**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 Бондаренко М.В.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

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. Створити список, що зберігає назву вхідного файлу.
2. Створити списки, що зберігають назви допоміжних файлів.
3. Створити пусті файли B та С для подальшого використання.
4. Розподілити дані основного файлу по допоміжним.
5. Ініціалізувати змінну flag = 1 для перемикання злиття файлів.
6. ПОКИ нема відсортованого файлу
   1. ЯКЩО flag = 1
      1. Злити B файли в С.
   2. ІНАКШЕ
      1. Злити С файли в В.
   3. ВСЕ ЯКЩО
7. ВСЕ ПОКИ

Алгоритм злиття:

1. Ініціалізувати список sequence для збереження послідовностей.
2. Ініціалізувати списки потоків зчитування та запису файлів.
3. Ініціалізувати змінну j = 0 для переміщення між файлами для запису.
4. ПОКИ не дійшли до кінця всіх файлів з даними
   1. Ініціалізувати змінну min\_number та min\_index.
   2. ЦИКЛ i ВІД 0 ДО кількість файлів для зчитування
      1. Зчитати значення файлу (number).
      2. ЯКЩО number існує
         1. ЯКЩО sequence пуста АБО number >= останнє значення sequence
            1. ЯКЩО number < min\_number

min\_number = number

min\_index = i

* + - * 1. ВСЕ ЯКЩО
      1. ВСЕ ЯКЩО
    1. ВСЕ ЯКЩО
  1. КІНЕЦЬ ЦИКЛУ
  2. ЯКЩО min\_index не змінився
     1. ЦИКЛ number В sequence
        1. Записати число в новий файл.
     2. ВСЕ ЦИКЛ
     3. Почистити sequence
     4. j = (j + 1) % кількість файлів
  3. ІНАКШЕ
     1. Записати в sequence min\_number
     2. Зчитуємо наступні дані з заданого файлу для зчитування.
  4. ВСЕ ЯКЩО

1. ВСЕ ПОКИ

## Програмна реалізація алгоритму

### Вихідний код

public static string BalancedKwayMerge(int files\_amount)  
{  
 List<string> A\_files\_list = new List<string>();  
 A\_files\_list.Add("A1.dat");  
 List<string> B\_files\_list = CreateFilesLists(files\_amount, 'B');  
 List<string> C\_files\_list = CreateFilesLists(files\_amount, 'C');  
 GenerateFile();  
 CreateFiles(files\_amount);  
   
 MultiwayMerge(A\_files\_list, B\_files\_list, files\_amount);  
  
 int flag = 1;  
 while (!isSorted(A\_files\_list, B\_files\_list, C\_files\_list))  
 {  
 if (flag == 1)  
 MultiwayMerge(B\_files\_list, C\_files\_list, files\_amount);  
 else  
 MultiwayMerge(C\_files\_list, B\_files\_list, files\_amount);  
 flag = -flag;  
 }  
   
 var A\_size = new FileInfo(A\_files\_list[0]).Length;  
 var B\_size = new FileInfo(B\_files\_list[0]).Length;  
 if (A\_size == B\_size)  
 return B\_files\_list[0];  
 return C\_files\_list[0];  
}  
  
private static Boolean isSorted(List<string> A\_files\_list, List<string> B\_files\_list, List<string> C\_files\_list)  
{  
 var A\_size = new FileInfo(A\_files\_list[0]).Length;  
 var B\_size = new FileInfo(B\_files\_list[0]).Length;  
 var C\_size = new FileInfo(C\_files\_list[0]).Length;  
 return A\_size == B\_size || A\_size == C\_size;  
}  
  
private static void MultiwayMerge(List<string> files\_to\_divide, List<string> divided\_files, int amount\_of\_files)  
{  
 List<int> sequence = new List<int>();  
  
 List<Reader> readers\_list = new List<Reader>();  
 foreach (var file in files\_to\_divide)  
 {  
 var reader = new Reader(file);  
 readers\_list.Add(reader);  
 }  
  
 List<BinaryWriter> writers\_list = new List<BinaryWriter>();  
 foreach (var file in divided\_files)  
 {  
 BinaryWriter writer = new BinaryWriter(File.Open(file, FileMode.**Create**));  
 writers\_list.Add(writer);  
 }  
   
 int j = 0; *// index to move between files for writing* while (!isMerged(readers\_list))  
 {  
 int min\_number = 150000;  
 int min\_index = -1;  
  
 for (int i = 0; i < readers\_list.Count; i++)  
 {  
 var bin\_num = readers\_list[i].GetCurrent();  
 if (bin\_num.Length > 0)  
 {  
 var int\_num = BitConverter.ToInt32(bin\_num, 0);  
 if (sequence.Count == 0 || int\_num >= sequence.Last())  
 {  
 if (int\_num <= min\_number)  
 {  
 min\_number = int\_num;  
 min\_index = i;   
 }  
 }  
 }  
 }  
 if (min\_index == -1)  
 {  
 foreach (var number in sequence)  
 writers\_list[j].Write(BitConverter.GetBytes(number));  
 sequence.Clear();  
 j = (j + 1) % amount\_of\_files;  
 }  
 else  
 {  
 sequence.Add(min\_number);  
 readers\_list[min\_index].GetNext();  
 }  
 }  
  
 foreach (var number in sequence)  
 writers\_list[j].Write(BitConverter.GetBytes(number));  
  
 foreach (var reader in readers\_list)  
 reader.Close();  
  
 foreach (var writer in writers\_list)  
 writer.Close();  
}  
private static Boolean isMerged(List<Reader> readers\_list)  
{  
 foreach (var reader in readers\_list)  
 {  
 if (reader.GetCurrent().Length != 0)  
 return false;  
 }  
 return true;  
}

Висновок

При виконанні даної лабораторної роботи дослідив та реалізував алгоритм збалансованого багатошляхового злиття. Це є алгоритм зовнішнього сортування, що передбачений для роботи з великими файлами, але для досягнення нормальної швидкості роботи його треба модифікувати. Моя реалізація алгоритму виконує задачу сортування файла розміром 10 Мб за 8 секунд. Для розгляду різних ситуацій провів тести з діленням на 3, 10 та 50 файлів. З трьома файлами впорався за 8 секунд, з десятьма за той же час, з п’ятдесятьма - за 15 секунд. Звісно, за такого обсягу даних не раціонально використовувати алгоритми зовнішнього сортування, бо вони значно повільніше, але вони сильно потрібні для роботи з файлами, що більші за обсяги ОЗУ серверів.

Алгоритми зовнішнього сортування мають місце в природі та потрібні у деяких ситуаціях, але вони менш ефективні за алгоритми внутрішнього сортування.

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%.