

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»  
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

**Звіт**

З лабораторної роботи № 4 з дисципліни  
«Технології паралельних обчислень»

Тема: «Розробка паралельних програм з використанням пулів потоків,  
екзекуторів та ForkJoinFramework»

**Виконав(ла)**

*ІП-14 Бабіч Денис*

(шифр, прізвище, ім'я, по батькові)

**Перевірів**

*Дифучина О. Ю.*

(шифр, прізвище, ім'я, по батькові)

Київ 2024

## ОСНОВНА ЧАСТИНА

**Мета роботи:** Розробка паралельних програм з використанням пулів потоків, екзекуторів та ForkJoinFramework.

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.

```
Execution time: 1 ms
Processed words: 1000
Average word length: 6.166
Standard deviation: 2.262
Dispersion: 0.367

Execution time: 3 ms
Processed words: 1000
Average word length: 6.166
Standard deviation: 2.262
Dispersion: 0.367

Execution time: 2 ms
Processed words: 10000
Average word length: 6.218
Standard deviation: 2.267
Dispersion: 0.365

Execution time: 2 ms
Processed words: 10000
Average word length: 6.218
Standard deviation: 2.267
Dispersion: 0.365

Execution time: 8 ms
Processed words: 100000
Average word length: 6.208
Standard deviation: 2.275
Dispersion: 0.366

Execution time: 6 ms
Processed words: 100000
Average word length: 6.208
Standard deviation: 2.275
Dispersion: 0.366
```

Рисунок 1.1 – Результати роботи алгоритму

### Лістинг методу `getWordsStatsDefault`

```

public static WordsStats getWordsStatsDefault(Document document)
{
    if (document == null)
    {
        throw new IllegalArgumentException("document cannot be null.");
    }

    long timestepStart = System.currentTimeMillis();

    int totalLength = 0;

    for (String word : document.getWords())
    {
        totalLength += word.length();
    }

    final double AVERAGE_WORD_LENGTH = (double)totalLength /
document.getWordsCount();

    double totalSquares = 0;

    for (String word : document.getWords())
    {
        totalSquares += Math.pow(word.length() - AVERAGE_WORD_LENGTH,
2);
    }

```

```

        final double STANDARD_DEVIATION = Math.sqrt(totalSquares /
document.getWordsCount());

        final double DISPERSION = STANDARD_DEVIATION /
AVERAGE_WORD_LENGTH;

        WordsStats stats = new WordsStats(document.getWordsCount(), DISPERSION,
STANDARD_DEVIATION, AVERAGE_WORD_LENGTH);

        long timestepEnd = System.currentTimeMillis();
        long executionTime = timestepEnd - timestepStart;

        System.out.println(String.format("Execution time: %d ms", executionTime));

        return stats;
    }

```

### **Лістинг методу `getWordsStatsEnhanced`**

```

public static WordsStats getWordsStatsEnhanced(Document document)
{
    if (document == null)
    {
        throw new IllegalArgumentException("document cannot be null.");
    }

    long timestepStart = System.currentTimeMillis();

    final int TOTAL_LENGTH = ForkJoinPool.commonPool().invoke(new
TotalLengthTask(document, 0, document.getWordsCount()));

```

```
final double AVERAGE_WORD_LENGTH = (double)TOTAL_LENGTH /
document.getWordsCount();
```

```
final double TOTAL_SQUARES = ForkJoinPool.commonPool().invoke(new
TotalSquaresTask(AVERAGE_WORD_LENGTH, document, 0,
document.getWordsCount()));
```

```
final double STANDARD_DEVIATION = Math.sqrt(TOTAL_SQUARES /
document.getWordsCount());
```

```
final double DISPERSION = STANDARD_DEVIATION /
AVERAGE_WORD_LENGTH;
```

```
long timestepEnd = System.currentTimeMillis();
```

```
long executionTime = timestepEnd - timestepStart;
```

```
WordsStats stats = new WordsStats(document.getWordsCount(), DISPERSION,
STANDARD_DEVIATION, AVERAGE_WORD_LENGTH);
```

```
System.out.println(String.format("Execution time: %d ms", executionTime));
```

```
return stats;
```

```
}
```

### **Лістинг класу TotalLengthTask**

```
package common;
```

```
import java.util.List;
```

```
import java.util.concurrent.RecursiveTask;
```

```

public final class TotalLengthTask extends RecursiveTask<Integer>
{
    private static int Threshold = 100;

    private final int INDEX_END;
    private final int INDEX_START;
    private final Document DOCUMENT;

    public TotalLengthTask(Document document, int indexStart, int indexEnd)
    {
        this.DOCUMENT = document;
        this.INDEX_END = indexEnd;
        this.INDEX_START = indexStart;
    }

    @Override
    protected Integer compute()
    {
        if ((this.INDEX_END - this.INDEX_START) <= TotalLengthTask.Threshold)
        {
            int totalLength = 0;

            List<String> wordsSlice =
this.DOCUMENT.getWordsSlice(this.INDEX_START, this.INDEX_END);

            for (String word : wordsSlice)
            {
                totalLength += word.length();
            }
        }
    }
}

```

```

        return totalLength;
    }
    else
    {
        final int INDEX_MID = (INDEX_END + INDEX_START) / 2;

        TotalLengthTask firstTask = new TotalLengthTask(this.DOCUMENT,
this.INDEX_START, INDEX_MID);

        TotalLengthTask secondTask = new TotalLengthTask(this.DOCUMENT,
INDEX_MID, this.INDEX_END);

        firstTask.fork();
        secondTask.fork();

        return firstTask.join() + secondTask.join();
    }
}

public static int getThreshold()
{
    return TotalLengthTask.Threshold;
}

public static void setThreshold(int value)
{
    TotalLengthTask.Threshold = value;
}
}

```

**Лістинг класу TotalSquaresTask**

```
package common;

import java.util.List;

import java.util.concurrent.RecursiveTask;

public class TotalSquaresTask extends RecursiveTask<Double>
{
    private static int Threshold = 100;

    private final int INDEX_END;
    private final int INDEX_START;
    private final Document DOCUMENT;

    private final double AVERAGE_WORD_LENGTH;

    public TotalSquaresTask(double averageWordLength, Document document, int
indexStart, int indexEnd)
    {
        this.DOCUMENT = document;
        this.INDEX_END = indexEnd;
        this.INDEX_START = indexStart;

        this.AVERAGE_WORD_LENGTH = averageWordLength;
    }

    @Override
    protected Double compute()
```



```

{
    if (this.INDEX_END - this.INDEX_START <= TotalSquaresTask.Threshold)
    {
        double totalSquares = 0;

        List<String> wordsSlice =
this.DOCUMENT.getWordsSlice(this.INDEX_START, this.INDEX_END);

        for (String word : wordsSlice)
        {
            totalSquares += Math.pow(word.length() -
this.AVERAGE_WORD_LENGTH, 2);
        }

        return totalSquares;
    }
    else
    {
        final int INDEX_MID = (INDEX_END + INDEX_START) / 2;

        TotalSquaresTask firstTask = new
TotalSquaresTask(this.AVERAGE_WORD_LENGTH, this.DOCUMENT,
this.INDEX_START, INDEX_MID);

        TotalSquaresTask secondTask = new
TotalSquaresTask(this.AVERAGE_WORD_LENGTH, this.DOCUMENT,
INDEX_MID, this.INDEX_END);

        firstTask.fork();
        secondTask.fork();
    }
}

```

```

        return firstTask.join() + secondTask.join();
    }
}

public static int getThreshold()
{
    return TotalSquaresTask.Threshold;
}

public static void setThreshold(int value)
{
    TotalSquaresTask.Threshold = value;
}
}

```

Таблиця 1.1 – Порівняння прискорення

| Кількість слів у файлі | Час виконання, мс  |                   | Прискорення |
|------------------------|--------------------|-------------------|-------------|
|                        | Послідовний підхід | ForkJoinFramework |             |
| 1000                   | 1                  | 3                 | 0.3         |
| 10000                  | 2                  | 2                 | 1           |
| 100000                 | 8                  | 6                 | 1.3         |

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

```
Execution time: 0.171 seconds, (171 milliseconds).
Execution time: 0.135 seconds, (135 milliseconds).
Threads: 4; Dimensions: 500; Threshold: 62 | true
Execution time: 1.061 seconds, (1061 milliseconds).
Execution time: 1.316 seconds, (1316 milliseconds).
Threads: 4; Dimensions: 1000; Threshold: 125 | true
Execution time: 5.417 seconds, (5417 milliseconds).
Execution time: 7.881 seconds, (7881 milliseconds).
Threads: 4; Dimensions: 1500; Threshold: 187 | true
Execution time: 15.905 seconds, (15905 milliseconds).
Execution time: 22.757 seconds, (22757 milliseconds).
Threads: 4; Dimensions: 2000; Threshold: 250 | true
Execution time: 30.440 seconds, (30440 milliseconds).
Execution time: 48.807 seconds, (48807 milliseconds).
```

Рисунок 1.2 – Результати виконання алгоритму

Таблиця 1.2 – Отримані значення прискорення

| Matrix<br>Size | Stripe<br>algorithm<br>(enhanced) | Stripe algorithm (default) |          |              |          |               |          |
|----------------|-----------------------------------|----------------------------|----------|--------------|----------|---------------|----------|
|                |                                   | 4 processors               |          | 9 processors |          | 16 processors |          |
|                |                                   | Time                       | Speed up | Time         | Speed up | Time          | Speed up |
| 500            | 0.171                             | 0.157                      | 1.08917  | 0.078        | 2.19231  | 0.092         | 1.8587   |
| 1000           | 1.061                             | 1.092                      | 0.97161  | 1.122        | 0.94563  | 1.21          | 0.87686  |
| 1500           | 5.417                             | 8.032                      | 0.67443  | 5.846        | 0.92662  | 5.298         | 1.02246  |
| 2000           | 15.905                            | 22.19                      | 0.71676  | 15.135       | 1.05088  | 13.941        | 1.14088  |
| 2500           | 30.44                             | 45.03                      | 0.67599  | 33.438       | 0.91034  | 28.5          | 1.06807  |
| 3000           | 60.544                            | 93.29                      | 0.64899  | 58.878       | 1.0283   | 54.36         | 1.11376  |

### Лістинг методу **multiplyStripedWithActions**

```

public static Result multiplyStripedWithActions(MatrixInt matrix1, MatrixInt
matrix2)
{
    if (!MatrixInt.areMultipliable(matrix1, matrix2))
    {
        throw new IllegalArgumentException("Matrices are not multipliable.");
    }

    MatrixInt resultingMatrix = new MatrixInt(matrix1.rows, matrix2.columns);

    long timestepStart = System.currentTimeMillis();

    ForkJoinPool.commonPool().invoke(new StripedMultiplicationTask(matrix1,
matrix2, resultingMatrix, 0, matrix1.rows * matrix2.columns));

    long timestepEnd = System.currentTimeMillis();
    long executionTime = timestepEnd - timestepStart;

    return new Result(resultingMatrix, executionTime);
}

```

### Лістинг класу **StripedMultiplicationTask**

```

public class StripedMultiplicationTask extends RecursiveAction {
    private static int Threshold = 100;

    private final MatrixInt MATRIX_1;
    private final MatrixInt MATRIX_2;
    private final MatrixInt MATRIX_RESULT;

```

```

private final int INDEX_START;
private final int INDEX_FINISH;

public StripedMultiplicationTask(MatrixInt matrix1, MatrixInt matrix2, MatrixInt
result, int indexStart, int indexFinish) {
    this.MATRIX_1 = matrix1;
    this.MATRIX_2 = matrix2;
    this.MATRIX_RESULT = result;

    this.INDEX_START = indexStart;
    this.INDEX_FINISH = indexFinish;
}

@Override
protected void compute() {
    if (this.INDEX_FINISH - this.INDEX_START <=
StripedMultiplicationTask.Threshold) {
        for (int i = this.INDEX_START; i < this.INDEX_FINISH; ++i)
        {
            final int INDEX_ROW = i / this.MATRIX_1.getColumns();
            final int INDEX_COLUMN = i % this.MATRIX_2.getColumns();

            final int[] ROW = this.MATRIX_1.getRow(INDEX_ROW);
            final int[] COLUMN = this.MATRIX_2.getColumn(INDEX_COLUMN);

            int result = 0;

            for (int j = 0; j < ROW.length; ++j) {

```

```

        result += ROW[j] * COLUMN[j];
    }

    this.MATRIX_RESULT.set(INDEX_ROW, INDEX_COLUMN, result);
}
}
else {
    final int INDEX_MID = (this.INDEX_START + this.INDEX_FINISH) / 2;

    StripedMultiplicationTask leftTask = new
    StripedMultiplicationTask(this.MATRIX_1, this.MATRIX_2,
    this.MATRIX_RESULT, this.INDEX_START, INDEX_MID);

    StripedMultiplicationTask rightTask = new
    StripedMultiplicationTask(this.MATRIX_1, this.MATRIX_2,
    this.MATRIX_RESULT, INDEX_MID, this.INDEX_FINISH);

    ForkJoinTask.invokeAll(leftTask, rightTask);
}
}

public static int getThreshold() {
    return StripedMultiplicationTask.Threshold;
}

public static void setThreshold(int value) {
    StripedMultiplicationTask.Threshold = value;
}
}

```

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

|       |           |               |
|-------|-----------|---------------|
| Word: | elementum | Quantity: 623 |
| Word: | tempor    | Quantity: 624 |
| Word: | mollis    | Quantity: 624 |
| Word: | potenti   | Quantity: 623 |
| Word: | purus     | Quantity: 623 |
| Word: | augue     | Quantity: 623 |
| Word: | justo     | Quantity: 624 |
| Word: | lorem     | Quantity: 623 |
| Word: | nam       | Quantity: 623 |
| Word: | id        | Quantity: 624 |
| Word: | habitant  | Quantity: 624 |
| Word: | per       | Quantity: 623 |
| Word: | semper    | Quantity: 624 |
| Word: | volutpat  | Quantity: 625 |
| Word: | ad        | Quantity: 625 |
| Word: | sodales   | Quantity: 623 |
| Word: | in        | Quantity: 624 |
| Word: | finibus   | Quantity: 625 |
| Word: | urna      | Quantity: 624 |
| Word: | velit     | Quantity: 624 |
| Word: | sociosqu  | Quantity: 624 |

Рисунок 1.3 – Результат роботи алгоритму

### Лістинг методу `getCommonWords`

```
public static HashMap<String, Integer> getCommonWords(Document... documents)
{
    HashMap<String, Integer> commonWords = new HashMap<>();

    for (Document document : documents)
    {
        HashMap<String, Integer> documentWords =
        ForkJoinPool.commonPool().invoke(new CommonWordsTask(document, 0,
        document.getWordsCount()));
```

```

    if (commonWords.isEmpty())
    {
        commonWords.putAll(documentWords);
    }
    else
    {
        HashMap<String, Integer> newCommonWords = new HashMap<>();

        for (String word : commonWords.keySet())
        {
            if (documentWords.containsKey(word))
            {
                newCommonWords.put(word, commonWords.get(word) +
documentWords.get(word));
            }
        }

        commonWords = newCommonWords;
    }

    return commonWords;
}

```

### **Лістинг класу CommonWordsTask**

```

final class CommonWordsTask extends RecursiveTask<HashMap<String, Integer>>{
    private static int Threshold = 100;

    private final Document DOCUMENT;

```



```

private final int INDEX_START;
private final int INDEX_FINISH;

public CommonWordsTask(Document document, int indexStart, int indexFinsih) {
    this.DOCUMENT = document;
    this.INDEX_START = indexStart;
    this.INDEX_FINISH = indexFinsih;
}

@Override
protected HashMap<String, Integer> compute() {
    HashMap<String, Integer> documentWords = new HashMap<>();

    if ((this.INDEX_FINISH - this.INDEX_START) <
CommonWordsTask.Threshold) {
        List<String> words = this.DOCUMENT.getWordsSlice(this.INDEX_START,
this.INDEX_FINISH);

        for (String word : words) {
            documentWords.compute(word, (key, value) -> (value == null) ? 1 : value
+ 1);
        }
    }
    else {
        final int INDEX_MID = this.INDEX_START + (this.INDEX_FINISH -
this.INDEX_START) / 2;

        CommonWordsTask firstTask = new CommonWordsTask(this.DOCUMENT,
this.INDEX_START, INDEX_MID);

```

```
CommonWordsTask secondTask = new  
CommonWordsTask(this.DOCUMENT, INDEX_MID, this.INDEX_FINISH);
```

```
ForkJoinTask.invokeAll(firstTask, secondTask);
```

```
HashMap<String, Integer> firstResult = firstTask.join();
```

```
HashMap<String, Integer> secondResult = secondTask.join();
```

```
for (Map.Entry<String, Integer> entry : firstResult.entrySet()) {  
    documentWords.put(entry.getKey(), entry.getValue());  
}
```

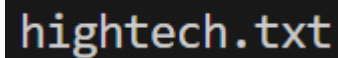
```
for (Map.Entry<String, Integer> entry : secondResult.entrySet()) {  
    documentWords.merge(entry.getKey(), entry.getValue(), Integer::sum);  
}  
}
```

```
return documentWords;  
}
```

```
public static int getThreshold() {  
    return CommonWordsTask.Threshold;  
}
```

```
public static void setThreshold(int value) {  
    CommonWordsTask.Threshold = value;  
}  
}
```

4. Розробіть та реалізуйте алгоритм пошуку текстових документів, які відповідають заданим ключовим словам (належать до області «Інформаційні технології»), з використанням ForkJoinFramework.



hightech.txt

Рисунок 1.4 – Результат роботи алгоритма

#### **Лістинг методу getDocumentsWithKeyWords**

```
public static List<Document> getDocumentsWithKeyWords(List<String> keywords,
Document... documents) {
    List<Document> result = new ArrayList<>();

    for (Document document : documents) {
        if (ForkJoinPool.commonPool().invoke(new KeywordsSearchTask(document,
keywords, 0, document.getWordsCount())) {
            result.add(document);
        }
    }

    return result;
}
```

#### **Лістинг класу KeywordsSearchTask**

```
package common;

import java.util.List;

import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.RecursiveTask;
```

```

public class KeywordsSearchTask extends RecursiveTask<Boolean>
{
    private static int Threshold = 100;
    private static int MinRequiredKeywordsCount = 5;

    private final Document DOCUMENT;
    private final List<String> KEYWORDS;

    private final int INDEX_START;
    private final int INDEX_FINISH;

    public KeywordsSearchTask(Document document, List<String> keywords, int
indexStart, int indexFinish)
    {
        this.DOCUMENT = document;
        this.KEYWORDS = keywords;

        this.INDEX_START = indexStart;
        this.INDEX_FINISH = indexFinish;
    }

    @Override
    protected Boolean compute()
    {
        if (this.INDEX_FINISH - this.INDEX_START < Threshold)
        {
            int count = 0;

            List<String> words = this.DOCUMENT.getWordsSlice(this.INDEX_START,
this.INDEX_FINISH);

```

```

    for (String word : words)
    {
        for (String keyword : KEYWORDS)
        {
            if (word.toLowerCase().contains(keyword.toLowerCase()))
            {
                ++count;

                if (count >= KeywordsSearchTask.MinRequiredKeywordsCount)
                {
                    return true;
                }
            }
        }
    }

    return false;
}

else
{
    final int INDEX_MID = this.INDEX_START + (this.INDEX_FINISH -
this.INDEX_START) / 2;

    KeywordsSearchTask firstTask = new
KeywordsSearchTask(this.DOCUMENT, this.KEYWORDS, this.INDEX_START,
INDEX_MID);

```

```

        KeywordsSearchTask secondTask = new
KeywordsSearchTask(this.DOCUMENT,    this.KEYWORDS,    INDEX_MID,
this.INDEX_FINISH);

```

```

        ForkJoinTask.invokeAll(firstTask, secondTask);

```

```

        return firstTask.join() || secondTask.join();

```

```

    }

```

```

}

```

```

public static int getThreshold()

```

```

{

```

```

    return KeywordsSearchTask.Threshold;

```

```

}

```

```

public static void setThreshold(int value)

```

```

{

```

```

    KeywordsSearchTask.Threshold = value;

```

```

}

```

```

public static int getMinRequiredKeywordsCount()

```

```

{

```

```

    return KeywordsSearchTask.MinRequiredKeywordsCount;

```

```

}

```

```

public static void setMinRequiredKeywordsCount(int value) {

```

```

    KeywordsSearchTask.MinRequiredKeywordsCount = value;

```

```

}

```

```

}

```

## ВИСНОВКИ

В ході виконання цієї лабораторної роботи було розроблено та реалізовано кілька алгоритмів з використанням ForkJoinFramework. Таким чином, був створений алгоритм статистичного аналізу тексту, який дозволив визначити характеристики випадкової величини «довжина слова в символах», де експериментальне дослідження показало ефективність побудованого алгоритму аналізу текстових документів з великою кількістю слів. Також було реалізовано алгоритм стрічкового множення матриць та було визначено прискорення, яке отримане за рахунок використання ForkJoinFramework. Додатково, було розроблено та реалізовано алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework і алгоритм пошуку текстових документів, які відповідають заданим ключовим словам.

Таким чином, використання ForkJoinFramework дозволило оптимізувати процес алгоритми, покращити продуктивність та забезпечити коректність роботи програм. Завдяки паралельному виконанню завдань було можливо значно прискорити обробку великих обсягів даних.