



Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики і програмної інженерії

Лабораторна робота №4
з дисципліни
Технології паралельних обчислень

Виконав:

студент групи ІІІ-13:
Бабашев О. Д.

Перевірив:

ст.викл.
Дифучин А. Ю.

Київ 2024

Завдання до комп'ютерного практикуму 4 «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.
2. Реалізуйте один з алгоритмів комп'ютерного практикуму 2 або 3 з використанням ForkJoinFramework та визначте прискорення, яке отримане за рахунок використання ForkJoinFramework.
3. Розробіть та реалізуйте алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework.
4. Розробіть та реалізуйте алгоритм пошуку текстових документів, які відповідають заданим ключовим словам (належать до області «Інформаційні технології»), з використанням ForkJoinFramework.

Завдання 1

Лістинг коду:

DirectoryTask.java

```
package Task1;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.RecursiveTask;

public class DirectoryTask extends RecursiveTask<Statistics> {
    private Path directory;

    public DirectoryTask(Path directory) {
        this.directory = directory;
    }
}
```

```

@Override
protected Statistics compute() {
    List<RecursiveTask<Statistics>> tasks = new ArrayList<>();

    try {
        Files.list(directory).forEach(path -> {
            if (Files.isDirectory(path)) {
                DirectoryTask directoryTask = new DirectoryTask(path);
                directoryTask.fork();
                tasks.add(directoryTask);
            } else if (path.toString().endsWith(".txt")) {
                FileTask fileTask = new FileTask(path);
                fileTask.fork();
                tasks.add(fileTask);
            }
        });

        Statistics result = new Statistics(0, 0);
        for (RecursiveTask<Statistics> task : tasks) {
            result.add(task.join());
        }
        return result;
    } catch (IOException e) {
        e.printStackTrace();
        return new Statistics(0, 0);
    }
}

```

FileTask.java

```
package Task1;

import java.io.IOException;
import java.nio.file.Path;
import java.util.concurrent.RecursiveTask;

public class FileTask extends RecursiveTask<Statistics> {
    private Path file;

    public FileTask(Path file) {
        this.file = file;
    }

    @Override
    protected Statistics compute() {
        try {
            String text = TextReader.readFile(file);
            return SequentialWordLengthStatistics.analyze(text);
        } catch (IOException e) {
            e.printStackTrace();
            return new Statistics(0, 0);
        }
    }
}
```

Main.java

```
package Task1;

import java.io.IOException;
import java.nio.file.Path;
```

```

import java.nio.file.Paths;
import java.util.List;
import java.util.concurrent.ForkJoinPool;

public class Main {
    public static void main(String[] args) {
        try {
            String directoryPath = "D:\\kpi 6\\ТПО\\parallel-labs\\lab4\\books";
            Path path = Paths.get(directoryPath);

            // Послідовний аналіз
            long sequentialStartTime = System.currentTimeMillis();
            Statistics sequentialStatistics = new Statistics(0, 0);
            List<Path> files = TextReader.getAllTextFiles(directoryPath);
            for (Path file : files) {
                String text = TextReader.readFile(file);
                Statistics fileStatistics = SequentialWordLengthStatistics.analyze(text);
                sequentialStatistics.add(fileStatistics);
            }
            long sequentialEndTime = System.currentTimeMillis();
            long sequentialDuration = sequentialEndTime - sequentialStartTime;

            System.out.println("\nSequential Analysis:");
            sequentialStatistics.print();
            System.out.println("Sequential execution time: " + sequentialDuration + "
ms");

            System.out.println("\n=====
\n");

```

```

// Паралельний аналіз
long parallelStartTime = System.currentTimeMillis();
ForkJoinPool pool = new ForkJoinPool();
DirectoryTask directoryTask = new DirectoryTask(path);
Statistics parallelStatistics = pool.invoke(directoryTask);
long parallelEndTime = System.currentTimeMillis();
long parallelDuration = parallelEndTime - parallelStartTime;

System.out.println("Parallel Analysis:");
parallelStatistics.print();
System.out.println("Parallel execution time: " + parallelDuration + "
ms\n");

double speedup = (double) sequentialDuration / parallelDuration;
System.out.println("Speedup: " + speedup);

} catch (IOException e) {
    e.printStackTrace();
}
}
}

```

SequentialWordLengthStatistics.java

```

package Task1;

public class SequentialWordLengthStatistics {
    public static Statistics analyze(String text) {
        // Розбиваємо текст на слова, використовуючи пробіли як роздільники
        String[] words = text.split("\\s+");
        int totalWords = words.length;
        int totalLength = 0;
    }
}

```

```

        for (String word : words) {
            totalLength += word.length();
        }
        return new Statistics(totalWords, totalLength);
    }
}

```

Statistics.java

```
package Task1;
```

```

public class Statistics {
    private int totalWords;
    private int totalLength;

    public Statistics(int totalWords, int totalLength) {
        this.totalWords = totalWords;
        this.totalLength = totalLength;
    }

    public void add(Statistics other) {
        this.totalWords += other.totalWords;
        this.totalLength += other.totalLength;
    }

    public void print() {
        System.out.println("Total words: " + totalWords);
        System.out.println("Total length: " + totalLength);
        System.out.println("Average word length: " + (totalWords == 0 ? 0 : (double)
totalLength / totalWords));
    }
}

```

TextReader.java

```
package Task1;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.List;
import java.util.stream.Collectors;

public class TextReader {

    public static String readFile(Path path) throws IOException {

        return new String(Files.readAllBytes(path));

    }

    public static List<Path> getAllTextFiles(String directoryPath) throws
IOException {

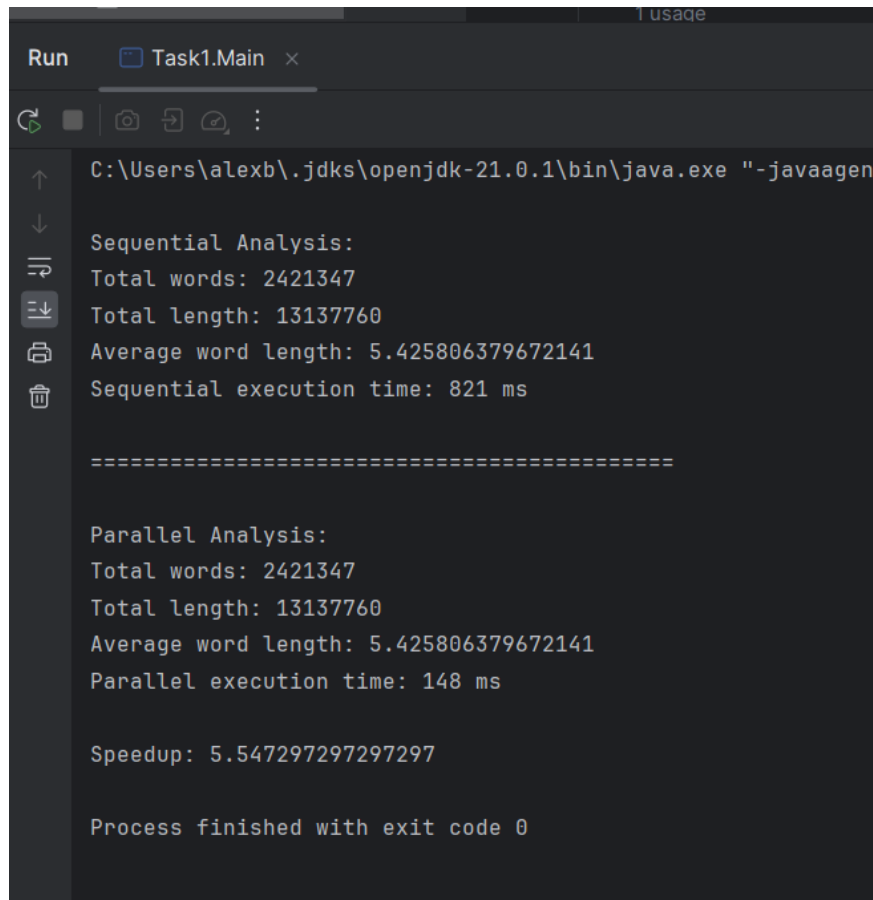
        return Files.walk(Paths.get(directoryPath))
            .filter(Files::isRegularFile)
            .filter(path -> path.toString().endsWith(".txt"))
            .collect(Collectors.toList());

    }

}
```


Результати виконання коду:

Реалізовано порівняння виконання послідовного алгоритму статистичного аналізу тексту та алгоритму статистичного аналізу тексту розробленого за допомогою ForkJoinFramework.



```
Run Task1.Main x
C:\Users\alexb\.jdk\openjdk-21.0.1\bin\java.exe "-javaagent
Sequential Analysis:
Total words: 2421347
Total length: 13137760
Average word length: 5.425806379672141
Sequential execution time: 821 ms

=====

Parallel Analysis:
Total words: 2421347
Total length: 13137760
Average word length: 5.425806379672141
Parallel execution time: 148 ms

Speedup: 5.547297297297297

Process finished with exit code 0
```

Рисунок 1.1 – Результат виконання коду

Як можемо зазначити алгоритми працюють належним чином. Алгоритм реалізований за допомогою ForkJoinFramework працює значно швидше, що є логічним і очікуваним результатом.

Завдання 2

Лістинг коду:

ForkJoinMultiplier.java

```
package Task2;
```

```
import java.util.concurrent.ForkJoinPool;
```

```
public class ForkJoinMultiplier implements IMultiplier {
```

```

private final int threads;

public ForkJoinMultiplier(int threads) {
    this.threads = threads;
}

@Override
public Result multiply(int[][] A, int[][] B) {
    int n = A.length;
    Result result = new Result(n, n);
    ForkJoinPool forkJoinPool = new ForkJoinPool(threads);

    // Create and invoke the ForkJoin task
    MultTask task = new MultTask(A, B, result, 0, n);
    forkJoinPool.invoke(task);
    return result;
}
}

```

IMultiplier.java

```

package Task2;

public interface IMultiplier {
    Result multiply(int[][] A, int[][] B);
}

```

Main.java

```

package Task2;

import java.util.Random;

public class Main {

```

```
private static final Random random = new Random();
```

```
public static int[][] initMatrix(int size) {  
    int[][] matrix = new int[size][size];  
    for (int i = 0; i < size; i++) {  
        for (int j = 0; j < size; j++) {  
            matrix[i][j] = random.nextInt(11);  
        }  
    }  
    return matrix;  
}
```

```
public static double measureTimeOfPerformance(IMultiplier multiplier, int[][]  
A, int[][] B, Result[] outResult) {  
    long totalTime = 0;  
    long startTime = System.currentTimeMillis();  
    outResult[0] = multiplier.multiply(A, B);  
    long endTime = System.currentTimeMillis();  
    totalTime += (endTime - startTime);  
    return totalTime;  
}
```

```
public static void main(String[] args){
```

```
    int size = 500;  
    int threads = 4;  
    int[][] A = initMatrix(size);  
    int[][] B = initMatrix(size);
```

```
    IMultiplier parallelMultiplier = new ParallelMultiplier(threads);
```

```

    Result[] parallelResult = new Result[1];

    double parallelTime = measureTimeOfPerformance(parallelMultiplier, A, B,
parallelResult);

    System.out.printf("Parallel: " + parallelTime + " ms\n");


    IMultiplier ForkJoinMultiplier = new ForkJoinMultiplier(threads);
    Result[] forkJoinResult = new Result[1];

    double forkJoinTime = measureTimeOfPerformance(ForkJoinMultiplier, A,
B, forkJoinResult);

    System.out.printf("ForkJoin: " + forkJoinTime + " ms\n");


//    System.out.println();
//    MatrixHelper.PrintMatrix(parallelResult[0].getMatrix());
//    System.out.println();
//    MatrixHelper.PrintMatrix(forkJoinResult[0].getMatrix());
//    System.out.println();


    if (MatrixHelper.NotEqual(forkJoinResult[0].getMatrix(),
parallelResult[0].getMatrix())) {
        System.err.println("\nresults NOT match!");
    } else {
        System.out.println("\nresults match!");
    }


    double speedup = parallelTime / forkJoinTime;

    System.out.printf("\nSpeedup: %.2f\n", speedup);


    System.out.println();
}

```

```
}
```

MatrixHelper.java

```
package Task2;
```

```
public class MatrixHelper {
```

```
    public static boolean NotEqual(int[][] matrix1, int[][] matrix2) {
```

```
        if (matrix1.length != matrix2.length || matrix1[0].length != matrix2[0].length)
        {
```

```
            return true;
```

```
        }
```

```
        for (int i = 0; i < matrix1.length; i++) {
```

```
            for (int j = 0; j < matrix1[i].length; j++) {
```

```
                if (matrix1[i][j] != matrix2[i][j]) {
```

```
                    return true;
```

```
                }
```

```
            }
```

```
        }
```

```
        return false;
```

```
    }
```

```
    public static void PrintMatrix(int[][] matrix) {
```

```
        for (int i = 0; i < matrix.length; i++) {
```

```
            for (int j = 0; j < matrix[i].length; j++) {
```

```
                System.out.print(matrix[i][j] + " ");
```

```
            }
```

```
            System.out.println(); // New line after each row
```

```
        }
```

```
    }
```

```
}
```

MultTask.java

```
package Task2;
```

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

```
public class MultTask extends RecursiveTask<Void> {
```

```
    private static final int THRESHOLD = 4;
```

```
    private final int[][] A;
```

```
    private final int[][] B;
```

```
    private final Result result;
```

```
    private final int startRow;
```

```
    private final int endRow;
```

```
    public MultTask(int[][] A, int[][] B, Result result, int startRow, int endRow) {
```

```
        this.A = A;
```

```
        this.B = B;
```

```
        this.result = result;
```

```
        this.startRow = startRow;
```

```
        this.endRow = endRow;
```

```
    }
```

```
    @Override
```

```
    protected Void compute() {
```

```
        int n = A.length;
```

```
        int colsA = A[0].length;
```

```
        int colsB = B[0].length;
```

```
        if (endRow - startRow <= THRESHOLD) {
```

```
            for (int i = startRow; i < endRow; i++) {
```

```

        for (int j = 0; j < colsB; j++) {
            int sum = 0;
            for (int k = 0; k < colsA; k++) {
                sum += A[i][k] * B[k][j];
            }
            result.setData(i, j, sum);
        }
    }
} else {
    int midRow = (startRow + endRow) / 2;

    MultTask task1 = new MultTask(A, B, result, startRow, midRow);
    MultTask task2 = new MultTask(A, B, result, midRow, endRow);
    invokeAll(task1, task2);
}

return null;
}
}

```

ParallelMultiplier.java

```
package Task2;
```

```
public class ParallelMultiplier implements IMultiplier {
```

```
    private final int threads;
```

```
    public ParallelMultiplier(int threads) {
```

```
        this.threads = threads;
```

```
    }
```

```
@Override
```

```

public Result multiply(int[][] A, int[][] B) {
    int n = A.length;
    Result result = new Result(n, n);
    Thread[] threads = new Thread[this.threads];

    for (int i = 0; i < this.threads; i++) {
        final int threadIndex = i;
        threads[i] = new Thread(() -> {
            int chunkSize = n / this.threads;
            int startRow = threadIndex * chunkSize;
            int endRow = (threadIndex == this.threads - 1) ? n : startRow +
chunkSize;

            for (int row = startRow; row < endRow; row++) {
                for (int col = 0; col < n; col++) {
                    int sum = 0;
                    for (int k = 0; k < n; k++) {
                        sum += A[row][k] * B[k][col];
                    }
                    result.setData(row, col, sum);
                }
            }
        });

        threads[i].start();
    }

    for (Thread thread : threads) {
        try {
            thread.join();

```



```

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    return result;
}
}

Result.java
package Task2;

public class Result {
    private final int[][] data;

    public Result(int rows, int cols) {
        data = new int[rows][cols];
    }

    public synchronized void setData(int row, int col, int value) {
        data[row][col] = value;
    }

    public synchronized int getData(int row, int col) {
        return data[row][col];
    }
}

//synchronized
public int[][] getMatrix() {
    return data;
}

```

```

public void printMatrix() {
    for (int[] row : data) {
        for (int val : row) {
            System.out.print(val + "\t");
        }
        System.out.println();
    }
}
}

```

Результати виконання коду:

Було реалізовано стрічковий алгоритм множення матриць за допомогою ForkJoinFramework. Протестуємо коректність роботи програми.

```

Run Main x
C:\Users\alexh\.jdk\openjdk-21.0.1\bin\java.exe "-javaagent:
Parallel: 200.0 ms
ForkJoin: 137.0 ms
results match!
Speedup: 1,46
Process finished with exit code 0

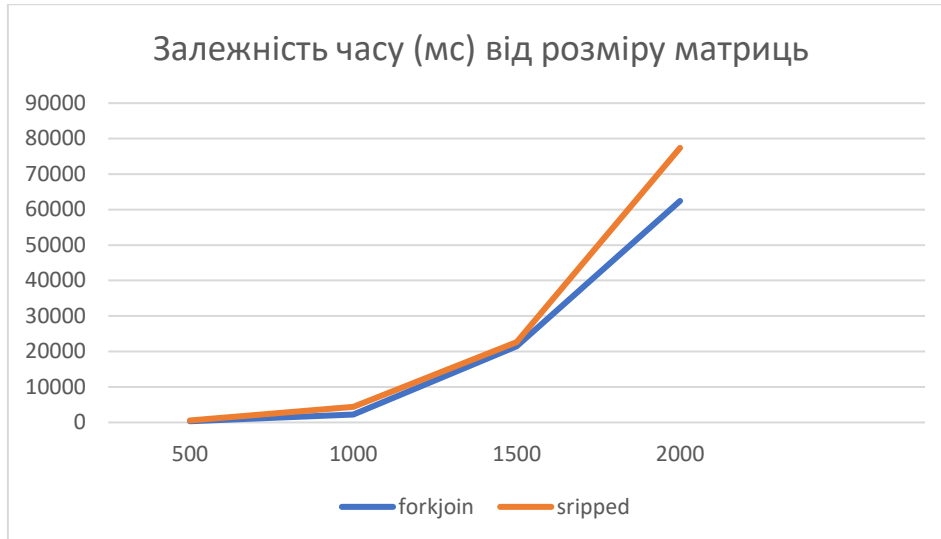
```

Рисунок 2.1 – Результат виконання коду

Як бачимо обидва алгоритми відпрацьовують належним чином. Також зазначимо з результатів, що алгоритм множення матриць за допомогою ForkJoinFramework в порівнянні із класичною його версією відпрацьовує швидше.

Порівняльні результати виконання класичної версії алгоритму та із застосуванням ForkJoinFramework на різних розмірах матриць наведено нижче.

	500	1000	1500	2000
forkjoin	339	2250	21467	62436
sripped	558	4385	22659	77392



В цілому з результатів експерименту бачимо, що стрічковий алгоритм множення матриць реалізований за допомогою ForkJoinFramework працює швидше і зі збільшенням розмірів матриць, збільшується розрив у часі.

Завдання 3

Лістинг коду:

CommSearch.java

```
package Task3;
```

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.Path;
import java.util.List;
import java.util.Set;
import java.util.HashSet;
import java.util.stream.Collectors;
import java.util.Arrays;
```

```

import java.util.stream.Stream;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

public class CommSearch {

    private static List<String> docReader(String directoryPath) {
        try (Stream<Path> paths = Files.walk(Paths.get(directoryPath))) {
            return paths.filter(Files::isRegularFile)
                .map(Path::toString)
                .map(CommSearch::fileReader)
                .collect(Collectors.toList());
        } catch (IOException e) {
            e.printStackTrace();
            return List.of();
        }
    }

    private static String fileReader(String filePath) {
        try {
            return new String(Files.readAllBytes(Paths.get(filePath)));
        } catch (IOException e) {
            e.printStackTrace();
            return "";
        }
    }

    static class CommonWordsTask extends RecursiveTask<Set<String>> {
        private static final int THRESHOLD = 1;
        private final List<String> documents;
    }

```

```
private final int startIndex;
```

```
private final int endIndex;
```

```
public CommonWordsTask(List<String> documents, int startIndex, int  
endIndex) {
```

```
    this.documents = documents;
```

```
    this.startIndex = startIndex;
```

```
    this.endIndex = endIndex;
```

```
}
```

```
@Override
```

```
protected Set<String> compute() {
```

```
    if (endIndex - startIndex <= THRESHOLD) {
```

```
        return findCommonWordsInDocuments(documents.subList(startIndex,  
endIndex));
```

```
    } else {
```

```
        int midIndex = (startIndex + endIndex) / 2;
```

```
        CommonWordsTask task1 = new CommonWordsTask(documents,  
startIndex, midIndex);
```

```
        CommonWordsTask task2 = new CommonWordsTask(documents,  
midIndex, endIndex);
```

```
        invokeAll(task1, task2);
```

```
        Set<String> res1 = task1.join();
```

```
        Set<String> res2 = task2.join();
```

```
        res1.retainAll(res2);
```

```
        return res1;
```

```
}  
}
```

```
private Set<String> findCommonWordsInDocuments(List<String> docs) {  
    if (docs.isEmpty()) {  
        return new HashSet<>();  
    }  
}
```

```
Set<String> commonWords = Arrays.stream(docs.get(0).split("\\W+"))  
    .collect(Collectors.toSet());
```

```
for (int i = 1; i < docs.size(); i++) {  
    Set<String> wordsInDoc = Arrays.stream(docs.get(i).split("\\W+"))  
        .collect(Collectors.toSet());  
    commonWords.retainAll(wordsInDoc);  
}
```

```
return commonWords;  
}  
}
```

```
public static void main(String[] args) {  
    // Читаємо документи з директорії  
    List<String> documents = docReader("D:\\kpi 6\\ТІО\\parallel-  
labs\\lab4\\commonWords");  
  
    ForkJoinPool pool = new ForkJoinPool();  
    CommonWordsTask task = new CommonWordsTask(documents, 0,  
documents.size());  
    Set<String> commonWords = pool.invoke(task);  
}
```

```

// Виведення результатів

System.out.println("Кількість слів у кожному файлі:");

for (int i = 0; i < documents.size(); i++) {

    int wordCount = Arrays.stream(documents.get(i).split("\\W+")).filter(word
-> !word.isEmpty()).toArray().length;

    System.out.println("Документ " + (i + 1) + ": " + wordCount);

}

System.out.println("Кількість спільних слів: " + commonWords.size());

System.out.println("Спільні слова: " + commonWords);

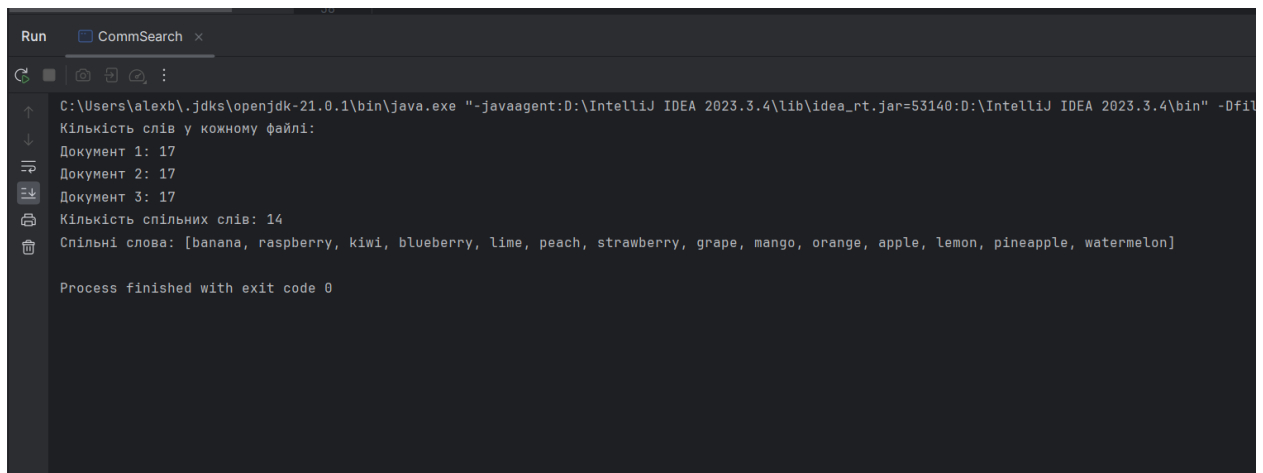
}

}

```

Результат виконання коду:

Було розроблено та реалізовано алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework.



```

Run CommSearch x
C:\Users\alexh\jdk\openjdk-21.0.1\bin\java.exe "-javaagent:D:\IntelliJ IDEA 2023.3.4\lib\idea_rt.jar=53140:D:\IntelliJ IDEA 2023.3.4\bin" -Dfile.encoding=UTF-8
Кількість слів у кожному файлі:
Документ 1: 17
Документ 2: 17
Документ 3: 17
Кількість спільних слів: 14
Спільні слова: [banana, raspberry, kiwi, blueberry, lime, peach, strawberry, grape, mango, orange, apple, lemon, pineapple, watermelon]
Process finished with exit code 0

```

Рисунок 3.1 – Результат виконання коду

В результаті виконання алгоритм знаходить всі спільні слова, які присутні одночасно в 3 наведених документах, слова унікальні проігноровано. Алгоритм відпрацьовує належним чином.

Завдання 4

Лістинг коду

DocSearch.java

```
package Task4;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.Path;
import java.util.*;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

public class DocSearch {

    static class DocSearchTask extends RecursiveAction {
        private final Path directory;

        public DocSearchTask(Path directory) {
            this.directory = directory;
        }

        @Override
        protected void compute() {
            if (Files.isDirectory(directory)) {
                try {
                    List<DocSearchTask> subTasks = new ArrayList<>();

                    Files.list(directory).forEach(path -> {
                        subTasks.add(new DocSearchTask(path));
                    });
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```



```

    });

    invokeAll(subTasks);

} catch (IOException e) {

    e.printStackTrace();

}
} else {
    fileSearcher(directory);
}
}
}

```

```

private static final Set<String> KEYWORDS = Set.of("machine learning",
"deep learning", "natural language processing",
    "mobile development", "Android", "iOS", "cross-platform development",
"native apps", "hybrid apps",
    "game development", "game engines", "graphics", "animation",
"artificial intelligence", "virtual reality",
    "software", "cloud", "network", "algorithm", "architecture", "database",
"web", "security", "server");

```

```

private void fileSearcher(Path filePath) {
    try {

        Set<String> foundKeywords = new HashSet<>();
        String content = Files.readString(filePath);
        String[] words = content.toLowerCase().split("\\W+");
        for (String word : words) {
            if (KEYWORDS.contains(word)) {
                foundKeywords.add(word);
            }
        }
    }
}

```

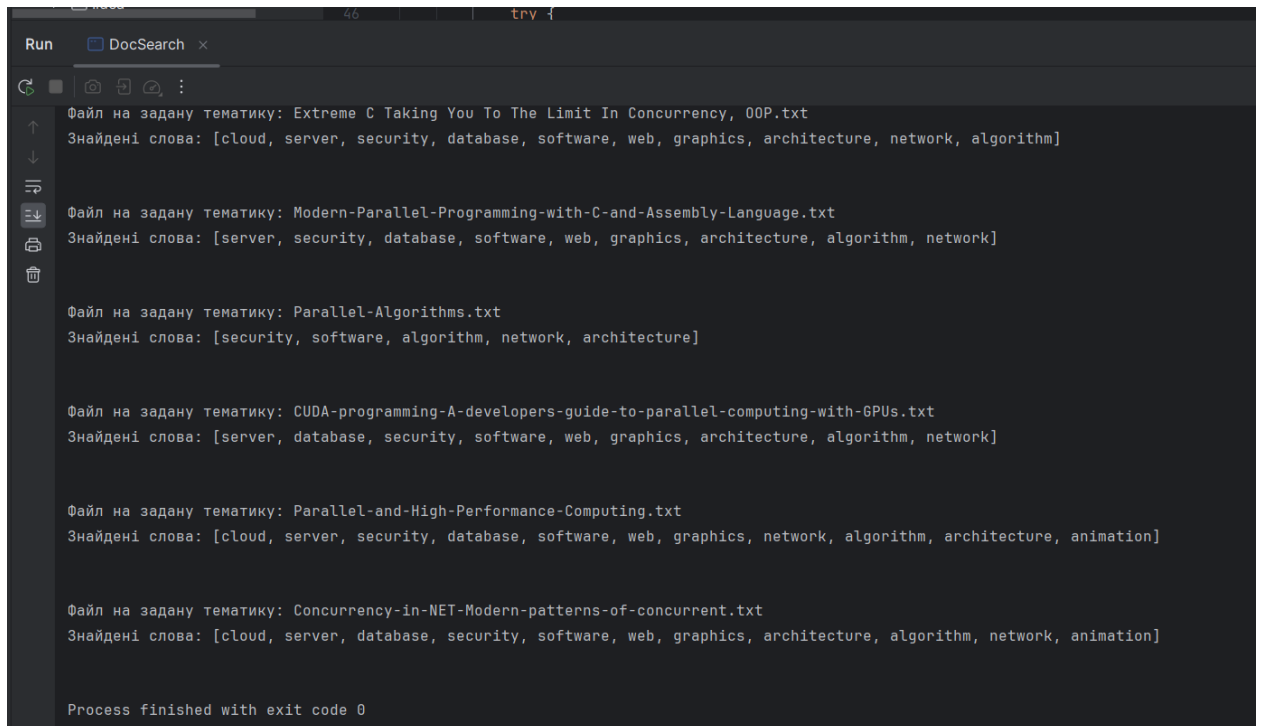
```
    }  
    if (!foundKeywords.isEmpty()) {  
        System.out.println("\nФайл на задану тематику: " +  
filePath.getFileName());  
        System.out.println("Знайдені слова: " + foundKeywords + "\n");  
    }  
  
}
```

```
    } catch (IOException e) {  
  
        e.printStackTrace();  
  
    }  
}  
}
```

```
public static void main(String[] args) {  
  
    ForkJoinPool pool = new ForkJoinPool();  
    DocSearchTask task = new DocSearchTask(Paths.get("D:\\kpi  
6\\ТПО\\parallel-labs\\lab4\\books"));  
  
    pool.invoke(task);  
  
    pool.shutdown();  
}  
}
```

Результат виконання коду:

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



```
Run DocSearch x
Файл на задану тематику: Extreme C Taking You To The Limit In Concurrency, OOP.txt
Знайдені слова: [cloud, server, security, database, software, web, graphics, architecture, network, algorithm]

Файл на задану тематику: Modern-Parallel-Programming-with-C-and-Assembly-Language.txt
Знайдені слова: [server, security, database, software, web, graphics, architecture, algorithm, network]

Файл на задану тематику: Parallel-Algorithms.txt
Знайдені слова: [security, software, algorithm, network, architecture]

Файл на задану тематику: CUDA-programming-A-developers-guide-to-parallel-computing-with-GPUs.txt
Знайдені слова: [server, database, security, software, web, graphics, architecture, algorithm, network]

Файл на задану тематику: Parallel-and-High-Performance-Computing.txt
Знайдені слова: [cloud, server, security, database, software, web, graphics, network, algorithm, architecture, animation]

Файл на задану тематику: Concurrency-in-NET-Modern-patterns-of-concurrent.txt
Знайдені слова: [cloud, server, database, security, software, web, graphics, architecture, algorithm, network, animation]

Process finished with exit code 0
```

Рисунок 4.1 – Результат виконання коду

Алгоритм пошуку текстових документів відпрацьовує коректно, виведено назви всіх файлів, що стосуються заданої тематики та ключові слова знайдені в них.

Висновок

У цій лабораторній роботі ми дослідили ефективність використання ForkJoinFramework для паралельного програмування на мові Java, реалізувавши кілька алгоритмів для обробки текстових документів.

Зокрема, ми побудували алгоритм статистичного аналізу тексту, визначивши характеристики випадкової величини «довжина слова в символах», алгоритм пошуку спільних слів у текстових документах та алгоритм пошуку текстових документів за заданою тематикою по ключовим словам.

Крім того, ми адаптували алгоритм з попереднього практикуму, щоб визначити прискорення за рахунок використання ForkJoinFramework. Наші експериментальні дослідження показали, що використання ForkJoinFramework значно підвищує ефективність обробки великих обсягів даних.