

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

Кафедра ІІІ(ІСТ)

Звіт

з лабораторної роботи № 6 з дисципліни
«Теорія алгоритмів»

„Піраміди ”

Виконали

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

Перевірив

Степанов А.С

Київ 2024

Завдання

В даній роботі необхідно розв'язати наступну задачу визначення послідовності медіан для заданого вхідного масиву. Нагадаємо, що медіаною для масиву називається елемент, який займає середнє положення у відсортованому масиві. Так, якщо кількість елементів у масиві непарна, то медіана одна та індекс її у відсортованому масиві визначається як $\lfloor n/2 \rfloor$ (де n — розмір вхідного масиву).

Якщо кількість елементів у масиві парна, то медіан буде дві та їх індекси визначаються за формулами $\lfloor n/2 \rfloor$ та $\lfloor n/2 \rfloor + 1$.

Задача формулюється наступним чином. Нехай заданий вхідний масив $A = [x_1, \dots, x_N]$. Припустимо, що елементи масиву поступають на вхід програми послідовно: в кожний момент часу розглядається новий елемент x_i . Необхідно для кожного i (від 1 до N) визначити медіану підмасиву $A' = [x_1, \dots, x_i]$, тобто медіану для масиву елементів, які були отримані програмою на даний момент часу. Необхідно розв'язати цю задачу, використовуючі структури даних пірамід і так, щоб кожна медіана визначалась за час $O(\log(i))$.

Цю задачу можна розв'язати, використовуючи дві піраміди (heap) наступним чином.

- Позначимо через H_{low} незростаючу піраміду (max-heap), яка буде містити елементи меншої половини масиву (тобто такі елементи, які у відсортованому поточному елементі A' будуть розташовуватись у першій, меншій половині масиву).
- Позначимо через H_{high} неспадну піраміду (min-heap), яка буде містити елементи більшої половини масиву (тобто такі елементи, які у відсортованому поточному елементі A' будуть розташовуватись у другій, більшій половині масиву).

Тепер розглянемо роботу процедури, яка розв'язує поставлену задачу із використанням двох наведених пірамід. Нехай додається черговий елемент x_i . На поточний момент сумарна кількість елементів, які зберігаються в обох пірамідах, становить $(i-1)$. Наступні кроки, які ми повинні виконати:

1. Визначимо, в яку піраміду (H_{low} або H_{high}) потрібно додати новий елемент. Якщо x_i менше ніж найбільший елемент з H_{low} (тобто новий елемент буде

розташовуватись в меншій поточній половині), то додаємо його у цю піраміду. В іншому випадку додаємо елемент в піраміду N_{high} .

2. В кожний момент часу, тобто на кожній ітерації роботи алгоритму, повинен зберігатись наступний інваріант: кількість елементів в піраміді N_{low} не повинна відрізнятись від кількості елементів в N_{high} не більше ніж на одиницю. Під час виконання попереднього етапу цей інваріант може порушитись. Тому тепер необхідно відновити даний інваріант: якщо у піраміді N_{low} елементів більше на 2 за N_{high} , то визначаємо найбільший елемент з N_{low} і вставляємо його у N_{high} ; якщо кількість елементів у N_{high} більше на 2 за N_{low} , то визначаємо найменший елемент з N_{high} і вставляємо його у N_{low} . Зрозуміло, що після кожної вставки нового елементу в піраміду необхідно перевіряти властивість піраміди: для N_{low} - властивість незростаючої піраміди, для N_{high} - властивість неспадної піраміди.

3. Визначити медіану для поточного масиву $A' = [x_1, \dots, x_i]$:

- Якщо кількість елементів у A' парна, то після збереження інваріанту у пункті 2, кількість елементів у пірамідах N_{low} та N_{high} буде рівною. Тому одна медіана буде найбільшим елементом N_{low} , а інша медіана — найменшим елементом N_{high} .
- Якщо кількість елементів у A' непарна, то єдина медіана буде знаходитись у тій піраміді, в якій кількість елементів буде більше (на одиницю) за кількість в іншій. Тому, якщо кількість елементів у N_{low} більше за N_{high} , то медіана — це найбільший елемент з N_{low} . Інакше медіана — найменший елемент з N_{high} .

Наведений алгоритм використовує процедури `extract_max` незростаючої піраміди N_{low} та

`extract_min` неспадної піраміди N_{high} , які виконуються за час $O(\log N)$, де N — розмір піраміди. Тому на кожній ітерації № i для поточного масиву $A' = [x_1, \dots, x_i]$ час роботи наведеної процедури становитиме $O(\log(i))$.

Формат вхідних/вихідних даних

Розроблена програма повинна зчитувати вхідні дані з файлу заданого формату та записувати дані у файл заданого формату. Вхідний файл представляє собою текстовий файл із $N+1$ рядків, де N — це розмірність вхідного масиву A .

Першим записом є число — кількість елементів в масиві; наступні N записів містять елементи вхідного масиву.

Вихідний файл представляє також текстовий файл із N рядків, де кожен рядок i містить медіани для вхідного підмасиву $[x_1, \dots, x_i]$. Якщо медіана одна, то в рядку буде одне число; якщо медіани дві, то вони записується через пробіл.

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

Нижче наведені приклади вхідного та вихідного файлу для $N = 10$.

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

```

namespace Classes
{
    public class BinaryTree
    {
        public static async Task<BinaryTree> CreateFromFile(string file)
        {
            using FileStream stream = File.OpenRead(file);
            using StreamReader reader = new StreamReader(stream);

            string line = await reader.ReadToEndAsync();

            List<int> items = new List<int>();

            string[] strs = line.Split(new char[] { ' ', '\n', '\t' });

            foreach (var s in strs)
            {
                if (s.Length > 0)
                    items.Add(int.Parse(s));
            }

            return new BinaryTree(items);
        }

        private TreeNode _Root { get; set; }

        public BinaryTree(List<int> items)
        {
            _Root = new TreeNode(items[0]);

            int cur = 1;

            _FillTree(ref cur, items, _Root);
        }

        private void _FillTree(ref int current, List<int> items, TreeNode node)
        {
            if (current >= items.Count)
                return;

            if (items[current] != 0)
            {
                node.Left = new TreeNode(items[current++]);
                _FillTree(ref current, items, node.Left);
            }
            else
            {
                current++;
            }

            if (items[current] != 0)
            {
                node.Right = new TreeNode(items[current++]);
            }
        }
    }
}

```

```

        _FillTree(ref current, items, node.Right);
    }
    else
    {
        current++;
    }
}

public List<int> GetList()
{
    List<int> result = new List<int>();

    InOrderTraversal(result, _Root);

    return result;
}

private void InOrderTraversal(List<int> result, TreeNode current)
{
    if (current.Left != null)
        InOrderTraversal(result, current.Left);

    result.Add(current.Value);

    if (current.Right != null)
        InOrderTraversal(result, current.Right);
}

public void WriteValues(List<int> values)
{
    int current = 0;
    InOrderTraversalWrite(ref current, values, _Root);
}

private void InOrderTraversalWrite(ref int current, List<int> values,
TreeNode node)
{
    if (node.Left != null)
        InOrderTraversalWrite(ref current, values, node.Left);

    node.Value = values[current++];

    if (node.Right != null)
        InOrderTraversalWrite(ref current, values, node.Right);
}

public List<List<int>> GetAllSums(int sum)
{
    List<List<int>> result = new List<List<int>>();

    _GetAllSums(sum, _Root, result, new List<int>());

    return result;
}

```

```

    }

    private void _GetAllSums(int sum, TreeNode current, List<List<int>> result,
List<int> trace)
    {
        trace.Add(current.Value);

        int sum_copy = sum;
        int i = trace.Count - 1;
        for (; i >= 0 && sum_copy > 0; i--)
        {
            sum_copy -= trace[i];
        }

        if (sum_copy == 0)
        {
            i++;
            List<int> new_res = new List<int>();
            while(i < trace.Count)
            {
                new_res.Add(trace[i++]);
            }
            result.Add(new_res);
        }

        if (current.Left != null)
            _GetAllSums(sum, current.Left, result, trace);

        if (current.Right != null)
            _GetAllSums(sum, current.Right, result, trace);

        trace.RemoveAt(trace.Count - 1);
    }
}

```

Код 1.1 – Реалізація алгоритму BinaryTree

```

namespace Classes
{
    public class TreeNode
    {
        public int Value { get; set; }
        public TreeNode? Left { get; set; } = null;
        public TreeNode? Right { get; set; } = null;

        public TreeNode(int value)
        {
            Value = value;
        }
    }
}

```

Код 1.2 – Реалізація алгоритму TreeNode

```
using Classes;

namespace Lab6
{
    internal class Program
    {
        static async Task Main(string[] args)
        {
            BinaryTree tree = await BinaryTree
                .CreateFromFile("D:\\LAB_TA\\teoriya-algoritmiv-6\\Lab6\\task_06_examples\\input_1000b.txt");

            List<int> getItems = tree.GetList();

            getItems.Sort();

            tree.WriteValues(getItems);

            int sum = 1025;
            var sums = tree.GetAllSums(sum);

            string outputFile = "D:\\LAB_TA\\teoriya-algoritmiv-6\\Lab6\\task_06_examples\\output_1000b_1025.txt";

            using FileStream stream = File.Create(outputFile);
            using StreamWriter writer = new StreamWriter(stream);

            foreach(var s in sums)
            {
                writer.WriteLine(string.Join(' ', s));
            }
        }
    }
}
```

Код 1.2 – Опрацювання вхідних файлів

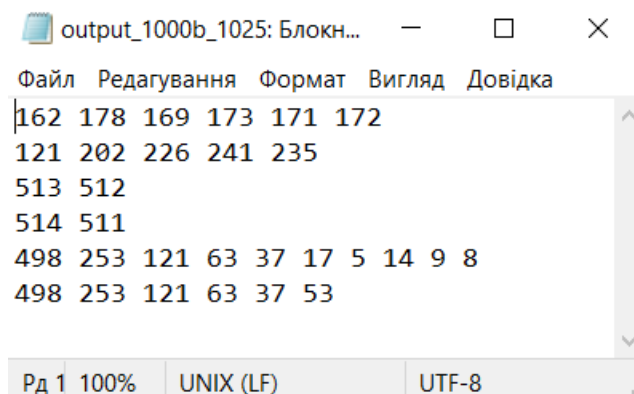
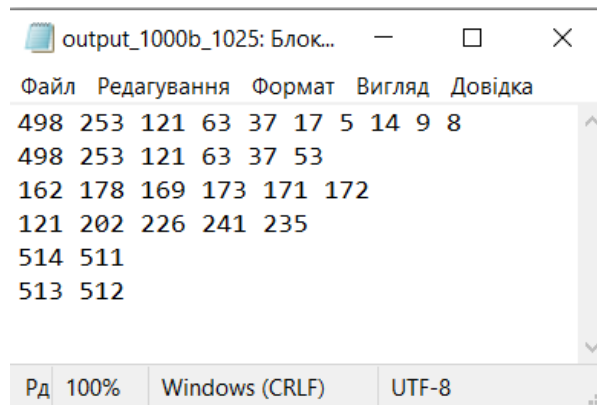


Рисунок 1.1 – Приклад вихідних даних



```
output_1000b_1025: Блок...
Файл  Редагування  Формат  Вигляд  Довідка
498 253 121 63 37 17 5 14 9 8
498 253 121 63 37 53
162 178 169 173 171 172
121 202 226 241 235
514 511
513 512
Рд 100% Windows (CRLF) UTF-8
```

Рисунок 1.2 – Результат роботи коду

Висновок: У цій лабораторній роботі ми досліджували поняття пірамід у контексті теорії алгоритмів. Ми вивчили основні властивості пірамід, такі як структура даних, яка представляє собою деревоподібну структуру з коренем у верхньому вузлі та вузлами-нащадками, розташованими на двох рівнях, та здатність до швидкого пошуку та видалення мінімального (або максимального) елемента. Ми також дослідили основні операції, пов'язані з пірамідами, такі як вставка та видалення елементів. Розглянули алгоритми цих операцій та їхню складність у різних варіаціях пірамід - міні-пірамід та макс-пірамід. Загалом, вивчення пірамід у теорії алгоритмів має велике значення для розуміння та розв'язання різних завдань у комп'ютерних науках, таких як сортування, пошук і оптимізація. Вони є важливим інструментом для розробки ефективних алгоритмів та покращення продуктивності програмного забезпечення.