**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-12 Сімчук Андрій Володимирович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов Олексій Олександрович*

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

iteration = 1

**while** (2 \*\* (iteration + 1) <= n) **do**

**open** ‘A.bin’ **for reading**

**open** ‘B.bin**’ for writing**

**open** ‘V.bin’ **for writing**

i = 0

**while** (i < n) **do**

k1 = 0

**while** (k1 != 2 \*\* iteration) **do**

**write element to** ‘B.bin’ **from** ‘A.bin’

i += 1

k1 += 1

**end while**

k2 = 0

**while** (k2 != 2 \*\* iteration) **do**

**write element to** ‘C.bin’ **from** ‘A.bin’

i += 1

k2 += 1

**end while**

**end while**

**close** ‘A.bin’

**close** ‘B.bin’

**close** ‘C.bin’

**open** ‘A.bin’ **for writing**

**open** ‘B.bin**’ for reading**

**open** ‘V.bin’ **for reading**

i = 0

k1 = 1

k2 = 1

el1 = **read element from** ‘B.bin’

k1 += 1

el2 = **read element from** ‘C.bin’

k2 += 1

**while** (i < n) **do**

k1 = 1

k2 = 1

**while**(k1 != 2 \*\* iteration + 1 and k2 != 2 \*\* iteration + 1)**do**

**if** (el1 <= el2) **do**

**write element** el1 **to** ‘A.bin’

el1 = **read element from** ‘B.bin’

k1 += 1

i += 1

**else do**

**write element** el2 **to** ‘A.bin’

el2 = **read element from** ‘C.bin’

k2 += 1

i += 1

**while** (k1 != 2 \*\* iteration + 1) **do**

**write element** el1 **to** ‘A.bin’

k1 += 1

i += 1

el1 = **read element from** ‘B.bin’

**end while**

**while** (k2 != 2 \*\* iteration + 1) **do**

**write element** el2 **to** ‘A.bin’

k2 += 1

i += 1

el2 = **read element from** ‘C.bin’

**end while**

**end while**

**close** ‘A.bin’

**close** ‘B.bin’

**close** ‘C.bin’

**end while**

## Програмна реалізація алгоритму

### Вихідний код

### Файл ‘Lab1.py’

from func import \*

import time

from shutil import copyfile as copyf

n = 4194304

i = 0

# Algorithm

def unmodified\_alg(n, i):

generate\_file\_a(n)

t1 = time.time()

while (2 \*\* (i + 1) <= n):

split\_to\_files(n, i)

merge(n, i)

i += 1

t2 = time.time()

if (is\_sorted('A.bin')):

print("\nSORTED")

else:

print("\nNOT SORTED")

print("Used time: " + str(t2 - t1) + "s")

return

def modified\_alg(n, i):

generate\_file\_a(n)

t1 = time.time()

i = pre\_sort('A.bin')

copyf('preSortedFile.bin', 'A.bin')

while(2 \*\* (i + 1) <= n):

split\_to\_files(n, i)

merge(n, i)

i += 1

t2 = time.time()

if(is\_sorted('A.bin')):

print("\nSORTED")

else:

print("\nNOT SORTED")

print("Used time: " + str(t2 - t1) + "s")

return

unmodified\_alg(n, i)

modified\_alg(n, i)

### Файл ‘func.py’

from random import randrange as rand

import psutil

import math

def print\_file(path):

with open(path, "rb") as file:

num = file.read(32)

while num:

num = int.from\_bytes(num, 'big')

print(num, end = ' ')

num = file.read(32)

return

def clear\_file(path):

with open(path, "wb") as file:

file.write(b'')

def print\_all():

print\_file("A.bin")

print()

print\_file("B.bin")

print()

print\_file("C.bin")

print()

return

def generate\_file\_a(n):

fa = open("A.bin", 'wb')

for i in range(n):

fa.write(rand(0, 100).to\_bytes(32,'big'))

fa.close()

return

def split\_to\_files(n, iteration):

fa = open('A.bin', 'br')

fb = open('B.bin', 'wb')

fc = open('C.bin', 'wb')

i = 0

while(i < n):

k1 = 0

while(k1 != 2 \*\* iteration):

fb.write(fa.read(32))

i += 1

k1 += 1

k2 = 0

while (k2 != 2 \*\* iteration):

fc.write(fa.read(32))

i += 1

k2 += 1

fa.close()

fb.close()

fc.close()

return

def merge(n, iteration):

fa = open('A.bin', 'wb')

fb = open('B.bin', 'rb')

fc = open('C.bin', 'rb')

i = 0

k1 = 1

k2 = 1

el1 = int.from\_bytes(fb.read(32),'big')

k1 += 1

el2 = int.from\_bytes(fc.read(32),'big')

k2 += 1

while (i < n):

k1 = 1

k2 = 1

while (k1 != 2 \*\* iteration + 1 and k2 != 2 \*\* iteration + 1):

if(el1 <= el2):

fa.write(el1.to\_bytes(32,'big'))

el1 = int.from\_bytes(fb.read(32),'big')

k1 += 1

i += 1

else:

fa.write(el2.to\_bytes(32,'big'))

el2 = int.from\_bytes(fc.read(32),'big')

k2 += 1

i += 1

while (k1 != 2 \*\* iteration + 1):

fa.write(el1.to\_bytes(32, 'big'))

k1 += 1

i += 1

el1 = int.from\_bytes(fb.read(32), 'big')

while (k2 != 2 \*\* iteration + 1):

fa.write(el2.to\_bytes(32, 'big'))

k2 += 1

i += 1

el2 = int.from\_bytes(fc.read(32), 'big')

fa.close()

fb.close()

fc.close()

return

def is\_sorted(path):

with open(path, "rb") as file:

prev = file.read(32)

num = file.read(32)

while num:

if (int.from\_bytes(prev, 'big') > int.from\_bytes(num, 'big')):

return False

prev = num

num = file.read(32)

return True

# Functions for modified algorithm

def pre\_sort(path):

available\_ints = int(psutil.virtual\_memory()[1]/32)

clear\_file("preSortedFile.bin")

with open(path, "rb") as file:

int\_list = []

while True:

num = file.read(32)

if num == b'':

break

for i in range(available\_ints):

int\_list.append(int.from\_bytes(num, 'big'))

num = file.read(32)

if num == b'':

break

int\_list.sort()

with open('preSortedFile.bin', "ab") as preFile:

for i in int\_list:

preFile.write(i.to\_bytes(32, 'big'))

return math.log2(available\_ints)

Висновок

При виконанні даної лабораторної роботи вивчив основні алгоритми зовнішнього сортування та способи їх модифікації, оцінив поріг їх ефективності.

Реалізував алгоритм «Пряме злиття». Модифікація представляє собою попереднє сортування груп чисел (скільки дозволить пам’ять) в початковому файлі і виконування алгоритму прямого злиття з меншою кількістю ітерацій.

Як ефект модифікації алгоритм почав сортувати файл набагато швидше за немодифікований.

Критерії оцінювання

У випадку здачі лабораторної роботи до 09.10.2022 включно максимальний бал дорівнює – 5. Після 09.10.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 15%;
* програмна реалізація алгоритму – 40%;
* програмна реалізація модифікацій – 40%;
* висновок – 5%.