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

Тема: РАБОТА С ДАННЫМИ ИЗ ФАЙЛА

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент: Павлов Никита
группа ИКБО-50-23

Москва 2024

СОДЕРЖАНИЕ

1. БИТОВЫЕ ОПЕРАЦИИ. СОРТИРОВКА ЧИСЛОВОГО ФАЙЛА С ПОМОЩЬЮ БИТОВОГО МАССИВА 4
1.1 ЦЕЛЬ РАБОТЫ 4
1.2 ЗАДАНИЕ 1А 5
1.2.1 Формулировка задачи 5
1.2.2 Математическая модель решения (описание алгоритма) 5
1.2.3 Код программы 5
1.2.4 Результаты тестирования 5
1.3 ЗАДАНИЕ 1Б 6
1.3.1 Формулировка задачи 6
1.3.2 Математическая модель решения (описание алгоритма) 6
1.3.3 Код программы 6
1.3.4 Результаты тестирования 6
1.4 ЗАДАНИЕ 1В 7
1.4.1 Формулировка задачи 7
1.4.2 Математическая модель решения (описание алгоритма) 7
1.4.3 Код программы 7
1.4.4 Результаты тестирования 7
1.5 ЗАДАНИЕ 2А 8
1.5.1 Формулировка задачи 8
1.5.2 Математическая модель решения (описание алгоритма) 8
1.5.3 Код программы 8
1.5.4 Результаты тестирования 8
1.6 ЗАДАНИЕ 2Б 9
1.6.1 Формулировка задачи 9
1.6.2 Математическая модель решения (описание алгоритма) 9
1.6.3 Код программы 9
1.6.4 Результаты тестирования 9
1.7 ЗАДАНИЕ 2В 10
1.7.1 Формулировка задачи 10

1.7.2 Математическая модель решения (описание алгоритма) 10
1.7.3 Код программы 11
1.7.4 Результаты тестирования 11
1.8 ЗАДАНИЕ 3 12
1.8.1 Формулировка задачи 12
1.8.2 Математическая модель решения (описание алгоритма) 12
1.8.3 Код программы 13
1.8.4 Результаты тестирования 13
1.10 ВЫВОДЫ 14
2. АЛГОРИТМЫ ПОИСКА В ТАБЛИЦЕ ПРИ РАБОТЕ С ДАННЫМИ ИЗ ФАЙЛА 15
2.1 ЦЕЛЬ РАБОТЫ 15
2.2 ЗАДАНИЕ 1 16
2.2.1 Постановка задачи 16
2.2.2 Описание подхода к решению 16
2.2.3 Код программы 16
2.2.4 Результаты тестирования 17
2.3 ЗАДАНИЕ 2 18
2.3.1 Постановка задачи 18
2.3.2 Описание подхода к решению 18
2.3.3 Код программы 19
2.3.4 Результаты тестирования 19
2.4 ЗАДАНИЕ 3 20
2.4.1 Постановка задачи 20
2.4.2 Описание подхода к решению 20
2.4.3 Код программы 22
2.4.4 Результаты тестирования 23
2.5 ВЫВОДЫ 24
СПИСОК ИНФОРМАЦТОННЫХ ИСТОЧНИКОВ 25

1. БИТОВЫЕ ОПЕРАЦИИ. СОРТИРОВКА ЧИСЛОВОГО ФАЙЛА С ПОМОЩЬЮ БИТОВОГО МАССИВА

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 1А

1.2.1 Формулировка задачи

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

Реализовать вышеприведённый пример, проверить правильность результата в том числе и на других значениях x

1.2.2 Математическая модель решения (описание алгоритма)

1. Объявление переменные x, maska_1 типа unsigned char и инициализация значениями 255 и 1 соответственно.

2. Увеличение числа двоичных разрядов переменной maska_1 на 4, заполняя нолями. Инвертирование и логическое перемножение с переменной x.

1.2.3 Код программы

```
unsigned int x = 255, maska_1 = 1;
cout << "Исходное число: " << x << endl;
x &= (~(maska_1 << 4));//пятый разряд в 0
cout << "После преобразования: " << x << endl;
```

Рисунок 1 – Код программы

1.2.4 Результаты тестирования

```
Исходное число: 255
После преобразования: 239
```

```
Исходное число: 200
После преобразования: 200
```

Рисунок 2 – Результаты тестирования

1.3 ЗАДАНИЕ 1Б

1.3.1 Формулировка задачи

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

1.3.2 Математическая модель решения (описание алгоритма)

1. Объявление переменные x, maska_1 типа unsigned char и инициализация значениями 5 и 1 соответственно.

2. Увеличение числа двоичных разрядов переменной maska_1 на 6, заполняя нолями. Логическое сложение с переменной x.

1.3.3 Код программы

```
x = 5;
maska_1 = 1;
cout << "Исходное число: " << x << endl;
x |= (maska_1 << 6);//седьмой разряд в 1
cout << "После преобразования: " << x << endl << endl;
```

Рисунок 3 – Код программы

1.3.4 Результаты тестирования

```
Исходное число: 5
После преобразования: 69

Исходное число: 11
После преобразования: 75
```

Рисунок 4 – Результаты тестирования

1.4 ЗАДАНИЕ 1В

1.4.1 Формулировка задачи

Реализуйте код листинга 1, объясните выводимый программой результат

1.4.2 Математическая модель решения (описание алгоритма)

1. Объявление переменные x, maska и константы n, которая равняется числу бит (разрядов) в int. Переменная maska дополняется до n разрядов (10..0)
2. Инициализация переменной i единицей. Пока значение i меньше n:

2.1.Поразрядное логическое умножение переменных maska и x, начиная со старшего разряда. Извлечение результата для текущего разряда.

2.2.Уменьшение числа разрядов переменной maska на 1

2.3.Увеличение i на 1

1.4.3 Код программы

```
SetConsoleCP(1251);
SetConsoleOutputCP(1251);
x = 25;
const int n = sizeof(int) * 8;
unsigned maska = (1 << n - 1);
cout << "Начальный вид маски: " << bitset<n>(maska) << endl;
cout << "Результат: ";
for (int i = 1; i <= n; i++) {
    cout << ((x & maska) >> (n - i));
    maska = maska >> 1;
}
cout << endl;
```

Рисунок 5 – Код программы

1.4.4 Результаты тестирования

```
Начальный вид маски: 10000000000000000000000000000000
Результат: 00000000000000000000000000011001
```

Рисунок 6 – Результаты тестирования

1.5 ЗАДАНИЕ 2А

1.5.1 Формулировка задачи

Реализовать 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. Проверить работу программы.

1.5.2 Математическая модель решения (описание алгоритма)

1. Ввод определённого пользователем количества чисел, меньших 8. Заполнение единицей разряда двоичной последовательности, номер которого равен значению вводимого числа.

2. Инициализация переменной i значением 7. Пока значение i больше либо равно 0:

2.1.Если i-тый разряд равен 1, вывод значения 7-i на экран

2.2.Уменьшение i на 1

1.5.3 Код программы

```
unsigned char bitmas = 0;
cout << "Введите до 8 чисел (значения от 0 до 7, 8 - завершение ввода): ";
cin >> x;
while (x != 8) {
    bitmas |= (1 << (7 - x));
    cin >> x;
}
cout << "Отсортированны массив:";
bitset<8> sort = bitmas;
for (int i = 7; i >= 0; i--) {
    if (sort[i] == 1) {
        cout << " " << 7 - i;
    }
}
```

Рисунок 7 – Код программы

1.5.4 Результаты тестирования

```
Введите до 8 чисел (значения от 0 до 7, 8 - завершение ввода): 4 6 1 0 3 8
Отсортированны массив: 0 1 3 4 6
```

Рисунок 8 – Результаты тестирования

1.6 ЗАДАНИЕ 2Б

1.6.1 Формулировка задачи

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.

1.6.2 Математическая модель решения (описание алгоритма)

1. Ввод определённого пользователем количества чисел, меньших 64. Заполнение единицей разряда двоичной последовательности, номер которого равен значению вводимого числа.

2. Инициализация переменной i значением 63. Пока значение i больше либо равно 0:

2.1.Если i-тый разряд равен 1, вывод значения 63-i на экран

2.2.Уменьшение i на 1

1.6.3 Код программы

```
unsigned long long mas = 0;
cout << "\nВведите до 64 чисел (значения от 0 до 63, 64 - завершение ввода): ";
cin >> x;
while (x != 64) {
    mas |= (static_cast<unsigned long long>(1) << (63 - x));
    cin >> x;
}
cout << "Отсортированны массив:";
bitset<64> srt = mas;
for (int i = 63; i >= 0; i--) {
    if (srt[i] == 1) {
        cout << " " << 63 - i;
    }
}
```

Рисунок 9 – Код программы

1.6.4 Результаты тестирования

```
Введите до 64 чисел (значения от 0 до 63, 64 - завершение ввода): 34 7 4 12 1 0 45 27 54 64
Отсортированны массив: 0 1 4 7 12 27 34 45 54
```

Рисунок 10 – Результаты тестирования

1.7 ЗАДАНИЕ 2В

1.7.1 Формулировка задачи

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

1.7.2 Математическая модель решения (описание алгоритма)

1. Объявление массива mass типа unsigned char на 8 элементов: i-тый элемент отвечает за хранение чисел от i*8 до i*8+7.
2. Ввод чисел с клавиатуры (x). В элементе массива mass под номером x/8 заполнение единицей разряда под номером 7-x%8.
3. Цикл для целочисленной переменной i со значениями от 0 до 7 включительно:
   - 3.1. Инициализация двоичного массива sorting значением i-того элемента массива mass
   - 3.2. Цикл для целочисленной переменной j со значениями от 7 до 0 включительно:
     - 3.2.1. Если j-тый элемент массива sorting равен 1, то вывод i*8+7-j
     - 3.2.2. Уменьшение j на 1
   - 3.3. Увеличение i на 1

1.7.3 Код программы

```
unsigned char mass[8]{ 0 };
cout << "\nВведите до 64 чисел (значения от 0 до 63, 64 - завершение ввода): ";
cin >> x;
while (x != 64) {
    mass[x / 8] |= (1 << (7 - x % 8));
    cin >> x;
}
cout << "Отсортированны массив:";
for (int i = 0; i < 8; i++) {
    bitset<8> sorting = mass[i];
    for (int j = 7; j >= 0; j--) {
        if (sorting[j] == 1) {
            cout << " " << i * 8 + 7 - j;
        }
    }
}
```

Рисунок 11 – Код программы

1.7.4 Результаты тестирования

```
Введите до 64 чисел (значения от 0 до 63, 64 - завершение ввода): 23 6 8 43 56 0 12 44 33 64
Отсортированны массив: 0 6 8 12 23 33 43 44 56
```

Рисунок 12 – Результаты тестирования

1.8 ЗАДАНИЕ 3

1.8.1 Формулировка задачи

Входные данные: файл, содержащий не более n=10^7 неотрицательных целых чисел2, 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.

Реализовать задачу сортировки числового файла с заданными условиями. Добавить в код возможность определения времени работы программы. Определить программно объём оперативной памяти, занимаемый битовым массивом.

1.8.2 Математическая модель решения (описание алгоритма)

1. Объявление константы bitset_size типа int и инициализация значением 1МБ.
2. Объявление указателя numbers на bitset размера bitset_size.
3. Считывание числа x из файла. Установка единицы в разряд x по адресу numbers.
4. Для всех i в диапазоне от 0 до bitset_size: если i-ый элемент numbers равен 1, то вывести i
5. Вывести время работы программы и размер массива numbers.

1.8.3 Код программы

```
unsigned int start = clock();
const long int bitset_size = 8 * 1024 * 1024;
bitset<bitset_size>* numbers = new bitset<bitset_size>;
fstream fin("input.txt");
while (fin >> x) {
    numbers->set(x, 1);
}
fin.close();
fout.open("output.txt");
for (long int i = 0; i < bitset_size; i++) {
    if (numbers->test(i)) {
        fout << i << " ";
    }
}
fout.close();
unsigned int end = clock();

cout << "Время работы алгоритма: " << end - start <<  " ms" << endl;
cout << "Размер массива: " << sizeof(*numbers) << " bytes" << endl;
```

Рисунок 13 – Код программы

1.8.4 Результаты тестирования

```
Время работы алгоритма: 282 ms
Размер массива: 1048576 bytes
```

Рисунок 14 – Результаты тестирования

1.10 ВЫВОДЫ

Цель работы и сопутствующие задачи были успешно выполнены. Освоены приёмы работы с битовым представлением беззнаковых целых чисел, реализован эффективный алгоритм внешней сортировки на основе битового массива.

2. АЛГОРИТМЫ ПОИСКА В ТАБЛИЦЕ ПРИ РАБОТЕ С ДАННЫМИ ИЗ ФАЙЛА

2.1 ЦЕЛЬ РАБОТЫ

Получить практический опыт по применению алгоритмов поиска в таблицах данных.

Персональный вариант: 24

Алгоритм поиска: бинарный однородный с таблицей смещений

Структура записи файла: Книга: ISBN – двенадцатизначное число, автор, название

2.2 ЗАДАНИЕ 1

2.2.1 Постановка 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.

2.2.2 Описание подхода к решению

Для реализации поставленной персональным вариантом задачи создана структура book (рис. 15), каждая запись которой занимает в памяти 88 байт.

```
struct book {

    long long isbn;
    string author;
    string name;

    string out() {
        return to_string(isbn) + ", " + author + ", " + name + "\n";
    }
};
```

Рисунок 15 – Структура организации записи в файле

2.2.3 Код программы

```
int x, size = rand() % 100;
book test;

ofstream fout("books.bin", ios::binary | ios::out);
for (int i = 0; i < size; i++) {
    test.isbn = 0;
    for (int j = 0; j < 12; j++) {
        x = rand() % 10;
        if (x == 0 && j == 11) {
            test.isbn += (x + 1) * pow(10, j);
        }
        else {
            test.isbn += x * pow(10, j);
        }
    }
    test.author = "Vasya_" + to_string(size - i);
    test.name = "book_" + to_string(i);
    fout.write((char*)&test, sizeof(book));
}
fout.close();
```

Рисунок 16 – Код программы

2.2.4 Результаты тестирования

```
639358891659, Vasya_15, book_0
808684324363, Vasya_14, book_1
186548766607, Vasya_13, book_2
246846170873, Vasya_12, book_3
803311186345, Vasya_11, book_4
133498091764, Vasya_10, book_5
745739838966, Vasya_9, book_6
970685802796, Vasya_8, book_7
662171933836, Vasya_7, book_8
820849600185, Vasya_6, book_9
381122613435, Vasya_5, book_10
576971126052, Vasya_4, book_11
232204705292, Vasya_3, book_12
136249202281, Vasya_2, book_13
711276001915, Vasya_1, book_14
```

Рисунок 17 – Результаты тестирования

2.3 ЗАДАНИЕ 2

2.3.1 Постановка 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

1. Открыть файл для чтения.
2. Установить индекс равным 0.
3. Повторять до конца файла:
   - Прочитать текущую запись из файла.
   - Если ключ текущей записи совпадает с заданным ключом, то: Вернуть индекс.
4. Вернуть -1 (запись не найдена).
5. Закрыть файл.

Пример реализации на псевдокоде:

```
function линейный_поиск(файл, ключ):
  индекс = 0
  открыть файл для чтения
  пока не достигнут конец файла:
    текущая_запись = прочитать запись из файла
    если текущая_запись.ключ == ключ:
      вернуть индекс
    индекс = индекс + 1
  вернуть -1
  закрыть файл
```

2.3.3 Код программы

```
unsigned int start = clock();
ifstream fin("books.bin", ios::binary | ios::in);
long long key;
int number = 0;
bool f = false;
cout << "Введите ключ для поиска: ";
cin >> key;
while (fin.read((char*)&test, sizeof(book))) {
    number++;
    if (test.isbn == key) {
        f = true;
        break;
    }
}
if (f) {
    cout << "Искомый ключ в " << number << " строке" << endl;
}
else {
    cout << "Ключ не найден" << endl;
}
fin.close();
unsigned int end = clock();
cout << "Время работы программы: " << end - start << " ms"  << endl;
```

Рисунок 18 – Код программы

2.3.4 Результаты тестирования

```
Введите ключ для поиска: 125867988537
Искомый ключ в 82 строке
Время работы программы: 7571 ms
```

Рисунок 19 – Результаты тестирования для 100 записей

Таблица 1 – Тестирование на разных размерах файла:

Количество записей	Время в мс
100	5
1000	7
10000	14

2.4 ЗАДАНИЕ 3

2.4.1 Постановка задачи

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

2. 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.

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

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

5. Составить таблицу с указанием результатов замера времени.

2.4.2 Описание подхода к решению

Прямой доступ к записям в бинарном файле позволяет нам обращаться к конкретным записям, минуя последовательное чтение всего файла.

Использование функции `seekg()`:
Эта функция позволяет перемещать указатель файла на заданную позицию.

Формат: `seekg(offset, ios:: откуда);`

* `offset`: количество байт, на которое нужно переместить указатель.
* `откуда`: откуда считать смещение:
  * `beg`: от начала файла.
  * `cur`: от текущего положения указателя.
  * `end`: от конца файла.

Реализация алгоритма на псевдокоде:

Функция: Поиск ключа в таблице

Входные данные:

- tabl: вектор пар, где каждая пара содержит книгу и ее индекс
- table: вектор целых чисел, представляющий таблицу поиска
- key: ключ, который нужно найти

Выходные данные:

- индекс: индекс книги в таблице, если ключ найден
- -1: если ключ не найден

Алгоритм:

1. Установить индекс i равным первому элементу таблицы поиска table минус 1.
2. Установить флаг f как false (ложь).
3. Если ключ находится в диапазоне между ISBN первой и последней книги в таблице tabl, то:
    - Повторять для всех элементов таблицы поиска table, начиная со второго элемента (j от 1 до размера table):
        - Если ISBN текущей книги tabl[i] равен ключу:
            - Установить флаг f как true (истина).
            - Выйти из цикла.
        - Иначе, если ISBN текущей книги tabl[i] меньше ключа:
            - Увеличить индекс i на значение j-го элемента таблицы table.
        - Иначе:
            - Уменьшить индекс i на значение j-го элемента таблицы table.
    - Если флаг f равен true, то:
        - Вернуть индекс текущей книги tabl[i].
    - Иначе:
        - Вернуть -1.
4. Иначе:
    - Вернуть -1.
5. Вернуть 0.

2.4.3 Код программы

```
fin.open("books.bin", ios::binary | ios::in);
for (int i = 0; i < size; i++) {
    fin.read((char*)&test, sizeof(book));
    tabl.push_back(make_pair(test, i + 1));
}
fin.close();

sort(tabl.begin(), tabl.end(), comp);

/*for (auto element : tabl) {
    cout << element.second << " " << element.first.out();
}*/

vector <int> table;
for (int j = 1; j <= log2(tabl.size()) + 2; j++) {
    table.push_back((tabl.size() + pow(2, j - 1)) / pow(2, j));
}

cout << "Введите ключ для поиска: ";
cin >> key;
int out = search_key(tabl, table, key);
if (out >= 0) {
    cout << "Искомый ключ в " << out << " строке" << endl;
}
else {
    cout << "Ключ не найден" << endl;
}
```

Рисунок 20 – Код программы

```
int search_key(vector<pair<book, int>> tabl, vector<int> table, long long key) {
    int i = table[0] - 1;
    bool f = false;
    if (key >= tabl[0].first.isbn && key <= tabl[tabl.size() - 1].first.isbn) {
        for (int j = 1; j < table.size(); j++) {
            if (tabl[i].first.isbn == key) {
                f = true;
                break;
            }
            else if (tabl[i].first.isbn < key) {
                i += table[j];
            }
            else {
                i -= table[j];
            }
        }
        if (f) {
            return tabl[i].second;
        }
        else {
            return -1;
        }
    }
    else {
        return -1;
    }
    return 0;
}
```

Рисунок 21 – Функция поиска элемента по ключу

```
string search_url(string s, int url) {
    ifstream fin(s, ios::binary | ios::in);
    fin.seekg(sizeof(book) * (url - 1), ios_base::beg);
    book test;
    fin.read((char*)&test, sizeof(book));
    fin.close();
    return test.out();
}
```

Рисунок 22 – Функция считывания элемента по ссылке

2.4.4 Результаты тестирования

```
Введите ключ для поиска: 819795719832
Время работы программы: 0 ms
Искомый ключ в 76 строке
Введите номер записи: 12
Время работы программы: 1 ms
Найдена запись: 389440258759, Vasya_89, book_11
```

Рисунок 23 – Результаты тестирование для 100 элементов

Таблица 2 – Тестирование на различных размерах файла

Количество записей	Время работы функции поиска в мс	Время работы функции считывания в мс
100	0	0
1000	3	0
10000	31	0

2.5 ВЫВОДЫ

Получен практический опыт по применению алгоритмов поиска в таблицах данных, а именно: линейный поиск и однородный бинарный поиск с таблицей смещений. Проведено тестирование алгоритмов на различных размерах исходных данных и зафиксировано время работы каждого.

СПИСОК ИНФОРМАЦТОННЫХ ИСТОЧНИКОВ

1. Рысин, М. Л. Введение в структуры и алгоритмы обработки данных. Часть 1. Сложность алгоритмов. Сортировки. Линейные структуры данных. Поиск в таблице. : учеб. пособие / М. Л. Рысин, М. В. Сартаков, М. Б. Туманова ; МИРЭА – Российский технологический университет, 2022. – 109 с. – ISBN 978-5-7339-1612-5.

2. ГОСТ 19.701-90. Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения : межгосударственный стандарт : дата введения 1992-01-01 / Федеральное агентство по техническому регулированию. – Изд. официальное. – Москва : Стандартинформ, 2010. – 23 с.