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 комп'ютерного практикуму №4 з дисципліни
«Технології паралельних обчислень»

«Розробка паралельних програм з використанням пулів потоків, екзекьюторів та ForkJoinFramework »

Виконав(ла) *ІП-11 Прищепа В. С.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Дифучин А. Ю.* _____
(прізвище, ім'я, по батькові)

Київ 2024

Завдання

1. Побудуйте алгоритм статистичного аналізу тексту та визначте характеристики випадкової величини «довжина слова в символах» з використанням ForkJoinFramework. 20 балів. Дослідіть побудований алгоритм аналізу текстових документів на ефективність експериментально. 10 балів.

2. Реалізуйте один з алгоритмів комп'ютерного практикуму 2 або 3 з використанням ForkJoinFramework та визначте прискорення, яке отримане за рахунок використання ForkJoinFramework. 20 балів.

3. Розробіть та реалізуйте алгоритм пошуку спільних слів в текстових документах з використанням ForkJoinFramework. 20 балів.

4. Розробіть та реалізуйте алгоритм пошуку текстових документів, які відповідають заданим ключовим словам (належать до області «Інформаційні технології»), з використанням ForkJoinFramework. 30 балів.

Виконання

Завдання 1

Лістинг

Main.java

```
package org.example.task1;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Main {
    public static void main(String[] args) throws IOException {
        String[] filenames = {"test2.txt","test3.txt","test4.txt"};
        int[] threadCounts = {1, 2, 4, 8};

        for (String filename : filenames) {
            for (int threadCount : threadCounts) {
                String text = new String(Files.readAllBytes(Paths.get("E:\\Install\\Projects\\
ParallelPrograming\\Lab4\\PPLab4\\src\\org\\example\\data\\"+filename)));
                String[] words = text.split("\\s+");
```

```
            long startTime = System.currentTimeMillis();
            WordStats stats = WordLengthAnalysis.analyze(words, threadCount);
            long endTime = System.currentTimeMillis();

            System.out.println("File: " + filename + ", Threads: " + threadCount +
                    ", Time: " + (endTime - startTime) + " ms, Average word length: " +
                    (double) stats.getTotalLength() / stats.getTotalCount());
        }
    }
}
```

WordLengthAnalysis.java

```
package org.example.task1;
import java.util.concurrent.RecursiveTask;
import java.util.concurrent.ForkJoinPool;

public class WordLengthAnalysis extends RecursiveTask<WordStats> {
    private static final int THRESHOLD = 10; // Поріг для розділення обробки

    private final String[] words;
    private final int start;
    private final int end;
    private final int threadCount;

    public WordLengthAnalysis(String[] words, int start, int end, int threadCount) {
        this.words = words;
        this.start = start;
        this.end = end;
        this.threadCount = threadCount;
    }

    @Override
    protected WordStats compute() {
        if (end - start < THRESHOLD) {
            return computeDirectly();
        } else {
            int middle = start + (end - start) / 2;
            WordLengthAnalysis leftTask = new WordLengthAnalysis(words, start,
middle, threadCount);
            WordLengthAnalysis rightTask = new WordLengthAnalysis(words, middle,
end, threadCount);

            leftTask.fork();
```

```
            WordStats rightResult = rightTask.compute();
            WordStats leftResult = leftTask.join();

            return leftResult.merge(rightResult);
        }
    }

    private WordStats computeDirectly() {
        int totalLength = 0;
        int totalCount = 0;

        for (int i = start; i < end; i++) {
            totalLength += words[i].length();
            totalCount++;
        }

        return new WordStats(totalLength, totalCount);
    }

    public static WordStats analyze(String[] words, int threadCount) {
        ForkJoinPool pool = new ForkJoinPool(threadCount);
        return pool.invoke(new WordLengthAnalysis(words, 0, words.length,
threadCount));
    }

    public static void main(String[] args) {
        String text = "Sample text for analysis. This is a test.";
        String[] words = text.split("\\s+");
        WordStats stats = WordLengthAnalysis.analyze(words, 4); // Приклад з
використанням 4 потоків
        System.out.println("Average word length: " + (double) stats.getTotalLength() /
stats.getTotalCount());
    }
}
```

WordStats.java

```
package org.example.task1;

class WordStats {
    private final int totalLength;
    private final int totalCount;

    public WordStats(int totalLength, int totalCount) {
        this.totalLength = totalLength;
        this.totalCount = totalCount;
```

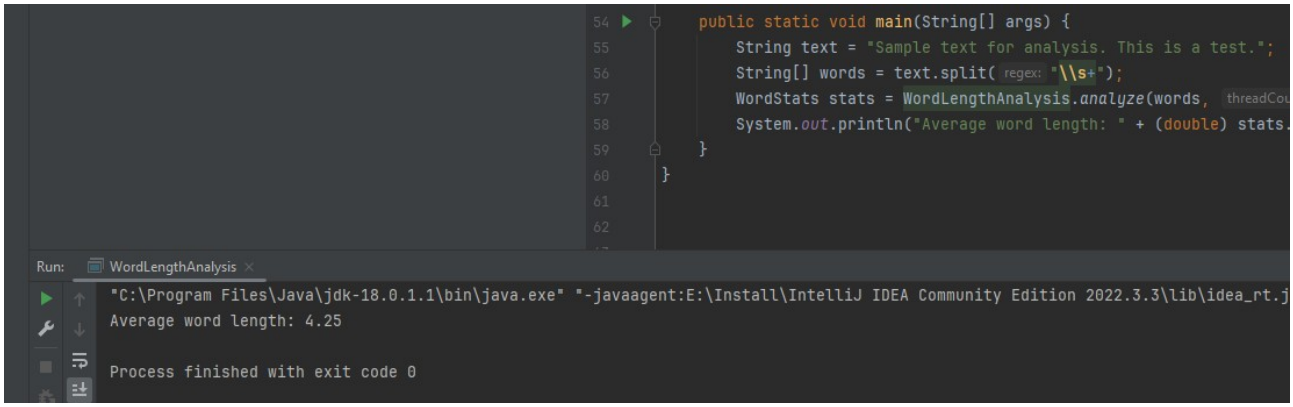
```
    }

    public WordStats merge(WordStats other) {
        return new WordStats(this.totalLength + other.totalLength, this.totalCount +
other.totalCount);
    }

    public int getTotalLength() {
        return totalLength;
    }

    public int getTotalCount() {
        return totalCount;
    }
}
```

Результат



Результат роботи алгоритму на невеличкому тексті.

```
File: test2.txt, Threads: 1, Time: 4 ms, Average word length: 4.288770053475936
File: test2.txt, Threads: 2, Time: 0 ms, Average word length: 4.288770053475936
File: test2.txt, Threads: 4, Time: 1 ms, Average word length: 4.288770053475936
File: test2.txt, Threads: 8, Time: 1 ms, Average word length: 4.288770053475936
File: test3.txt, Threads: 1, Time: 2 ms, Average word length: 5.1070678796361095
File: test3.txt, Threads: 2, Time: 0 ms, Average word length: 5.1070678796361095
File: test3.txt, Threads: 4, Time: 1 ms, Average word length: 5.1070678796361095
File: test3.txt, Threads: 8, Time: 1 ms, Average word length: 5.1070678796361095
File: test4.txt, Threads: 1, Time: 3 ms, Average word length: 5.7635986237879955
File: test4.txt, Threads: 2, Time: 1 ms, Average word length: 5.7635986237879955
File: test4.txt, Threads: 4, Time: 1 ms, Average word length: 5.7635986237879955
File: test4.txt, Threads: 8, Time: 2 ms, Average word length: 5.7635986237879955
```

Дослідження ефективності роботи алгоритму із різною кількістю потоків та розмірами текстових файлів.

Завдання 2

Лістинг

ForkFoxMatrixMultiplier.java

```
package org.example.task2;
import java.util.concurrent.Callable;

import java.util.concurrent.RecursiveTask;
import java.util.concurrent.ForkJoinPool;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class ForkFoxMatrixMultiplier {
    private static class Result {
        private final int[][] matrix;
        private final int rowOffset;
        private final int colOffset;

        public Result(int[][] matrix, int rowOffset, int colOffset) {
            this.matrix = matrix;
            this.rowOffset = rowOffset;
            this.colOffset = colOffset;
        }

        public int[][] getMatrix() {
            return matrix;
        }

        public int getRowOffset() {
            return rowOffset;
        }

        public int getColOffset() {
            return colOffset;
        }
    }

    private static class MatrixMultiplicationTask extends RecursiveTask<Result> {
        private final int[][] matrixA;
        private final int[][] matrixB;
        private final int rowOffset;
        private final int colOffset;
```

```
        private final int k;
        private final int blockSize;

        public MatrixMultiplicationTask(int[][] matrixA, int[][] matrixB, int rowOffset,
int colOffset, int k, int blockSize) {
            this.matrixA = matrixA;
            this.matrixB = matrixB;
            this.rowOffset = rowOffset;
            this.colOffset = colOffset;
            this.k = k;
            this.blockSize = blockSize;
        }

        @Override
        protected Result compute() {
            int[][] result = new int[blockSize][blockSize];
            for (int i = 0; i < blockSize; i++) {
                for (int j = 0; j < blockSize; j++) {
                    for (int x = 0; x < blockSize; x++) {
                        result[i][j] += matrixA[rowOffset + i][k + x] * matrixB[k + x]
[colOffset + j];
                    }
                }
            }
            return new Result(result, rowOffset, colOffset);
        }
    }

    public static int[][] multiply2(int[][] matrixA, int[][] matrixB, int blockSize, int
numThreads) {
        ForkJoinPool pool = new ForkJoinPool(numThreads);
        int[][] result = new int[matrixA.length][matrixB[0].length];
        try {
            for (int i = 0; i < matrixA.length; i += blockSize) {
                for (int j = 0; j < matrixB[0].length; j += blockSize) {
                    List<MatrixMultiplicationTask> tasks = new ArrayList<>();
                    for (int k = 0; k < matrixA[0].length; k += blockSize) {
                        tasks.add(new MatrixMultiplicationTask(matrixA, matrixB, i, j, k,
blockSize));
                    }

                    List<Callable<Result>> callableTasks = tasks.stream().map(task ->
(Callable<Result>) task::compute).collect(Collectors.toList());

                    List<Result> results =
```

```
pool.invokeAll(callableTasks).stream().map(future -> {
                    try {
                        return future.get();
                    } catch (Exception e) {
                        e.printStackTrace();
                        return null;
                    }
                }).collect(Collectors.toList());

                for (Result res : results) {
                    for (int x = 0; x < blockSize; x++) {
                        for (int y = 0; y < blockSize; y++) {
                            result[res.getRowOffset() + x][res.getColOffset() + y] +=
res.getMatrix()[x][y];
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            pool.shutdown();
        }
        return result;
    }

    public static void main(String[] args) {
        int[][] matrix1 = {
                {23, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
        };
        int[][] matrix2 = {
                {2, 25, 2},
                {4, 25, 6},
                {2, 25, 29}
        };

        int[][] result = ForkFoxMatrixMultiplier.multiply2(matrix1, matrix2, 1, 4);
        printMatrix(result);
    }

    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();
        }
    }

}
```

FoxMatrixMultiplier.java

```
package org.example.task2;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

class FoxMatrixMultiplier {
    private static class Result {
        private final int[][] matrix;
        private final int rowOffset;
        private final int colOffset;

        public Result(int[][] matrix, int rowOffset, int colOffset) {
            this.matrix = matrix;
            this.rowOffset = rowOffset;
            this.colOffset = colOffset;
        }

        public int[][] getMatrix() {
            return matrix;
        }

        public int getRowOffset() {
            return rowOffset;
        }

        public int getColOffset() {
            return colOffset;
        }
    }
```

```
    private static class MatrixMultiplicationTask implements Callable<Result> {
        private final int[][] matrixA;
        private final int[][] matrixB;
        private final int rowOffset;
        private final int colOffset;
        private final int k;
        private final int blockSize;

        public MatrixMultiplicationTask(int[][] matrixA, int[][] matrixB, int rowOffset,
int colOffset, int k, int blockSize) {
            this.matrixA = matrixA;
            this.matrixB = matrixB;
            this.rowOffset = rowOffset;
            this.colOffset = colOffset;
            this.k = k;
            this.blockSize = blockSize;
        }

        @Override
        public Result call() {
            int[][] result = new int[blockSize][blockSize];
            for (int i = 0; i < blockSize; i++) {
                for (int j = 0; j < blockSize; j++) {
                    for (int x = 0; x < blockSize; x++) {
                        result[i][j] += matrixA[rowOffset + i][k + x] * matrixB[k + x]
[colOffset + j];
                    }
                }
            }
            return new Result(result, rowOffset, colOffset);
        }
    }

    public static int[][] multiply2(int[][] matrixA, int[][] matrixB, int blockSize, int
numThreads) {
        ExecutorService executor = Executors.newFixedThreadPool(numThreads);
        int[][] result = new int[matrixA.length][matrixB[0].length];
        try {
            for (int i = 0; i < matrixA.length; i += blockSize) {
                for (int j = 0; j < matrixB[0].length; j += blockSize) {
                    List<Future<Result>> futures = new ArrayList<>();
                    for (int k = 0; k < matrixA[0].length; k += blockSize) {
                        futures.add(executor.submit(new MatrixMultiplicationTask(matrixA,
matrixB, i, j, k, blockSize)));
                    }
```

```
                for (Future<Result> future : futures) {
                    Result res = future.get();
                    for (int x = 0; x < blockSize; x++) {
                        for (int y = 0; y < blockSize; y++) {
                            result[res.getRowOffset() + x][res.getColOffset() + y] +=
res.getMatrix()[x][y];
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            executor.shutdown();
        }
        return result;
    }
    private static int[][] generateMatrix(int rows, int cols) {
        Random random = new Random();
        int[][] matrix = new int[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                int randomInt = random.nextInt(5) + 1; // Генеруємо випадкове ціле
число від 1 до 5
                matrix[i][j] = randomInt; // Присвоюємо це ціле число як частину
дійсного числа
            }
        }
        return matrix;
    }
    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();
        }
    }

    public static void main(String[] args) {
        int[][] matrix1 = {
                {23, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
```

```
        };
        int[][] matrix2 = {
                {2, 25, 2},
                {4, 25, 6},
                {2, 25, 29}
        };

        int[][] result = FoxMatrixMultiplier.multiply2(matrix1, matrix2, 1, 4);
        printMatrix(result);
    }
}
```

MatrixMultiplicationExperiment.java

```
package org.example.task2;

import java.util.Random;

public class MatrixMultiplicationExperiment {

    private static int[][] generateRandomMatrix(int rows, int cols) {
        Random random = new Random();
        int[][] matrix = new int[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                int randomInt = random.nextInt(5) + 1; // Генеруємо випадкове ціле
число від 1 до 5
                matrix[i][j] = randomInt; // Присвоюємо це ціле число як частину
дійсного числа
            }
        }
        return matrix;
    }


    private static long measureExecutionTime(Runnable task) {
        long startTime = System.nanoTime();
        task.run();
        long endTime = System.nanoTime();
        return endTime - startTime;
    }
    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();
        }
    }
    public static void main(String[] args) {
        final int[] dimensions = {50,500,1000,1500};
        final int numThreads = Runtime.getRuntime().availableProcessors();

        for (int dim : dimensions) {
            int[][] matrixA = generateRandomMatrix(dim, dim);
            int[][] matrixC = matrixA;
            int[][] matrixB = generateRandomMatrix(dim, dim);
            int[][] matrixD = matrixB;
            long ForkTime = measureExecutionTime(() -> {
                int[][] result1 = ForkFoxMatrixMultiplier.multiply2(matrixA, matrixB,
5,8);
            });
            long foxTime = measureExecutionTime(() -> {
                int[][] result = FoxMatrixMultiplier.multiply2(matrixC, matrixD, 5, 8);
            });

            System.out.println("Matrix Dimension: " + dim);
            System.out.println("Fork Fox Algorithm Time: " + ForkTime / 1_000_000 +
"ms");
            System.out.println("Fox Algorithm Time: " + foxTime / 1_000_000 + "ms");
            System.out.println();
        }

    }
}
```

Результат

```
Matrix Dimension: 50
Fork Fox Algorithm Time: 22ms
Fox Algorithm Time: 11ms

Matrix Dimension: 500
Fork Fox Algorithm Time: 718ms
Fox Algorithm Time: 629ms

Matrix Dimension: 1000
Fork Fox Algorithm Time: 4077ms
Fox Algorithm Time: 3983ms

Matrix Dimension: 1500
Fork Fox Algorithm Time: 11584ms
Fox Algorithm Time: 12740ms
```

Результат виконання програми порівняння алгоритму множення Фокса з і без ForkJoinFramework.

Таблиця результатів порівняння (у мс.):

Розмірність	2 потоки		4 потоки		8 потоків	
	Із	Без	Із	Без	Із	Без
50	22	11	22	12	21	13
500	718	629	705	877	803	877
1000	4077	3983	3485	6524	4177	6084
1500	11584	12740	9266	20269	12540	19858

Бачимо, що ForkJoinFramework дає прискорення майже у два рази в середньому. Чим більша розмірність і кількість задієних потоків, тим більше прискорення.

Завдання 3

Лістинг

CommonWordsFinder.java

```java
package org.example.task3;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;
import java.util.concurrent.RecursiveTask;
import java.util.concurrent.ForkJoinPool;
```

```java
public class CommonWordsFinder {

    private static class CommonWordsTask extends RecursiveTask<Set<String>> {
        private final List<String> words1;
        private final List<String> words2;

        public CommonWordsTask(List<String> words1, List<String> words2) {
            this.words1 = words1;
            this.words2 = words2;
        }

        @Override
        protected Set<String> compute() {
            Set<String> commonWords = new HashSet<>();
            for (String word : words1) {
                if (words2.contains(word)) {
                    commonWords.add(word);
                }
            }
            return commonWords;
        }
    }

    public static Set<String> findCommonWords(String filePath1, String filePath2)
throws Exception {
        String content1 = new String(Files.readAllBytes(Paths.get(filePath1)));
        String content2 = new String(Files.readAllBytes(Paths.get(filePath2)));

        List<String> words1 = Arrays.asList(content1.split("\\s+"));
        List<String> words2 = Arrays.asList(content2.split("\\s+"));

        ForkJoinPool forkJoinPool = new ForkJoinPool();
        return forkJoinPool.invoke(new CommonWordsTask(words1, words2));
    }


    public static void main(String[] args) throws Exception {
        String filePath1 = "E:\\Install\\Projects\\ParallelPrograming\\Lab4\\PPLab4\\
src\\org\\example\\data\\test2.txt";
        String filePath2 = "E:\\Install\\Projects\\ParallelPrograming\\Lab4\\PPLab4\\
src\\org\\example\\data\\test3.txt";
        Set<String> commonWords = findCommonWords(filePath1, filePath2);
        System.out.println("Common words: " + commonWords);
    }
}
```

Результат

```
Common words: [all, a, be, in, was, for, is, an, him, the, The, as, at, his, her, and, of, to, first]

Process finished with exit code 0
```

Із двох англомовних текстів було виявлено усі спільні слова.

Завдання 4

Лістинг

KeywordMatcher.java

```
package org.example.task4;

import java.util.Arrays;
import java.util.Map;
import java.util.concurrent.ForkJoinTask;
import java.util.concurrent.RecursiveAction;
import java.util.stream.Collectors;

public class KeywordMatcher {
    public static boolean matchesKeywords(String[] keywords, String text) {
        text = text.trim().replaceAll("\\s+", " ");

        final Map<String, Boolean> keywordsMap = Arrays.stream(keywords)
                .collect(Collectors.toMap(keyword -> keyword, keyword -> false));

        final TextMatchRecursive textMatch = new TextMatchRecursive(keywordsMap,
text, 0, text.length());
        TextMatchRecursive.invokeAll(textMatch);

        return keywordsMap
                .values()
                .stream()
                .allMatch(bool -> bool);
    }

    private static class TextMatchRecursive extends RecursiveAction {
        private static final String WORD_DELIMITER = " ";

        private final Map<String, Boolean> keywords;
        private final String subtext;
        private final int beginText;
```

```
    private final int endText;

    public TextMatchRecursive(Map<String, Boolean> keywords, String subtext, int
beginText, int endText) {
        this.keywords = keywords;
        this.subtext = subtext;
        this.beginText = beginText;
        this.endText = endText;
    }

    @Override
    protected void compute() {
        final int center = subtext.length() / 2;
        final int indexRight = subtext.indexOf(WORD_DELIMITER, center);
        final int indexLeft = subtext.lastIndexOf(WORD_DELIMITER, center);

        if (indexRight != -1 || indexLeft != -1) {
            if (center - indexLeft > Math.abs(indexRight - center)) {
                this.splitJoin(indexRight);
            } else {
                this.splitJoin(indexLeft);
            }
        } else {
            this.verifyWord();
        }
    }

    private void splitJoin(int index) {
        final TextMatchRecursive splitLeft = new TextMatchRecursive(
                keywords,
                subtext.substring(0, index),
                beginText,
                beginText + index
        );
        final TextMatchRecursive splitRight = new TextMatchRecursive(
                keywords,
                subtext.substring(index + WORD_DELIMITER.length()),
                beginText + index + WORD_DELIMITER.length(),
                endText
        );
        ForkJoinTask.invokeAll(splitLeft, splitRight);
    }

    private void verifyWord() {
        for (String keyword : keywords.keySet()) {
```

```
            if (subtext.equalsIgnoreCase(keyword)) {
                synchronized (keywords) {
                    keywords.put(keyword, true);
                }
            }
        }
    }
}
```

Main.java

```
package org.example.task4;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;

class Main {
    public static void main(String[] args) throws IOException {
        File[] files = new File("E:\\Install\\Projects\\ParallelPrograming\\Lab4\\PPLab4\\
src\\org\\example\\data\\").listFiles();
        assert files != null;
        final String[] keywords = {"lorem" };

        for (File file : files) {
            final String text = Files.readString(file.toPath());
            if (KeywordMatcher.matchesKeywords(keywords, text)) {
                System.out.println("Found matches in file: " + file.getPath());
            }
        }
    }
}
```

Результат

```
Found matches in file: E:\Install\Projects\ParallelPrograming\Lab4\PPLab4\src\org\example\data\test1.txt
Found matches in file: E:\Install\Projects\ParallelPrograming\Lab4\PPLab4\src\org\example\data\test4.txt

Process finished with exit code 0
```

Програма відшукала вказане ключове слово у двох текстах.

Висновок: у даній лабораторній роботі я навчився використовувати ForkJoinFramework, з’ясував на практиці, що він дає прискорення майже у два рази, застосував його для виконання обробки текстів.