

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний
інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 1 з дисципліни
«Проектування алгоритмів»

«Проектування і аналіз алгоритмів зовнішнього сортування»

Виконав:

ІІ-13, Музичук Віталій Андрійович _____
(шифр, прізвище, ім'я, по батькові)

Перевірив

Головченко М.М. _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ.....	5
3.1	ПСЕВДОКОД АЛГОРИТМУ.....	5
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	8
3.2.1	<i>Вихідний код.....</i>	8
	ВИСНОВОК	10
	КРИТЕРІЇ ОЦІНЮВАННЯ	ПОМИЛКА! ЗАКЛАДКУ НЕ ВИЗНАЧЕНО.

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні алгоритми зовнішнього сортування та способи їх модифікації, оцінити поріг їх ефективності.

2 ЗАВДАННЯ

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм зовнішнього сортування за допомогою псевдокоду (чи іншого способу за вибором).

Виконати програмну реалізацію алгоритму на будь-якій мові програмування та відсортувати випадковим чином згенерований масив цілих чисел, що зберігається у файлі (розмір файлу має бути не менше 10 Мб, можна значно більше).

Здійснити модифікацію програми і відсортувати випадковим чином згенерований масив цілих чисел, що зберігається у файлі розміром не менше ніж двократний обсяг ОП вашого ПК. Досягти швидкості сортування з розрахунку 1Гб на 3хв. або менше.

Рекомендується попередньо впорядкувати серії елементів довжиною, що займає не менше 100Мб або використати інші підходи для пришвидшення процесу сортування.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти базову та модифіковану програми. У висновку деталізувати, які саме модифікації було виконано і який ефект вони дали.

Таблиця 2.1 – Варіанти алгоритмів

№	Алгоритм сортування
17	Пряме злиття

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

Псевдокод алгоритму сортування:

1. ПОЧАТОК
2. `sequence_size = 1`
3. `quantity_of_number = size / 4`
4. ПОКИ `sequence_size < quantity_of_number`
 - 4.1. **split**(`file_name`, `sequence_size`)
 - 4.2. **merge**(`file_name`, `sequence_size`, `quantity_of_numbers`)
5. Вивести дані файлу на екран
6. КІНЕЦЬ

Псевдокод процедури **split**(`file_name`, `sequence_size`):

1. ПОЧАТОК
2. Відкрити файл з ім'ям `file_name` для читання
3. Створити два допоміжних файла з назвами "B.bin" і "C.bin" та відкрити їх для запису
4. `value = 0`
5. `flag = true`
6. Читати ПОКИ не дійдемо до кінця файлу `file_name`
 - 6.1. ЯКЩО `flag = true`, ТО
 - 6.1.1. Записати зчитане число з файлу `file_name` у перший файл "B.bin"
 - 6.1.2. `value = value + 1`
 - 6.1.3. ЯКЩО `value >= sequence_size`, ТО
 - 6.1.3.1. `flag = false`
 - 6.1.3.2. `value = 0`
 - 6.2. ІНАКШЕ
 - 6.2.1. Записати зчитане число з файлу `file_name` у другий файл "C.bin"
 - 6.2.2. `value = value + 1`

6.2.3. ЯКЩО $value \geq sequence_size$, ТО

6.2.3.1. $flag = true$

6.2.3.2. $value = 0$

7. Закрити усі файли

8. КІНЕЦЬ

Псевдокод процедури **merge**(file_name, sequence_size, quantity_of_numbers):

1. ПОЧАТОК

2. Створити файл з ім'ям file_name та відкрити його для запису

3. Відкрити два допоміжних файли з назвами "B.bin" і "C.bin"

4. ПОКИ правда

4.1. Зчитати з другого файлу «C.bin» число у змінну first_number

4.2. ЯКЩО first_number є кінцем файлу, ТО

4.2.1. Завершити цикл

4.3. Зчитати з першого файлу «B.bin» число у змінну second_number

4.4. $valueA = 0$; $valueB = 0$

4.5. ПОКИ valueA не дорівнює sequence_size ТА valueB не дорівнює sequence_size

4.5.1. ЯКЩО first_number < second_number, ТО

4.5.1.1. Записати у вихідний файл file_name число first_number

4.5.1.2. $valueA = valueA + 1$

4.5.1.3. ЯКЩО valueA не дорівнює sequence_size, ТО

4.5.1.3.1. Зчитати наступне число з першого файлу "B.bin" у змінну first_number

4.5.2. ІНАКШЕ

4.5.2.1. Записати у вихідний файл file_name число second_number

4.5.2.2. $valueB = valueB + 1$

4.5.2.3. ЯКЩО valueB не дорівнює sequence_size, ТО

4.5.2.3.1. Зчитати наступне число з другого файлу "C.bin" у змінну second_number

- 4.6. ЯКЩО $\text{valueA} > \text{valueB}$, ТО
 - 4.6.1. Записати у вихідний файл число `second_number`
 - 4.6.2. $\text{valueB} = \text{valueB} + 1$
 - 4.6.3. ПОКИ $\text{valueB} < \text{sequence_size}$ ТА не дійшли до кінця другого файлу “C.bin”, ТО
 - 4.6.3.1.Зчитати наступне число з другого файлу “C.bin” у змінну `second_number`
 - 4.6.3.2.Записати у вихідний файл число `second_number`
 - 4.6.3.3. $\text{valueB} = \text{valueB} + 1$
- 4.7. ІНАКШЕ ЯКЩО $\text{valueA} < \text{valueB}$, ТО
 - 4.7.1. Записати у вихідний файл число `first_number`
 - 4.7.2. $\text{valueA} = \text{valueA} + 1$
 - 4.7.3. ПОКИ $\text{valueA} < \text{sequence_size}$ ТА не дійшли до кінця першого файлу “B.bin”, ТО
 - 4.7.3.1.Зчитати наступне число з першого файлу “B.bin” у змінну `first_number`
 - 4.7.3.2.Записати у вихідний файл число `first_number`
 - 4.7.3.3. $\text{valueA} = \text{valueA} + 1$
- 5. ПОКИ не дійдемо до кінця першого файлу «B.bin», ТО
 - 5.1. Записувати у вихідний файл `file_name` число `first_number`
- 6. Закрити всі файли
- 7. КІНЕЦЬ

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

Базовий алгоритм

Процедура sort()

```
void sort(string file_name) {
    int quantity_of_number = get_size(file_name);
    copy_file(file_name, quantity_of_number);

    for (long sequence_size = 1; sequence_size < quantity_of_number; sequence_size *=2) {
        split(COPY_FILE, sequence_size);

        //file_output(FIRST_FILE);
        //file_output(SECOND_FILE);

        merge(COPY_FILE, sequence_size, quantity_of_number);

        file_output(COPY_FILE);
    }
    //file_output(COPY_FILE);
}
```

Процедура split()

```
void split(string name, int sequence_size) {
    ifstream file_out(FOLDER_NAME + name, ios::binary);
    ofstream A(FOLDER_NAME + FIRST_FILE, ios::binary);
    ofstream B(FOLDER_NAME + SECOND_FILE, ios::binary);

    int number;
    bool flag = true;
    int value = 0;

    while (file_out.read((char*)&number, sizeof(number))) {
        if (flag) {
            A.write((char*)&number, sizeof(number));
            value++;
            if (value >= sequence_size) {
                flag = false;
                value = 0;
            }
        }
        else {
            B.write((char*)&number, sizeof(number));
            value++;
            if (value >= sequence_size) {
                flag = true;
                value = 0;
            }
        }
    }
    A.close();
    B.close();
    file_out.close();
}
```

Процедура merge()

```
void merge(string file_name, int sequence_size, int size) {
    ofstream file_out(FOLDER_NAME + file_name, ios::binary);
```



```

ifstream A(FOLDER_NAME + FIRST_FILE, ios::binary);
ifstream B(FOLDER_NAME + SECOND_FILE, ios::binary);
int first_number; int second_number;

while (true) {
    B.read((char*)&second_number, sizeof(int));
    if (B.eof())
        break;
    A.read((char*)&first_number, sizeof(int));
    int valueA = 0, valueB = 0;

    while (valueA != sequence_size && valueB != sequence_size) {
        if (first_number < second_number) {
            file_out.write((char*)&first_number, sizeof(int));
            valueA++;
            if (valueA != sequence_size)
                A.read((char*)&first_number, sizeof(int));
        }
        else {
            file_out.write((char*)&second_number, sizeof(int));
            valueB++;
            if (valueB != sequence_size)
                B.read((char*)&second_number, sizeof(int));
        }
    }
    if (valueA > valueB) {
        file_out.write((char*)&second_number, sizeof(int));
        valueB++;
        while (valueB < sequence_size) {
            B.read((char*)&second_number, sizeof(int));
            if (B.eof())
                break;
            file_out.write((char*)&second_number, sizeof(int));
            valueB++;
        }
    }
    else if (valueA < valueB) {
        file_out.write((char*)&first_number, sizeof(int));
        valueA++;
        while (valueA < sequence_size) {
            A.read((char*)&first_number, sizeof(int));
            if (A.eof())
                break;
            file_out.write((char*)&first_number, sizeof(int));
            valueA++;
        }
    }
}

while (A.read((char*)&first_number, sizeof(int))) {
    file_out.write((char*)&first_number, sizeof(int));
}

file_out.close();
A.close();
B.close();
}

```

[GitHub](#)

ВИСНОВОК

При виконанні даної лабораторної роботи я детально ознайомився з проблемою відсортовування файлів великих розмірів, дізнався про можливі розв'язки цієї задачі та самостійно реалізував псевдокод та програмну реалізацію методу прямого злиття. Був проведений аналіз роботи алгоритму, який показав, що базовий алгоритм без модифікацій сортує файл розміром 10Мб в середньому за 2 хвилини. Це є доволі непродуктивно, тому базовий алгоритм не варто використовувати для реальних задач.