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

Лабораторна робота №4
Технології паралельних обчислень

Виконав студент групи ІТ-03: Чабан А.Є.		Перевірив: Дифучина О.Ю
		Дата:
		Оцінка:

Київ 2023

Завдання:

5.4 Завдання до комп'ютерного практикуму 4 «Розробка паралельних програм з використанням пулів потоків, екзекьюторів та ForkJoinFramework»

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.**

Хід виконання:

Завдання 1:

Було реалізовано 2 класи DirLengthStat та FileLengthStat, в класі для директорії відбувається читання файлів у директорії та створення підзадач для кожного файлу. У класі FileLengthStat відбувається безпосередньо обчислення статистики слів для конкретного файлу, в методі compute() перевіряється чи розділено файл на слова, далі в залежності від кількості слів відбувається підрахунок статистики, або розділення на 2 підзадачі і рекурсивний виклик. Цей клас реалізує рекурсивний алгоритм розбиття файлу на підзавдання та підрахунок статистики слів.

DirLengthStat.java | FileLengthStat.java | Main.java

```java
package lab4.task1;

import ...

1 usage  Anton Chaban
public class DirLengthStat extends RecursiveTask<HashMap<Integer, Integer>> {
    2 usages
    private final List<String> filePaths;
    1 usage  Anton Chaban
    public DirLengthStat(String dirPath) {
        try (Stream<Path> walk = Files.walk(Paths.get(dirPath))) {
            filePaths = walk.filter(Files::isRegularFile) Stream<Path>
                    .map(Path::toString) Stream<String>
                    .collect(Collectors.toList());
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException("Error while reading files");
        }
    }

    Anton Chaban
    @Override
    protected HashMap<Integer, Integer> compute() {
        var tasks = new ArrayList<FileLengthStat>();

        for(String filePath : filePaths) {
            var task = new FileLengthStat(filePath);
            task.fork();
            tasks.add(task);
        }

        var result = new HashMap<Integer, Integer>();

        for(FileLengthStat task : tasks) {
            task.join().forEach((lengthKey, count) ->
                    result.merge(lengthKey, count, Integer::sum)
            );
        }

        return result;
    }
}
```

```java
5 usages  Anton Chaban
public class FileLengthStat extends RecursiveTask<HashMap<Integer, Integer>> {
    5 usages
    public final String filePath;
    6 usages
    private List<String> wordsList;
    6 usages
    private int start;
    6 usages
    private int end;
    3 usages
    private final boolean splitted;

    1 usage  Anton Chaban
    public FileLengthStat(String filePath) {
        this.filePath = filePath;
        splitted = false;
    }

    2 usages  Anton Chaban
    public FileLengthStat(String filePath, List<String> wordsList, int start, int end) {
        this.filePath = filePath;
        this.wordsList = wordsList;
        this.start = start;
        this.end = end;
        splitted = true;
    }

    Anton Chaban
    @Override
    protected HashMap<Integer, Integer> compute() {
        if (!splitted) {
```

```
        if (!splitted) {
            initWordsList();
        }

        if (end - start < 200_000) {
            return getWordsData();
        }

        var midIndex = (end + start) / 2;

        var leftTask = new FileLengthStat(filePath, wordsList, start, midIndex);
        leftTask.fork();

        var rightTask = new FileLengthStat(filePath, wordsList, midIndex, end);

        var result = rightTask.compute();
        leftTask.join().forEach((lengthKey, count) ->
                result.merge(lengthKey, count, Integer::sum)
        );

        return result;
    }

    1 usage  Anton Chaban
    private HashMap<Integer, Integer> getWordsData() {
        var lengthsMapper = new HashMap<Integer, Integer>();

        wordsList.subList(start, end).forEach(word -> {
            var wordLength = word.length();

            if (lengthsMapper.containsKey(wordLength)) {
```

```
60
61                  if (lengthsMapper.containsKey(wordLength)) {
62                      lengthsMapper.put(wordLength, lengthsMapper.get(wordLength) + 1);
63                  } else {
64                      lengthsMapper.put(wordLength, 1);
65                  }
66              });
67
68              return lengthsMapper;
69          }
70
            1 usage  Anton Chaban
71          private void initWordsList() {
72              try {
73                  String content = Files.readString(Paths.get(filePath));
74                  wordsList = List.of(content.split( regex: "\\s+"));
75                  start = 0;
76                  end = wordsList.size();
77              } catch (IOException e) {
78                  e.printStackTrace();
79                  throw new RuntimeException("Error while reading file");
80              }
81          }
82      }
83
```

Послідовна реалізація:

```
1 usage  Anton Chaban
private static void seqAnalyseDir(String dirPath) {
    var filePaths = new ArrayList<Path>();
    try {
        Files.walk(Paths.get(dirPath))
                .filter(path -> Files.isRegularFile(path))
                .forEach(e -> filePaths.add(e));
    } catch (IOException e) {
        e.printStackTrace();
    }

    var result = new HashMap<Integer, Integer>();

    for (Path filePath : filePaths) {
        try {
            var content = Files.readString(filePath);
            String[] words = content.split( regex: "\\s+");

            for (String word : words) {
                int length = word.length();
                result.put(length, result.getOrDefault(length,  defaultValue: 0) + 1);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    printStatistics(result);
}
```

Статистика:

```
C:\Users\anton\.jdks\corretto-11.0.18\bin\java.
Total amount of words: 1053375
Avg length of words: 4.57
Time taken for sequential analyse : 396 ms

Total amount of words: 1053375
Avg length of words: 4.57
Time taken for parallel analyse : 156 ms
```

Завдання 2:

У класі FJFoxCalculatorTask створюються підзадачі для заданих для множення блоків матриці і записуються у результуючу матрицю. У класі FJFoxCalculator створюється пул для розподілення задач між заданою кількістю потоків, в залежності від кількості доступних потоків визначається «крок» в обчисленнях і створюється відповідна підзадача для блоку матриці в межах цього кроку.

```java
1 usage  Anton Chaban
public class FJFoxCalculatorTask extends RecursiveAction {
    7 usages
    private final MatrixEntity matrixEntity1;
    6 usages
    private final MatrixEntity matrixEntity2;
    9 usages
    private final int curRowShift;
    9 usages
    private final int curColShift;
    10 usages
    private final int blockSize;
    3 usages
    private final MatrixEntity resultMatrix;

    1 usage  Anton Chaban
    public FJFoxCalculatorTask(MatrixEntity matrixEntity1, MatrixEntity matrixEntity2, int curRowShift,
                               int curColShift, int blockSize, MatrixEntity resultMatrix) {
        this.resultMatrix = resultMatrix;
        this.matrixEntity1 = matrixEntity1;
        this.matrixEntity2 = matrixEntity2;
        this.curRowShift = curRowShift;
        this.curColShift = curColShift;
        this.blockSize = blockSize;
    }

    Anton Chaban
    @Override
    protected void compute() {
        var m1RowSize = blockSize;
        var m2ColSize = blockSize;

        var m2ColSize = blockSize;

        if (curRowShift + blockSize > matrixEntity1.getRowsSize())
            m1RowSize = matrixEntity1.getRowsSize() - curRowShift;

        if (curColShift + blockSize > matrixEntity2.getColumnsSize())
            m2ColSize = matrixEntity2.getColumnsSize() - curColShift;

        for (int k = 0; k < matrixEntity1.getColumnsSize(); k += blockSize) {
            var m1ColSize = blockSize;
            var m2RowSize = blockSize;

            if (curRowShift + blockSize > matrixEntity1.getRowsSize()) {
                m1ColSize = matrixEntity1.getRowsSize() - curRowShift;
            }

            if (curColShift + blockSize > matrixEntity2.getRowsSize()) {
                m2RowSize = matrixEntity2.getRowsSize() - curColShift;
            }

            var blockFirst = copyBlock(matrixEntity1, curRowShift, rowFinish: curRowShift + m1RowSize, k, colFinish: k + m1ColSize);
            var blockSecond = copyBlock(matrixEntity2, k, rowFinish: k + m2RowSize, curColShift, colFinish: curColShift + m2ColSize);

            var resBlock = new SequentialCalculator().multiplyMatrix(blockFirst, blockSecond);
            for (int i = 0; i < resBlock.getRowsSize(); i++) {
                for (int j = 0; j < resBlock.getColumnsSize(); j++) {
                    resultMatrix.set( i: i + curRowShift,  j: j + curColShift,
                            value: resBlock.get(i, j) + resultMatrix.get( i: i + curRowShift,  j: j + curColShift));
                }
            }
        }
    }
```

```java
@RequiredArgsConstructor
public class FJFoxCalculator {
    private MatrixEntity matrixEntity1;
    private MatrixEntity matrixEntity2;
    private int threadsCount;
    private MatrixEntity resultMatrix;

    1 usage  Anton Chaban
    public FJFoxCalculator(MatrixEntity matrixEntity1, MatrixEntity matrixEntity2, int threadsCount) {
        this.matrixEntity1 = matrixEntity1;
        this.matrixEntity2 = matrixEntity2;
        this.resultMatrix = new MatrixEntity(matrixEntity1.getRowsSize(), matrixEntity2.getColumnsSize());

        if (threadsCount > matrixEntity1.getRowsSize() * matrixEntity2.getColumnsSize() / 4) {
            this.threadsCount = matrixEntity1.getRowsSize() * matrixEntity2.getColumnsSize() / 4;
        } else {
            this.threadsCount = Math.max(threadsCount, 1);
        }
    }

    Anton Chaban
    public MatrixEntity multiplyMatrix() {
        var step = (int) Math.ceil(1.0 * matrixEntity1.getRowsSize() / (int) Math.sqrt(threadsCount));

        ForkJoinPool pool = new ForkJoinPool(threadsCount);
        System.out.println("threads count: " + pool.getParallelism());

        for (int i = 0; i < matrixEntity1.getRowsSize(); i += step) {
            for (int j = 0; j < matrixEntity2.getColumnsSize(); j += step) {
                pool.invoke(new FJFoxCalculatorTask(matrixEntity1, matrixEntity2, i, j, step, resultMatrix));
            }
        }
```

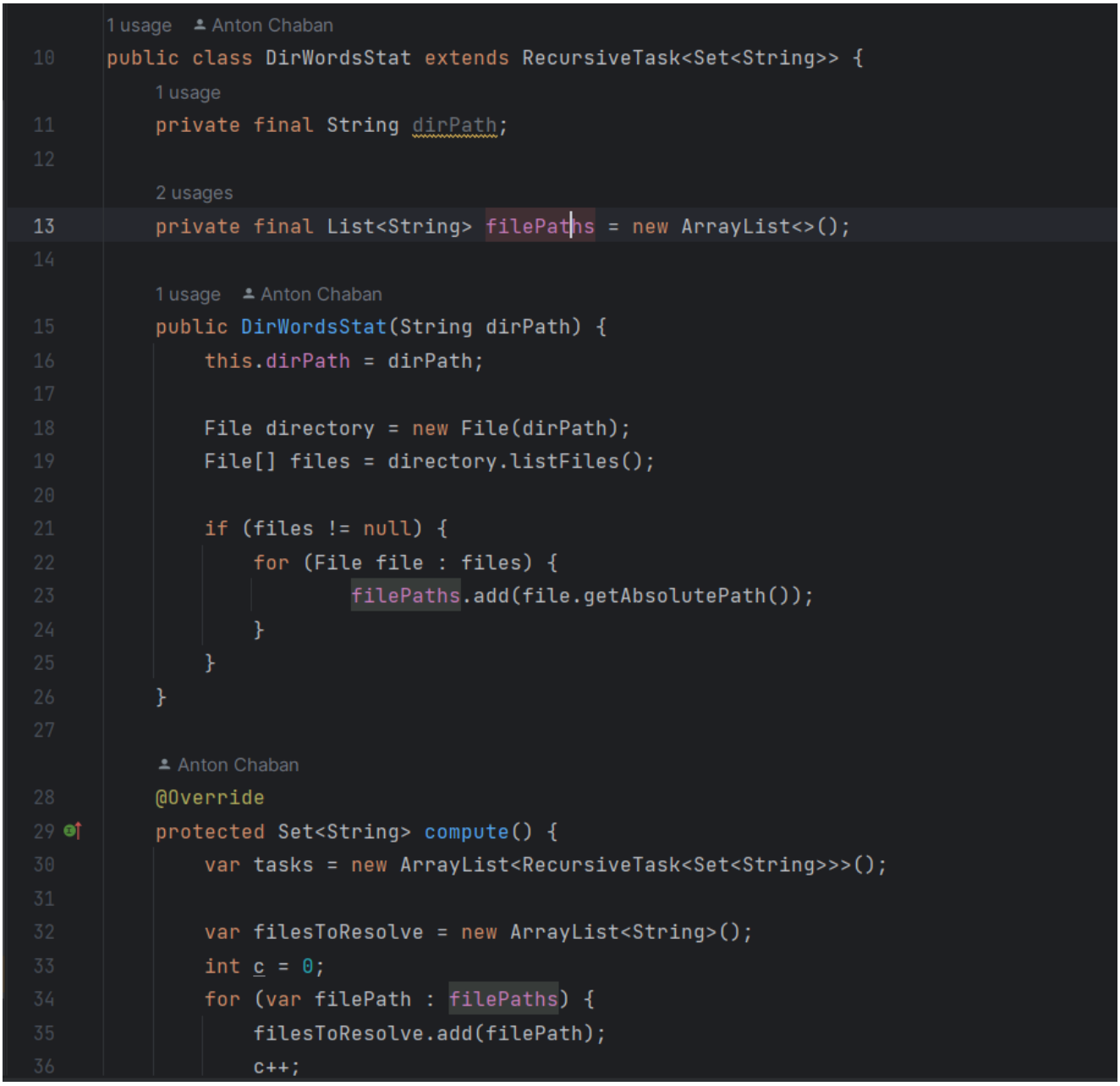
Результати:

```
C:\Users\anton\.jdks\corretto-11.0.18\bin\java.exe
sequential result:
43996
fox result:
threads count: 4
7927
```

Прискорення для матриці 1500x1500 становить 5,55.

Завдання 3:

В класі DirWordsStat виконується обробка директорії та створення задач для кожного файлу в директорії. Якщо кількість файлів для обробки перевищує 2, то створюється ще одна задача FileWordStat. Після обробки задач у FileWordStat до сету слів використовується метод retainAll(), щоб знайти взаємний перетин між ними.



```
1 usage  Anton Chaban
public class DirWordsStat extends RecursiveTask<Set<String>> {
    1 usage
    private final String dirPath;

    2 usages
    private final List<String> filePaths = new ArrayList<>();

    1 usage  Anton Chaban
    public DirWordsStat(String dirPath) {
        this.dirPath = dirPath;

        File directory = new File(dirPath);
        File[] files = directory.listFiles();

        if (files != null) {
            for (File file : files) {
                    filePaths.add(file.getAbsolutePath());
            }
        }
    }

    Anton Chaban
    @Override
    protected Set<String> compute() {
        var tasks = new ArrayList<RecursiveTask<Set<String>>>();

        var filesToResolve = new ArrayList<String>();
        int c = 0;
        for (var filePath : filePaths) {
            filesToResolve.add(filePath);
            c++;
```

```
37
38              if (c >= 2) {
39                  var task = new FileWordsStat(new ArrayList<>(filesToResolve));
40                  tasks.add(task);
41                  task.fork();
42
43                  c = 0;
44                  filesToResolve.clear();
45              }
46          }
47          if (!filesToResolve.isEmpty()) {
48              var task = new FileWordsStat(new ArrayList<>(filesToResolve));
49              tasks.add(task);
50              task.fork();
51          }
52
53          var setsToIntersect = new ArrayList<Set<String>>();
54          for (RecursiveTask<Set<String>> task : tasks) {
55              setsToIntersect.add(task.join());
56          }
57
58          Set<String> intersectionOfSets = new HashSet<>(setsToIntersect.get(0));
59          for (int i = 1; i < setsToIntersect.size(); i++) {
60              intersectionOfSets.retainAll(setsToIntersect.get(i));
61          }
62
63          return intersectionOfSets;
64      }
65  }
66
```

```java
@AllArgsConstructor
public class FileWordsStat extends RecursiveTask<Set<String>> {
    private final ArrayList<String> filePaths;


    Anton Chaban
    @Override
    protected Set<String> compute() {
        var setsToIntersect = new ArrayList<Set<String>>();

        for (String filePath : filePaths) {
            setsToIntersect.add(getSetFromFile(filePath));
        }

        Set<String> intersectionOfSets = new HashSet<>(setsToIntersect.get(0));
        for (int i = 1; i < setsToIntersect.size(); i++) {
            intersectionOfSets.retainAll(setsToIntersect.get(i));
        }

        return intersectionOfSets;
    }

    1 usage  Anton Chaban
    private Set<String> getSetFromFile(String filePath) {
        try {
            return Files.lines(Paths.get(filePath))
                    .flatMap(line -> List.of(line.split( regex: " ")).stream())
                    .map(word -> word.toLowerCase().replaceAll( regex: "[^\\p{L}]",  replacement: ""))
                    .filter(word -> !word.isEmpty())
                    .collect(Collectors.toSet());
        } catch (IOException e) {
            throw new RuntimeException(e);
```

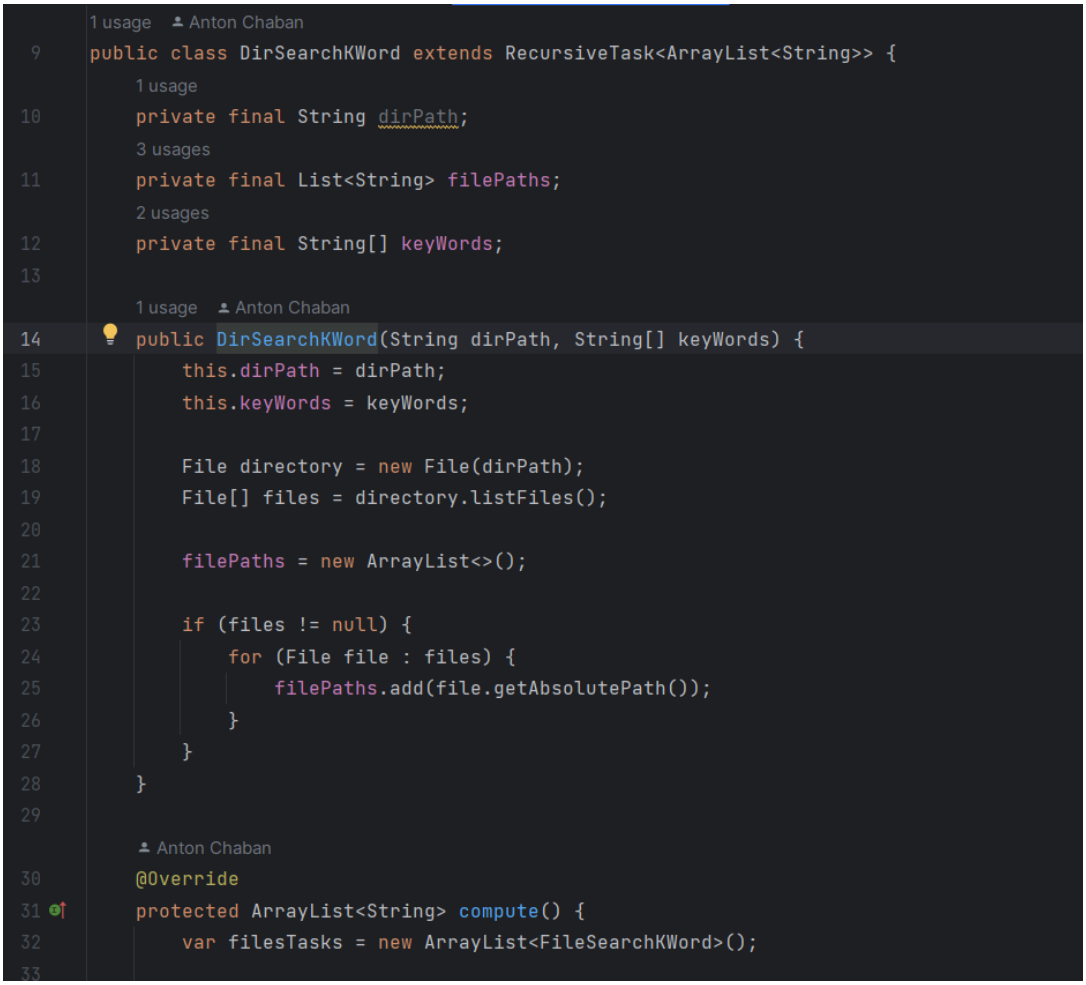
Результат:

```
C:\Users\anton\.jdks\corretto-11.0.18\bin\java.exe -javaagent:C:\Users\anton\AppData\Local\JetBrains\Toolbox\apps\IDEA-U\ch-0\231.8770.65\lib\idea
Common words for all files: [half, spoke, fifty, shooting, hall, owners, pretend, guards, drunk, would, pick, legs, gloomy, rattled, ten, daring,

Process finished with exit code 0
```

Завдання 4:

Клас для директорії аналогічно попереднім класам зчитує директорію та створює задачі для читання файлів. У класі FileSearchKWord виконується перевірка на розмір файла, у випадку перевищення 200 тис. слів – виконується розділення на 2 підзадачі і задача виконується рекурсивно, поки не буде виконана умова для виходу з рекурсії та пошуку ключових слів у файлі за допомогою методу wordsListContainsSearchWord(). Після завершення задачі в FileSearchKWord алгоритм повертається до DirSearch, і у випадку, якщо ключове слово було знайдено – шлях до файлу додається до результуючого списку.



```
1 usage  Anton Chaban
public class DirSearchKWord extends RecursiveTask<ArrayList<String>> {
    1 usage
    private final String dirPath;
    3 usages
    private final List<String> filePaths;
    2 usages
    private final String[] keyWords;

    1 usage  Anton Chaban
    public DirSearchKWord(String dirPath, String[] keyWords) {
        this.dirPath = dirPath;
        this.keyWords = keyWords;

        File directory = new File(dirPath);
        File[] files = directory.listFiles();

        filePaths = new ArrayList<>();

        if (files != null) {
            for (File file : files) {
                filePaths.add(file.getAbsolutePath());
            }
        }
    }

    Anton Chaban
    @Override
    protected ArrayList<String> compute() {
        var filesTasks = new ArrayList<FileSearchKWord>();
```

```
        for (String filePath : filePaths) {
            var task = new FileSearchKWord(filePath, keyWords);
            filesTasks.add(task);

            task.fork();
        }

        var results = new ArrayList<String>();

        for (FileSearchKWord task : filesTasks) {
            if (task.join()) {
                results.add(task.filePath);
            }
        }

        return results;
    }
}
```

```
public class FileSearchKWord extends RecursiveTask<Boolean> {
    public final String filePath;
    private final String[] keyWords;
    private final List<String> wordsList;
    private final int start;
    private final int end;


    1 usage  Anton Chaban
    FileSearchKWord(String filePath, String[] keyWords) {
        this.filePath = filePath;
        this.keyWords = keyWords;

        Scanner scanner;
        try {
            scanner = new Scanner(Paths.get(filePath), StandardCharsets.UTF_8);
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException(e);
        }

        var content = scanner.useDelimiter( pattern: "\\A").next();
        scanner.close();

        wordsList = List.of(content.split( regex: "\\s+"));
        start = 0;
        end = wordsList.size();
    }

```

```
    Anton Chaban
    @Override
    protected Boolean compute() {
        if (end - start < 200_000) {
            return wordsListContainsSearchWord();
        }

        var middleIndex = (end + start) / 2;

        var leftTask = new FileSearchKWord(
                filePath, keyWords, wordsList, start, middleIndex);
        leftTask.fork();

        var rightTask = new FileSearchKWord(
                filePath, keyWords, wordsList, middleIndex, end);

        return leftTask.join() || rightTask.compute();
    }

    1 usage  Anton Chaban
    private boolean wordsListContainsSearchWord() {
        var pattern = Pattern.compile( regex: "\\p{Punct}");
        for (String str : wordsList) {
            String[] words = pattern.split(str.toLowerCase());
            for (String word : words) {
                for (var keyWord : keyWords) {
                    if (word.equals(keyWord.toLowerCase())) {
                        return true;
                    }
                }

            }
        }

        return false;
    }
}
```

Результати:

```
C:\Users\anton\.jdks\corretto-11.0.18\bin\java.exe -javaagent:C:\Users\anton\AppData\Local\JetBrains\Toolbox\apps\IDEA-U
Keywords [world, hello] were found in files:
- C:\Study\3 course\6 sem\MultiThreading\MultiThreading\src\main\java\lab4\texts\Book 1 - The Philosopher's Stone.txt
- C:\Study\3 course\6 sem\MultiThreading\MultiThreading\src\main\java\lab4\texts\Book 7 - The Deathly Hallows.txt
- C:\Study\3 course\6 sem\MultiThreading\MultiThreading\src\main\java\lab4\texts\mk.txt
- C:\Study\3 course\6 sem\MultiThreading\MultiThreading\src\main\java\lab4\texts\pg996.txt
```

```
C:\Users\anton\.jdks\corretto-11.0.18\bin\java.exe -javaagent:C:\Users\anton\AppData\Local\JetBrains\Toolbox\apps\IDEA-U
Keywords [computer] were found in files:
- C:\Study\3 course\6 sem\MultiThreading\MultiThreading\src\main\java\lab4\texts\Book 1 - The Philosopher's Stone.txt
- C:\Study\3 course\6 sem\MultiThreading\MultiThreading\src\main\java\lab4\texts\Book 7 - The Deathly Hallows.txt
- C:\Study\3 course\6 sem\MultiThreading\MultiThreading\src\main\java\lab4\texts\pg996.txt
```