**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(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-13 Павленко М. А.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc109342184)

[2 ЗаВдання 4](#_Toc109342185)

[3 Виконання 6](#_Toc109342186)

[3.1 Псевдокод алгоритму 6](#_Toc109342187)

[3.2 Програмна реалізація алгоритму 6](#_Toc109342188)

[3.2.1 Вихідний код 6](#_Toc109342189)

[Висновок 7](#_Toc109342190)

[Критерії оцінювання 8](#_Toc109342191)

# Мета лабораторної роботи

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

# 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** |
| 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 |

# Виконання

## Псевдокод алгоритму

1. Розділити послідовність чисел у файлі на серії чисел
2. Записати отримані серії у робочі файли згідно їх розподілу за числами Фібоначі

2.1) Якщо не вдається «дотягнути» число серій у файлах, то заповнити їх за допомогою пустих серій (dummy runs)

3) Процедура злиття послідовностей у більші серії

4) Робочий файл, що містить 1 серію та у який вони зливались в останній раз, автоматично стає відсортованим початковим файлом.

## Програмна реалізація алгоритму

### Вихідний код

import java.io.\*;

import java.util.Arrays;

import java.util.PriorityQueue;

import java.util.Random;

public class App {

public record Element(int value, int file\_index) implements Comparable<Element> {

public int getValue() {

return value;

}

public int getFileIndex() {

return file\_index;

}

@Override

public int compareTo(Element o) {

return Integer.compare(this.value, o.value);

}

static int intNull = Integer.MAX\_VALUE, intSize = 4, N = 5;

static long processedData;

static int nextRunFirstElem;

static int outFileIndex;

static int previousOutFileIndex;

static int runsPerLevel;

static int[] dummyRuns = new int[N + 1];

static int[] runsDistribution = new int[N + 1];

static boolean[] fileCanBeRead = new boolean[N + 1];

static int[] runsLastElems = new int[N + 1];

static int[] run\_last\_elements = new int[N + 1];

static Element[] nextRunFirstElems = new Element[N + 1];

static PriorityQueue<Element> queue = new PriorityQueue<>();

static byte[] byteBuffer = new byte[4];

static long unsortedFileLen;

public static void main(String[] args) throws IOException {

File file = initFile("main.bin");

unsortedFileLen = file.length();

MergeSort(file);

}

public static File initFile(String path) throws IOException {

System.out.print("Do you want to use existing file or generate new one [0-existing / 1-new]? ");

BufferedReader console = new BufferedReader(new InputStreamReader(System.in));

File data;

if (Integer.parseInt(console.readLine()) > 0) {

System.out.print("What is size of wanted file in MB? ");

int mb = Integer.parseInt(console.readLine());

DataOutputStream dos = new DataOutputStream(new FileOutputStream(data = new File(path)));

Random random = new Random();

System.out.println("Generating file");

long start = System.currentTimeMillis();

if (mb <= 1 << 8) {

int[] tmp = new int[mb << 18];

for (int i = 0; i < mb << 18; i++)

tmp[i] = random.nextInt(intNull);

dos.write(toByte(tmp));

} else {

int[] tmp = new int[1 << 8];

for (int i = 0; i < mb << 10; i++) {

for (int j = 0; j < 1 << 8; j++)

tmp[j] = random.nextInt(intNull);

dos.write(toByte(tmp));

}

}

System.out.println("Successfully generated file of " + mb + " MB ");

System.out.println("Generation time:" + (System.currentTimeMillis() - start) + " ms");

dos.close();

} else {

System.out.print("Write path to the file: ");

data = new File(console.readLine());

}

console.close();

return data;

}

private static byte[] toByte(int[] d) throws IOException {

ByteArrayOutputStream bos = new ByteArrayOutputStream(d.length << 2);

DataOutputStream dos = new DataOutputStream(bos);

for (int i : d)

dos.writeInt(i);

return bos.toByteArray();

}

private static int ByteBufferToInt() {

return byteBuffer[0] << 24 | (byteBuffer[1] & 0xFF) << 16 | (byteBuffer[2] & 0xFF) << 8

| (byteBuffer[3] & 0xFF);

}

private static void MergeSort(File main\_file) throws IOException {

DataInputStream[] run\_files\_dis = new DataInputStream[N + 1];

File[] working\_files = new File[N + 1];

for (int i = 0; i < working\_files.length; i++)

working\_files[i] = new File("work\_temp\_" + (i + 1) + ".bin");

DistributeRuns(N, working\_files, new DataInputStream(new FileInputStream(main\_file)));

long start = System.currentTimeMillis();

int min\_dummy\_values = GetMinDummyValue();

InitMerge(min\_dummy\_values);

DataOutputStream dos = new DataOutputStream(new FileOutputStream(working\_files[outFileIndex]));

for (int i = 0; i < run\_files\_dis.length - 1; i++)

run\_files\_dis[i] = new DataInputStream(new FileInputStream(working\_files[i]));

while (runsPerLevel > 0) {

runsLastElems[outFileIndex] = intNull;

MergeProcedure(runsDistribution[getMinFileIndex()] - min\_dummy\_values, run\_files\_dis, dos);

setPreviousRunDistributionLevel();

outFileIndex = (outFileIndex > 0 ? outFileIndex : runsDistribution.length) - 1;

resetAllowReadArray();

min\_dummy\_values = GetMinDummyValue();

dos = new DataOutputStream(new FileOutputStream(working\_files[outFileIndex]));

run\_files\_dis[previousOutFileIndex] = new DataInputStream(

new FileInputStream(working\_files[previousOutFileIndex]));

}

System.out.println("Merging done " + (System.currentTimeMillis() - start) + " ms");

dos.close();

closeAll(run\_files\_dis);

outHelpFiles(main\_file, working\_files);

}

private static void resetAllowReadArray() {

Arrays.fill(fileCanBeRead, true);

fileCanBeRead[outFileIndex] = false;

}

private static int GetMinDummyValue() {

int min = dummyRuns[0];

for (int i = 1; i < dummyRuns.length; i++)

if (dummyRuns[i] < min)

if (i != outFileIndex)

min = dummyRuns[i];

return min;

}

private static void DistributeRuns(int temp\_files, File[] working\_files, DataInputStream main\_file\_dis)

throws IOException {

long start = System.currentTimeMillis();

runsPerLevel = 1;

runsDistribution[0] = 1;

outFileIndex = working\_files.length - 1;

int[] write\_sentinel = new int[temp\_files];

DataOutputStream[] run\_files\_dos = new DataOutputStream[temp\_files];

for (int i = 0; i < temp\_files; i++)

run\_files\_dos[i] = new DataOutputStream(new FileOutputStream(working\_files[i]));

while (processedData < unsortedFileLen) {

for (int i = 0; i < temp\_files; i++)

while (write\_sentinel[i] != runsDistribution[i]) {

while (processedData < unsortedFileLen && nextRunFirstElem != intNull

&& run\_last\_elements[i] <= nextRunFirstElem)

WriteNextRun(main\_file\_dis, run\_files\_dos[i], i);

WriteNextRun(main\_file\_dis, run\_files\_dos[i], i);

dummyRuns[i]++;

write\_sentinel[i]++;

}

setNextDistributionLevel();

}

setPreviousRunDistributionLevel();

for (int i = 0; i < runsDistribution.length - 1; i++)

dummyRuns[i] = runsDistribution[i] - dummyRuns[i];

System.out.println("Distribution done in " + (System.currentTimeMillis() - start) + " ms");

}

private static void WriteNextRun(DataInputStream main\_file\_dis, DataOutputStream run\_file\_dos,

int file\_index) throws IOException {

if (processedData >= unsortedFileLen) {

dummyRuns[file\_index]--;

return;

}

if (nextRunFirstElem != intNull) {

run\_file\_dos.writeInt(nextRunFirstElem);

processedData += intSize;

}

int min\_value = Integer.MIN\_VALUE;

if (main\_file\_dis.read(byteBuffer) < intSize)

return;

int current\_int = ByteBufferToInt();

while (current\_int != intNull) {

if (current\_int >= min\_value) {

run\_file\_dos.writeInt(current\_int);

processedData += intSize;

min\_value = current\_int;

if (main\_file\_dis.read(byteBuffer) < intSize)

break;

current\_int = ByteBufferToInt();

} else {

nextRunFirstElem = current\_int;

run\_last\_elements[file\_index] = min\_value;

break;

}

}

}

private static void InitMerge(int min\_dummy) {

for (int i = 0; i < dummyRuns.length - 1; i++)

dummyRuns[i] -= min\_dummy;

dummyRuns[outFileIndex] += min\_dummy;

resetAllowReadArray();

}

private static void MergeProcedure(int min\_file\_values, DataInputStream[] run\_files\_dis,

DataOutputStream writer)

throws IOException {

int num, min\_file, heap\_empty = 0;

Element element;

FillQueue(run\_files\_dis);

while (heap\_empty != min\_file\_values) {

if ((element = queue.poll()) == null)

return;

writer.writeInt(element.getValue());

min\_file = element.getFileIndex();

if (fileCanBeRead[min\_file])

if ((num = ReadInteger(run\_files\_dis[min\_file], min\_file)) != intNull)

queue.add(new Element(num, min\_file));

if (queue.size() == 0) {

heap\_empty++;

for (int i = 0; i < nextRunFirstElems.length; i++) {

if (nextRunFirstElems[i] == null)

break;

queue.add(new Element(nextRunFirstElems[i].getValue(), i));

runsLastElems[i] = nextRunFirstElems[i].getValue();

}

FillQueue(run\_files\_dis);

resetAllowReadArray();

}

}

}

private static void FillQueue(DataInputStream[] run\_files\_dis) throws IOException {

for (int i = 0; i < run\_files\_dis.length; i++)

if (dummyRuns[i] == 0) {

if (fileCanBeRead[i])

queue.add(new Element(ReadInteger(run\_files\_dis[i], i), i));

} else

dummyRuns[i]--;

}

private static int ReadInteger(DataInputStream file\_dis, int file\_index) throws IOException {

if (file\_dis.read(byteBuffer) < intSize)

return intSize;

int current\_int = ByteBufferToInt();

if (runsLastElems[file\_index] != intSize)

if (current\_int < runsLastElems[file\_index]) {

nextRunFirstElems[file\_index] = new Element(current\_int, file\_index);

fileCanBeRead[file\_index] = false;

return intSize;

}

return runsLastElems[file\_index] = current\_int;

}

private static int getMinFileIndex() {

int min\_file\_index = 0, min = runsDistribution[0];

for (int i = 1; i < runsDistribution.length; i++)

if (runsDistribution[i] != 0)

if (runsDistribution[i] < min)

min\_file\_index = i;

return min\_file\_index;

}

private static void setNextDistributionLevel() {

runsPerLevel = 0;

int[] clone = runsDistribution.clone();

for (int i = 0; i < clone.length - 1; runsPerLevel += runsDistribution[++i])

runsDistribution[i] = clone[0] + clone[i + 1];

}

private static void setPreviousRunDistributionLevel() {

int[] clone = runsDistribution.clone();

previousOutFileIndex = outFileIndex;

runsDistribution[0] = runsPerLevel = clone[clone.length - 2];

for (int diff, i = clone.length - 3; i >= 0; i--, runsPerLevel += diff)

runsDistribution[i + 1] = diff = clone[i] - clone[clone.length - 2];

}

private static void outHelpFiles(File main\_file, File[] temp\_files) throws IOException {

for (File temp\_file : temp\_files) {

System.out.print(

"Unsorted file size: " + main\_file.length() + " - Work file size: " + temp\_file.length());

System.out.println(temp\_file.length() != 0

? " - Is file " + temp\_file.getName() + " sorted? - " + isFileSorted(temp\_file)

: "");

}

}

private static boolean isFileSorted(File temp\_file) throws IOException {

DataInputStream dis = new DataInputStream(new FileInputStream(temp\_file));

if (dis.read(byteBuffer) < intSize)

return false;

int first = ByteBufferToInt();

if (dis.skip(temp\_file.length() - 8) <= 0)

return false;

if (dis.read(byteBuffer) < intSize)

return false;

dis.close();

return first <= ByteBufferToInt();

}

private static void closeAll(Closeable[] c) throws IOException {

for (Closeable o : c)

o.close();

}

}

}

Висновок

При виконанні даної лабораторної роботи я вивчив теоретичну частину основних алгоритмів зовнішнього сортування та способів їх модифікації, оцінив поріг їх ефективності та використав набуті знання на практиці.