**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 Василишин Михайло Михайлович*

**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

pointerB = 0

pointerC = 0

while (!finB.eof())do

in(tempB)

if (finC.eof)do

if (tempB < tempC)do

Out(tempB)

pointerB++

end if

if (tempB >= tempC && !endC)do

endC = true;

out(tempC)

pointerC++

out(tempB)

pointerB++

else if (tempB >= tempC && endC)do

out(tempB)

pointerB++;

end if

end if

if (isstartSegment && !finC.eof)do

while (!finC.eof() and pointerC < length)do

in(tempC)

if (tempB < tempC)do

out(tempB)

pointerB++;

break;

else fo

out(tempC)

pointerC++;

if (finC.eof())do endC = true

end if

end if

End while

if (tempC <= tempB) do

foutA << " " << tempB;

pointerB++;

end if

else

if (!isstartSegment and !finC.eof )do

if (pointerC == length or tempB < tempC)do

out(tempB)

pointerB++;

end if

else

do

out(tempC)

pointerC++

while (!finC.eof() and pointerC < length)do

int(tempC)

if (tempB < tempC)do

out(tempB)

pointerB++;

break;

end if

else do

out(tempC)

pointerC++

if (finC.eof())do endC = true

end if

end else

End while

if (tempC <= tempB)fo

out(tempB)

pointerB++

end if

end if

end of

if (pointerB == length && pointerC != length)do

if (finC.eof() and tempB < tempC and !endC

do

Out(tempC)

pointerC++

if (finC.eof())do endC = true

end if

end of

else do

if (!finC.eof())do

out(tempC)

pointerC++

if (finC.eof())do endC = true;

end if

while (!finC.eof() && pointerC < length)do

in(tempC)

out(tempC)

if (finC.eof())do endC = true

pointerC++

end while

End if

End if

if (pointerC == length and pointerB == length)do

isstartSegment = true

pointerB = 0

pointerC = 0

end if

else do isstartSegment = false;

end else

end while

## Програмна реалізація алгоритму

### Вихідний код

#pragma once

#include<iostream>

#include<fstream>

#include<time.h>

#include<windows.h>

#include<string.h>

using namespace std;

void generateFile(string fileName, double sizeArray);

int generateNumber(int min, int max);

void openFile(string FileName);

#define \_CRT\_SECURE\_NO\_WARNINGS

#include"auxiliary.h"

void generateFile(string fileName, double sizeArray)

{

ofstream fout(fileName);

srand(time(NULL));

for (int i = 0; i < sizeArray; i++)fout << generateNumber(1, 100) << " ";

fout.close();

}

int generateNumber(int min, int max)

{

return min + rand() % (max - min + 1);

}

void openFile(string fileName)

{

char\* fileNameChar = new char[fileName.length()];

strcpy(fileNameChar, fileName.c\_str());

system(fileNameChar);

}

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<fstream>

#include<sstream>

#include<algorithm>

using namespace std;

void sort(string fileNameA, string fileNameB, string fileNameC,int length);

void splitIntoFiles(string fileNameA, string fileNameB, string fileNameC, int length);

void mergingFile(string fileNameA, string fileNameB, string fileNameC, int length);

void preliminarySorting(string fileNameA, string fileNameB,int size);

#include"sort.h"

void splitIntoFiles(string fileNameA, string fileNameB, string fileNameC, int length)

{

ifstream finA(fileNameA);

ofstream foutB(fileNameB);

ofstream foutC(fileNameC);

int temp, count = 1;

bool writeToB = true;

while (!finA.eof())

{

finA >> temp;

if (writeToB)foutB << " " << temp;

else foutC << " " << temp;

count++;

if (count > length)

{

count = 1;

writeToB = !writeToB;

}

}

foutB << "";

foutC << "";

finA.close();

foutB.close();

foutC.close();

}

void mergingFile(string fileNameA, string fileNameB, string fileNameC, int length)

{

ifstream finB(fileNameB);

ifstream finC(fileNameC);

ofstream foutA(fileNameA);

int pointerB = 0, pointerC = 0;

int tempB, tempC;

bool isstartSegment = true;

bool endC = false;

while (!finB.eof())

{

finB >> tempB;

if (finC.eof())// досягнутий кінець файлу C, але не відомо, чи записався останній елемент

{

if (tempB < tempC)

{

foutA << " " << tempB;

pointerB++;

}

if (tempB >= tempC && !endC)

{

endC = true;

foutA << " " << tempC;

pointerC++;

foutA << " " << tempB;

pointerB++;

}

else if (tempB >= tempC && endC)

{

foutA << " " << tempB;

pointerB++;

}

}

if (isstartSegment && !finC.eof())//об'єднання нових нової пари сегментів

{

while (!finC.eof() && pointerC < length)

{

finC >> tempC;

if (tempB < tempC)

{

foutA << " " << tempB;

pointerB++;

break;

}

else

{

foutA << " " << tempC;

pointerC++;

if (finC.eof())endC = true;

}

}

if (tempC <= tempB)

{

foutA << " " << tempB;

pointerB++;

}

}

else

if (!isstartSegment && !finC.eof())//об'єднання пари сегментів, що вже опрацбовувались на попередніх ітераціях

{

if (pointerC == length || tempB < tempC)

{

foutA << " " << tempB;

pointerB++;

}

else

{

foutA << " " << tempC;

pointerC++;

while (!finC.eof() && pointerC < length)

{

finC >> tempC;

if (tempB < tempC)

{

foutA << " " << tempB;

pointerB++;

break;

}

else

{

foutA << " " << tempC;

pointerC++;

if (finC.eof())endC = true;

}

}

if (tempC <= tempB)

{

foutA << " " << tempB;

pointerB++;

}

}

}

if (pointerB == length && pointerC != length)//досягнутий кінець файлу B, при цьому останній елемент файлу успішно записаний

{

if (finC.eof() && tempB < tempC && !endC)//записуємо останній елемент файлу C, при умові що зчитали весь файл C,але останній елемент поки що не записався

{

foutA << " " << tempC;

pointerC++;

if (finC.eof())endC = true;

}

else

if (!finC.eof())//записуємо всі елемент файлу C, що залишились

{

foutA << " " << tempC;

pointerC++;

if (finC.eof())endC = true;

while (!finC.eof() && pointerC < length)

{

finC >> tempC;

foutA << " " << tempC;

if (finC.eof())endC = true;

pointerC++;

}

}

}

if (pointerC == length && pointerB == length)// опрацювання пари сегментів з файлу B та С завершилось

{

isstartSegment = true;

pointerB = 0;

pointerC = 0;

}

else isstartSegment = false;

}

finB.close();

finC.close();

foutA.close();

}

void sort(string fileNameA, string fileNameB, string fileNameC,int length)

{

while (1)

{

splitIntoFiles(fileNameA, fileNameB, fileNameC, length);

ifstream fileC(fileNameC);

if (fileC.peek() == std::ifstream::traits\_type::eof())break;

mergingFile(fileNameA, fileNameB, fileNameC, length);

length \*= 2;

cout << length << endl;

}

}

void preliminarySorting(string fileNameA, string fileNameB,int size)

{

int\* arrayT = new int[size];

ifstream finA(fileNameA);

ofstream foutB(fileNameB);

int temp,iter=0;

while (!finA.eof())

{

finA >> arrayT[iter];

iter++;

if (iter == size)

{

sort(arrayT, arrayT + size);

for (int i = 0; i < size; i++)foutB << " " << arrayT[i];

iter = 0;

}

}

if (iter != 0)

{

sort(arrayT, arrayT + iter);

for (int i = 0; i < iter; i++)foutB << " " << arrayT[i];

}

finA.close();

foutB.close();

}

#include <iostream>

#include"auxiliary.h"

#include"sort.h"

int main()

{

generateFile("t.txt", 7\*1024\*512);

preliminarySorting("t.txt", "in.txt", 1048576);

sort("in.txt", "b.txt", "c.txt", 1048576);

std::cout << "Hello World!\n";

return 0;

}

Висновок

При виконанні даної лабораторної роботи було досліджено методи зовнішнього сортування, а також ознайомлено з основними варіантами реалізації таких алгоритмів. Особливістю мого варіанту було використання прямого злиття, яке в порівнянні з іншими реалізаціями алгоритму є менш ефективним. При опрацюванні файлу з 10MB програма витрачала при немодифікованому алгоритмі 100с на одну процедуру розподілу і злиття, а оскільки асимптотична складність алгоритму є логарифмічною, то при кількості елементів більше 2 млн, кількість таких процедур дорівнювала 22-ом. А отже, загальний час визначався як – 100с\*22. У модифікованому алгоритмі процедури розподілу та злиття починались не з відрізків розмірності 1 елемента, а з 1048576 елементів. Тому що ідея модифікація полягала у тому, що заздалегідь до виконання основного алгоритму, було підготовлено серії великої розмірності, які в свою чергу було результатом внутрішнього сортування, а саме quicksort.

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%.