МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №5.1

Тема:

«Битовые операции. Сортировка числового файла с помощью битового массива»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Платонов Л.А

Группа: ИКБО-13-23

Москва – 2024

1. Цель работы:

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

2. Ход работы:

2.1. Задача 1.

2.1.1. Постановка задачи 1:

1.а. Установить 5-й бит произвольного целого числа в 0 и посмотреть, что получится в результате.

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

1.в. Реализовать код листинга 1, объяснить выводимый программой результат.

2.1.2. Математическая модель решения

1.а. Чтобы установить 5-й бит числа в 0, нужно воспользоваться побитовой операцией «и» с маской, в которой все биты равны 1, кроме 5-го, который равен 0. Например, маска для 5-го бита выглядит как ~(1 << 4).

1.б. Чтобы установить 7-й бит числа в 1, нужно использовать побитовую операцию «или» с маской, где только 7-й бит равен 1. Маска для 7-го бита — это 1 << 6.

1.в. Программа создает маску, в которой только самый старший бит равен 1, а остальные 0. Затем она выводит это число в виде битовой строки. В цикле проверяется каждый бит числа, начиная с самого старшего, путем побитового «и» с текущей маской, после чего маска сдвигается вправо.

2.1.3. Код программы:

Листинг главной функции:

```
int main() {
    cout << "Choose number of exercise (1, 2, 3): ";
    char letter;
    int n;
    cin >> n;
    switch (n) {
        case 1:
            cout << "Choose a, b or c: ";
            cin >> letter;
            if (letter == 'a')
                first_a();
            else if (letter == 'b')
                first_b();
            else
                first_v();
            break;
        case 2:
            cout << "Choose a, b or c: ";
            cin >> letter;
            if (letter == 'a')
                second_a();
            else if (letter == 'b')
                second_b();
            else
                second_v();
            break;
        case 3:
            //create_file();
            third();
            break;
        default:
            break;
    }
    return 0;
```

Листинг кода 1.a.:

```
inline void first_a() {
    unsigned char x = 255;
    unsigned char maska = 1;
    x = x & (~(maska << 4));

    cout << (int)x;
}
```

Листинг кода 1.б.:

```
inline void first_b() {
    unsigned char x = 57;
    unsigned char maska = 1;
    x = x | (maska << 6);

    cout << (int)x;
}
```

Листинг кода 1.в.:

```
void first_v() {
    unsigned int x = 25;
    const int n = sizeof(int) * 8;
    unsigned maska = (1 << (n - 1));
    cout << "Mask: " << bitset<n>(maska) << endl;
    cout << "Result: ";
    for (int i = 1; i <= n; i++) {

        cout << (((x & maska)) >> (n - i));
        maska = maska >> 1;
    }

    cout << endl;
    system("pause");
}
```

2.1.4. Результаты тестирования

Тестирование кодов для задач 1.а., 1.б., 1.в. соответственно:

```
Choose number of exercise (1, 2, 3): 1
Choose a, b or c: a
239
```

```
Choose number of exercise (1, 2, 3): 1
Choose a, b or c: b
121
```

```
Choose number of exercise (1, 2, 3): 1
Choose a, b or c: c
Mask: 10000000000000000000000000000000
Result: 00000000000000000000000000011001
```

2.2. Задача 2.

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.

2.2.1. Постановка задачи 2:

2.а. Реализовать 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. 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.

2.б. Адаптировать 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. Исправить 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.

2.2.2. Математическая модель решения

2.а. Для сортировки до 8 чисел, каждое из которых находится в диапазоне от 0 до 7, создается переменная типа unsigned char, где каждый бит представляет одно число. Инициализируется переменная unsigned char = 0. Для каждого числа в наборе устанавливается соответствующий бит в 1 с помощью побитовой операции OR. Затем программа последовательно проходит по битам переменной и выводит индексы единичных битов, что дает отсортированный набор чисел.

2.б. Для работы с числовым диапазоном от 0 до 63 используется тип unsigned long long, где каждый бит отвечает за одно число. Инициализируется переменная unsigned long long = 0. Для каждого числа в наборе устанавливается соответствующий бит в 1. После этого программа проходит по всем 64 битам и выводит индексы единичных битов, что обеспечивает сортировку.

2.в. Вместо одного большого числа типа unsigned long long используется массив из 8 переменных типа unsigned char, каждая из которых отвечает за 8 бит. Инициализируется массив из 8 элементов unsigned char. Для каждого числа вычисляется индекс элемента массива и позиция бита в этом элементе. После установки соответствующего бита программа выводит индексы единичных битов, что позволяет получить отсортированный набор чисел.

2.2.3. Код программы

Листинг кода 2.а.:

```
void second_a() {
    cout << "Enter size from 1 to 8" << endl;
    int n, num;
    unsigned char maska, bit_mas = 0;
    cin >> n;
    int* mas = new int[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter an element from 0 to 7: ";
        cin >> num;
        mas[i] = num;
    }
    for (int i = 0; i < n; i++) {
        maska = 1;
        maska = (maska << (mas[i]));
        bit_mas = bit_mas | maska;
    }
    for (int i = 0; i < 8; i++) {
        if ((bit_mas & 1) == 1)
            cout << i << " ";
        bit_mas >>= 1;
    }
    cout << endl;
}
```

Листинг кода 2.б.:

```
void second_b() {
    int n, num;
    unsigned long long int maska, bit_mas = 0;
    cout << "Enter size from 0 to 64: ";
    cin >> n;
    int* mas = new int[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter an element from 0 to 64: ";
        cin >> num;
        mas[i] = num;
    }
    for (int i = 0; i < n; i++) {
        maska = 1;
        maska = (maska << (mas[i]));
        bit_mas = bit_mas | maska;
    }
    for (int i = 0; i < sizeof(bit_mas) * 8; i++) {
        if ((bit_mas & 1) == 1)
            cout << i << " ";
        bit_mas >>= 1;
    }
    cout << endl;
}
```

Листинг кода 2.в.:

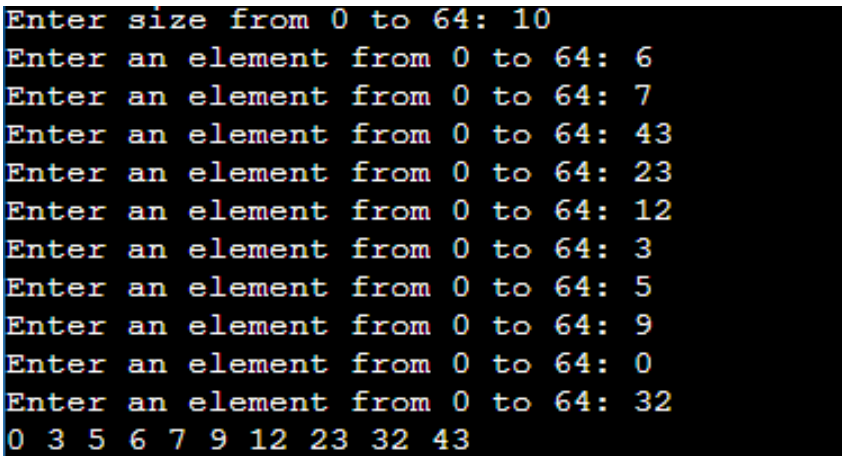
```
void second_v() {
    int n, ch;
    unsigned char maska = 1;
    cout << "Enter size from 0 to 64: ";
    cin >> n;
    vector <unsigned char> mas(8);
    for (int i = 0; i < n; i++) {
        cout << "Enter an element from 0 to 64: ";
        cin >> ch;
        mas[ch / 8] = mas[ch / 8] | (maska << ch % 8);
    }
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if ((mas[i] & 1) == 1)
                cout << 8 * i + j << " ";
            mas[i] >>= 1;
        }
    }
}
```

2.2.4. Результаты тестирования

Тестирование кодов для задач 2.а., 2.б., 2.в. соответственно:

```
Choose number of exercise (1, 2, 3): 2
Choose a, b or c: a
Enter size from 1 to 8
5
Enter an element from 0 to 7: 6
Enter an element from 0 to 7: 1
Enter an element from 0 to 7: 2
Enter an element from 0 to 7: 3
Enter an element from 0 to 7: 4
1 2 3 4 6
```

```
Choose number of exercise (1, 2, 3): 2
Choose a, b or c: b
Enter size from 0 to 64: 7
Enter an element from 0 to 64: 1
Enter an element from 0 to 64: 63
Enter an element from 0 to 64: 12
Enter an element from 0 to 64: 54
Enter an element from 0 to 64: 12
Enter an element from 0 to 64: 53
Enter an element from 0 to 64: 2
1 2 12 53 54 63
```



2.3. Задача 3.

На практике может возникнуть задача внешней сортировки, т.е. упорядочения значений, расположенных во внешней памяти компьютера, размер которых превышает допустимый объём ОЗУ (например, 1 МБ стека, выделяемый по умолчанию программе операционной системой).

Возможный способ – это алгоритм внешней сортировки слиянием, рассмотренный в одной из предыдущих практических работ. Считывание исходного файла при этом происходит один раз, но в процессе сортировки создаются и многократно считываются вспомогательные файлы, что существенно снижает быстродействие.

Второй возможный приём – считывание входного файла порциями, размер каждой из которых не превышает лимит ОЗУ. Результат записывается в выходной файл за один раз, при этом не используются вспомогательные файлы. Программа будет работать быстрее, но всё-таки есть алгоритм, существенно превосходящий перечисленные.

Реализовать высокоэффективную сортировку большого объёма числовых данных в файле можно на идее битового массива. Достаточно один раз считать содержимое файла, заполнив при этом в памяти ЭВМ битовый массив и на его основе быстро сформировать содержимое выходного файла в уже отсортированном виде.

2.3.1. Постановка задачи 3.

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.

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

2.3.2. Математическая модель решения.

3.а. Для сортировки чисел в файле считываем данные и используем битовый массив, где каждый бит соответствует наличию числа в наборе. Вводится битовый массив размером 1 МБ (8 миллионов бит), который позволяет хранить числа до 8 миллионов. Числа из входного файла отмечаются установкой соответствующего бита в 1. После этого программа проходит по битовому массиву и записывает числа в отсортированном порядке в выходной файл. Время работы программы измеряется с помощью встроенного таймера.

3.б. Для определения объема оперативной памяти, занимаемой битовым массивом, вычисляем количество бит, необходимое для хранения чисел (1 бит на число). Поскольку массив ограничен 1 МБ, это позволяет хранить до 8 миллионов чисел (8 * 1024 * 1024 = 8,388,608 битов). Программа выводит размер памяти, занимаемой массивом, в байтах.

2.3.3. Код программы.

Листинг кода:

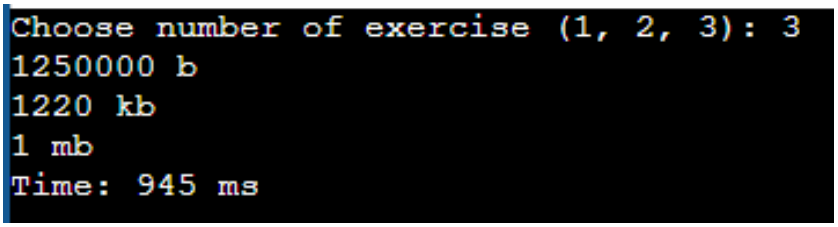
```
void third() {
    const int n = 10000000/8;
    int start = clock();
    unsigned char maska = 1;
    vector <unsigned char> bit_mas(n);
    string path = "Test.txt";
    ifstream fin;
    fin.open(path);
    if (!fin.is_open())
        cout << "Open error" << endl;
    else {

        int ch;

        while (!fin.eof()) {
            fin >> ch;
            bit_mas[ch / 8] = bit_mas[ch / 8] | (maska << ch % 8);
        }
    }
    fin.close();
    int stop = clock();
    ofstream fout;
    fout.open(path);
    if (!fout.is_open())
        cout << "Open error" << endl;
    else {
        for (int i = 0; i < (n); i++) {
            for (int j = 0; j < 8; j++) {
                if ((bit_mas[i] & 1) == 1)
                    fout << 8 * i + j << endl;
                bit_mas[i] >>= 1;
            }
        }
    }
```

```
    }
    fout.close();
    int resTime = stop - start;
    bit_mas.shrink_to_fit();
    cout << bit_mas.capacity() << " b" << endl;
    cout << bit_mas.capacity() / 1024 << " kb" << endl;
    cout << bit_mas.capacity() / (1024 * 1024) << " mb" << endl;
    cout << "Time: " << resTime << " ms \n";
}

void create_file() {

    const long n = 1000000;
    const long n_max = 9999999;
    const int len = n_max - n + 1;
    long* array = new long [len];

    for (long i = 0; i < len; i++) {
        array[i] = n + i;
    }

    srand(time(NULL));

    for (long i = 0; i < len; i++)
        swap(array[i], array[rand() % len]);

    ofstream fout;
    fout.open("Test.txt");


    if (!fout.is_open())
        cout << "Open error" << endl;
    else {
        for (int i = 0; i < len; i++)
```

```
        if (!fout.is_open())
            cout << "Open error" << endl;
        else {
            for (int i = 0; i < len; i++)
                fout << array[i] << endl;
        }
        fout.close();
    }
```

2.3.4. Результаты тестирования

Пример тестирования кода для данной задачи:



3. Вывод:

В работе рассмотрены методы побитовой обработки данных и их применение для сортировки чисел. Реализованы алгоритмы сортировки с использованием битовых массивов, что обеспечивает эффективное использование памяти. Применение битовых операций позволило достичь высокой производительности при сортировке больших объемов данных.