Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 2

з дисципліни «Алгоритми та структури даних» на тему:

«ДЕРЕВА ТА ГЕШ-ТАБЛИЦІ»

Варіант №8

Виконав:

ст. гр. КНТ-113сп Іван Щедровський

Прийняв:

Старший викладач Лариса ДЕЙНЕГА

2023

1 Мета роботи:

1.1 Засвоїти основні концепції геш-таблиць та бінарних дерев пошуку і B-дерев зокрема.

1.2 Навчитися використовувати геш-таблиці та B-дерева на практиці.

2 Завдання до лабораторної роботи:

Розробити програмне забезпечення, що виконує базові операції з геш-таблицями та B-деревами.

2.1 Розроблюваний програмний проєкт має складатися з окремих класів, що реалізують структури даних геш-таблиця та бінарне дерево пошуку, а також має містити окремий модуль, що забезпечує інтерфейсну взаємодію з користувачем для роботи зі створеними класами.

2.2 Клас, що реалізує геш-таблицю, має дозволяти виконувати наступні операції на основі окремих методів: вставлення елементу, видалення елементу, пошук елементу, відображення структури геш-таблиці на основі використання параметрів, обраних у відповідності з варіантом індивідуального завдання з п. 2.3.4.

2.3 Клас, що реалізує B-дерево, має дозволяти виконувати наступні операції на основі окремих методів: створення порожнього дерева, відображення структури дерева, пошук у дереві, вставлення ключа, видалення ключа.

2.4 Розв’язати індивідуальне завдання, що складається з 2 задач, наведених нижче, за допомогою розроблених модулів програмного забезпечення.

2.3.1 Задача 2 B. Створити геш-таблицю, що використовує метод ланцюжків для розв’язання колізій та геш-функцію множення. Геш-таблицю заповнити на основі виділення інформації з текстового файлу, в якому містяться прізвища, ім’я і по батькові співробітників фірми та займані ними посади. Визначити посаду заданого співробітника.

2.3.2 Задача 2 Б. Дані про власників автомобілів включають ідентифікаційний номер транспортного засобу, дату реєстрації та власника (прізвище, ім’я, по батькові). Сформувати дерево з інформації про власників автомобілів. Реалізувати пошук інформації про автомобіль за заданим ідентифікаційним номером транспортного засобу, визначення осіб, які володіють більше ніж одним автомобілем.

3 Текст розробленого програмного забезпечення з коментарями:

src\AVLTree.class.ts

```
import { LogTreeNode } from './helper/logTree'
/*
- [x] створення порожнього дерева
- [x] відображення структури дерева
- [x] пошук у дереві
- [x] вставлення ключа
- [x] видалення ключа
*/
class TreeNode<T> {
    key: number
    value: T
    left: TreeNode<T> | null = null
    right: TreeNode<T> | null = null
    height = 0
    constructor(key: number, value: T) {
        this.key = key
        this.value = value
    }
    insert(node: TreeNode<T>, key: number, value: T) {
        if (key < node.key) {
            if (node.left === null) {
                node.left = new TreeNode<T>(key, value)
            } else {
                this.insert(node.left, key, value)
            }
        } else if (node.right === null) {
            node.right = new TreeNode(key, value)
        } else {
            this.insert(node.right, key, value)
        }
        this.updateHeight(node)
        this.balance(node)
    }
    search(node: TreeNode<T>, key: number) {
```

```
    if (node === null) return null
    if (node.key === key) return node
    return key < node.key
        ? this.search(node.left, key)
        : this.search(node.right, key)
}
getMin(node: TreeNode<T>): TreeNode<T> | null {
    if (node === null) return null
    if (node.left === null) return node
    return this.getMin(node.left)
}
getMax(node: TreeNode<T>): TreeNode<T> | null {
    if (node === null) return null
    if (node.right === null) return node
    return this.getMax(node.right)
}
delete(node: TreeNode<T>, key: number) {
    if (node === null) return null
    if (key < node.key) node.left = this.delete(node.left, key)
    else if (key > node.key) node.right = this.delete(node.right, key)
    else {
        if (node.left === null || node.right === null) {
            node = node.left === null ? node.right : node.left
        } else {
            const maxInLeft = this.getMax(node.left)
            node.key = maxInLeft.key
            node.value = maxInLeft.value
            node.left = this.delete(node.left, maxInLeft.key)
        }
    }
    if (node !== null) {
        this.updateHeight(node)
        this.balance(node)
    }
    return node
}
treeForOutput(node: TreeNode<T>) {
    if (node === null) return
    const left = this.treeForOutput(node.left)
    const right = this.treeForOutput(node.right)
    const EMPTY_NAME = ''
    let children = [left, right].map(item => {
        if (item !== undefined) return item
        return {
            name: EMPTY_NAME
        }
    })
    if (children[0].name === EMPTY_NAME && children[1].name === EMPTY_NAME) {
        children = []
    }
    const result: LogTreeNode = {
        name: String(node.key)
```

```
    }
    if (children.length > 0) {
      result.children = children
    }
    return result
}
// Симетричний обхід
inorderTreeWalkPrint(node: TreeNode<T>) {
    if (node === null) return
    const result = {}
    const left = this.inorderTreeWalkPrint(node.left)
    const right = this.inorderTreeWalkPrint(node.right)
    if (left) result['left'] = left
    result['root'] = node.value
    if (right) result['right'] = right
    return result
}
// Зворотній обхід
preorderTreeWalkPrint(node: TreeNode<T>) {
    if (node === null) return
    this.preorderTreeWalkPrint(node.left)
    this.preorderTreeWalkPrint(node.right)
    console.log(node.value)
}
// Прямий обхід
postorderTreeWalkPrint(node: TreeNode<T>) {
    if (node === null) return
    console.log(node.value)
    this.postorderTreeWalkPrint(node.left)
    this.postorderTreeWalkPrint(node.right)
}
updateHeight(node: TreeNode<T>) {
    node.height =
      Math.max(this.getHeight(node.left), this.getHeight(node.right)) + 1
}
getHeight(node: TreeNode<T>) {
    return node === null ? -1 : node.height
}
getBalance(node: TreeNode<T>) {
    return node === null
      ? 0
      : this.getHeight(node.right) - this.getHeight(node.left)
}
swap(a: TreeNode<T>, b: TreeNode<T>) {
    const a_key = a.key
    a.key = b.key
    b.key = a_key
    const a_value = a.value
    a.value = b.value
    b.value = a_value
}
rightRotate(node: TreeNode<T>) {
```

```
        this.swap(node, node.left)
        const buffer = node.right
        node.right = node.left
        node.left = node.right.left
        node.right.left = node.right.right
        node.right.right = buffer
        this.updateHeight(node.right)
        this.updateHeight(node)
    }
    leftRotate(node: TreeNode<T>) {
        this.swap(node, node.right)
        const buffer = node.left
        node.left = node.right
        node.right = node.left.right
        node.left.right = node.left.left
        node.left.left = buffer
        this.updateHeight(node.left)
        this.updateHeight(node)
    }
    balance(node: TreeNode<T>) {
        const balance = this.getBalance(node)
        if (balance === -2) {
            if (this.getBalance(node.left) === 1) this.leftRotate(node.left)
            this.rightRotate(node)
        } else if (balance === 2) {
            if (this.getBalance(node.right) === -1) this.rightRotate(node.right)
            this.leftRotate(node)
        }
    }
}
class AVLTree<T> {
    headNode: TreeNode<T> | null = null
    insert(key: number, value: T) {
        if (this.headNode === null) {
            this.headNode = new TreeNode<T>(key, value)
            return
        }
        this.headNode.insert(this.headNode, key, value)
    }
    search(key: number) {
        return this.headNode.search(this.headNode, key)?.value
    }
    delete(key: number) {
        return this.headNode.delete(this.headNode, key)
    }
    showStructure(type: 'inorder' | 'preorder' | 'postorder' = 'inorder') {
        if (type === 'inorder')
            return this.headNode.inorderTreeWalkPrint(this.headNode)
        if (type === 'preorder')
            return this.headNode.preorderTreeWalkPrint(this.headNode)
        if (type === 'postorder')
            return this.headNode.postorderTreeWalkPrint(this.headNode)
```

```
    }
    treeForOutput() {
      return this.headNode.treeForOutput(this.headNode)
    }
  }
  export default AVLTree
```

src\HashTable.class.ts

```
  /*
  - [x] Вставлення елементу
  - [x] Видалення елементу
  - [x] Пошук елементу
  - [x] Відображення структури геш-таблиці на основі використання параметрів, обраних у відповідності з варіантом індивідуального завдання з п. 2.3.4.
  */
  type KeyType = number | string
  class List {
    private key: KeyType
    private value: unknown
    public next: List | null
    addOrUpdate(key: KeyType, value: unknown) {
      if (this.key == undefined || this.value == undefined || this.key === key) {
        this.key = key
        this.value = value
        return this
      }
      if (this.next) {
        this.next.addOrUpdate(key, value)
        return this
      }
      this.next = new List().addOrUpdate(key, value)
    }
    get(key: KeyType): unknown | null {
      if (this.key === key) {
        return this.value
      }
      if (!this.next) return null
      return this.next.get(key)
    }
    remove(key: KeyType): List | null {
      const dummy = new List()
      dummy.next = this.next
      let prev = dummy
      let current = new List()
      if (this.key === key) {
        prev.next = current.next
        current = current.next
      } else {
        prev = current
        current = current.next
      }
      while (current) {
```

```
            if (current.key === key) {
                prev.next = current.next
                current = current.next
            } else {
                prev = current
                current = current.next
            }
        }
        return dummy.next
    }
    __getInfoWithRemove(): null | [KeyType, unknown, List | null] {
        this.remove(this.key)
        return [this.key, this.value, this.next]
    }
    getShow(): string[] {
        if (this.next == null) return [String(this)]
        return [String(this), ...this.next.getShow()]
    }
    toString() {
        return `${this.key}: ${this.value}`
    }
}
class HashTable {
    private readonly sizes: number[] = [
        5, 11, 23, 47, 97, 193, 389, 769, 1543, 3072, 3079, 12289, 24593, 49157,
        98317, 196613, 393241, 786433, 1572869, 3145739, 6291469, 12582917,
        25165843, 50331653, 100663319, 201326611, 402653189, 805306457, 1610612736,
        2147483629
    ]
    private sizes_index: number = 0
    private factor = 0.75
    private count: number = 0
    private values: List[] = Array(this.sizes[this.sizes_index])
    addOrUpdate(key: KeyType, value: unknown) {
        if (this.checkMemory()) {
            this.addMemory()
        }
        const index = this.getIndex(key)
        if (this.values[index] == null) {
            this.values[index] = new List()
            this.count++
        }
        this.values[index].addOrUpdate(key, value)
    }
    get(key: KeyType) {
        const index = this.getIndex(key)
        if (this.values[index] == null) {
            return null
        }
        return this.values[index].get(key)
    }
    remove(key: KeyType) {
```

```
    const index = this.getIndex(key)
    if (this.values[index] == null) {
        return null
    }
    const removeResult = this.values[index].remove(key)
    if (removeResult !== null) {
        this.values[index] = removeResult
        return null
    }
    this.values[index] = null
}
show() {
    return this.values
        .map((item, index) => {
            if (item) return { index: index, value: item.getShow().join(' => ') }
        })
        .filter(item => item != undefined)
}
private getHash(key: KeyType): number {
    const value = typeof key === 'number' ? key : this.stringToNumber(key)
    const w = 10
    const A = Math.sqrt(5) / 2 ** w
    const M = 2 ** 16
    const resultOfMultiple = value * A
    return Math.ceil(M * (resultOfMultiple % 1))
}
private stringToNumber(key: string) {
    let hash = 0
    for (let i = 0; i < key.length; i++) {
        hash = (hash << 5) - hash + key.charCodeAt(i)
    }
    return Math.abs(hash)
}
private getIndex(key: KeyType): number {
    const hash = this.getHash(key)
    return hash % this.sizes[this.sizes_index]
}
private checkMemory() {
    return this.count / this.sizes[this.sizes_index] >= this.factor
}
private addMemory() {
    this.count = 0
    const oldValues = [...this.values]
    this.sizes_index += 1
    this.values = Array(this.sizes[this.sizes_index])
    for (let i = 0; i < this.sizes[this.sizes_index - 1]; i++) {
        if (!oldValues[i]) continue
        let node = oldValues[i].__getInfoWithRemove()
        while (node != null) {
            this.addOrUpdate(node[0], node[1])
            if (node[2] == null) {
                break
```

```
            }
            node = node[2].__getInfoWithRemove()
          }
        }
    }
}
export default HashTable

src\main.ts
import AVLTree from './AVLTree.class'
import HashTable from './HashTable.class'
import { logTree } from './helper/logTree'
import * as fs from 'fs'
console.log('Var 8 => Task 1 В)')
/*
В. Створити геш-таблицю, що використовує метод ланцюжків
для розв’язання колізій та геш-функцію множення. Геш-таблицю
заповнити на основі виділення інформації з текстового файлу, в якому
містяться прізвища, ім’я і по батькові співробітників фірми та займані
ними посади. Визначити посаду заданого співробітника.
*/
const hashData = JSON.parse(fs.readFileSync('./src/data/task1.json', 'utf8'))
const hashTable = new HashTable()
const start = []
hashData.map(([name, job], index) => {
    start.push({ name, job })
    hashTable.addOrUpdate(`${index} - ${name}`, job)
})
console.log('Data from file:')
console.table(start)
console.log()
console.log('How save in HashTable:')
console.table(hashTable.show())
console.log()
hashTable.addOrUpdate(`24 - Tara Cremin Skiles`, 'Accountability')
console.log(
    'How save in HashTable after add "24 - Tara Cremin Skiles: Accountability" people:'
)
console.table(hashTable.show())
console.log()
console.log("Search '13 - Javier Kilback Rodriguez' value: ")
console.log('value = ', hashTable.get('13 - Javier Kilback Rodriguez'))
console.log('\n ====================\n')
console.log('Var 8 => Task 2 Б)')
/*
    2 Б) Дані про власників автомобілів включають ідентифікаційний
    номер транспортного засобу, дату реєстрації та власника (прізвище,
    ім’я, по батькові). Сформувати дерево з інформації про власників
    автомобілів. Реалізувати пошук інформації про автомобіль за заданим
    ідентифікаційним номером транспортного засобу, визначення осіб, які
    володіють більше ніж одним автомобілем.
*/
```

```
class CarOwner {
    id: number
    registerDate: Date
    owner: string
    constructor(id: number, registerDate: Date, owner: string) {
        this.id = id
        this.registerDate = registerDate
        this.owner = owner
    }
}
const carOwners = [
    new CarOwner(67324, new Date('Sun Jan 06 2075 23:37:10'), 'Alexandra Becker'),
    new CarOwner(29497, new Date('Tue May 03 2016 23:40:18'), 'Mr. Lela Kessler'),
    new CarOwner(22486, new Date('Tue Apr 28 2020 02:28:00'), 'Roosevelt Crooks'),
    new CarOwner(85849, new Date('Tue Jan 13 2054 10:55:12'), 'Cory Schowalter'),
    new CarOwner(74389, new Date('Sat Aug 05 2017 19:04:01'), 'Wendell Hessel'),
    new CarOwner(44563, new Date('Tue Oct 05 2094 03:34:48'), 'Joe Lesch'),
    new CarOwner(61297, new Date('Fri Dec 21 2012 16:56:12'), 'Karla Simonis'),
    new CarOwner(53376, new Date('Mon Nov 05 2074 12:16:53'), 'Marty Beahan'),
    new CarOwner(5613, new Date('Sat Jan 26 2069 01:07:46'), 'Kim Lockman'),
    new CarOwner(15435, new Date('Sun Dec 17 1995 18:56:09'), 'Mr. Victor Kunze'),
    new CarOwner(15673, new Date('Wed Sep 17 2053 01:54:01'), 'Arturo Robel IV'),
    new CarOwner(63325, new Date('Fri Sep 02 2033 01:25:31'), 'Ms. Donna Kessler')
]
const binaryTree = new AVLTree<CarOwner>()
carOwners.map(item => {
    binaryTree.insert(item.id, item)
})
console.log('Tree structure after creating:')
console.log(logTree(binaryTree.treeForOutput()))
binaryTree.delete(74389)
binaryTree.delete(29497)
binaryTree.delete(53376)
binaryTree.delete(15673)
console.log('Tree structure after delete 4 items:')
console.log(logTree(binaryTree.treeForOutput()))
console.log("Find 5613, 61297 and 44563 id's:")
console.table([
    binaryTree.search(5613),
    binaryTree.search(61297),
    binaryTree.search(44563)
])

src\helper\logTree.ts
export type LogTreeNode = {
    name: string
    children?: Array<LogTreeNode>
}
function logTree(
    tree: LogTreeNode | LogTreeNode[],
    level = 0,
    parentPre = '',
```

```
    treeStr = ''
) {
    if (!Array.isArray(tree)) {
        const children = tree['children']
        treeStr = `${tree['name']}\n`
        if (children) {
            treeStr += logTree(children, level + 1)
        }
        return treeStr
    }
    tree.forEach((child, index) => {
        const hasNext = tree[index + 1] ? true : false
        const children = child['children']
        treeStr += `${parentPre}${hasNext ? '├' : '└'}── ${child['name']}\n`
        if (children) {
            treeStr += logTree(
                children,
                level + 1,
                `${parentPre}${hasNext ? '│ ' : '  '}   `
            )
        }
    })
    return treeStr
}
export { logTree }
```

4 Результати роботи програмного забезпечення:

На рисунках 4.1, 4.2 та 4.3 показано виконання програми:

Var 8 => Task 1 B)
Data from file:

(index)	name	job
0	'Loy Graham Zboncak'	'Directives'
1	'Roy Breitenberg Runte'	'Accountability'
2	'Erika Emard Feest'	'Security'
3	'Vance Flatley Thiel'	'Research'
4	'Mallory Hoppe O`Hara'	'Applications'
5	'Dulce Douglas Boyer'	'Interactions'
6	'Dedrick Jerde Kozey'	'Accounts'
7	'Hudson Langosh Mayert'	'Applications'
8	'Georgianna Bergstrom VonRueden'	'Solutions'
9	'Marjorie Rolfson Bashirian'	'Integration'
10	'Mitchell O`Keefe Shanahan'	'Branding'
11	'Filiberto Gottlieb Marquardt'	'Accountability'
12	'Murphy Cassin Franey'	'Configuration'
13	'Javier Kilback Rodriguez'	'Branding'
14	'Kayley Powlowski Kuphal'	'Assurance'
15	'Liliana Johnston Ebert'	'Metrics'
16	'Everette Little Cartwright'	'Accounts'
17	'Otilia Fadel Spinka'	'Implementation'
18	'Watson Schuppe Lowe'	'Web'
19	'Duane Emmerich Rohan'	'Paradigm'
20	'Linwood Huel VonRueden'	'Marketing'
21	'Toni Johns Wiegand'	'Accounts'
22	'Taylor Kreiger Kihn'	'Functionality'

How save in HashTable:

(index)	index	value
0	0	'1 - Roy Breitenberg Runte: Accountability => 21 - Toni Johns Wiegand: Accounts'
1	1	'0 - Loy Graham Zboncak: Directives'
2	2	'15 - Liliana Johnston Ebert: Metrics => 18 - Watson Schuppe Lowe: Web'
3	4	'19 - Duane Emmerich Rohan: Paradigm'
4	5	'9 - Marjorie Rolfson Bashirian: Integration'
5	7	'20 - Linwood Huel VonRueden: Marketing'
6	8	'13 - Javier Kilback Rodriguez: Branding => 12 - Murphy Cassin Franey: Configuration'
7	9	'16 - Everette Little Cartwright: Accounts'
8	12	'17 - Otilia Fadel Spinka: Implementation'
9	13	'7 - Hudson Langosh Mayert: Applications'
10	14	'5 - Dulce Douglas Boyer: Interactions'
11	15	'8 - Georgianna Bergstrom VonRueden: Solutions'
12	16	'3 - Vance Flatley Thiel: Research => 6 - Dedrick Jerde Kozey: Accounts'
13	17	'2 - Erika Emard Feest: Security'
14	19	'14 - Kayley Powlowski Kuphal: Assurance'
15	20	'11 - Filiberto Gottlieb Marquardt: Accountability => 22 - Taylor Kreiger Kihn: Functionality'
16	21	'4 - Mallory Hoppe O`Hara: Applications => 10 - Mitchell O`Keefe Shanahan: Branding'

Рисунок 4.1 – Виконання програми 1 завдання перша половина

How save in HashTable after add "24 - Tara Cremin Skiles: Accountability" people:

(index)	index	value
0	0	'1 - Roy Breitenberg Runte: Accountability => 21 - Toni Johns Wiegand: Accounts'
1	1	'0 - Loy Graham Zboncak: Directives'
2	2	'15 - Liliana Johnston Ebert: Metrics => 18 - Watson Schuppe Lowe: Web'
3	4	'19 - Duane Emmerich Rohan: Paradigm'
4	5	'9 - Marjorie Rolfson Bashirian: Integration => 24 - Tara Cremin Skiles: Accountability'
5	7	'20 - Linwood Huel VonRueden: Marketing'
6	8	'13 - Javier Kilback Rodriguez: Branding => 12 - Murphy Cassin Franey: Configuration'
7	9	'16 - Everette Little Cartwright: Accounts'
8	12	'17 - Otilia Fadel Spinka: Implementation'
9	13	'7 - Hudson Langosh Mayert: Applications'
10	14	'5 - Dulce Douglas Boyer: Interactions'
11	15	'8 - Georgianna Bergstrom VonRueden: Solutions'
12	16	'3 - Vance Flatley Thiel: Research => 6 - Dedrick Jerde Kozey: Accounts'
13	17	'2 - Erika Emard Feest: Security'
14	19	'14 - Kayley Powlowski Kuphal: Assurance'
15	20	'11 - Filiberto Gottlieb Marquardt: Accountability => 22 - Taylor Kreiger Kihn: Functionality'
16	21	'4 - Mallory Hoppe O`Hara: Applications => 10 - Mitchell O`Keefe Shanahan: Branding'

Search '13 - Javier Kilback Rodriguez' value:
value = Branding

Рисунок 4.2 – Виконання програми 1 завдання 2 половина

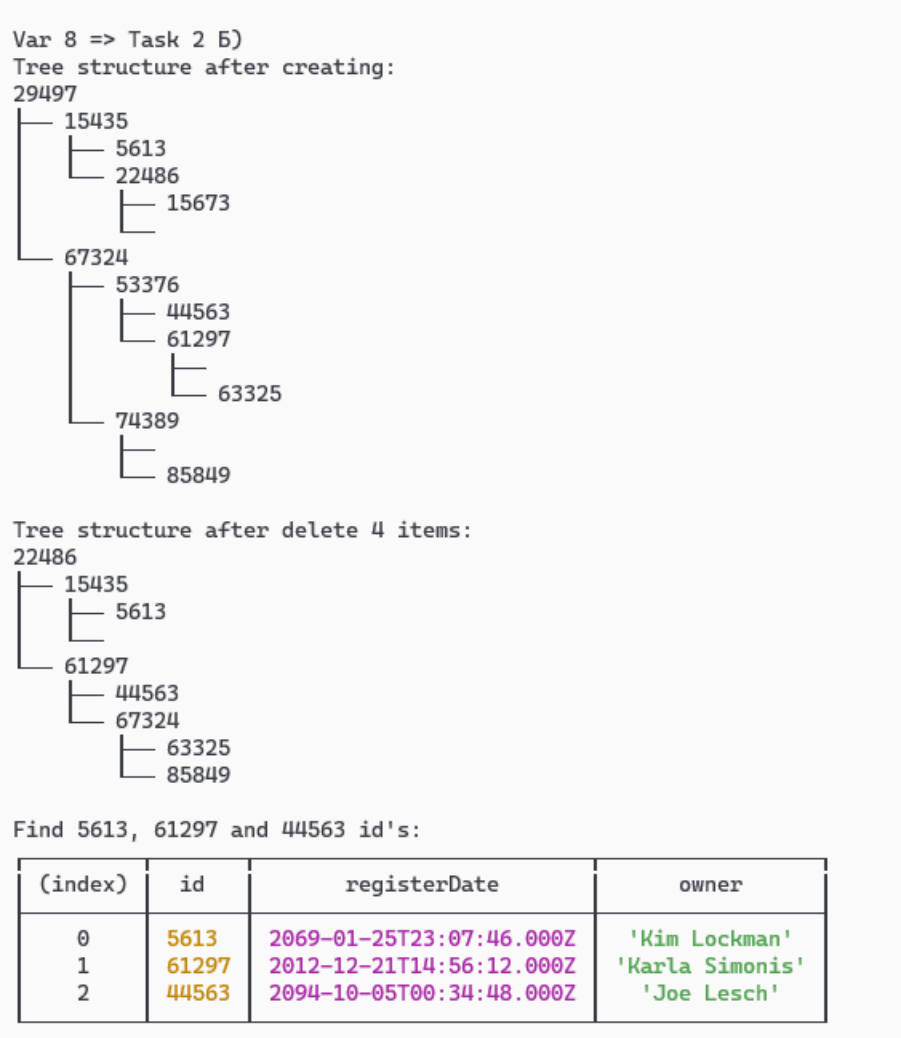

Рисунок 4.3 – Виконання програми 1 завдання

5 Висновки:

У ході цієї лабораторної роботи я навчився фундаментальним концепціям геш-таблиць та бінарних дерев пошуку, зокрема B-дерев. Розуміння цих структур даних стало для мене ключовим, оскільки вони відкривають широкий спектр можливостей для ефективного зберігання та операцій над даними.

На практиці я здобув навички використання геш-таблиць та B-дерев у програмному забезпеченні. Розробка окремих класів для кожної з цих структур

дозволила мені ефективно виконувати операції вставки, видалення, пошуку та відображення даних.

Ця лабораторна робота надала мені важливі навички роботи зі складними структурами даних та їх практичне застосування. Тепер я готовий використовувати геш-таблиці та B-дерева в реальних програмних проєктах для ефективної обробки та організації великих обсягів інформації.