чисел, реализовать эффективный алгоритм внешней сортировки на основе битового массива.

ЗАДАНИЕ 1

**Формулировка задачи**

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

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.

**Математическая модель решения (описание алгоритма)**

1.а.1) На вход подаётся число х,

1.а.2) Создается 8-битная маска, её 5ый бит устанавливается на 0, остальные на 1,

1.а.3) Маска применяется к числу x при помощи поразрядного И.

1.б.1) На вход подаётся число х,

1.б.2) Создается 8-битная маска, её 7ой бит устанавливается на 1, остальные на 0,

1.б.3) Маска применяется к числу x при помощи поразрядного ИЛИ.

1.в.1) На вход подаётся число х,

1.в.2) Создается 32-битная маска, её старший разряд устанавливается на 1,

1.в.3) Осуществляется цикл, для i = 1..32,

1.в.3) При помощи комбинации поразрядных операций выводится одно число – значение бита на текущей позиции i,

1.в.4) Маска сдвигается на один разряд вправо.

**Код программы**

void task1a() {

// 1.а \\

unsigned char x, maska = 1;

x = 255; // 11111111

cout << "Начальное значение х: " << (int)x;

x = x & (~(maska << 4));

cout << ", х после применения маски: " << (int)x << endl;

x = 132; // 10000100

cout << "Начальное значение х: " << (int)x;

x = x & (~(maska << 4));

cout << ", х после применения маски: " << (int)x << endl;

}

void task1b() {

// 1.б \\

unsigned char x, maska = 1;

x = 255; // 11111111

cout << "Начальное значение х: " << (int)x;

x = x | (maska << 6);

cout << ", х после применения маски: " << (int)x << endl;

x = 132; // 10000100

cout << "Начальное значение х: " << (int)x;

x = x | (maska << 6);

cout << ", х после применения маски: " << (int)x << endl;

}

void task1c() {

// 1.c \\

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

unsigned int x = 25;

const int n = sizeof(int) \* 8; // 32 - количество разрядов в числе типа int

unsigned int maska = (1 << (n - 1)); // 1 в старшем бите 32-разрядной сетки

cout << "Начальный вид маски: " << bitset<32>(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;

system("pause");

}

**Результаты тестирования**

Результаты тестирования представлены на рисунках 1-3



Рисунок 1 – Тестирование программы 1.а)



Рисунок 2 - Тестирование программы 1.б)



Рисунок 3 – Тестирование программы 1.в)

ЗАДАНИЕ 2

**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. 1 Под значение типа bool выделяется 1 байт (8 бит) памяти, но в классе vector происходит оптимизация, в результате которой одно логическое значение занимает 1 бит. СиАОД – 5.1 4 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.

**Математическая модель решения (описание алгоритма)**

2.а.1) Создаётся 8-разрядный битсет bitSet,

2.а.2) Биты этого битсета, которые соответствуют значениям, присутствующем во входном списке, устанавливаются на 1.

2.а.3) Создаётся пустой массив sortedNums

2.а.4) Битсет поразрядно проходится, если очередной бит равен 1, то индекс этого бита добавляется в sortedNums.

2.а.5) возвращается sortedNums

2.б.1) Создаётся 64-разрядный битсет bitSet, основанный на числе типа Unsigned Long Long.

2.б.2) Биты этого битсета, которые соответствуют значениям, присутствующем во входном списке, устанавливаются на 1.

2.б.3) Создаётся пустой массив sortedNums

2.б.4) Битсет поразрядно проходится, если очередной бит равен 1, то индекс этого бита добавляется в sortedNums.

2.б.5) возвращается sortedNums

2.в.1) Создаётся 64-разрядный битсет bitSet, основанный 8-и числах типа unsigned char, хранящие 8 бит каждый.

2.в.2) Биты этого битсета, которые соответствуют значениям, присутствующем во входном списке, устанавливаются на 1.

2.в.3) Создаётся пустой массив sortedNums

2.в.4) Битсет поразрядно проходится, если очередной бит равен 1, то индекс этого бита добавляется в sortedNums.

2.в.5) возвращается sortedNums

**Код программы**

vector<int> bitSort8(vector<int>& nums) {

unsigned char bitSet = 0;

for (auto x : nums) {

bitSet = bitSet | (1 << x);

}

vector<int> sortedNums;

cout << "BitSet = ";

for (int i = 0; i < 8; i++) {

if (bitSet & (1 << i)) sortedNums.push\_back(i);

cout << ((bitSet & (1 << i)) >> (i));

}

cout << endl;

return sortedNums;

}

vector<int> bitSort64ULL(vector<int>& nums) {

unsigned long long bitSet = 0;

for (auto x : nums) {

bitSet = bitSet | (1ULL << x);

}

vector<int> sortedNums;

cout << "BitSet = ";

for (int i = 0; i < 64; i++) {

if (bitSet & (1ULL << i)) sortedNums.push\_back(i);

cout << ((bitSet & (1ULL << i)) >> (i));

}

cout << endl;

return sortedNums;

}

vector<int> bitSort64UC(vector<int>& nums) {

unsigned char\* bitSet = new unsigned char[8] {};

for (auto x : nums) {

bitSet[7 - x / 8] = bitSet[7 - x / 8] | (1 << (x % 8));

}

vector<int> sortedNums;

cout << "BitSet = | ";

for (int i = 0; i < 64; i++) {

if (bitSet[7 - i / 8] & (1 << (i % 8))) sortedNums.push\_back(i);

cout << ((bitSet[7 - i / 8] & (1 << (i % 8))) >> (i % 8));

if ((i + 1) % 8 == 0) cout << " | ";

}

cout << endl;

return sortedNums;

}**Результаты тестирования**

Результаты тестирования представлены на рисунках 4 - 6

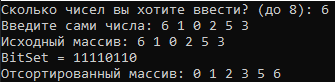


Рисунок 4 – Тестирование программы 2.а)

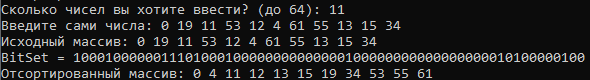


Рисунок 5 - Тестирование программы 2.б)

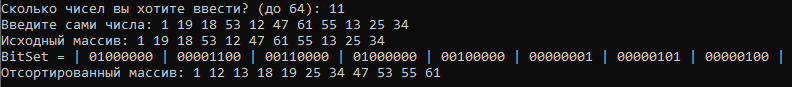


Рисунок 6 – Тестирование программы 2.в)

ЗАДАНИЕ 3

**Формулировка задачи**

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.

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

**Математическая модель решения (описание алгоритма)**

**Алгоритм выполнения**

**1. Инициализация параметров:**

1.1) Задать максимальное число для сортировки (MAX\_NUMBER) и рассчитать размер массива битов (ARR\_SIZE).

1.2) Создать массив битов (bitSet) и заполнить его нулями.

1.3) Вычислить объем памяти, занимаемой массивом битов, и сохранить значение в переменную memoryUsed.

**2. Открытие исходного файла для чтения:**

2.1) Попытаться открыть файл sourceFileName.

2.2) Если файл открыть не удалось, вывести сообщение об ошибке и завершить выполнение.

**3. Считывание чисел и установка битов:**

3.а) Для каждого числа, считанного из файла:

3.а.1) Определить, в каком байте массива находится число.

3.а.2) Определить позицию числа внутри этого байта.

3.а.3) Установить соответствующий бит, чтобы пометить, что число присутствует.

3.б) Закрыть исходный файл после завершения чтения.

**4. Открытие файла для записи:**

4.1) Попытаться открыть файл destinationFileName.

4.2) Если файл открыть не удалось, вывести сообщение об ошибке и завершить выполнение.

**5. Сортировка чисел и запись результата:**

5.а) Зафиксировать время начала сортировки.

5.б) Для каждого байта в массиве битов:

5.б.1) Проверить каждый бит в текущем байте.

5.б.2) Если бит установлен, вычислить число, соответствующее этому биту.

5.б.3) Записать это число в файл.

5.в) Зафиксировать время окончания сортировки.

5.г) Вычислить затраченное время и сохранить его в переменную timeSpent.

**6. Закрытие файла для записи:**

6.1) Закрыть файл после завершения записи данных.

**7. Завершение работы функции:**

7.1) Вернуться к вызывающему коду, предоставив результаты работы.

**Код программы**

void bitSortFile(const string& sourceFileName, const string& destinationFileName, double& timeSpent, int& memoryUsed) {

const int MAX\_NUMBER = 10'000'000;

const int ARR\_SIZE = (MAX\_NUMBER + 1) / 8 + 1;

vector<unsigned char> bitSet(ARR\_SIZE, 0);

memoryUsed = bitSet.capacity() \* sizeof(unsigned char);

ifstream inFile(sourceFileName);

if (!inFile) {

cerr << "Не удалось открыть файл для чтения: " << sourceFileName << endl;

return;

}

int num;

while (inFile >> num) {

bitSet[ARR\_SIZE - num / 8 - 1] = bitSet[ARR\_SIZE - num / 8 - 1] | (1 << (num % 8));

}

inFile.close();

ofstream outFile(destinationFileName);

if (!outFile) {

cerr << "Не удалось открыть файл для записи: " << destinationFileName << endl;

return;

}

auto start = chrono::high\_resolution\_clock::now();

for (int charByte = 0; charByte < ARR\_SIZE; charByte++) {

for (int charBit = 0; charBit < 8; charBit++) {

if (bitSet[ARR\_SIZE - charByte - 1] & (1 << (charBit))) {

outFile << charByte \* 8 + charBit << endl;

}

}

}

auto end = chrono::high\_resolution\_clock::now();

chrono::duration<double> elapsed = end - start;

timeSpent = elapsed.count();

outFile.close();

}

void generateRandomNumbersToFile(const string& filename, int amount) {

ofstream outFile(filename);

if (!outFile) {

cerr << "Не удалось открыть файл для записи: " << filename << endl;

return;

}

vector<int> numbers(amount);

for (int i = 0; i < amount; i++) {

numbers[i] = i;

}

random\_device rd;

mt19937 gen(rd());

shuffle(numbers.begin(), numbers.end(), gen);

for (int i = 0; i < amount; ++i) {

outFile << numbers[i] << endl;

}

outFile.close();

}**Результаты тестирования**

Результаты тестирования представлены на рисунках 7-9



Рисунок 7 – Тестирование программы 3

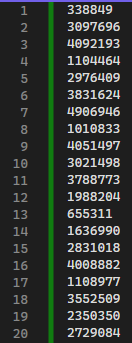


Рисунок 8 – Файл с числами в случайном порядке

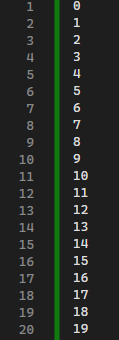


Рисунок 6 – Файл с числами в отсортированном порядке

ВЫВОДЫ

В ходе работы были изучены битовые операции, маски, сортировки при помощи битсетов, сортировки внешних файлов и многое другое. Были реализованы на практике все требуемые алгоритмы

ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

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

2. Документация по языку С++ [Электронный ресурс]. URL: https://docs.microsoft.com/ruru/cpp/cpp/ (дата обращения 01.09.2021).

3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=4020 (дата обращения 01.09.2021).