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

Кафедра ІПІ

Звіт

з лабораторної роботи №1 з дисципліни
«Проєктування алгоритмів»

„ **Проектування і аналіз алгоритмів зовнішнього сортування**”

Варіант 10

Виконав ІП-45 Янов Богдан Євгенійович

Перевірив Соколовський Владислав Володимирович

Київ 2025

Лабораторна робота №1

Проектування і аналіз алгоритмів зовнішнього сортування

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

Згідно з варіантом, розробити та записати алгоритм зовнішнього сортування за допомогою псевдокоду (чи іншого способу за вибором).

Виконати програмну реалізацію алгоритму будь-якою мовою програмування та відсортувати випадковим чином згенерований файл за ключем (розмір файлу має бути не менше 10 Мб, можна значно більше).

Структура файлу наступна:

− латинська літера (ключ) - рядок довжиною до 45 символів - номер телефону.

Здійснити модифікацію програми та відсортувати файл за ключем. Досягти швидкості сортування з розрахунку 1 ГБ на 3-5 хв або менше.

За допомогою чат-боту (на власний вибір) створити модифіковану версію програми сортування файлу дослідити її, і вказати переваги та недоліки. Алгоритм сортування має відрізнятись від лекційного, але обмеження взяти відповідно до варіанту.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти базову та модифіковані програми. У висновку деталізувати, які саме модифікації було виконано і який ефект вони дали. Описати переваги та недоліки кожної версії.

Варіант 10 – Природне (адаптивне) злиття за зростанням, 300 МБ.

Псевдокод алгоритмів

```
function distribute(source, fileA, fileB):
    open reader from source
    open writerA for fileA
    open writerB for fileB

    currentWriter = writerA
    otherWriter = writerB

    lastRecord = null
    hasLast = false
    runCount = 0

    while reader has next record x:
        if not hasLast:
            runCount += 1
            currentWriter.write(x)
            lastRecord = x
            hasLast = true

        else if x >= lastRecord:
            currentWriter.write(x)
            lastRecord = x

        else:
            swap(currentWriter, otherWriter)
            runCount += 1
            currentWriter.write(x)
            lastRecord = x

    return runCount

function merge_files(file1, file2, outputFile):
    open runReader1 for file1
    open runReader2 for file2
    open writer for outputFile

    runCount = 0

    while runReader1 not empty OR runReader2 not empty:
        runCount += 1

        # merge one run
        loop:
            a = runReader1 has value AND not at boundary
            b = runReader2 has value AND not at boundary

            if not a and not b:
                break
```

```
            if a and b:
                if runReader1.peek() <= runReader2.peek():
                    writer.write(runReader1.peek())
                    runReader1.consume()
                else:
                    writer.write(runReader2.peek())
                    runReader2.consume()

            else if a:
                writer.write(runReader1.peek())
                runReader1.consume()

            else:
                writer.write(runReader2.peek())
                runReader2.consume()

        if runReader1 at boundary:
            runReader1.clear_boundary()
        if runReader2 at boundary:
            runReader2.clear_boundary()

    return runCount

function natural_merge_sort(path, fileA = "data/a.txt", fileB = "data/b.txt"):
    loop forever:
        runs = distribute(path, fileA, fileB)
        if runs <= 1:
            break

        mergedRuns = merge_files(fileA, fileB, path)
        if mergedRuns <= 1:
            break

    delete fileA
    delete fileB
```

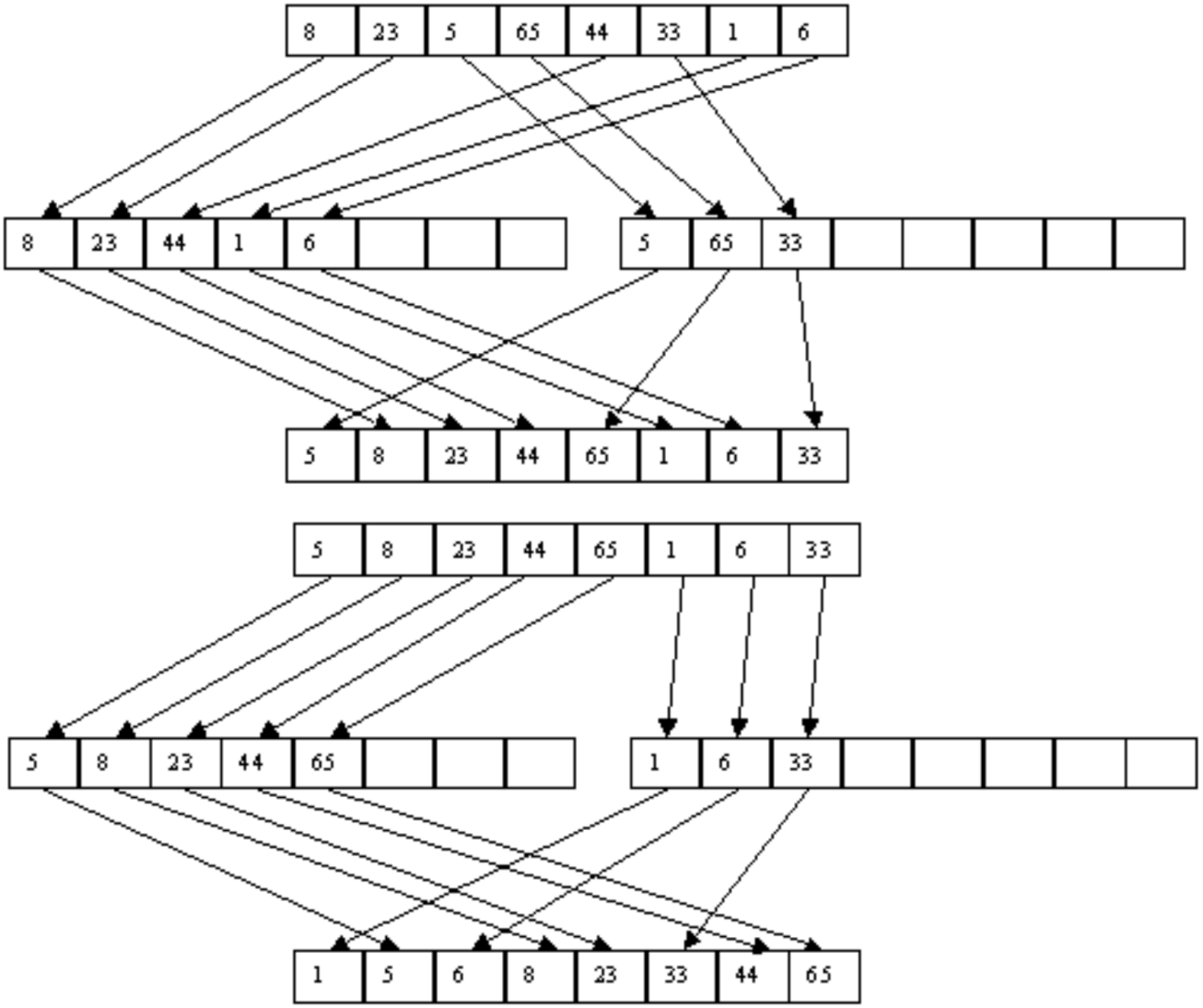
Вихідний код

https://github.com/Partur-dev/KPI/tree/main/3 Sem/PA/lab1

Запит до ШІ

Write a C++ program that sorts a large text file up to 1 GB or more in ascending order. Each line of the file has the following format: key\t45characters\nphone-number. The key is a 5-character string. The memory limit is 300 MB. Sorting must be completed in less than 5 minutes.

Приклад роботи



Аналіз алгоритмів

Під час аналізу класичного алгоритму природного злиття виявлено значний недолік — формування великої кількості серій. Через це серії мають невеликий розмір, а їх загальна кількість суттєво зростає. Складність формування серії оцінюється як O(n), оскільки кожен запис файлу зчитується один раз, а всі операції у циклі (парсинг рядка, перевірка умови, запис у файл) виконуються за сталий час. На кожному етапі кількість серій зменшується приблизно вдвічі. Таким чином, загальна складність алгоритму оцінюється як O(n)*O(logn)=O(nlogn).

У модифікованому алгоритмі реалізовано метод initial_distribute, який на початковому етапі розбиває вихідний файл на блоки по 1 000 000 записів та сортує їх у пам'яті перед записом у проміжні файли. Це дозволяє формувати більш оптимальні серії на початку сортування, зменшуючи кількість фаз злиття. Для сортування блоку використовується std::sort, який базується на алгоритмі швидкого сортування: у середньому його складність O(nlogn), а в найгіршому випадку O(n^2). Завдяки такій модифікації зменшується кількість серій на початку алгоритму, що дозволяє скоротити загальний час сортування та підвищити ефективність зовнішнього злиття.

Третій алгоритм реалізує зовнішнє сортування з використанням буферизації та priority_queue для злиття серій. Дані читаються у блоки, що поміщаються у виділену пам'ять (наприклад, 100 МБ), сортуються у пам'яті (std::sort) і записуються у тимчасові файли. На фазі злиття всі тимчасові файли зчитуються паралельно, а мінімальний елемент серед доступних блоків витягується за допомогою priority_queue. Завдяки цій структурі даних злиття має складність O(logk) на кожну вставку, де k — кількість тимчасових файлів, а загальна складність сортування оцінюється як O(nlogn) для великих обсягів даних.

У порівнянні з природним злиттям, алгоритм із priority_queue менш чутливий до початкової структури даних, оскільки завжди формує великі відсортовані блоки, і злиття виконується ефективно навіть при високій фрагментації вхідного файлу. Модифіковане природне злиття на початку намагається зменшити кількість серій через попереднє сортування блоків, що робить його більш адаптивним до локальної впорядкованості даних, але в гіршому випадку може вимагати більше фаз злиття.

Висновок

У ході лабораторної роботи реалізовано класичний алгоритм природного злиття, модифіковану версію з попереднім сортуванням блоків у пам’яті та алгоритм зовнішнього сортування, згенерований за допомогою ШІ. Проведено аналіз їх ефективності за часом виконання та використанням пам’яті. Модифіковане природне злиття показало кращу адаптивність до локальної впорядкованості даних, зменшуючи кількість фаз злиття, тоді як алгоритм на основі priority_queue забезпечує стабільну продуктивність при великих обсягах файлів. Отримані результати дозволяють оцінити переваги та недоліки різних підходів до зовнішнього сортування та вибрати оптимальний метод залежно від обмежень пам’яті та структури вхідних даних.