**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-14 Медвідь Олександр*

**Перевірив**

(прізвище, ім'я, по батькові)

*Соколовський В. В.*

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 7](#_Toc109342188)

[3.2.1 Вихідний код 7](#_Toc109342189)

[Висновок 16](#_Toc109342190)

[Критерії оцінювання 17](#_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

F2.open()

**while** (InBlockR.NumNRead(&Number, F2) **do**

**while** (Number >= temp) **do**

BlockW[i].NumNWrite(Number, \*(StreamOf[i]));

temp = Number

**if** (!InBlockR.NumNRead(&Number, F2))

break

**if** (i == Num - 1)

Bused = true

i = (i + 1) % Num

BlockW[i].NumNWrite(Number, \*(StreamOf[i]))

temp = Number

F2.close();

**for** (int i = 0; i < Num; i++)

BlockW[i].BlockWrite(\*(StreamOf[i]))

CloseStream<ofstream\*\*>(StreamOf, Num)

**if** (!Bused)

Num = i + 1

flag = 1

**while** (flag) **do**

**if** (flag % 2 == 1)

FileName1 = "B"

FileName2 = "C"

**else**

FileName1 = "C"

FileName2 = "B"

OpenStream<ifstream>(FileName1, Num, StreamIf)

OpenStream<ofstream>(FileName2, Num, StreamOf)

SeriesFunctions\* series = new SeriesFunctions[Num]

**for** (int i = 0; i < Num; i++)

**if** (BlockR[i].NumNRead(&temp, \*(StreamIf[i])))

series[i].AddElement(temp)

**else**

series[i].SetFE(true)

counter = 0

AllDone = false

InFlag = true

**while** (InFlag) **do**

CheckNum = false;

**while** (!CheckNum) **do**

CheckNum = true;

unsigned long long min = ULLONG\_MAX

**for** (int j = 0; j < Num; j++)

**if** (series[j].GetSeries() && !series[j].GetFE())

**if** (series[j].GetCurrent() <= min)

min = series[j].GetCurrent()

pos = j

CheckNum = false

**if** (!CheckNum)

BlockW[counter].NumNWrite(min, \*(StreamOf[counter]))

**if** (BlockR[pos].NumNRead(&temp, \*(StreamIf[pos])))

series[pos].AddElement(temp)

**else**

series[pos].SetFE(true)

counter++;

**if** (counter == Num)

counter = 0;

AllDone = true;

InFlag = false;

**for** (int j = 0; j < Num; j++)

series[j].SetSeries(true)

**if** (!series[j].GetFE())

InFlag = true

**for** (int i = 0; i < Num; i++)

BlockR[i].Clear()

BlockW[i].BlockWrite(\*(StreamOf[i]))

CloseStream(StreamIf, Num)

CloseStream(StreamOf, Num)

**if** (!AllDone)

Num = counter

flag++

**if** (Num == 1)

flag = 0

## Програмна реалізація алгоритму

### Вихідний код

Файл Main.cpp

#include "FileFunctions.h"

#include <string>

int main()

{

srand(time(NULL));

int Choice, Size;

string FileName, EndFile;

for (int i = 0; ; i++)

{

cout << "Enter 1 to create a file\nEnter 2 to read a file\nEnter 3 to sort a file\nEnter anything else to exit\nEnter the number ";

cin >> Choice;

if (Choice == 1)

{

cout << "Enter the name of the file ";

cin.ignore();

getline(cin, FileName);

cout << endl << "Enter the size of the file in mb ";

cin >> Size;

FileCreate(FileName, Size \* 131072);

cout << "File " << FileName << " is created" << endl;

}

else if (Choice == 2)

{

cin.ignore();

cout << "Enter the name of file: ";

getline(cin, FileName);

cout << "Enter the the quantity of displayed numbers ";

cin >> Size;

FileRead(FileName, Size);

cout << endl;

}

else if (Choice == 3)

{

if (FileName == "")

{

cout << "Enter the name of the file ";

cin.ignore();

getline(cin, FileName);

}

for (int i = 0;; i++)

{

cout << "Enter the number of files to divide the original one from 2 to 10 ";

cin >> Size;

if (Size < 2 || Size > 10)

{

cout << "\nIncorrect input, please try again" << endl;

}

else

{

break;

}

}

EndFile = FileSort(Size, FileName);

cout << "The file is sorted, to check the results, read the file " << EndFile << "1.txt\n\n";

}

else

{

break;

}

}

}

Файл FileFunctions.h

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "SeriesFunctions.h"

#include "Block.h"

#include <iostream>

#include <fstream>

using namespace std;

void FileCreate(string FileName, unsigned long long Count);

void FileRead(string FileName, int Count);

string FileSort(int, string);

template <typename T>

void OpenStream(string Letter, int Quant, T\*\* StreamArr)

{

for (int i = 0; i < Quant; i++)

{

char\* Pos = new char;

\_itoa(i + 1, Pos, 10);

StreamArr[i] = new T(Letter + string(Pos) + ".txt", ios::binary);

}

}

template <typename T>

void CloseStream(T StreamArr, int Quant)

{

for (int i = 0; i < Quant; i++)

{

StreamArr[i]->close();

}

}

Файл FileFunctions.cpp

#include "FileFunctions.h"

void FileCreate(string FileName, unsigned long long Count)

{

ofstream F1(FileName, ios::binary);

unsigned long long Num;

Block block;

for (int i = 0; i < Count; i++)

{

Num = rand() + 1;

block.NumNWrite(Num, F1);

}

block.BlockWrite(F1);

F1.close();

}

void FileRead(string FileName, int Count)

{

ifstream F2(FileName, ios::binary);

unsigned long long Num, Prev;

Prev = 0;

Block block;

bool flag = true;

//Ми зчитуєм з файла число Num і якщо не дублюєтся, то виводимо

for (int i = 0; i < Count && block.NumNRead(&Num, F2); i++)

{

if (Num != Prev)

{

cout << Num << " ";

Prev = Num;

}

else

{

i--;

}

}

F2.close();

}

string FileSort(int Num, string FileName)

{

//Відкриваємо масиви ifstream та ofstream для запису/зчитування розподільчих файлів

ofstream\*\* StreamOf = new ofstream\*[Num];

OpenStream<ofstream>("B", Num, StreamOf);

ifstream\*\* StreamIf = new ifstream\*[Num];

ifstream F2(FileName, ios::binary);

unsigned long long Number, temp = 0;

bool Bused = false;

int i = 0;

Block InBlockR;

Block\* BlockR = new Block[Num];

Block\* BlockW = new Block[Num];

//Доки

while (InBlockR.NumNRead(&Number, F2))

{

while (Number >= temp)

{

BlockW[i].NumNWrite(Number, \*(StreamOf[i]));

temp = Number;

if (!InBlockR.NumNRead(&Number, F2))

{

break;

}

}

//Перевірка на запис усіх файлів

if (i == Num - 1)

{

Bused = true;

}

//Перенос ітератора-покажчика на наступний файл для щапису серій

i = (i + 1) % Num;

//Запис числа, яке не ввійшло у попередню серію

BlockW[i].NumNWrite(Number, \*(StreamOf[i]));

temp = Number;

}

F2.close();

for (int i = 0; i < Num; i++)

{

BlockW[i].BlockWrite(\*(StreamOf[i]));

}

CloseStream<ofstream\*\*>(StreamOf, Num);

//Якщо не усі файли були використані, то ми змінюємо загальну кількість файлів

if (!Bused)

{

Num = i + 1;

}

int flag = 1;

string FileName1, FileName2;

int counter;

while (flag)

{

//Почерговий запис із файлі В у С і навпаки

if (flag % 2 == 1)

{

FileName1 = "B";

FileName2 = "C";

}

else

{

FileName1 = "C";

FileName2 = "B";

}

OpenStream<ifstream>(FileName1, Num, StreamIf);

OpenStream<ofstream>(FileName2, Num, StreamOf);

SeriesFunctions\* series = new SeriesFunctions[Num];

//Збирання перших чисел розподільчих файлів

for (int i = 0; i < Num; i++)

{

if (BlockR[i].NumNRead(&temp, \*(StreamIf[i])))

{

series[i].AddElement(temp);

}

else

{

series[i].SetFE(true);

}

}

counter = 0;

bool AllDone = false;

bool InFlag = true;

//Злиття і запис серій

while (InFlag)

{

bool CheckNum = false;

while (!CheckNum)

{

CheckNum = true;

int pos;

unsigned long long min = ULLONG\_MAX;

//Пошук мінімального числа з початків серій

for (int j = 0; j < Num; j++)

{

if (series[j].GetSeries() && !series[j].GetFE())

{

if (series[j].GetCurrent() <= min)

{

min = series[j].GetCurrent();

pos = j;

}

CheckNum = false;

}

}

//Запис мінімального числа у файл та взяття наступного числа з того ж файлу

if (!CheckNum)

{

BlockW[counter].NumNWrite(min, \*(StreamOf[counter]));

if (BlockR[pos].NumNRead(&temp, \*(StreamIf[pos])))

{

series[pos].AddElement(temp);

}

else

{

series[pos].SetFE(true);

}

}

}

counter++;

//Перевірка на запис усіх файлів

if (counter == Num)

{

counter = 0;

AllDone = true;

}

InFlag = false;

//Перевірка на зчитування всіх файлів

for (int j = 0; j < Num; j++)

{

series[j].SetSeries(true);

if (!series[j].GetFE())

{

InFlag = true;

}

}

}

//Дозаписування та доочищування усіх блоків

for (int i = 0; i < Num; i++)

{

BlockR[i].Clear();

BlockW[i].BlockWrite(\*(StreamOf[i]));

}

CloseStream(StreamIf, Num);

CloseStream(StreamOf, Num);

if (!AllDone)

{

Num = counter;

}

flag++;

if (Num == 1)

{

flag = 0;

}

delete[] series;

}

delete[] BlockR;

delete[] BlockW;

delete[] StreamIf;

delete[] StreamOf;

return FileName2;

}

Файл Block.h

#pragma once

#include <fstream>

using namespace std;

class Block

{

unsigned long long\* Data;

int Size;

int Iter;

public:

Block(int num = 4 \* 1048576);

bool BlockRead(ifstream& file);

bool GetNum(unsigned long long\* num);

bool NumNRead(unsigned long long\* num, ifstream& file);

bool AddNum(unsigned long long num);

void BlockWrite(ofstream& file);

void NumNWrite(unsigned long long num, ofstream& file);

void Clear();

~Block();

};

Файл Block.cpp

#include "Block.h"

Block::Block(int num)

{

Data = new unsigned long long[num];

Size = num;

Clear();

}

bool Block::BlockRead(ifstream& file)

{

Clear();

//Зчитуєм з файла блок

if (file.read((char\*)Data, Size \* sizeof(unsigned long long)))

{

return true;

}

else

{

return false;

}

}

bool Block::GetNum(unsigned long long\* num)

{

if (Iter < Size)

{

//Якщо ітератор не вийшов за межі блока, то повертаєм поточне значення

if (Data[Iter] != ULLONG\_MAX)

{

\*num = Data[Iter];

Iter++;

return true;

}

}

return false;

}

bool Block::NumNRead(unsigned long long\* num, ifstream& file)

{

//Якщо ітератор вийшов за межі, то заново починаєм зчитувати блок

if (!GetNum(num))

{

if (BlockRead(file))

{

return GetNum(num);

}

else

{

return false;

}

}

else

{

return true;

}

}

bool Block::AddNum(unsigned long long num)

{

//Додаємо число до блоку, якщо є вільне місце

if (Iter < Size)

{

Data[Iter] = num;

Iter++;

return true;

}

return false;

}

void Block::BlockWrite(ofstream& file)

{

//Якщо наявен хоча б один елемент у блоці, то записуєм його у файл

if (Iter > 0)

{

file.write((char\*)Data, sizeof(unsigned long long) \* Size);

}

Clear();

}

void Block::NumNWrite(unsigned long long num, ofstream& file)

{

//Якщо блок повний, то записуєм існуюючі числа у файл, очищаєм та записуєм число

if (!AddNum(num))

{

BlockWrite(file);

Clear();

AddNum(num);

}

}

void Block::Clear()

{

//Звільняєм блок від значень

for (int i = 0; i < Size; i++)

{

Data[i] = ULLONG\_MAX;

}

Iter = 0;

}

Block::~Block()

{

delete[Size] Data;

}

Файл SeriesFunctions.h

#pragma once

#include <climits>

class SeriesFunctions

{

unsigned long long Previous, Current;

bool Series, FileEnd;

public:

SeriesFunctions();

bool GetSeries();

void SetSeries(bool flag);

void SetFE(bool flag);

bool GetFE();

bool AddElement(unsigned long long num);

unsigned long long GetCurrent();

void Clear();

};

Файл SeriesFunctions.cpp

#include "SeriesFunctions.h"

SeriesFunctions::SeriesFunctions()

{

Current = Previous = 0;

Series = true;

FileEnd = false;

}

bool SeriesFunctions::GetSeries()

{

return Series;

}

void SeriesFunctions::SetSeries(bool flag)

{

Series = flag;

}

void SeriesFunctions::SetFE(bool flag)

{

FileEnd = flag;

}

bool SeriesFunctions::GetFE()

{

return FileEnd;

}

//Додавання елементу та подальше визначення сутності серії

bool SeriesFunctions::AddElement(unsigned long long num)

{

Previous = Current;

Current = num;

if (Previous <= Current && Current != ULLONG\_MAX)

{

Series = true;

}

else

{

Series = false;

}

return Series;

}

unsigned long long SeriesFunctions::GetCurrent()

{

return Current;

}

void SeriesFunctions::Clear()

{

Previous = 0;

Current = 0;

Series = true;

}

Висновок

Під час даної лабораторної роботи були досліджені алгоритми зовнішнього сортування. Були набуті навички їх використання у програмних специфікаціях. Був розлянут метод збалансованого багатошляхового злиття. Для обчислення виразу була створена елементарна програма. Результати програми виявилися правильними, що стверджує на її дієвість. Завдання було виконано на мові програмування С++.

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%.