

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

## ЛАБОРАТОРНА РОБОТА № 2

з дисципліни «Теорія алгоритмів»

на тему «Метод декомпозиції. Пошук інверсій»

ВИКОНАВ:  
студент 4 курсу  
групи ІП-72з  
Сахнюк Антон Юрійович  
Залікова - 6224

ПЕРЕВІРИВ:  
Доцент кафедри ОТ  
к.т.н., с.н.с.  
Антонюк А.І.

Київ - 2021

## ЗАВДАННЯ

**Тема:** “Метод декомпозиції. Пошук інверсій”

**Мета:** за допомогою методу декомпозиції розробити алгоритм та знайдення числа степенів схожості вподобань (кількість інверсій).

**Завдання:** За допомогою методу декомпозиції розробити алгоритм, який буде розв’язувати наступну задачу.

**Вхідні дані.** Матриця  $D$  натуральних чисел розмірності  $u$  на  $m$ , де  $u$  — це кількість користувачів,  $m$  — кількість фільмів. Кожний елемент  $D[i, j]$  на позицію фільму  $j$  в списку вподобань користувача  $i$ . Іншим вхідним елементом є  $x$  — номер користувача, з яким будуть порівнюватись номери всіх інших користувачів.

**Вихідні дані.** Список з впорядкованих за зростанням другого елементу пар  $(i, c)$ , де  $i$  — номер користувача,  $c$  — число, яке вказує на ступінь схожості

вподобань користувачів  $x$  та  $c$  (кількість інверсій).

## ПРОГРАМНИЙ КОД

```
import java.util.*
import kotlin.Comparator

/**
 * Допоміжна функція для вводу числа з консолі
 * @return Повертає зчитане число
 */
fun readNumber(prompt: String = "Input number: "): Int {
    print(prompt);
    val s = readLine()
    return s?.toInt() ?: throw IllegalStateException("Could not read a number")
}

/**
 * Допоміжна функція для вводу масиву чисел з консолі
 * @return Повертає зчитаний масив чисел
 */
fun readNumberArray(prompt: String = "Input number array (separate with commas): "): IntArray {
    print(prompt)
```

```

    val s = readLine() ?: throw IllegalStateException("Could not read a line")
    return s.split(',').map { it.trim().toInt() }.toIntArray()
}

/**
 * Процедура злиття підмасивів left, right у масив arr
 *
 * @return Повертає кількість інверсій що була обчислена під час злиття
 */
fun merge(arr: IntArray, left: IntArray, right: IntArray): Long {
    var i = 0
    var j = 0
    var inversionCount = 0
    while (i < left.size || j < right.size) {
        if (i == left.size) { // якщо усі числа з лівого масиву уже опрацьовані
            arr[i + j] = right[j]
            j++
        } else if (j == right.size) { // якщо усі числа з правого масиву уже
            опрацьовані
            arr[i + j] = left[i]
            i++
        } else if (left[i] <= right[j]) { // найменший елемент знаходиться у
            лівому підмасиві. Просто додаємо його до результуючого масиву
            arr[i + j] = left[i]
            i++
        } else {
            // Найменший елемент знаходиться у правому підмасиві, в той час як в
            лівому все ще присутні елементи.
            // Це означає що існує ще стільки інверсій, скільки залишилось
            неопрацьованих елементів з лівого масиву
            // Тому ми додаємо це значення до загальної к-сті інверсій
            arr[i + j] = right[j]
            inversionCount += left.size - i
            j++
        }
    }
    return inversionCount.toLong()
}

```

```

/**
 * Процедура підрахунку кількості інверсій у заданому масиві методом сортування
 злиттям і підрахунку інверсій в процесі
 * цього сортування (див. процедуру [merge])
 *
 * @return Повертає кількість інверсій
 */
fun countInversions(arr: IntArray): Long {
    if (arr.size < 2) return 0

    // Розбиття масиву arr на два підмасиви:
    val m = (arr.size + 1) / 2
    val left = Arrays.copyOfRange(arr, 0, m)
    val right = Arrays.copyOfRange(arr, m, arr.size)

    // Результуюча кількість інверсій буде дорівнювати к-сті інверсій лівого
    підмасиву плюс кількість інверсій правого

    // підмасиву і плюс кількість інверсій що виявлена при злитті двох
    підмасивів:
    return countInversions(left) + countInversions(right) + merge(
        arr,
        left,
        right
    )
}

/**
 * Структура що представляє пару результатів обчислення інверсій між юзерами
 [userIndex] та x
 * @param inversions - обчислена кількість інверсій
 * @param userIndex - порядковий номер юзера якого ми порівнювали з юзером x
 */
data class Result(val inversions: Long, val userIndex: Int)

val resultComparator = Comparator<Result> { p0, p1 ->
    p0.inversions.compareTo(p1.inversions) }

fun main(){

```

```

val u = readNumber("Input number of users (u): ")
val m = readNumber("Input number of movies (m): ")
val x = readNumber("Input index of user to compare with others (x): ")
val D = mutableListOf<IntArray>()
for (i in 0 until u){
    D.add(readNumberArray("Input D[$i] array: "))
}
val results = sortedSetOf<Result>(comparator = resultComparator)
for (i in 0 until u){
    if (i == x) continue
    // Обчислимо "масив пріоритетів" на основі масивів D[x] і D[i], в якому
    і будем розраховувати інверсії
    val p = IntArray(m) { 0 }
    for (j in 0 until m){
        p[j] = D[x].indexOf(D[i][j])
    }
    val inversions = countInversions(p)
    results.add(Result(inversions, i))
}
println("\nResults:")
println(results.joinToString(separator = "\n") { "(i = ${it.userIndex}; c =
${it.inversions})" })
}

```

## РЕЗУЛЬТАТИ РОБОТИ ПРОГРАМИ

1. Вхідні дані:

$u = 4$

$m = 5$

$x = 0$

Матриця D:

2,4,3,1,0

0,1,2,3,4

1,2,3,4,0

4,3,1,2,0

Результати:

( $i = 3$ ;  $c = 3$ )

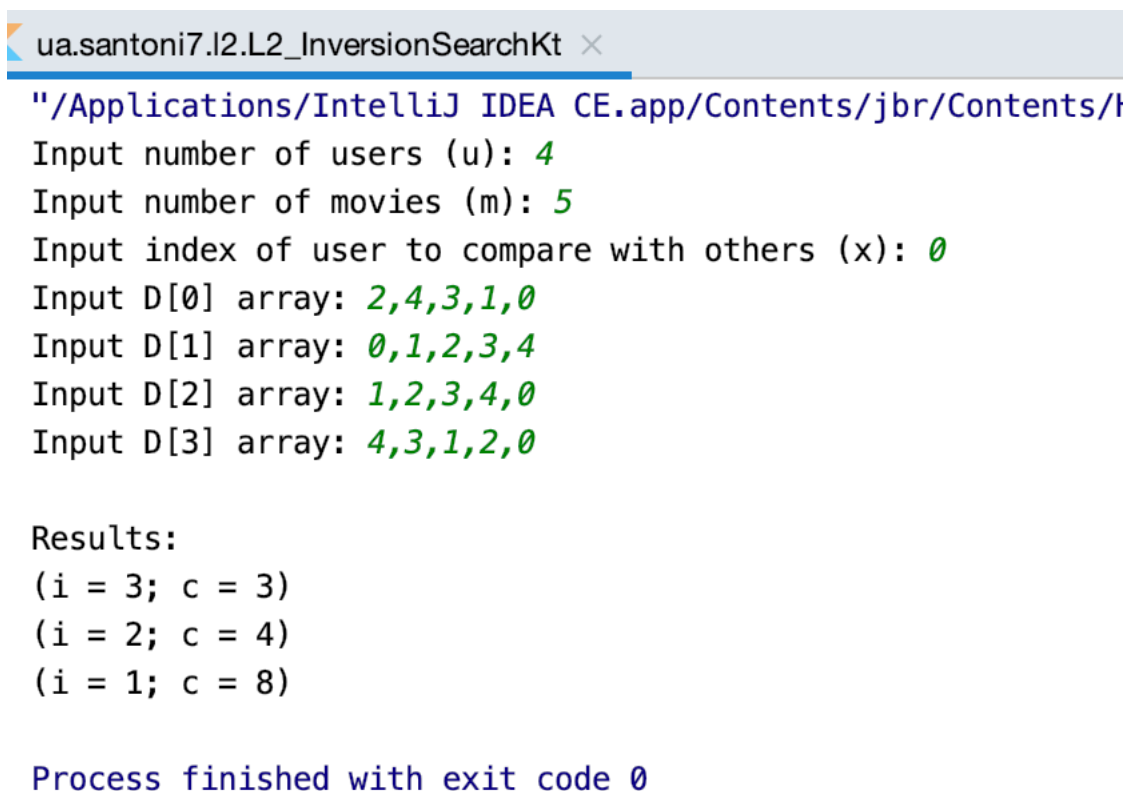
( $i = 2$ ;  $c = 4$ )

( $i = 1$ ;  $c = 8$ )

Інтерпретація результатів:

Даний результат означає, що найбільш схожі в подібання фільмів до користувача 0 у останнього користувача, під номером 3, так як кількість підрахованих інверсій найменша і дорівнює 3. Далі в порядку спадання схожості в подібань ідуть користувачі 2 і 1 з кількістю інверсій 4 та 8 відповідно.

Скріншот виконання програми:



```
ua.santoni7.l2.L2_InversionSearchKt x
"/Applications/IntelliJ IDEA CE.app/Contents/jbr/Contents/I
Input number of users (u): 4
Input number of movies (m): 5
Input index of user to compare with others (x): 0
Input D[0] array: 2,4,3,1,0
Input D[1] array: 0,1,2,3,4
Input D[2] array: 1,2,3,4,0
Input D[3] array: 4,3,1,2,0

Results:
(i = 3; c = 3)
(i = 2; c = 4)
(i = 1; c = 8)

Process finished with exit code 0
```

2. Вхідні дані:

u = 4

m = 6

x = 1

Матриця D:

0,1,2,3,4,5

5,4,3,2,1,0

0,5,1,4,3,2

2,1,0,5,4,3

Результати:

(i = 2; c = 8)

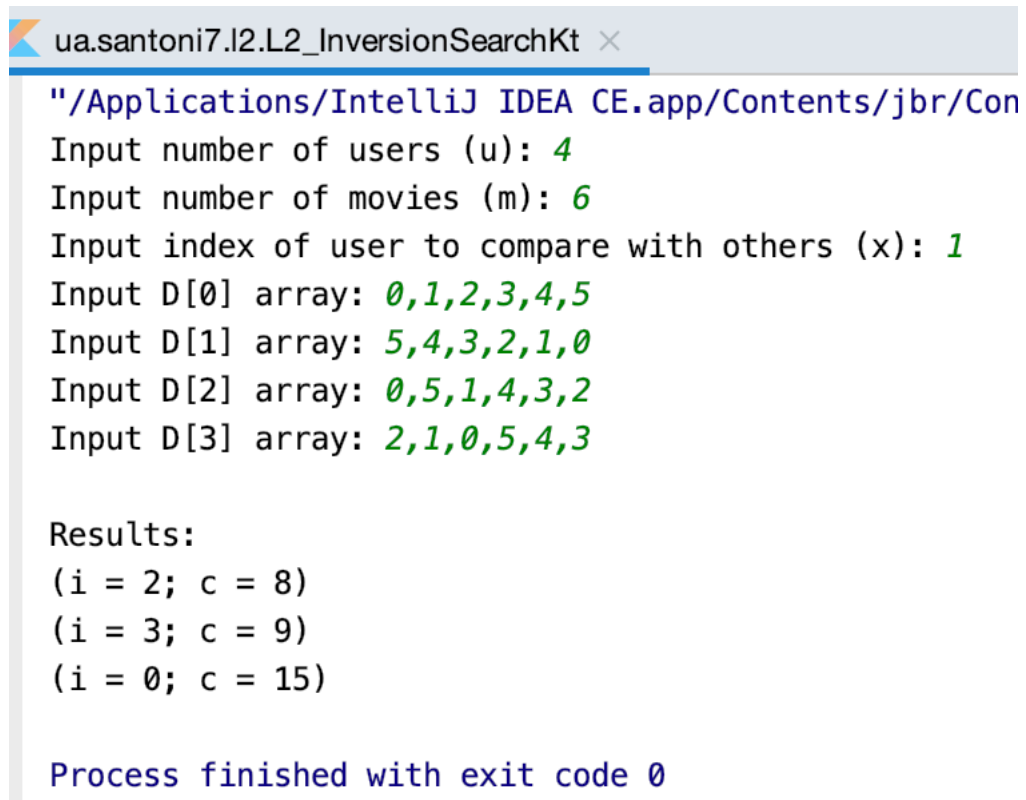
(i = 3; c = 9)

(i = 0; c = 15)

Інтерпретація результатів:

Даний результат означає, що найбільш схожі вподобання фільмів до користувача 1 є у користувача під номером 2, так як кількість підрахованих інверсій найменша і дорівнює 8. Далі в порядку спадання схожості вподобань ідуть користувачі 3 і 0 з кількістю інверсій 9 та 15 відповідно.

Скріншот виконання програми:



```
ua.santoni7.l2.L2_InversionSearchKt ×
"/Applications/IntelliJ IDEA CE.app/Contents/jbr/Con
Input number of users (u): 4
Input number of movies (m): 6
Input index of user to compare with others (x): 1
Input D[0] array: 0,1,2,3,4,5
Input D[1] array: 5,4,3,2,1,0
Input D[2] array: 0,5,1,4,3,2
Input D[3] array: 2,1,0,5,4,3

Results:
(i = 2; c = 8)
(i = 3; c = 9)
(i = 0; c = 15)

Process finished with exit code 0
```

## ВИСНОВКИ

У даній роботі ми ознайомились з побудовою алгоритмів за методом декомпозиції - тобто розділення задачі на підзадачі і потім опрацювання результату виконання підзадач. Прикладом такого алгоритму є сортування злиттям - в його основі лежить принцип “розділяй і властуй”. Для виконання завдання нам потрібно було підрахувати кількість інверсій у масиві для визначення степені схожості вподобань користувачів. Чим менша кількість інверсій - тим більше схожі вподобання. Для підрахунку кількості інверсій ми модифікували алгоритм сортування злиттям аби паралельно з сортуванням підраховувати кількість інверсій. Основна логіка в тому, що при злитті двох підмасивів у момент коли найменший елемент знаходиться у правому підмасиві, це означає що існує стільки інверсій скільки залишилось неопрацьованих елементів лівого підмасиву. Користуючись цим фактом, ми розробили алгоритм що підраховує загальну кількість інверсій масиву, шляхом сумування кількості інверсій в кожному підмасиві а також кількості інверсій виявлених при злитті цих двох підмасивів.

Також, ми перевірили правильність роботи цього алгоритму на невеликому наборі вхідних даних користувачів і отримали результат що відображає ступінь схожості вподобань певного користувача із усіма іншими користувачами. Таким чином, ми змогли відсортувати користувачів у порядку зростання схожості вподобань до обраного користувача.