

Міністерство освіти і науки України

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря
Сікорського»**

Факультет інформатики та обчислювальної техніки

Кафедра ІСТ

Звіт

з лабораторної роботи № 2 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

„Метод декомпозиції. Пошук інверсій”

Виконали

ІА-31 Клим'юк В.Л, Самелюк А.С, Дук М.Д, Сақун Д.С

Перевірів

Степанов А.С

Київ 2024

Практичне завдання №2

“Метод декомпозиції. Пошук інверсій”

Мета лабораторної роботи: Вивчити метод декомпозиції для пошуку інверсій у послідовностях даних. Розібрати алгоритм дії методу та вивчити його застосування для ефективного виявлення та аналізу інверсій у різноманітних наборах даних.

Завдання

Існує веб сервіс, який надає своїм користувачам можливість перегляду фільмів онлайн. Періодично система надає нові рекомендації користувачам — які фільми, що їх користувач ще не дивився, можливо будуть йому або їй цікаві.

В основі рекомендаційного алгоритму лежить ідея, що користувачі, які подивились однакові фільми та також оцінили їх схожим чином, мають схожі смаки. Наприклад, нехай є два користувача: Аліса та Богдан. Обидва вони переглянули наступні фільми: “Зоряні війни”, “Гравітація”, “Пірати карибського моря”, “Володар перснів”, “Матриця”.

Спочатку система просить користувачів оцінити ці фільми і розташувати їх у порядку вподобання, іншими словами — створити власний хіт-парад. Так Аліса розташувала вказані фільми у порядку від найбільш до найменш вподобаного: “Пірати карибського моря”, “Володар перснів”, “Матриця”, “Гравітація”, “Зоряні війни”. Хіт-парад Богдана: “Зоряні війни”, “Володар перснів”, “Гравітація”, “Матриця”, “Пірати карибського моря”.

Після цього система може надати кількісну оцінку наскільки схожими є смаки двох користувачів. Для цього використовується алгоритм підрахунку інверсій поміж двома масивами.

Нехай $A[1..n]$ — масив з n чисел. Якщо $i < j$ та $A[i] > A[j]$, то пара (i, j) — інверсія в A .

Щоб звести задачу порівняння двох хіт-парадів до задачі підрахунку інверсій у нашому прикладі, побудуємо два масиви A та B . Масив $A = [1, 2, 3, 4, 5]$. Масив B будується наступним чином: елементом $B[j]$ є число, яке відповідає позиції фільму в хіт-параді Богдана, який в хіт-параді Аліси посідав місце j . Наприклад, $j = 1$ у хіт-параді Аліси відповідає фільму “Пірати карибського моря”. Цей фільм в списку Богдана стоїть на позиції 5, тому $B[1] = 5$. Загалом отримуємо масив $B = [5, 2, 4, 3, 1]$.

Масив $B = [5, 2, 4, 3, 1]$ має наступні інверсії (вказуються індекси елементів, а не їх значення): $(1,2)$, $(1,3)$, $(1,4)$, $(1,5)$, $(2,5)$, $(3,4)$, $(3,5)$, $(4,5)$. Загалом 8 інверсій. І це число вказує наскільки сильно відрізняється список вподобань Аліси від списку вподобань Богдана. Ми порахували віддаленість списку Аліси від списку Богдана. Якщо порахувати цю відстань в іншому напрямку, то чи буде вона такою самою? Тобто визначити кількість інверсій в списку Аліси по відношенню до списку Богдана.

Сервіс перегляду фільмів онлайн має базу даних D вподобань користувачів. Ця база є матрицею.

Рядки цієї матриці відповідають користувачам, а стовпці — фільмам. Її розмірність $u \times m$, де u — це кількість користувачів, m — кількість фільмів. Кожний елемент матриці $D[i, j]$ вказує на позицію фільму j в списку вподобань користувача i . Для спрощення

припускаємо, що всі користувачі переглянули всі фільми.

Тепер щоб визначити наскільки подібні смаки деякого користувача x до смаків інших користувачів, система попарно порівнює списки вподобань x та всіх інших користувачів i не дорівнює x : за вказаним вище принципом підраховується кількість інверсій у масиві $D[x]$ відносно масиву $D[i]$.

Визначене число інверсій буде кількісною оцінкою наскільки смаки x є близькими до смаків кожного i — чим менше значення цього числа, тим більш подібними є смаки двох користувачів.

Формальна постановка задачі

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

Вхідні дані. Матриця D натуральних чисел розмірності $u \times m$, 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).

Формат вхідних/вихідних даних

Розроблена програма повинна зчитувати вхідні дані з файлу заданого формату та записувати дані у файл заданого формату. У вхідному файлі зберігається матриця вподобань всіх користувачів D .

Номер користувача X , з яким відбувається порівняння всіх інших користувачів, передається аргументом виклику програми через командний рядок.

Вхідний файл представляє собою текстовий файл із $U+1$ рядків. Перший рядок містить два числа: U та M , де U — кількість користувачів, M — кількість фільмів. Кожен наступний рядок представляє список вподобань (хіт-парад) фільмів відповідних користувачів і містить $M+1$ число, розділених пробілом. Перше число в рядку є номером користувача (від 1 до U). Решта M чисел є номерами фільмів 1, ..., M у хіт-параді відповідного користувача.

Вихідний файл представляє також текстовий файл із U рядків. Перший рядок містить одне число — номер користувача X , з яким відбувалось порівняння всіх інших користувачів. Далі йде $U-1$ рядків, кожен з яких містить два числа через пробіл: номер користувача i та число c , яке визначає степінь подібності списків вподобань користувачів x та i . Рядки з парами i та c впорядковані за значенням елементу c .

До документу завдання також додаються приклади вхідних і вихідних файлів різної розмірності.

Нижче наведені приклади вхідного та вихідного файлу для $U = 10$ та $M = 5$ і користувача $X = 6$.

Вхідний файл	Вихідний файл
10 5	6
1 5 2 1 3 4	3 3
2 3 2 4 1 5	2 4
3 4 5 3 2 1	7 4
4 5 1 4 3 2	9 4
5 1 2 5 4 3	5 5
6 2 5 4 1 3	1 7
7 2 4 5 3 1	4 7
8 5 3 1 4 2	8 7
9 4 5 2 3 1	10 8
10 3 1 2 4 5	6

Вимоги до програмного забезпечення

- Розробляти програму можна на одній з наступних мов програмування: C/C++ (версія C++11), C# (версія C# 5.0), Java (версія Java SE 8), Python .
- Програма повинна розміщуватись в окремому вихідному файлі, без використання додаткових нестандартних зовнішніх модулів.
- Не дозволяється використовувати будь-які нестандартні бібліотеки та розширення. Програма не повинна залежати від операційної системи.
- Не реалізуйте жодного інтерфейсу користувача (окрім командного рядку). Програма не повинна запитувати через пристрій вводу в користувача жодної додаткової інформації. Вашу програму будуть використовувати виключно у вигляді “чорного ящика”.
- Назва вихідного файлу вашої програми повинна задовольняти наступному формату:

НомерГрупи_ПрізвищеСтудента_НомерЗавдання.Розширення, де НомерГрупи — це один з рядків is01, is02, is03; ПрізвищеСтудента — прізвище студента записане латинськими літерами; НомерЗавдання — двозначний номер завдання (01, 02, ...);

Розширення — розширення файлу, відповідно до мови програмування (.c, .cpp, .cs, .java, .py). Приклад назви вихідного файлу: is31_ivanenko_01.cs.

- Розроблена програма повинна зчитувати з командного рядку назву вхідного файлу та записувати результат у вихідний файл. При запуску першим і єдиним аргументом командного рядку повинна бути назва вхідного файлу (наприклад, input_10.txt). Назва вихідного файлу повинна складатись із назви файлу самої програми разом із суфіксом “_output” і мати розширення .txt. Приклад назви вихідного файлу:

is01_ivanenko_01_output.txt.

```

namespace Lab2Classes
{
    public class Algorithm
    {
        //Calculate Rating for user
        public int CalculateRating(User comparable, User compared)
        {
            int[] inverseArray = new int[comparable.Films.Count];

            FillArray(comparable, compared, inverseArray);

            return CountInversion2(inverseArray);
            //return CountInversion(inverseArray);
        }

        public int CalculateRating(User comparable, string[] userParams)
        {
            int[] inverseArray = new int[comparable.Films.Count];

            FillArray(comparable, userParams, inverseArray);

            return CountInversion2(inverseArray);
            //return CountInversion(inverseArray);
        }

        //Fill inversion array
        private void FillArray(User comparable, User compared, int[] arr)
        {
            //Setup array
            for (int j = 0; j < comparable.Films.Count; j++)
            {
                arr[comparable.Films[j] - 1] = compared.Films[j];
            }
        }

        private void FillArray(User comparable, string[] userParams, int[] arr)
        {
            //Setup array
            for (int j = 0; j < comparable.Films.Count; j++)
            {
                arr[comparable.Films[j] - 1] = int.Parse(userParams[j+1]);
            }
        }

        //Count inversion O(n^2)
        private int CountInversion(int[] arr)
        {
            int res = 0;

            //Count inversion O(n^2)
            for (int j = 0; j < arr.Length - 1; j++)
            {
                for (int k = j + 1; k < arr.Length; k++)
                {
                    if (arr[j] > arr[k])

```

```

        {
            res++;
        }
    }
}
return res;
}

//Count inversion O(n*logn)
private int CountInversion2(int[] arr)
{
    int[] buf = new int[arr.Length];
    return MergeInversion(arr, buf, 0, arr.Length - 1);
}

private int MergeInversion(int[] arr, int[] buf, int left, int right)
{
    int res = 0;

    if (right > left)
    {
        int m = (left + right) / 2;

        res += MergeInversion(arr, buf, left, m);
        res += MergeInversion(arr, buf, m + 1, right);
        res += MergeInversionSort(arr, buf, left, m + 1, right);
    }

    return res;
}

private int MergeInversionSort(int[] arr, int[] buf, int left, int mid, int
right)
{
    int res = 0;

    int i = left;
    int j = mid;
    int cur = left;

    while ((i <= mid - 1) && (j <= right))
    {
        if (arr[i] <= arr[j])
        {
            buf[cur++] = arr[i++];
        }
        else
        {
            buf[cur++] = arr[j++];
            res = res + mid - i;
        }
    }

    while(i <= mid - 1)
    {

```

```

        buf[cur++] = arr[i++];
    }

    while(j <= right)
    {
        buf[cur++] = arr[j++];
    }

    for(int k = left; k <= right; k++)
    {
        arr[k] = buf[k];
    }

    return res;
}
}
}

```

Код 1.1 – Algorithm

```

using Lab2Classes;

namespace Lab2
{
    public class FileHelper
    {
        //Read results from file
        public async Task<Dictionary<int, List<UserResult>>> ReadResults(string file,
int userId)
        {
            if (userId < 1)
            {
                throw new ArgumentException("userId must be >= 1");
            }

            if (!File.Exists(file))
            {
                throw new FileNotFoundException("File does not exist!");
            }

            //Open file
            using var fileStream = File.OpenRead(file);
            using var reader = new StreamReader(fileStream);

            //Read data parameters
            string? parameters = await reader.ReadLineAsync();

            if (string.IsNullOrEmpty(parameters))
            {
                throw new InvalidDataException("Invalid file data parameters");
            }

            //Split data parameters
            string[] strs = parameters.Split(' ');
            int usersAmount = 0;
            int filmsAmount = 1;

```

```

switch (strs.Length)
{
    case 2:
    {
        usersAmount = int.Parse(strs[0]);
        filmsAmount = int.Parse(strs[1]);
        break;
    }
    case 1:
    {
        usersAmount = int.Parse(strs[0]);
        break;
    }
    default:
        throw new InvalidDataException("Invalid amount of file data
parameters");
}

if (userId > usersAmount)
{
    throw new ArgumentException("userId exceeds total amount of users
present in file");
}

//Results hashTable
Dictionary<int, List<UserResult>> result = new Dictionary<int,
List<UserResult>>();
List<User> usersBuffer = new List<User>();
Algorithm alg = new Algorithm();
User? comparedUser = null;

//Reading users data and filling results
for (int i = 1; i <= usersAmount; i++)
{
    //Reading line
    string? userData = await reader.ReadLineAsync();

    if (string.IsNullOrEmpty(userData))
    {
        throw new InvalidDataException("Invalid user data");
    }

    //Splitting parameters
    string[] userParams = userData.Split(" ");

    if (userParams.Length != filmsAmount + 1)
    {
        throw new InvalidDataException("Invalid user data");
    }

    int id = int.Parse(userParams[0]);

    //Decide what to do with user

```



```

        //Found compared user
        if (id == userId)
        {
            comparedUser = CreateUser(id, userParams);

            //If buffer is not empty, fill with results
            if (usersBuffer.Count != 0)
            {
                foreach(var u in usersBuffer)
                {
                    UserResult newRes = new UserResult()
                    {
                        Id = u.Id,
                        Rating = alg.CalculateRating(comparedUser, u)
                    };

                    AddResult(result, newRes);
                }
            }
        }
        else
        {
            //Add new user to buffer because compared user isn't found yet
            if (comparedUser == null)
            {
                User newUser = CreateUser(id, userParams);
                usersBuffer.Add(newUser);
            }
            //Compared user found, calculate inversion and add into dictionary
            else
            {
                UserResult newRes = new UserResult()
                {
                    Id = id,
                    Rating = alg.CalculateRating(comparedUser, userParams)
                };
                AddResult(result, newRes);
            }
        }
    }

    return result;
}

//Add result into dictionary
private void AddResult(Dictionary<int, List<UserResult>> dic, UserResult res)
{
    if (!dic.ContainsKey(res.Rating))
    {
        dic.Add(res.Rating, new List<UserResult>() { res });
    }
    else
    {
        dic[res.Rating].Add(res);
    }
}

```

```

    }

    //Create user object
    private User CreateUser(int id, string[] userParams)
    {
        User user = new User()
        {
            Id = id
        };

        for(int i = 1; i < userParams.Length; i++)
        {
            user.Films.Add(int.Parse(userParams[i]));
        }

        return user;
    }

    public async Task WriteUserResultsInFile(string name, Dictionary<int,
List<UserResult>> results, int id)
    {
        //Open file
        using FileStream fileStream = File.Create(name);
        using StreamWriter writer = new StreamWriter(fileStream);

        //Write compared user id
        await writer.WriteLineAsync(id.ToString());

        int[] keys = results.Keys.ToArray();

        Array.Sort(keys);

        //Write results
        foreach(var key in keys)
        {
            foreach(var res in results[key])
            {
                await writer.WriteLineAsync(res.Id + " " + res.Rating);
            }
        }
    }
}

```

Код 1.2 – FileHelper

```

namespace Lab2Classes
{
    public class User
    {
        public int Id { get; set; }
        public List<int> Films { get; set; } = new List<int>();
    }
}

```

Код 1.3 – User

```
namespace Lab2Classes
{
    public class UserResult
    {
        public int Id { get; set; }
        public int Rating { get; set; }
    }
}
```

Код 1.4 – UserResult

```
using Lab2;
using System.Diagnostics;
using Xunit.Abstractions;

namespace Lab2Tests
{
    public class Tests10_5
    {
        private readonly ITestOutputHelper output;

        public Tests10_5(ITestOutputHelper output)
        {
            this.output = output;
        }

        [Theory]
        [InlineData("task_02_data_examples\\input_10_5.txt", "results\\IA-31_Brigade1_01_10-5-10.txt", 10)]
        [InlineData("task_02_data_examples\\input_10_5.txt", "results\\IA-31_Brigade1_01_10-5-6.txt", 6)]
        [InlineData("task_02_data_examples\\input_10_5.txt", "results\\IA-31_Brigade1_01_10-5-9.txt", 9)]
        public async Task Test(string inputFile, string outputFile, int id)
        {
            FileHelper helper = new FileHelper();
            Stopwatch sw = Stopwatch.StartNew();

            var results = await helper.ReadResults(inputFile, id);
            await helper.WriteUserResultsInFile(outputFile, results, id);
            sw.Stop();

            output.WriteLine("Elapsed: {0}", sw.Elapsed);
        }
    }
}
```

Код 1.5 – Приклад тесту

Test	Duration
✓ Lab2Tests (30)	538 ms
✓ Lab2Tests (30)	538 ms
✓ Tests10_5 (3)	29 ms
✓ Tests1000_100 (10)	271 ms
✓ Tests1000_5 (3)	41 ms
✓ Tests2_1000 (2)	29 ms
✓ Tests2_10000 (2)	23 ms
✓ Tests22_7 (2)	28 ms
✓ Tests240_5 (2)	20 ms
✓ Tests5_10 (2)	29 ms
✓ Tests5_5 (2)	29 ms
✓ Tests720_6 (2)	39 ms

Рисунок 1.1 – Час виконання тестів(30)

IA-31_Brigade1_01_10-5-10...	IA-31_Brigade1_01_10-5-9...
Файл Редагування Формат Вигляд Довідка	Файл Редагування Формат Вигляд Довідка
10	9
1 3	3 1
2 4	8 3
4 5	6 4
5 5	7 4
8 5	1 5
6 8	4 5
7 8	5 7
9 8	2 8
3 9	10 8
Рд 1 100% Windows (CRLF) UTF-8	Рд 1 100% Windows (CRLF) UTF-8

Рисунок 1.2(3) – Приклади вихідних файлів

Висновок: Проведена лабораторна робота підтвердила ефективність методу декомпозиції для пошуку інверсій у послідовностях даних. Використання цього методу дозволяє швидко та точно виявляти зміни у порядку елементів послідовності, що є корисним у багатьох областях, таких як геноміка, обробка сигналів та аналіз фінансових даних. Потенційні застосування методу декомпозиції включають покращення алгоритмів сортування, оптимізацію обробки даних та побудову прогностичних моделей.

