Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**Комп’ютерний практикум №4**

з дисципліни «Технології паралельних обчислень»

Тема: «Розробка паралельних програм з використанням пулів потоків, екзек’юторів та ForkJoinFramework»

|  |  |
| --- | --- |
| **Виконав:**  студент групи ІТ-04  Коновальчук Андрій Володимирович  Дата здачі \_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_ | **Перевірила**:  асистент кафедри ІПІ  Дифучина Олександра Юріївна |

Київ 2023

**Хід роботи:**

***Завдання 1:*** Побудуйте алгоритм статистичного аналізу тексту та визначте характеристики випадкової величини «довжина слова в символах» з використанням ForkJoinFramework. Дослідіть побудований алгоритм аналізу текстових документів на ефективність експериментально.

*Опис пункту:*

Для виконання завдання нам потрібно проробити наступні дії:

1. Прочитати файл з певним текстом.
2. Розбити текст на окремі слова, використовуючи пробіли і знаки пунктуації.
3. Реалізувати стандартний алгоритм підрахунку слів.
4. Реалізувати клас оптимізованого алгоритму з ForkJoin.
5. Підрахувати різницю швидкості роботи алгоритмів.
6. Підрахувати кількість слів та унікальних слів.
7. Для кожного слова визначити його довжину в символах та створити словник з цими значеннями.
8. Обчислити середню довжину слова в символах (математичне очікування).
9. Обчислити розмах.
10. Обчистити дисперсію.
11. Обчислити стандартне відхилення.

*Фрагмент коду:*

Суть оптимізованого алгоритму дуже проста. По-перше, імлементуємо клас, що розширяє клас RecursiveTask та перевизначимо метод compute. Створимо додатковий метод createSubtasks, який рекурсивно буде розбивати набір слів навпіл. Методом invokeAll паралельно виконуємо усі subtasks (під-задачі). Обчислюємо довжини слів та заносимо їх у словник.

Створюємо пул потоків ForkJoin та сабмітимо (submit) об’єкт описаного вище класу. Опісля порівнюємо алгоритми та визначаємо характеристики випадкової величини.

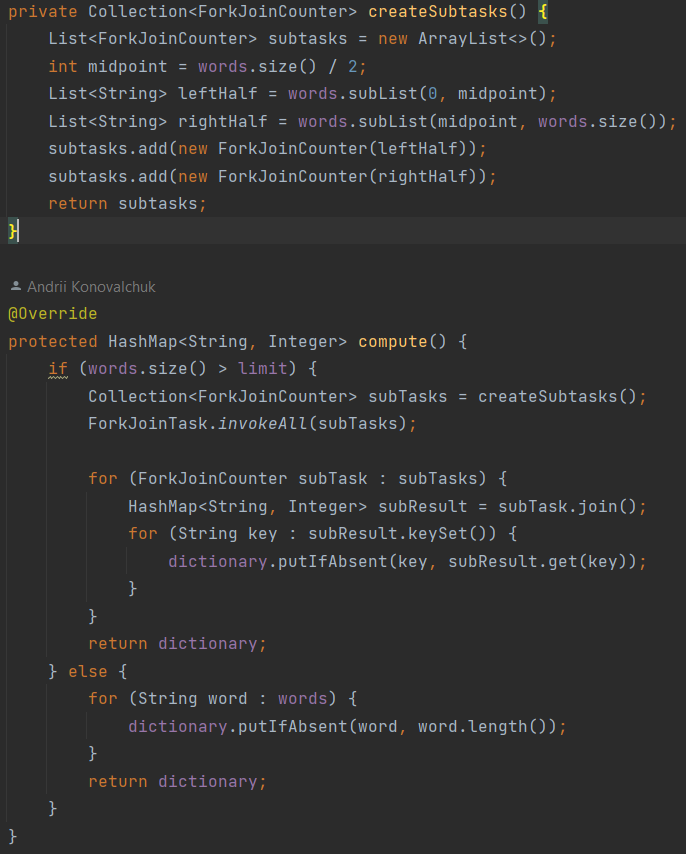


Рисунок 1 – Методи compute та createSubtask оптимізованого методу

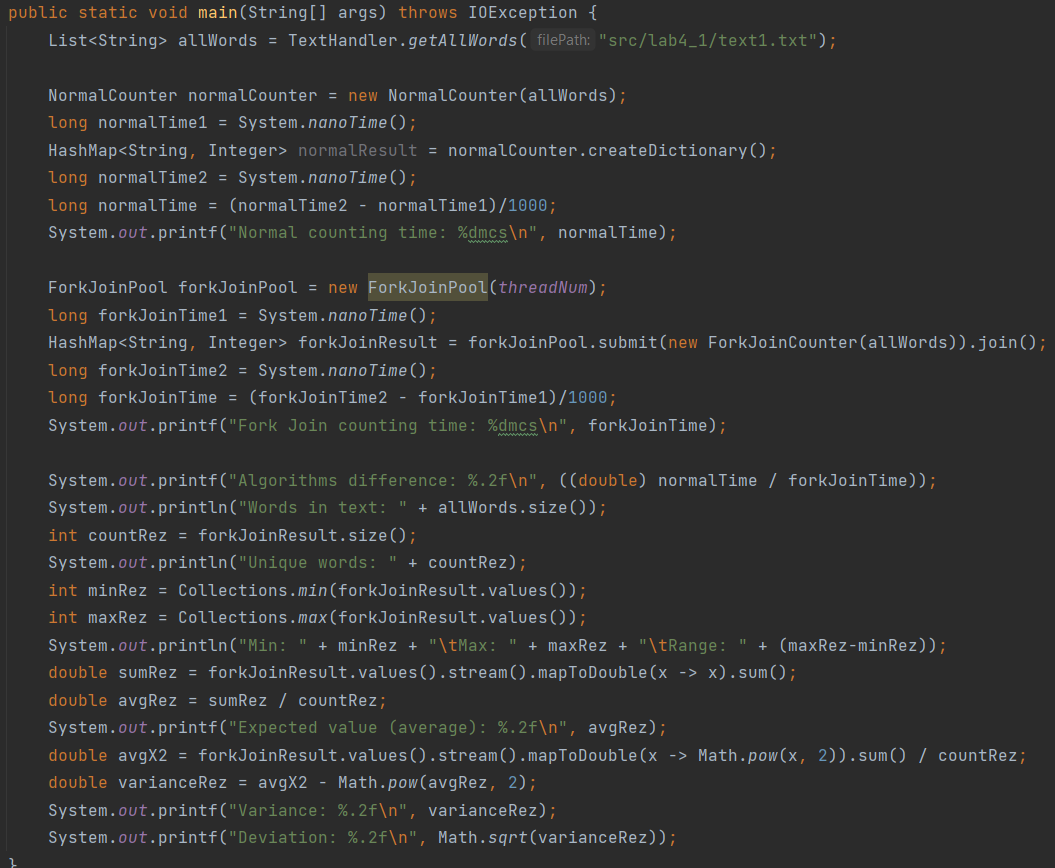


Рисунок 2 – Підрахунок характеристики випадкової величини

*Спостереження, результати, висновки:*

Бачимо прискорення програми, з використанням ForkJoinFramework. Програма працює дуже швидко (вимірюється у мікросекундах), і тому при повторних тестах різниця може бути різною. В основному сповільнюється робота звичайного алгоритму, а пришвидшений метод коливається з невеликими значеннями. Пов’язано це все з роботою процесора і іншими процесами які він обробляє в цей момент.

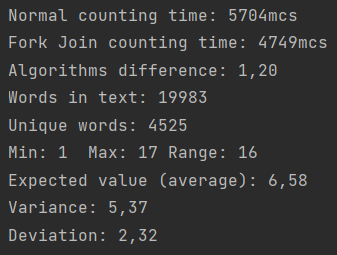
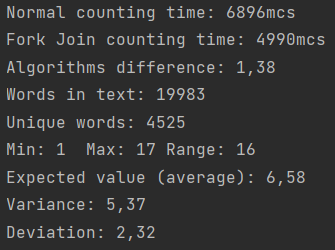
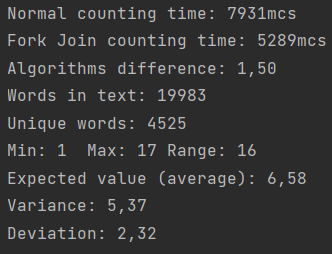
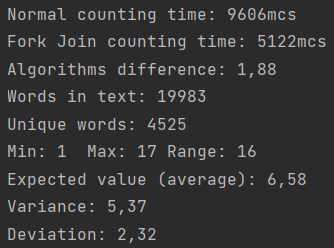


Рисунок 3 – Результати роботи алгоритмів

***Завдання 2:*** Реалізуйте один з алгоритмів комп’ютерного практикуму 2 або 3 з використанням ForkJoinFramework та визначте прискорення, яке отримане за рахунок використання ForkJoinFramework.

*Опис пункту:*

За основу взятий алгорим Фокса з другої лабораторної роботи. Вибір впав через те, що алгоритм виявився найшвидшим, а також більш підходящий за принципом для рекурсивних обчислень. Тож перевіримо чи можна його прискорити ще краще.

*Фрагмент коду:*

Переписуємо клас алгоритму наступним чином:  
Переносимо код у метод compute новоствореного класу. Ключові змінні заносимо в поля класу. Полям класу присвоюємо значення в конструкторі класу. Розділимо попереднє множення на два окремих методи: множення блоків і присвоєння значень перемножених блоків результатній матриці. Де перемноження блоків відбувається у класі що розширює клас RecursiveTask, у методі compute. Результати записуємо у hashmaps і обробляємо у попередньому класі. Замість пулу потоків і методу execute використовувався метод fork, де створені задачі після join-ились при записі результатів у результуючу матрицю.

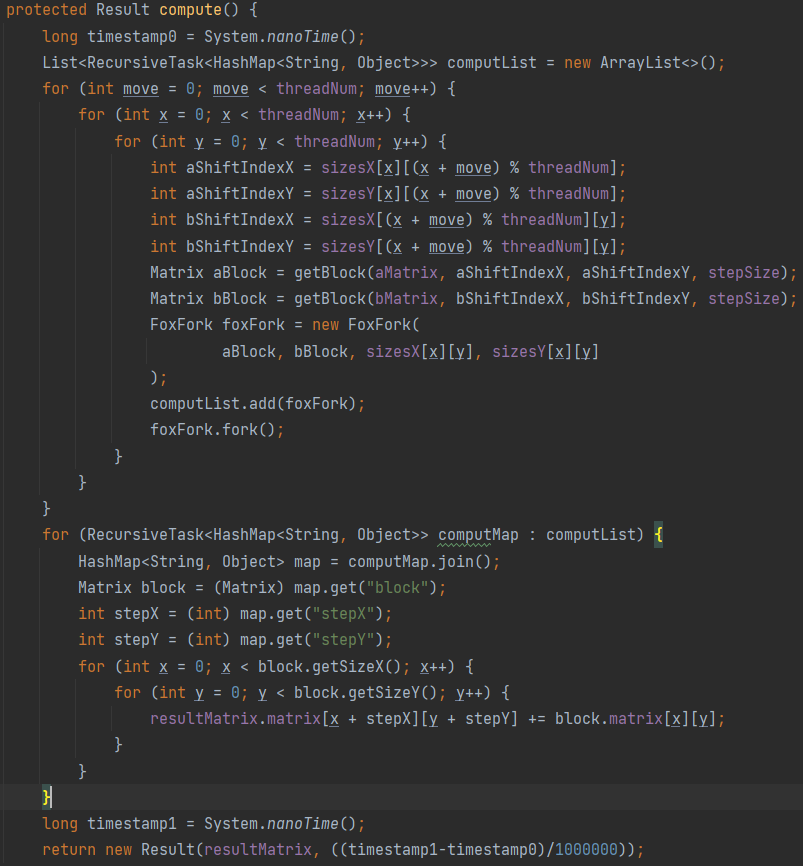
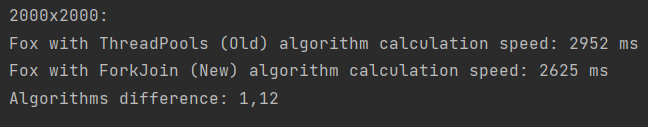


Рисунок 4 – Метод compute прискореного класу

*Спостереження, результати, висновки:*

Метод прискорений приблизно на 10%.



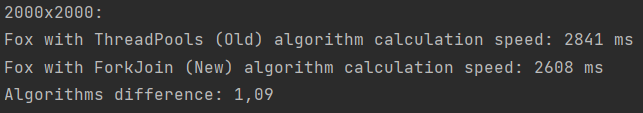


Рисунок 5 – Результати прискорення методу Фокса

***Завдання 3:*** Розробіть та реалізуйте алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework.

*Опис пункту:*

Оптимізація за допомогою ForkJoinFramework виконується наступним чином:

1. Відбираємо два тексти, котрі розбиті на рядки (абзаци).
2. Встановлюємо ліміт рядків на підзадачу.
3. Розбиваємо задачу на підзадачі, де рекурсивно текст розбивається на рядки за індексами (кожна підзадача з одного файлу обробляє свої підрядки).
4. Коли кількість рядків не перевищує ліміт, то розбиваємо рядки на унікальні слова.
5. Збираємо результат з підзадач.
6. Порівнюємо два набори слів.

*Фрагмент коду:*

Структура коду схожа на підзадачу 1 – compute та createSubtasks, проте ми працюємо не з HashMap, а з HashSet. Метод createSubtasks майже ідентичний: розділяємо задачу на дві підзадачі, де ділимо файл по номерах рядків на набір рядків. Збираємо результат з підзадач в HashSet і повертаємо (return) назад. Самі слова сплітимо регуляркою порядково і повертаємо (return).

Два набори унікальних слів порівнюємо в методі main методом contains та заносимо у ліст спільних слів.



Рисунок 6 – Метод compute та createSubtasks

*Спостереження, результати, висновки:*

Алгоритм працює швидко і коректно при перезапусках, з різними варіантами тексту та лімітами.

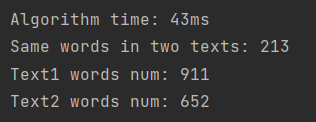


Рисунок 7 – Результати роботи алгоритму пошуку спільних слів

***Завдання 4:***

*Опис пункту:*

*Фрагмент коду:*

Рисунок 1 –

*Спостереження, результати, висновки:*

Рисунок 2 –

**Висновок:**

В ході виконання практикуму було ...

У звіті наявні скріншоти з прикладами результатів роботи коду, а також опис роботи коду та пояснення результатів. Вперше використано такі класи, інтерфейси, методи та ключові слова: ForkJoinTask, ForkJoinPool, invokeAll().

**Лістинг коду у вигляді GitHub-репозиторію:**

<https://github.com/m4cy43/parallel_programming/tree/master/lab4>