МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО» ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт

З компʼютерного практикуму № 4 з дисципліни

«Технології паралельних обчислень»

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

**Виконав(ла)** *ІП-15 Мєшков Андрій*

(шифр, прізвище, ім'я, по батькові)

**Перевірив** *Дифучина О. Ю.*

(шифр, 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.

# ХІД РОБОТИ

Завдання 1

Лістинг коду:

WordCounter.java

package task1;

import java.io.File;

import java.io.IOException;

import java.util.concurrent.ForkJoinPool;

public class WordCounter {

public static void main(String[] args) throws IOException {

File firstDirectory = *new* File("/Users/andrey/Documents/DOcument\_study/Year 3.2/ТПО/Lab 4/kp4/src/Library");

ForkJoinPool forkJoinPool = *new* ForkJoinPool(8);

CalculateInFolderTask firstTask = *new* CalculateInFolderTask(firstDirectory);

long startTime = System.currentTimeMillis();

CalcResult result = forkJoinPool.invoke(firstTask);

System.out.println(Double.valueOf(result.summaryLength) / Double.valueOf(result.wordsCount));

System.out.print("Time: ");

System.out.println(System.currentTimeMillis() - startTime);

}

}

CalcResult.java

package task1;

public class CalcResult {

long wordsCount;

long summaryLength;

public CalcResult(long wordsCount, long summaryLength) {

*this*.wordsCount = wordsCount;

*this*.summaryLength = summaryLength;

}

}

CalculateInFolderTask.java

package task1;

import java.io.File;

import java.util.LinkedList;

import java.util.List;

import java.util.Objects;

import java.util.concurrent.RecursiveTask;

public class CalculateInFolderTask extends RecursiveTask<CalcResult> {

File dir;

public CalculateInFolderTask(File dir) {

*this*.dir = dir;

}

@Override

protected CalcResult compute() {

*// System.out.println(dir.getPath());*

List<RecursiveTask<CalcResult>> tasks = *new* LinkedList<>();

*for* (File entry *:* Objects.requireNonNull(dir.listFiles())) {

*if* (entry.isDirectory()) {

CalculateInFolderTask task = *new* CalculateInFolderTask(entry);

tasks.add(task);

task.fork();

} *else* {

String fileName = entry.getName();

int dotIndex = fileName.lastIndexOf(".");

String extension = fileName.substring(dotIndex + 1);

*if* (extension.equals("txt")) {

CalculateInDocTask task = *new* CalculateInDocTask(entry);

tasks.add(task);

task.fork();

}

}

}

long wordsCount = 0;

long summary = 0;

*for* (RecursiveTask<CalcResult> task *:* tasks) {

CalcResult result = task.join();

wordsCount += result.wordsCount;

summary += result.summaryLength;

}

*return* *new* CalcResult(wordsCount, summary);

}

}

CalculateInDocTask.java

package task1;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

import java.util.concurrent.RecursiveTask;

public class CalculateInDocTask extends RecursiveTask<CalcResult> {

File dir;

public CalculateInDocTask(File dir) {

*this*.dir = dir;

}

@Override

protected CalcResult compute() {

*try* {

*// System.out.println(dir.getPath());*

int wordCount = 0;

int totalLength = 0;

Scanner scanner = *new* Scanner(dir);

*while* (scanner.hasNext()) {

String word = scanner.next();

wordCount++;

totalLength += word.length();

}

*return* *new* CalcResult(wordCount, totalLength);

} *catch* (FileNotFoundException e) {

*throw* *new* RuntimeException(e);

}

}

}

**Результат:**



Рисунок 1 – Результат запуску програми

Було проведено експеримент зі зміненням кількості файлів та кількості потоків.

|  |  |  |
| --- | --- | --- |
| Розмір бібліотеки | 8 | |
| Середня довжина слова | Час роботи алгоритму |
| 1,1MB | 4,483 | 314 |
| 105,9MB | 6.571 | 4008 |
| 210,8MB | 6,586 | 5935 |
| Розмір бібліотеки | | 16 | |
| Середня довжина слова | Час роботи алгоритму |
| 1,1MB | 4,483 | 290 |
| 105,9MB | 6.571 | 3888 |
| 210,8MB | 6,586 | 5382 |
| Розмір бібліотеки | | 32 | |
| Середня довжина слова | Час роботи алгоритму |
| 1,1MB | 4,483 | 320 |
| 105,9MB | 6.571 | 4264 |
| 210,8MB | 6,586 | 4168 |

Було досліджено, що алгоритм ефективний, проте для пришвидшення роботи при великих даних, потрібно підібрати найкращі параметри, в даному випадку – кількість потоків.

Завдання 2.

Лістинг коду:

Main.java

package task2;

public class Main {

public static void main(String[] args) {

Matrix matrix1 = *new* Matrix(1000);

Matrix matrix2 = *new* Matrix(1000);

int threadsCount = 8;

RowMultiplier rowMultiplier = *new* RowMultiplier(

matrix1, matrix2,

threadsCount

);

long startTime = System.currentTimeMillis();

Matrix rowResult = rowMultiplier.mult();

long rowAlgoEnd = System.currentTimeMillis();

System.out.println();

*// rowResult.printMatrix();*

System.out.println(rowAlgoEnd - startTime);

}

}

Matrix.java

package task2;

public class Matrix {

private int[][] matrix;

private int matrixSize;

static Matrix zeroMatrix(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] = 0;

}

}

*return* *new* Matrix(matrix);

}

public Matrix(int[][] matrix) {

*this*.matrix = matrix;

*this*.matrixSize = matrix.length;

}

public Matrix(int size) {

*this*.matrixSize = size;

int[][] matrix = *new* int[size][size];

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

*for* (int j = 0; j < size; j++) {

matrix[i][j] = 1;

}

}

*this*.matrix = matrix;

}

public int[] getRow(int index) {

*return* matrix[index];

}

public int[][] getMatrix() {

*return* matrix;

}

public int size() {

*return* matrixSize;

}

public void printMatrix() {

*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 Matrix transpose() {

int size = size();

int[][] transposedMatrix = *new* int[size][size];

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

*for* (int j = 0; j < size; j++) {

transposedMatrix[j][i] = matrix[i][j];

}

}

*return* *new* Matrix(transposedMatrix);

}

void add(Matrix addedMatrix) {

int size = size();

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

int[] addedRow = addedMatrix.getRow(i);

*for* (int j = 0; j < size; j++) {

matrix[i][j] += addedRow[j];

}

}

}

boolean isEqual(Matrix matrix) {

int size = size();

int[][] matrixValues = matrix.getMatrix();

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

*for* (int j = 0; j < size; j++) {

*if* (matrixValues[i][j] != *this*.matrix[i][j]) {

*return* false;

}

}

}

*return* true;

}

}

RowMultiplier.java

package task2;

import java.util.concurrent.*\**;

public class RowMultiplier {

Matrix matrix1;

Matrix matrix2;

int threadsCount;

public RowMultiplier(Matrix matrix1, Matrix matrix2, int threadsCount) {

*this*.matrix1 = matrix1;

*this*.matrix2 = matrix2;

*this*.threadsCount = threadsCount;

}

public Matrix mult() {

ForkJoinPool forkJoinPool = *new* ForkJoinPool(threadsCount);

*return* *new* Matrix(forkJoinPool.invoke(*new* RowMultiplyTask(

matrix1.getMatrix(),

matrix2.transpose().getMatrix(),

100, threadsCount

)));

}

}

RowMultiplyTask.java

package task2;

import java.util.concurrent.RecursiveTask;

public class RowMultiplyTask extends RecursiveTask<int[][]> {

int[][] aRows;

int[][] bColumns;

int minSize;

int numThreads;

RowMultiplyTask(int[][] aRows, int[][] bColumns, int minSize, int numThreads) {

*this*.aRows = aRows;

*this*.bColumns = bColumns;

*this*.numThreads = numThreads;

*this*.minSize = minSize;

}

@Override

public int[][] compute() {

*if* (aRows.length <= minSize) {

int[][] result = *new* int[aRows.length][bColumns.length];

*for* (int i = 0; i < aRows.length; i++) {

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

int sum = 0;

*for* (int k = 0; k < aRows[0].length; k++) {

sum += aRows[i][k] \* bColumns[j][k];

}

result[i][j] = sum;

}

}

*return* result;

}

int nearestInt = (int) Math.round(Math.sqrt(numThreads));

int smallestDivisor = 2;

int divisor = 0;

int distance = 0;

*while* (true) {

*if* (nearestInt + distance >= smallestDivisor) {

*if* (aRows.length % (nearestInt + distance) == 0) {

divisor = nearestInt + distance;

*break*;

}

}

*if* (nearestInt - distance >= smallestDivisor) {

*if* (aRows.length % (nearestInt - distance) == 0) {

divisor = nearestInt - distance;

*break*;

}

}

distance++;

}

int blockSize = aRows.length / divisor;

int[][][] manyARows = *new* int[divisor][blockSize][aRows[0].length];

int[][][] manyBColumns = *new* int[divisor][blockSize][aRows[0].length];

*for* (int i = 0; i < divisor; i++) {

*for* (int j = 0; j < blockSize; j++) {

manyARows[i][j] = aRows[i \* blockSize + j];

manyBColumns[i][j] = bColumns[i \* blockSize + j];

}

}

int[][] result = *new* int[aRows.length][bColumns.length];

RowMultiplyTask[][] tasks = *new* RowMultiplyTask[divisor][divisor];

*for*(int i = 0; i < divisor; i++) {

*for* (int j = 0; j < divisor; j++) {

RowMultiplyTask task = *new* RowMultiplyTask(manyARows[i], manyBColumns[j], minSize, numThreads);

task.fork();

tasks[i][j] = task;

}

}

*for*(int i = 0; i < divisor; i++) {

*for* (int j = 0; j < divisor; j++) {

int[][] subMatrix = tasks[i][j].join();

*for* (int ii = 0; ii < subMatrix.length; ii++) {

*for* (int jj = 0; jj < subMatrix.length; jj++) {

result[blockSize \* i + ii][blockSize \* j + jj] = subMatrix[ii][jj];

}

}

}

}

*return* result;

}

}

**Результат:**



Рисунок 2 – Результат запуску програми

Таблиця з порівняннями програми 2 та стрічкового алгоритму з 2го кп

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Розмір | Звичайний, мс | 2 | | | |
| Стрічковий, мс | Прискорення | Стрічковий ForkJoinPool, мс | Прискорення |
| 1000 | 956 | 457 | 2,09190372 | 534 | 1,79026217 |
| 1500 | 3375 | 1573 | 2,14558169 | 1418 | 2,38011283 |
| 2000 | 8566 | 3782 | 2,26493919 | 2967 | 2,88709134 |
| 2500 | 16838 | 6603 | 2,55005301 | 6222 | 2,70620379 |
| 3000 | 28085 | 10423 | 2,69452173 | 9397 | 2,9887198 |
| Розмір | Звичайний, мс | 4 | | | |
| Стрічковий, мс | Прискорення | Стрічковий ForkJoinPool, мс | Прискорення |
| 1000 | 956 | 405 | 2,36049383 | 408 | 2,34313725 |
| 1500 | 3375 | 1234 | 2,7350081 | 1075 | 3,13953488 |
| 2000 | 8566 | 3020 | 2,83642384 | 2324 | 3,6858864 |
| 2500 | 16838 | 5608 | 3,00249643 | 3603 | 4,67332778 |
| 3000 | 28085 | 7961 | 3,52782314 | 5703 | 4,92460109 |
| Розмір | Звичайний, мс | 8 | | | |
| Стрічковий, мс | Прискорення | Стрічковий ForkJoinPool, мс | Прискорення |
| 1000 | 956 | 495 | 1,93131313 | 543 | 1,76058932 |
| 1500 | 3375 | 1562 | 2,16069142 | 1109 | 3,04328224 |
| 2000 | 8566 | 3521 | 2,43283158 | 2367 | 3,61892691 |
| 2500 | 16838 | 6192 | 2,71931525 | 3880 | 4,33969072 |
| 3000 | 28085 | 10203 | 2,75262178 | 6283 | 4,46999841 |

Завдання 3.

Лістинг коду:

IntersectionFinder.java

package task3;

import java.io.File;

import java.io.IOException;

import java.util.Set;

import java.util.concurrent.ForkJoinPool;

public class IntersectionFinder {

public static void main(String[] args) throws IOException {

File firstDirectory = *new* File("/Users/andrey/Documents/DOcument\_study/Year 3.2/ТПО/Lab 4/kp4/src/Library");

ForkJoinPool forkJoinPool = *new* ForkJoinPool(8);

CalculateInFolderTask firstTask = *new* CalculateInFolderTask(firstDirectory);

Set<String> result = forkJoinPool.invoke(firstTask);

System.out.println(result);

}

}

CalculateInFolderTask.java

package task3;

import java.io.File;

import java.util.*\**;

import java.util.concurrent.RecursiveTask;

public class CalculateInFolderTask extends RecursiveTask<Set<String>> {

File dir;

public CalculateInFolderTask(File dir) {

*this*.dir = dir;

}

@Override

protected Set<String> compute() {

*// System.out.println(dir.getPath());*

List<RecursiveTask<Set<String>>> tasks = *new* LinkedList<>();

*for* (File entry *:* Objects.requireNonNull(dir.listFiles())) {

*if* (entry.isDirectory()) {

CalculateInFolderTask task = *new* CalculateInFolderTask(entry);

tasks.add(task);

task.fork();

} *else* {

String fileName = entry.getName();

int dotIndex = fileName.lastIndexOf(".");

String extension = fileName.substring(dotIndex + 1);

*if* (extension.equals("txt")) {

CalculateInDocTask task = *new* CalculateInDocTask(entry);

tasks.add(task);

task.fork();

}

}

}

boolean isFirst = true;

Set<String> result = *new* HashSet<>();;

*for* (RecursiveTask<Set<String>> task *:* tasks) {

*if* (isFirst) {

isFirst = false;

result = task.join();

} *else* {

Set<String> taskResult = task.join();

result.retainAll(taskResult);

}

}

*return* result;

}

}

CalculateInDocTask.java

package task3;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

import java.util.concurrent.RecursiveTask;

public class CalculateInDocTask extends RecursiveTask<Set<String>> {

File dir;

public CalculateInDocTask(File dir) {

*this*.dir = dir;

}

@Override

protected Set<String> compute() {

Set<String> result = *new* HashSet<>();

*try* {

*// System.out.println(dir.getPath());*

Scanner scanner = *new* Scanner(dir);

*while* (scanner.hasNext()) {

result.add(scanner.next().replace(",", "").replace(".", "").toLowerCase());

}

*return* result;

} *catch* (FileNotFoundException e) {

*throw* *new* RuntimeException(e);

}

}

}

**Результат:**

|  |
| --- |
|  |
| Рисунок 3 – Результат запуску програми |

Завдання 4.

Лістинг коду:

WordFinder.java

package task4;

import java.io.File;

import java.io.IOException;

import java.util.*\**;

import java.util.concurrent.ForkJoinPool;

public class WordFinder {

public static void main(String[] args) throws IOException {

File firstDirectory = *new* File("/Users/andrey/Documents/DOcument\_study/Year 3.2/ТПО/Lab 4/kp4/src/Library");

ForkJoinPool forkJoinPool = *new* ForkJoinPool(8);

Set<String> foundedWords = *new* HashSet<String>(Arrays.asList("romeo", "prince", "boy"));

CalculateInFolderTask firstTask = *new* CalculateInFolderTask(firstDirectory, foundedWords);

HashMap<String, Long> result = forkJoinPool.invoke(firstTask);

*for* (Map.Entry<String, Long> entry *:* result.entrySet()) {

String key = entry.getKey();

long value = entry.getValue();

System.out.println(key + " - " + value + "/" + foundedWords.size());

}

}

}

CalculateInFolderTask.java

package task4;

import java.io.File;

import java.util.*\**;

import java.util.concurrent.RecursiveTask;

public class CalculateInFolderTask extends RecursiveTask<HashMap<String, Long>> {

File dir;

Set<String> foundedWords;

public CalculateInFolderTask(File dir, Set<String> foundedWords) {

*this*.dir = dir;

*this*.foundedWords = foundedWords;

}

@Override

protected HashMap<String, Long> compute() {

*// System.out.println(dir.getPath());*

List<RecursiveTask<HashMap<String, Long>>> tasks = *new* LinkedList<>();

*for* (File entry *:* Objects.requireNonNull(dir.listFiles())) {

*if* (entry.isDirectory()) {

CalculateInFolderTask task = *new* CalculateInFolderTask(entry, foundedWords);

tasks.add(task);

task.fork();

} *else* {

String fileName = entry.getName();

int dotIndex = fileName.lastIndexOf(".");

String extension = fileName.substring(dotIndex + 1);

*if* (extension.equals("txt")) {

CalculateInDocTask task = *new* CalculateInDocTask(entry, foundedWords);

tasks.add(task);

task.fork();

}

}

}

HashMap<String, Long> result = *new* HashMap<String, Long>();;

*for* (RecursiveTask<HashMap<String, Long>> task *:* tasks) {

result.putAll(task.join());

}

*return* result;

}

}

CalculateInDocTask.java

package task4;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

import java.util.concurrent.RecursiveTask;

public class CalculateInDocTask extends RecursiveTask<HashMap<String, Long>> {

File dir;

Set<String> foundedWords;

public CalculateInDocTask(File dir, Set<String> foundedWords) {

*this*.dir = dir;

*this*.foundedWords = foundedWords;

}

@Override

protected HashMap<String, Long> compute() {

HashMap<String, Long> result = *new* HashMap<String, Long>();

long count = 0;

Set<String> findedWords = *new* HashSet<>();

*try* {

*// System.out.println(dir.getPath());*

Scanner scanner = *new* Scanner(dir);

*while* (scanner.hasNext()) {

String word = scanner.next().replace(",", "").replace(".", "").toLowerCase();

*if* (foundedWords.contains(word) && !findedWords.contains(word)) {

count++;

findedWords.add(word);

}

}

result.put(dir.getPath(), count);

*return* result;

} *catch* (FileNotFoundException e) {

*throw* *new* RuntimeException(e);

}

}

}

**Результат:**

|  |
| --- |
|  |
| Рисунок 4 – Результат запуску програми |

# ВИСНОВКИ

В результаті роботи над комп’ютерним практикумом було виконано 4 завдання за допомогою ForkJoinPool на мові програмування Java.

Завдання 1: Було реалізовано програму, що розраховує статистичні показники довжини слова у директорії.

Завдання 2: Було реалізовано програму, що виконує паралельне множення матриць стрічковим алгоритмом

Завдання 3: Було реалізовано програму, що знаходить спільні слова у всіх файлів у директорії

Завдання 4: Було реалізовано програму, що розраховує відповідність кожного файлу до масиву слів.