

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”
Варіант 7

Виконав(ла)

ІП-15, Гуменюк О.В.

(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	6
3.1	ПСЕВДОКОД АЛГОРИТМУ	6
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	8
3.2.1	<i>Вихідний код</i>	8
	ВИСНОВОК	14
	КРИТЕРІЇ ОЦІНЮВАННЯ	16

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 1Гб за 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
1	Пряме злиття
2	Природне (адаптивне) злиття
3	Збалансоване багатошляхове злиття
4	Багатофазне сортування
5	Пряме злиття
6	Природне (адаптивне) злиття
7	Збалансоване багатошляхове злиття
8	Багатофазне сортування
9	Пряме злиття
10	Природне (адаптивне) злиття

11	Збалансоване багатошляхове злиття
12	Багатофазне сортування
13	Пряме злиття
14	Природне (адаптивне) злиття
15	Збалансоване багатошляхове злиття
16	Багатофазне сортування
17	Пряме злиття
18	Природне (адаптивне) злиття
19	Збалансоване багатошляхове злиття
20	Багатофазне сортування
21	Пряме злиття
22	Природне (адаптивне) злиття
23	Збалансоване багатошляхове злиття
24	Багатофазне сортування
25	Пряме злиття
26	Природне (адаптивне) злиття
27	Збалансоване багатошляхове злиття
28	Багатофазне сортування
29	Пряме злиття
30	Природне (адаптивне) злиття
31	Збалансоване багатошляхове злиття
32	Багатофазне сортування
33	Пряме злиття
34	Природне (адаптивне) злиття
35	Збалансоване багатошляхове злиття

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

```
Function MultiwayMergeSort(input: Int[], m: Int)
  arrsA : Int[1][] = {input}
  arrsB : Int[m][]
  arrsC : Int[m][]
  for i = 1 to m do
    arrsB[i] = Int[]
    arrsC[i] = Int[]
  end for

  MultiwayMerge(arrsA, arrsB)

  flag : Int = 1

  while (!isSorted(arrsA[1], arrsB[1], arrsC[1])) do
    if (flag == 1) do
      MultiwayMerge(arrsB, arrsC)
    end if
    else do
      MultiwayMerge(arrsC, arrsB)
    end else
    flag = -flag
  end while

  if (arrsB[1].length == input.length)
    for i = 1 to m do
      input[i] = arrsB[i]
    end for
  end if
  else do
    for i = 1 to m do
      input[i] = arrsC[i]
    end for
  end else

  return input
end

Function isSorted(arrA: Int[], arrB: Int[], arrC: Int[])
  return (arrB[1].length == arrA.length or
    arrC[1].length == arrA.length)
end
```

```

Function MultiwayMerge(inputArrs: Int[][], outputArrs: Int[][])
    pointerArr: Int[]
    for i = 1 to inputArrs.length do
        pointerArr[i] = 1
    end for

    i : Int = 0
    set : Int[]

    while (!isMerged(pointerArr, inputArrs)) do

        minIndex : Int = null
        minValue : Int = inf

        for k = 1 to inputArrs.length do:
            if (pointerArr[k] <= inputArrs[k].length) do
                if (set.length == 0 or
                    inputArrs[k][pointerArr[k]] >= set[set.length]) do
                    if (inputArrs[k][pointerArr[k]] <= minValue) do
                        minIndex = k
                        minValue = inputArrs[k][pointerArr[k]]
                    end if
                end if
            end if
        end for

        if (minIndex == null) do
            outputArrs[i+1].append(set)
            set.clear()
            i = (i+1)% outputArrs.length
        end if
        else do
            set.append(minValue)
            pointerArr[minIndex]++
        end else
    end while

    return outputArrs
end

Function isMerged(pointerArr: Int[], arrs: Int[][])
    for i = 1 to arrs.length do
        if (pointerArr[i] <= arrs[i].length) do
            return False
        end if
    end for
    return True
end

```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код неоптимізованого алгоритму

```
fun oldMultiwayMergeSort(inputFileName: String, outputFileName: String, m:
Int) {
    val fileA = arrayOf(File(inputFileName))
    val filesB = Array(m) { i -> File("B${i + 1}.txt").also { it.delete();
it.createNewFile() } }
    val filesC = Array(m) { i -> File("C${i + 1}.txt").also { it.delete();
it.createNewFile() } }

    multiwayMerge(fileA, filesB)

    var flag = 1
    while (!isSorted(fileA[0], filesB[0], filesC[0])) {
        if (flag == 1) {
            multiwayMerge(filesB, filesC)
        } else {
            multiwayMerge(filesC, filesB)
        }
        flag = -flag
    }

    val outputFile = File(outputFileName)
    outputFile.createNewFile()

    if (filesB[0].length() == fileA[0].length()) {
        filesB[0].copyTo(outputFile, true)
    } else {
        filesC[0].copyTo(outputFile, true)
    }
}

private fun isSorted(arrA: File, arrB: File, arrC: File): Boolean {
    return arrB.length() == arrA.length() || arrC.length() ==
arrA.length()
}

private fun multiwayMerge(inputFiles: Array<File>, outputFiles:
Array<File>) {

    val bufferedReaders = Array(inputFiles.size) { i ->
BufferedReader(FileReader(inputFiles[i])) }
    outputFiles.forEach { it.writeText("") }

    var j = 0
    val set = ArrayList<Int>(0)

    while (!isMerged(bufferedReaders)) {
        var minValue = Int.MAX_VALUE
        var minIndex: Int? = null

        for (i in bufferedReaders.indices) {

            val text = peek(bufferedReaders[i])

            if (text != null) {
                val num = text.toInt()

                if (set.isEmpty() || num >= set.last()) {
```



```

        if (num <= minValue) {
            minValue = num
            minIndex = i
        }
    }
}

if (minIndex == null) {
    if (outputFiles[j].length() > 0L) {
        outputFiles[j].appendText("\n")
    }
    outputFiles[j].appendText(set.joinToString("\n"))

    set.clear()
    j = (j + 1) % outputFiles.size
} else {
    set.add(minValue)
    bufferedReaders[minIndex].readLine()
}

if (outputFiles[j].length() > 0L) {
    outputFiles[j].appendText("\n")
}
outputFiles[j].appendText(set.joinToString("\n"))

bufferedReaders.forEach { it.close() }
}

private fun isMerged(readers: Array<BufferedReader>): Boolean {
    for (reader in readers) {
        val s = peek(reader)
        if (s != null) {
            return false
        }
    }
    return true
}

private fun peek(reader: BufferedReader): String? {
    reader.mark(100)
    val line = reader.readLine()
    reader.reset()
    return line
}

```

3.2.2 Тестування неоптимізованого алгоритму

Приклад роботи неоптимізованої версії програми наведений на рисунку

3.1.

```

Do you want enter file size in GB or MB? (gb/mb/kb/b): mb
Enter size of the file in MB: 10
Number of generated elements: 1000000
File with name input.txt and size 10 in MB was generated
Enter number m: 7
Do you want to use new algorithm (yes/no): no
Running time of the program is 3 minutes and 17 seconds

Process finished with exit code 0

```

Рисунок 3.1 - Приклад роботи неоптимізованої версії програми

3.2.3 Вихідний код оптимізованого алгоритму

```

fun newMultiwayMergeSort(inputFileName: String, outputFileName: String, m:
Int = 5) {
    val fileA = File(inputFileName)
    val filesB = Array(m) { i -> File("B${i + 1}.txt").also { it.delete();
it.createNewFile() } }
    val filesC = Array(m) { i -> File("C${i + 1}.txt").also { it.delete();
it.createNewFile() } }

    initialSorting(fileA, filesB)

    var flag = 1

    while (!isSorted(fileA, filesB[0], filesC[0])) {
        when (flag){
            1 -> multiwayMerge(filesB, filesC)
            else -> multiwayMerge(filesC, filesB)
        }
        flag = -flag
    }

    val outputFile = File(outputFileName)
    outputFile.createNewFile()

    if (filesB[0].length() == fileA.length()) {
        filesB[0].copyTo(outputFile, true)
    } else {
        filesC[0].copyTo(outputFile, true)
    }
}

private fun initialSorting(inputFile: File, outputFiles: Array<File>){
    val br = BufferedReader(FileReader(inputFile), BUFFER_SIZE)
    val bufferedWritersB = Array(outputFiles.size) {i ->
BufferedWriter(FileWriter(outputFiles[i]), BUFFER_SIZE)}

    var i = 0
    while (true) {
        val charArr = readChunk(br, (CHUNK_SIZE/2))

        if (charArr.isEmpty()){
            break
        }

        val intArr = charToIntArray(charArr)

```

```

        quickSort(intArr, 0, intArr.size - 1)

        writeToFile(bufferedWritersB[i], outputFiles[i],
intArr.joinToString("\n"))

        i = (i + 1) % outputFiles.size
    }

    br.close()
    bufferedWritersB.forEach { it.close() }
}

private fun isSorted(arrA: File, arrB: File, arrC: File): Boolean {
    return arrB.length() == arrA.length() || arrC.length() ==
arrA.length()
}

private fun multiwayMerge(inputFiles: Array<File>, outputFiles:
Array<File>) {

    val bufferedWriters = Array(outputFiles.size) { i ->
BufferedWriter(FileWriter(outputFiles[i]), BUFFER_SIZE) }
    val bufferedReaders = Array(inputFiles.size) { i ->
BufferedReader(FileReader(inputFiles[i]), BUFFER_SIZE) }

    val outputFilesEmpty = Array(outputFiles.size){true}
    val pointerArr = IntArray(inputFiles.size){-1}

    val chunkSize = CHUNK_SIZE / inputFiles.size
    val bufferedArrays = ArrayList<IntArray>(0)

    bufferedReaders.withIndex().forEach{
        if (peek(it.value) != null){

bufferedArrays.add(charToIntArray(readChunk(bufferedReaders[it.index],
chunkSize)))

            pointerArr[it.index] = 0
        }
    }

    var j = 0
    val set = ArrayList<Int>(0)

    while (!isMerged(pointerArr)) {

        val minIndex = findMin(bufferedArrays, pointerArr, set)

        if (minIndex == null || set.size >= chunkSize/4) {
            writeToFile(bufferedWriters[j], !outputFilesEmpty[j],
set.joinToString("\n"))
            outputFilesEmpty[j] = false
            set.clear()
            if (minIndex == null) j = (j + 1) % outputFiles.size
        }

        if (minIndex != null) {

            set.add(bufferedArrays[minIndex][pointerArr[minIndex]])

            pointerArr[minIndex] ++
            if (pointerArr[minIndex] >= bufferedArrays[minIndex].size){

                val s = peek(bufferedReaders[minIndex])

```

```

        if (s != null) {
            bufferedArrays[minIndex] =
charToIntArray(readChunk(bufferedReaders[minIndex], chunkSize))
            pointerArr[minIndex] = 0
        }
        else{
            pointerArr[minIndex] = -1
            bufferedArrays[minIndex] = IntArray(0)
        }
    }
}

writeToFile(bufferedWriters[j], !outputFilesEmpty[j],
set.joinToString("\n"))

bufferedReaders.forEach { it.close() }
bufferedWriters.forEach { it.close() }
}

private fun writeToFile(writer: BufferedWriter, file: File, text: String){
    if (file.length() > 0L) {
        writer.write("\n")
    }
    writer.write(text)
}

private fun writeToFile(writer: BufferedWriter, isEmpty: Boolean, text:
String){
    if (isEmpty) {
        writer.write("\n")
    }
    writer.write(text)
}

private fun findMin(bufferedArrays: ArrayList<IntArray>, pointerArr:
IntArray, set: ArrayList<Int>): Int?{
    var minValue = Int.MAX_VALUE
    var minIndex: Int? = null

    for (i in bufferedArrays.indices) {
        if (pointerArr[i] != -1) {
            val num = bufferedArrays[i][pointerArr[i]]

            if (set.isEmpty() || num >= set.last()) {
                if (num <= minValue) {
                    minValue = num
                    minIndex = i
                }
            }
        }
    }
    return minIndex
}

private fun isMerged(pointerArrs: IntArray): Boolean {
    for (pointer in pointerArrs) {
        if (pointer != -1) {
            return false
        }
    }
}

```

```

    }
    return true
}

private fun peek(reader: BufferedReader): String? {
    reader.mark(100)
    val line = reader.readLine()
    reader.reset()
    return line
}

private fun charToIntArray(charArray: CharArray): IntArray {
    return
    charArray.joinToString("").split("\n").map{it.toInt()}.toIntArray()
}

private fun readChunk(br: BufferedReader, chunkSize: Int): CharArray{
    var charArr = CharArray(chunkSize)
    br.read(charArr)

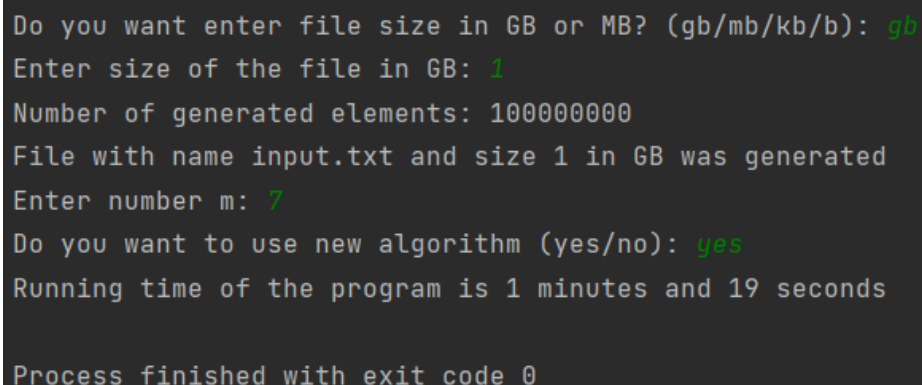
    if (charArr.last().code != 0){
        val arr = ArrayList<Char>()
        while(true){
            val char = br.read().toChar()
            if (char == '\n' || char.code == 0){
                charArr += arr
                break
            }
            else{
                arr.add(char)
            }
        }
    }
    else{
        charArr = charArr.filter{it.code != 0}.toCharArray()
    }

    return charArr
}

```

3.2.4 Тестування оптимізованого алгоритму

Приклад роботи оптимізованої версії програми наведений на рисунку 3.2.



```

Do you want enter file size in GB or MB? (gb/mb/kb/b): gb
Enter size of the file in GB: 1
Number of generated elements: 100000000
File with name input.txt and size 1 in GB was generated
Enter number m: 7
Do you want to use new algorithm (yes/no): yes
Running time of the program is 1 minutes and 19 seconds

Process finished with exit code 0

```

Рисунок 3.2 – Приклад роботи оптимізованої версії програми

ВИСНОВОК

При виконанні даної лабораторної роботи я отримав практичні навички роботи з алгоритмами зовнішнього сортування, а саме записав псевдокод алгоритму збалансованого багатошляхового злиття, виконав програмну реалізацію цього метода, а також протестував і оптимізував її.

Збалансоване багатошляхове злиття використовує $2 * m$ допоміжних файлів ($B_1 - B_m, C_1 - C_m$), тому вибір числа m сильно впливає на швидкість роботи алгоритму. Використовуючи тестування було визначено, що для обох версій програмної реалізації алгоритму оптимальним є $m = 7$.

Для оптимізації програмної реалізації алгоритму було виконано дві головні модифікації:

1. попереднє сортування частин файлу;
2. буферизація процесів читання і запису в файли.

У першій версії алгоритму з самого початку відбувається багатошляхове злиття спочатку з файлу A в допоміжні файли $B_1 - B_m$, потім з $B_1 - B_m$ в $C_1 - C_m$, потім з цих файлів назад в $B_1 - B_m$ і так далі, поки в B_1 або в C_1 не з'явиться повністю відсортовані елементи файлу A . У оптимізованій версії програми перед тим як використати збалансоване багатошляхове злиття, програма читає, зберігає та сортує серії елементів довжиною приблизно 250 МБ. Це значно пришвидшує роботу алгоритму, так як кількість потрібних злиттів сильно зменшується.

У неоптимізованій версії програми майже не використовується буферизація: числа читаються і записуються по-одному. Для кожного окремого запису або читання виходить окреме звертання до файлу, що є набагато повільнішим чим звертання до внутрішньої пам'яті. У оптимізованій версії програми використовуються буфери, а саме буферні читачі (BufferedReader) і буферні записники (BufferedWriter). Також замість читання чисел по-одному, програма читає великі шматки файли, розмір яких залежить від кількості

допоміжних файлів. Для всіх буферних виставлений розмір 4096 байтів; збільшення розміру буферів не змінювало швидкість роботи програми.

КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 25.09.2022 включно
максимальний бал дорівнює – 5. Після 25.09.2022 максимальний бал дорівнює –

1.

Критерії оцінювання у відсотках від максимального балу:

- псевдокод алгоритму – 15%;
- програмна реалізація алгоритму – 40%;
- програмна реалізація модифікованого алгоритму – 40%;
- висновок – 5%.