Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 1

з дисципліни «Алгоритми та структури даних» на тему:

«ЛІНІЙНІ СТРУКТУРИ ДАНИХ»

Варіант №8

Виконав:

ст. гр. КНТ-113сп Іван Щедровський

Прийняв:

Старший викладач Лариса ДЕЙНЕГА

2023

1 Мета роботи:

1.1.1 Вивчити основні концепції побудови лінійних структур даних: зв’язних списків, стеків, куп та черг з пріоритетами.

1.1.2 Навчитися обирати та реалізовувати структури даних для сортування, вставки, видалення та пошуку елементів.

1.1.3 Навчитися реалізовувати та застосовувати алгоритм пірамідального сортування на практиці.

2 Завдання до лабораторної роботи:

Розробити програмне забезпечення, що виконує базові операції з лінійними структурами даних.

2.1 Клас, що реалізує двозв’язний список, має дозволяти виконувати наступні операції на основі окремих методів: додавання вузла в початок списку, додавання вузла після заданого, пошук вузла в списку, видалення вузла, виведення вузлів на екран з початку та з кінця.

2.2 Клас, що реалізує купу (чергу з пріоритетами), має дозволяти виконувати наступні операції на основі окремих методів: вставлення елементу, сортування елементів, побудова купи з невпорядкованого масиву, видалення елементу, сортування елементів із використанням купи, виведення елементів на екран.

2.3 Розробити окремий модуль програмного забезпечення для реалізації пірамідального сортування на основі розробленого класу.

2.4 Розв’язати індивідуальне завдання за допомогою розробленої реалізації пірамідального сортування.

Варіант № 8

Книжки в бібліотеці характеризуються наступними даними:

– автор;

– назва;

– жанр;

– видавництво;

– рік публікації;

– кількість сторінок;

– загальна кількість екземплярів;

– кількість екземплярів у читачів.

Визначити книжки, кількість наявних екземплярів яких у бібліотеці в поточний момент входить у перші 50 %. Обчислити сумарну кількість наявних екземплярів таких книжок.

3 Текст розробленого програмного забезпечення з коментарями:

./Book.ts

```
/*
Книжки в бібліотеці характеризуються наступними даними:
– автор;
– назва;
– жанр;
– видавництво;
– рік публікації;
– кількість сторінок;
– загальна кількість екземплярів;
– кількість екземплярів у читачів.
*/

class Book {
	author: string
	title: string
	genre: string
	publisher: string
	publicationYear: number
	pageCount: number
	totalCopies: number
	copiesCheckedOut: number

	constructor(
		author: string,
		title: string,
		genre: string,
		publisher: string,
		publicationYear: number,
		pageCount: number,
```

```
            totalCopies: number,
            copiesCheckedOut: number
        ) {
            this.author = author
            this.title = title
            this.genre = genre
            this.publisher = publisher
            this.publicationYear = publicationYear
            this.pageCount = pageCount
            this.totalCopies = totalCopies
            this.copiesCheckedOut = copiesCheckedOut
        }

        valueOf() {
            return this.totalCopies - this.copiesCheckedOut
        }

        clone() {
            return new Book(
                this.author,
                this.title,
                this.genre,
                this.publisher,
                this.publicationYear,
                this.pageCount,
                this.totalCopies,
                this.copiesCheckedOut
            )
        }
}

export { Book }

./doubly-linked-list.ts
class DoublyLinkedListItem<T> {
        previous: DoublyLinkedListItem<T> | null = null
        value: T
        next: DoublyLinkedListItem<T> | null = null

        constructor(
            value: T,
            previous: DoublyLinkedListItem<T> | null = null,
            next: DoublyLinkedListItem<T> | null = null
        ) {
            this.value = value
            this.previous = previous
            this.next = next
        }
}

/*
Клас, що реалізує двозв’язний список, має дозволяти
```

виконувати наступні операції на основі окремих методів:
[x] додавання вузла в початок списку
[x] додавання вузла після заданого
[x] пошук вузла в списку
[x] видалення вузла
[ ] виведення вузлів на екран з початку та з кінця.
*/

```
class DoublyLinkedList<T> {
	head: DoublyLinkedListItem<T>

	constructor(value: T) {
		this.head = new DoublyLinkedListItem(value)
	}

	addAsHead(value: T) {
		const newHead = new DoublyLinkedListItem(value, null, this.head)
		this.head.previous = newHead

		this.head = newHead
	}

	addAfter(node: DoublyLinkedListItem<T>, value: T) {
		const newNode = new DoublyLinkedListItem(value, node, node.next)

		node.next = newNode

		if (newNode.next) {
			newNode.next.previous = newNode
		}

		return this
	}

	find(value: T): DoublyLinkedListItem<T> | null {
		let current: DoublyLinkedListItem<T> | null = this.head

		while (current !== null) {
			if (current.value === value) {
				return current
			}

			current = current.next
		}

		return null
	}

	delete(node: DoublyLinkedListItem<T>) {
		if (node.previous === null && node.next === null) return

		if (node === this.head) {
```

```
                if (this.head.next === null) {
                        throw new Error('List must have minimum one node - head')
                }

                if (this.head.next !== null) {
                        this.head = this.head.next
                        this.head.previous = null

                        node.next = null
                        return
                }
        }

        if (node.previous === null) {
                console.log('What? How?!')
                return
        }

        node.previous.next = node.next

        if (node.next) {
                node.next.previous = node.previous
        }
    }

    getPrint(valueToString: (value: T) => string) {
        const result: string[] = []

        let current: DoublyLinkedListItem<T> | null = this.head

        while (current !== null) {
                result.push(valueToString(current.value))

                current = current.next
        }

        return result.join(' ⇄ ')
    }
}

export { DoublyLinkedList }
```

./heap.ts

```
/*
Клас, що реалізує купу (чергу з пріоритетами), має
дозволяти виконувати наступні операції на основі окремих методів:
[x] вставлення елементу
[x] побудова купи з невпорядкованого масиву
[x] видалення елементу
[x] сортування елементів
[ ] виведення елементів на екран
```

```
*/

import { Book } from './Book'
import { logHeap } from './helpers/logHeap'

class Heap<T> {
	private heap: T[] = []

	add(element: T) {
		const index = this.heap.push(element)

		this.siftup(index)

		return this
	}

	fromArray(array: T[]) {
		this.heap = array

		this.heap.forEach((_, index) => {
			this.siftdown(this.heap, this.heap.length - 1 - index)
		})

		return this
	}

	extractTop(heap: T[] = this.heap) {
		const last = heap.length - 1

		;[heap[0], heap[last]] = [heap[last], heap[0]]

		const element = heap.pop() as T

		this.siftdown(heap, 0)

		return element
	}

	getSortedArray() {
		const heapCopy = this.heap.map(item => {
			if (item instanceof Book) {
				return item.clone()
			}

			return { ...item }
		}) as T[]

		const result: T[] = []

		while (heapCopy.length >= 1) {
			result.push(this.extractTop(heapCopy))
		}
```

```
            return result
        }

        getPrint(toString: (value: T) => string) {
            console.log(logHeap<T>(this.heap, toString))
        }

        private siftup(i: number) {
            let parent = Math.floor(i - 1 / 2)

            while (i !== 0 && Number(this.heap[i]) < Number(this.heap[parent])) {
                ;[this.heap[i], this.heap[parent]] = [this.heap[parent], this.heap[i]]

                i = parent
                parent = Math.floor(i - 1 / 2)
            }
        }

        private siftdown(heap: T[], i: number) {
            let left = i * 2 + 2
            let right = i * 2 + 1

            while (
                (left < heap.length && Number(heap[i]) > Number(heap[left])) ||
                (right < heap.length && Number(heap[i]) > Number(heap[right]))
            ) {
                let smallest = right

                if (right >= heap.length || Number(heap[left]) < Number(heap[right])) {
                    smallest = left
                }

                ;[heap[i], heap[smallest]] = [heap[smallest], heap[i]]
                i = smallest
                left = i * 2 + 2
                right = i * 2 + 1
            }
        }
    }

    export default Heap


    ./HeapSort.ts
    import Heap from './heap'

    class HeapSort<T> {
        heap: Heap<T>

        constructor(heap: Heap<T>) {
            this.heap = heap
```

```
        }

        getSorted() {
                return this.heap.getSortedArray()
        }
}

export { HeapSort }
```

./main.ts

```
import { HeapSort } from './HeapSort'
import { DoublyLinkedList } from './doubly-linked-list'
import Heap from './heap'
import { Book } from './Book'

const list = new DoublyLinkedList<number>(1)

list.addAfter(list.head, 2)
list.addAfter(list.head, 6)
list.addAfter(list.head, 7)
list.addAfter(list.head, 9)

console.log('DoublyLinkedList: ')
console.log(list.getPrint((value: number) => String(value)))
console.log('')

const books = [
        new Book(
                'Camille Predovic',
                'Armenian Gampr dog',
                'Electronics',
                'weepy-status.org',
                2022,
                213,
                11,
                5
        ),
        new Book(
                'Chris Von',
                'Black Norwegian Elkhound',
                'Automotive',
                'illiterate-antigen.biz',
                2001,
                308,
                24,
                6
        ),
        new Book(
                'Alonzo Fahey II',
                'Pekingese',
                'Outdoors',
```

```
        'concrete-dashboard.org',
        2021,
        399,
        50,
        43
),
new Book(
        'Fred Buckridge',
        'Fila Brasileiro',
        'Outdoors',
        'glamorous-relative.org',
        2011,
        208,
        11,
        7
),
new Book(
        'Colin O`Connell',
        'Armant',
        'Beauty',
        'graceful-territory.com',
        2018,
        365,
        38,
        30
),
new Book(
        'Shelly Greenfelder',
        'Hygen Hound',
        'Games',
        'quirky-mother-in-law.com',
        2010,
        355,
        24,
        6
),
new Book(
        'Margarita Franecki Jr.',
        'Bracco Italiano',
        'Home',
        'direct-lymphocyte.name',
        2021,
        302,
        25,
        12
),
new Book(
        'Margaret Hills',
        'Montenegrin Mountain Hound',
        'Baby',
        'attached-wake.name',
        2002,
```

```
                423,
                41,
                33
        ),
        new Book(
                'Julio Nikolaus',
                'Silky Terrier',
                'Electronics',
                'ragged-jelly.name',
                2006,
                321,
                46,
                37
        ),
        new Book(
                'Alan Gibson I',
                'Lancashire Heeler',
                'Sports',
                'definite-garbage.com',
                2006,
                356,
                44,
                4
        )
]

console.log('Individual with heap(Variant #8): ')

const heapBook = new Heap<Book>().fromArray(books)

const bookToString = (value: Book, index?: number) => {
        return `${index !== undefined ? index + ' | ' : ''}${value.author} \x1b[33m${
                value.title
        }\x1b[0m: \x1b[32m${value.totalCopies - value.copiesCheckedOut}\x1b[0m | ${
                value.copiesCheckedOut
        }/${value.totalCopies}`
}

books.map((item, index) => {
        console.log(bookToString(item, index))
})

console.log('')

heapBook.getPrint(bookToString)

const heapSort = new HeapSort<Book>(heapBook)

const a = heapSort.getSorted()

let summ = 0
```

```
a.slice(0, a.length / 2).map(item => {
	// Визначити книжки, кількість наявних екземплярів яких у бібліотеці в поточний момент входить у перші 50 %.
	console.log(bookToString(item))

	summ += Number(item)
})

// Обчислити сумарну кількість наявних екземплярів таких книжок.
console.log('\nSumm: ' + summ)


./helpers/logHeap.ts
type TreeNode<T = string> = {
	name: T
	children?: Array<TreeNode>
}

function logTree(
	tree: TreeNode<string> | TreeNode<string>[],
	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
```

```
}

function heapToTree<T>(
	heap: T[],
	toString: (value: T) => string,
	index: number = 0
): TreeNode<string> | null {
	if (index >= heap.length) return null

	const left = heapToTree(heap, toString, index * 2 + 1)
	const right = heapToTree(heap, toString, index * 2 + 2)

	return {
		name: toString(heap[index]),
		children: [left, right].filter(item => item !== null) as TreeNode<string>[]
	}
}

function logHeap<T>(heap: T[], toString: (value: T) => string) {
	return logTree(heapToTree(heap, toString) as TreeNode)
}

export { logHeap }
```

4 Результати роботи програмного забезпечення:

На рисунку 4.1 показано виконання програми:

```
DoublyLinkedList:
1 ⇄ 9 ⇄ 7 ⇄ 6 ⇄ 2

Individual with heap(Variant #8):
0 | Fred Buckridge Fila Brasileiro: 4 | 7/11
1 | Camille Predovic Armenian Gampr dog: 6 | 5/11
2 | Alonzo Fahey II Pekingese: 7 | 43/50
3 | Margaret Hills Montenegrin Mountain Hound: 8 | 33/41
4 | Colin O'Connell Armant: 8 | 30/38
5 | Shelly Greenfelder Hygen Hound: 18 | 6/24
6 | Margarita Franecki Jr. Bracco Italiano: 13 | 12/25
7 | Chris Von Black Norwegian Elkhound: 18 | 6/24
8 | Julio Nikolaus Silky Terrier: 9 | 37/46
9 | Alan Gibson I Lancashire Heeler: 40 | 4/44

Fred Buckridge Fila Brasileiro: 4 | 7/11
├── Camille Predovic Armenian Gampr dog: 6 | 5/11
│   ├── Margaret Hills Montenegrin Mountain Hound: 8 | 33/41
│   │   ├── Chris Von Black Norwegian Elkhound: 18 | 6/24
│   │   └── Julio Nikolaus Silky Terrier: 9 | 37/46
│   └── Colin O'Connell Armant: 8 | 30/38
│       └── Alan Gibson I Lancashire Heeler: 40 | 4/44
└── Alonzo Fahey II Pekingese: 7 | 43/50
    ├── Shelly Greenfelder Hygen Hound: 18 | 6/24
    └── Margarita Franecki Jr. Bracco Italiano: 13 | 12/25

Fred Buckridge Fila Brasileiro: 4 | 7/11
Camille Predovic Armenian Gampr dog: 6 | 5/11
Alonzo Fahey II Pekingese: 7 | 43/50
Margaret Hills Montenegrin Mountain Hound: 8 | 33/41
Colin O'Connell Armant: 8 | 30/38

Summ: 33
```

Рисунок 4.1 – Виконання програми

5 Висновки:

Під час виконання лабораторної роботи, я вивчив основні концепції побудови лінійних структур даних, таких як зв’язні списки, стеки, купи та черги з пріоритетами. Я розумію, що кожна з цих структур має свої унікальні особливості та використання.

Далі, я навчився обирати та реалізовувати відповідні структури даних для виконання операцій сортування, вставки, видалення та пошуку елементів. Це

дозволяє мені ефективно працювати з даними, враховуючи їхню природу та вимоги завдання.

Окрему увагу я приділив алгоритму пірамідального сортування. Я навчився його реалізовувати та застосовувати на практиці. Розумію, що цей алгоритм є ефективним для сортування даних у вигляді купи.

У цілому, завдяки вивченню цих концепцій та алгоритмів, я здатен ефективно працювати з лінійними структурами даних та використовувати їх для вирішення практичних завдань.