Національний технічний університет України «КПІ ім. Ігоря Сікорського»

Факультет Інформатики та Обчислювальної Техніки

Кафедра Автоматизованих Систем Обробки Інформації та Управління

Лабораторна робота № 3

з дисципліни «Теорія алгоритмів»

на тему:

**"Метод декомпозиції. Пошук інверсій"**

Виконав:

студент гр. ІС-02

Плостак Ілля

Викладач:

Cт. Вик. Новікова П. А.

Київ – 2021

**1. Формальна постановка задачі**

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

Вхідні дані. Матриця D натуральних чисел розмірності u\*m, де u — ці кількість користувачів, m — кількість фільмів. Кожний елемент матриці D[i, j] вказує на позицію фільму j в списку

вподобань користувача i. Іншим вхідним елементом є x — номер користувача, з яким будуть порівнюватись всі інші користувачі.

Вихідні дані. Список з впорядкованих за зростанням другого елементу пар (i, c), де i — номер користувача, c — число, яке вказує на степінь схожості вподобань користувачів x та c (кількість інверсій).

**2. Програмний код:**

[main.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab3/main.py):

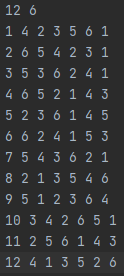
import os  
  
import algorithms as alg  
  
fileName = "input.txt"  
  
if (os.path.exists(fileName) == False):  
 print("Error accessing the file!")  
 exit(1)  
  
fileIn = open(fileName, 'r')  
userNum = 0  
filmsNum = 0  
matrixOfMarks = []  
userList = []  
  
for i, line in enumerate(fileIn):  
 temp = line.split(" ")  
 map\_temp = map(int, temp)  
 temp = list(map\_temp)  
 if i == 0:  
 userNum = temp[0]  
 filmsNum = temp[1]  
 else:  
 userList.append(temp[0])  
 temp.pop(0)  
 matrixOfMarks.append(temp)  
  
fileIn.close()  
  
print("Type a usernumber:")  
userToCompare = int(input())  
if ((userToCompare in userList) == False):  
 print("There is no user with choosed number!")  
 exit(1)  
  
curUserMarks = matrixOfMarks[userToCompare - 1]  
matrixOfMarks.pop(userToCompare - 1)  
userList.pop(userToCompare - 1)  
  
matrixOut = []  
for i in range(len(matrixOfMarks)):  
 temp = alg.sortToCount(curUserMarks, matrixOfMarks[i])  
 countInv = alg.InvCount(temp, 1, len(temp))  
 matrixOut.append([userList[i], countInv])  
  
alg.MergeSortBySecond(matrixOut, 1, len(matrixOut))  
matrixOut.insert(0, [userToCompare])  
matrixOut.append([userToCompare])  
  
fileOut = open("output.txt", 'w')  
for i in range(len(matrixOut)):  
 strOut = ""  
 for j in range(len(matrixOut[i])):  
 strOut += str(matrixOut[i][j])  
 if j != len(matrixOut[i]) - 1:  
 strOut += ' '  
 if i != len(matrixOut) - 1:  
 strOut += '\n'  
 fileOut.write(strOut)  
  
fileOut.close()

[algorithms.py](https://github.com/feedblackg44/kpilabs2/blob/master/TA/Lab3/algorithms.py):

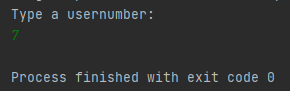
import math  
  
def Merge(A, p, q, r): # Принимаем параметры: сортируемый массив, точка начала, середина, точка конца  
 n1 = q - p + 1 # Длина левой части  
 n2 = r - q # Длина правой части  
 L = [] # Создаём пустую левую часть  
 R = [] # Создаём пустую правую часть  
 for i in range(0, n1): # Заполняем левую часть левой частью сортируемого массива  
 L.append(A[p + i - 1]) #  
 for j in range(0, n2): # Заполняем правую часть правой частью сортируемого массива  
 R.append(A[q + j]) #  
 L.append([math.inf, math.inf]) # Добавляем ещё один элемент что бы не выйти за границы массива при объединении массивов в один  
 R.append([math.inf, math.inf]) # Добавляем ещё один элемент что бы не выйти за границы массива при объединении массивов в один  
 i = 0 # Инициализируем итератор для левой части  
 j = 0 # Инициализируем итератор для правой части  
 for k in range(p - 1, r): # Объединяем два массива в один, сортируем  
 if L[i][1] <= R[j][1]: # Если текущий элемент левой части меньше или равен текущему элементу правой  
 A[k] = L[i] # Вставляем текущий элемент левой части  
 i += 1 # Увеличиваем итератор левой части  
 else: # Иначе если текущий элемент левой части больше текущего элемента правой  
 A[k] = R[j] # Вставляем текущий элемент правой части  
 j += 1 # Увеличиваем итератор правой части  
  
def MergeSortBySecond(A, p, r): # Принимаем параметры: сортируемый массив, точка начала, точка конца  
 if p < r: # Если точка начала меньше точки конца  
 q = math.floor((p + r) / 2) # Вычисляем середину массива округлённую вниз  
 MergeSortBySecond(A, p, q) # Рекурсивно вызываем эту функцию для левой части сортируемого массива  
 MergeSortBySecond(A, q + 1, r) # Рекурсивно вызываем эту функцию для правой части сортируемого массива  
 Merge(A, p, q, r) # Вызываем процедуру слияния  
  
def MergeInvCount(A, p, q, r): # Принимаем параметры: сортируемый массив, точка начала, середина, точка конца  
 n1 = q - p + 1 # Длина левой части  
 n2 = r - q # Длина правой части  
 L = [math.inf for i in range(n1 + 1)] # Создаём пустую левую часть  
 R = [math.inf for i in range(n2 + 1)] # Создаём пустую правую часть  
 for i in range(0, n1): # Заполняем левую часть левой частью сортируемого массива  
 L[i] = A[p + i - 1] #  
 for j in range(0, n2): # Заполняем правую часть правой частью сортируемого массива  
 R[j] = A[q + j] #  
 i = 0 # Добавляем ещё один элемент что бы не выйти за границы массива при объединении массивов в один  
 j = 0 # Добавляем ещё один элемент что бы не выйти за границы массива при объединении массивов в один  
 counter = 0 # Инициализируем счётчик инверсий  
 for k in range(p - 1, r): # Объединяем два массива в один, сортируем   
 if L[i] <= R[j]: # Если текущий элемент левой части меньше или равен текущему элементу правой  
 A[k] = L[i] # Вставляем текущий элемент левой части   
 i += 1 # Увеличиваем итератор левой части  
 else: # Иначе если текущий элемент левой части больше текущего элемента правой   
 A[k] = R[j] # Вставляем текущий элемент правой части   
 j += 1 # Увеличиваем итератор правой части   
 counter += n1 - i # Количество инверсий на каждом этапе merge равно оставшемуся количеству элементов в левой части  
 return counter # Возвращаем количество итераций  
  
def InvCount(A, p, r): # Принимаем параметры: сортируемый массив, точка начала, точка конца  
 if p < r: # Если точка начала меньше точки конца  
 q = math.floor((p + r) / 2) # Вычисляем середину массива округлённую вниз  
 x = InvCount(A, p, q) # Рекурсивно вызываем эту функцию для левой части сортируемого массива, получаем количество левых итераций  
 y = InvCount(A, q + 1, r) # Рекурсивно вызываем эту функцию для правой части сортируемого массива, получаем количество правых итераций  
 z = MergeInvCount(A, p, q, r) # Вызываем процедуру слияния, получаем количество разделённых итераций  
 return x + y + z # Возвращаем общее количество итераций  
 else: # Иначе если точка начала больше или равна точке конца  
 return 0 # Возвращаем 0  
  
def sortToCount(arr1, arr2): # Принимаем параметры: массив индексов для результирующего массива, массив значений для перестановки  
 temp = arr1.copy() # Копируем исходный массив индексов в новый временный  
 for i in range (len(arr1)): # Проходимся по всему массиву индексов  
 temp[arr1[i] - 1] = arr2[i] # Во временный массив на элемент с индексом соответствующим текущему элементу массива индексов ставим текущий элемент массива значений  
 return temp # Возвращаем полученный массив

**3. Приклад вхідного файлу, результат запуску програми та приклад вихідного файлу:**

Приклад вхідного файлу:



Результат запуску програми:



Приклад вихідного файлу:



**4. Висновок:**

В цій лабораторній роботі ми вивчили алгоритм сортування злиттям (merge sort). Цей алгоритм має складність θ(n\*log(n)) та використовує додаткову пам’ять розміром n. Ми навчились використовувати цей алгоритм сортування на практиці, а саме для підрахунку кількості інверсій та сортування двовимірного масиву за другим елементом кожного підмасиву. Вивід та ввід у програмі здійснюється через файли.

Правильність сортування за другим елементом можна побачити наочно нічого не рахуючи. Але правильність підрахунку інверсій потрібно довести. Доведемо:

Візьмемо в нашому прикладі сьомий рядок та порахуємо кількість інверсій відносно першого рядка:

7й рядок – 5 4 3 6 2 1

1й рядок – 4 2 3 5 6 1

Програма порахувавши отримала кількість інверсій: 5.

Перевіримо:

Формуємо масив відмінностей:

1. B[5] = 4;
2. B[4] = 2;
3. B[3] = 3;
4. B[6] = 5;
5. B[2] = 6;
6. B[1] = 1;

B = 1 6 3 2 4 5 – масив відмінностей.

Підрахуємо кількість інверсій:

(2, 3), (2, 4), (2, 5), (2, 6), (3, 4) – 5 інверсій.

Результати співпали, тобто програма працює коректно.