Національний Технічний Університет України

Київський Політехнічний Інститут імені Ігоря Сікорського

Інститут Прикладного Системного Аналізу

Кафедра Системного Проектування

Паралельні обчислення

Лабораторна робота №4

Виконала:

Желєзнова В.С.

Група: ДА-81

Перевірив: Яременко В.С.

Київ – 2021

ЗМІСТ

[Мета роботи 3](#_Toc68637040)

[Завдання 3](#_Toc68637041)

[Лістинг програми мовою Kotlin 4](#_Toc68637042)

[Результати роботи програми 6](#_Toc68637043)

[Опис роботи програми 7](#_Toc68637044)

[Висновки 8](#_Toc68637045)

Мета роботи

Розробка і реалізація паралельного алгоритму з використанням механізмів асинхронних обчислень.

Завдання

Розробити програму, яка за допомогою Java класу CompletableFuture та відповідних методів асинхронних розрахунків (supplyAsync(), thenApplyAsync() та ін.) виконує завдання згідно варіанту. Можливе виконання мовою програмування за власним вибором з використанням відповідних конструкцій асинхронних алгоритмів.

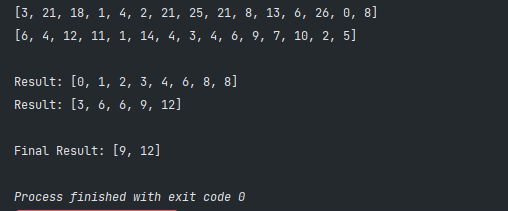
**Варіант 10**

Створити 2 масиви (або колекції) з випадковими числами. У першому масиві – залишити елементи, які вдвічі менші максимального значення масиву (уточнення, погоджене з Яременко В.С.: елементи, які менші за максимальне значення масиву, поділене на 2), в другому – залишити елементи кратні 3. Відсортувати масиви і злити в один відсортований масив тільки ті елементи, які входять в другий масив і не входять в перший.

Лістинг програми мовою Kotlin

import java.util.Collections.sort  
import java.util.concurrent.CompletableFuture  
import java.util.concurrent.ExecutionException  
import kotlin.collections.ArrayList  
import kotlin.random.Random  
  
  
const val *SIZE*: Int = 15  
  
fun main(args: Array<String>) {  
 // Создаю и заполняю 2 массива случайными числами  
 val a1 = IntArray(*SIZE*) **{** Random.nextInt(*SIZE*)**}**.*toList*()  
 val a2 = IntArray(*SIZE*) **{** Random.nextInt(*SIZE*)**}**.*toList*()  
  
 // Вывод массивов  
 *println*(a1)  
 *println*(a2)  
  
 // Первый фьючерс  
 val firstFuture: CompletableFuture<*List*<Int>?> = CompletableFuture  
 .supplyAsync **{** a1 **}** .thenApplyAsync **{** first: *List*<Int>? **->** // Создаю промежуточную переменную, присваиваю ей  
 // массив содержащий элементы, вдвое меньше максимального значения массива  
 val a: *List*<Int>? = first  
 ?.*filter* **{ it** < first.*maxOrNull*()!!/2 **}** // Сортируем  
 sort(a)  
 a  
 **}** // Создание второго фьючерса  
  
 val secondFuture: CompletableFuture<*List*<Int>?> = CompletableFuture  
 .supplyAsync **{** a2 **}** .thenApplyAsync **{** second: *List*<Int>? **->** // Создаю промежуточную переменную, присваиваю ей  
 // массив содержащий элементы, кратные 3-м  
 val a: *List*<Int>? = second?.*filter* **{ it** % 3 == 0 **}** // Сортируем  
 sort(a)  
 a  
 **}**// println()  
// println("Result: " + firstFuture.get())  
// println("Result: " + secondFuture.get())  
  
  
 // Финальный фьючерс, в котором получаем финальный массив  
 val finalResultFuture: CompletableFuture<*List*<Int>> = firstFuture.thenCombine(  
 secondFuture  
 ) **{** first: *List*<Int>?, second: *List*<Int>? **->** // Промежуточная переменная равная значению первого массива  
 val a: *MutableList*<Int> = ArrayList(second!!)  
 // Удаляю элементы со второго массива, которые содержатся в первом  
 first.*let* **{** if (**it** != null) {  
 a.removeAll(**it**)  
 }  
 **}** a  
 **}** // Вывод значений с последнего фьючерса  
 try {  
 *println*("\nFinal Result: " + finalResultFuture.get())  
 } catch (e: InterruptedException) {  
 e.printStackTrace()  
 } catch (e: ExecutionException) {  
 e.printStackTrace()  
 }  
}

Результати роботи програми



Опис роботи програми

В першу чергу, підключаю усі необхідні бібліотеки та задаю розмір масивів константним значенням **SIZE**. Далі відбувається безпосередньо вхід до функції **main()**.

Всередині функції **main()** я створюю та заповнюю 2 масиви випадковими числами в діапазоні **[0, SIZE-1]** та вивожу їх на екран задля того, аби побачити їх наповнення. Переходжу до створення ф’ючерсів.

Створюю перший CompletableFuture (ф’ючерс, який може бути завершений явно), для створення якого використовую метод .supplyAsync - для повернення результату роботи ф’ючерса, та метод .thenApplyAsync - для асинхронного виконання задачі в ф’ючерсі. У першому ф’ючерсі наявна лямбда – функція, в тілі якої я створюю проміжну змінну, в яку привласнюю значення переданого масиву **a1**, в якому через .filter залишаю тільки ті елементи, які менші максимального значення масиву, поділеного на 2. Наступним кроком я сортую масив у проміжній змінній та задаю її результатом ф’ючерса.

Аналогічно створюю другий ф’ючерс, який відрізняється від першого тим, що я проводжу операції над масивом **a2**, та залишаю тільки ті елементи, що кратні трьом.

Наступний крок – створення фінального ф’ючерса, що буде об’єднувати результати роботи перших двох ф’ючерсів. У цьому випадку я використовую .thenCombine для того, щоб дочекатися завершення виконання перших двох ф’ючерсів, щоб почати виконання фінального ф’ючерса. Функція .removeAll видаляє з масиву другого ф’ючерса елементи, що входять в масив першого ф’ючерса.

Вивожу результат виконання останнього ф’ючерса, який отримую через метод .get(), що чекає моменту, коли фінальний ф’ючерс поверне результат.

Висновки

У результаті виконання лабораторної роботи було створено програму, яка реалізує завдання згідно варіанту, за допомогою Java класу CompletableFuture та відповідних методів асинхронних розрахунків:

* .supplyAsync()
* .thenApplyAsync()
* .thenCombine

Ознайомилася з механізмами створення ф’ючерсів та оперування ними.

Під час виконання лабораторної роботи виникла проблема, при реалізації початкового завдання до уточнення, що у більшості випадків перший масив залишався пустий, бо співпадіння, що в масиві є елемент, що вдвічі більше максимального, спостерігалося дуже рідко. Саме тому, я звернулася до Яременко В.С., аби спитати про цю умову, і він уточнив, що малося на увазі саме «елементи, що менші максимального значення поділеного на 2».