Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 4

з дисципліни «Алгоритми та структури даних» на тему:

«ДИНАМІЧНЕ ПРОГРАМУВАННЯ»

Варіант №28

Виконав:

ст. гр. КНТ-113сп Іван Щедровський

Прийняв:

Старший викладач Лариса ДЕЙНЕГА

2023

1 Мета роботи:

1.1 Вивчити основні принципи динамічного програмування.

1.2 Навчитися використовувати динамічне програмування для розв’язання практичних завдань.

2 Завдання до лабораторної роботи:

1 Розробити програмне забезпечення, що розв’язує задачу у відповідності з індивідуальним завданням з пункту 4.3.2.3 із використанням принципів динамічного програмування. Згідно варіанту 28 = варіанту 8 = Задачі В та Д

1.1 Задача 1 В. Послідовність складається з деякого набору цілих чисел. Елементи послідовності можуть бути, як від’ємними, так і невід’ємними цілими числами. Визначити найдовшу спадну підпослідовність даної послідовності. Вивести на екран довжину такої послідовності та всі її члени.

1.2 Задача 2 Д. На одній з вулиць містечка будинки класифіковано за трьома типами: перший – звичайні житлові споруди, другий – промислові споруди, а третій – міські заклади (лікарні, школи тощо). У результаті вулиця схематично зображена набором літер, кожна з яких визначає тип будинку. У процесі збору інформації про місто була створена матриця – таблиця, в якій кожен стовпчик і рядок відповідають одному з типів будівель. Відповідно клітинка такої таблиці визначає, чи розташовані на даній вулиці міста поруч будівлі заданого типу. Матриця симетрична. Визначити, скільки існує способів взаємного розташування будинків даних типів між собою за заданою матрицею для заданої кількості будинків на вулиці, тобто кількість можливих наборів літер заданої довжини, що відповідають заданій матриці.

3 Текст розробленого програмного забезпечення з коментарями:

src\FirstTask.class.ts

```
/*
В. Послідовність складається з деякого набору цілих чисел.
Елементи послідовності можуть бути, як від’ємними, так і
невід’ємними цілими числами.

Визначити найдовшу спадну підпослідовність даної послідовності.

Вивести на екран довжину такої послідовності та всі її члени.
*/

class FirstTask {
	resolve(
		nums: number[],
		index: number = 0,
		previous: number = Infinity
	): number[] {
		if (index === nums.length) return []

		const excludeCurrent = this.resolve(nums, index + 1, previous)

		if (nums[index] >= previous) return excludeCurrent

		const includeCurrent: number[] = [
			nums[index],
			...this.resolve(nums, index + 1, nums[index])
		]

		return includeCurrent.length > excludeCurrent.length
			? includeCurrent
			: excludeCurrent
	}
}

export { FirstTask }
```

src\main.ts

```
import * as chalk from 'chalk'
import { FirstTask } from './FirstTask.class'
import { SecondTask } from './SecondTask.class'

console.log(chalk.yellow('Lab 4 | Algoritms\n'))

console.log(chalk.bgRed('Lab 4 | First task\n'))
const firstTask = new FirstTask()

const firstTaskCheckData = [
	{
		input: [0, 1, 0, 3, 2, 3],
		expectedOutput: [3, 2]
	},
	{
		input: [10, 9, 2, 5, 3, 7, 101, 18],
```

```
                expectedOutput: [10, 9, 5, 3]
        },
        {
                input: [5, 8, 7, 1, 2, 10, 3],
                expectedOutput: [8, 7, 3]
        },
        {
                input: [1, 3, 6, 7, 9, 4, 10, 5, 6],
                expectedOutput: [10, 6]
        },
        {
                input: [3, 4, 2, 8, 10],
                expectedOutput: [4, 2]
        },
        {
                input: [1, 2, 3, 4, 5],
                expectedOutput: [5]
        },
        {
                input: [5, 4, 3, 2, 1],
                expectedOutput: [5, 4, 3, 2, 1]
        },
        {
                input: [1],
                expectedOutput: [1]
        },
        {
                input: [],
                expectedOutput: []
        },
        {
                input: [2, 2, 2, 2, 2],
                expectedOutput: [2]
        }
]

const testData = []

firstTaskCheckData.forEach(test => {
        const actual = firstTask.resolve(test.input)

        testData.push({
                pass: JSON.stringify(test.expectedOutput) === JSON.stringify(actual),
                input: JSON.stringify(test.input),
                expected: JSON.stringify(test.expectedOutput),
                actual: JSON.stringify(actual),
                l: actual.length
        })
})

console.table(testData)

console.log(chalk.bgBlue('\nLab 4 | Second task\n'))
const secondTask = new SecondTask()

const dataForSecond = [
        {
                matrix: [
                        [1, 0, 0],
```

```
                [0, 1, 0],
                [0, 0, 1]
            ],
            count: 6
        },
        {
            matrix: [
                [1, 1, 0],
                [1, 0, 1],
                [0, 1, 1]
            ],
            count: 6
        },
        {
            matrix: [
                [1, 1, 1],
                [1, 1, 1],
                [1, 1, 1]
            ],
            count: 6
        },
        {
            matrix: [
                [1, 1, 1],
                [1, 1, 1],
                [1, 1, 1]
            ],
            count: 7
        }
]

dataForSecond.forEach(data => {
    const result = secondTask.resolve(data.matrix, data.count)

    console.log(
        `On matrix = ${JSON.stringify(data.matrix)}, count = ${
            data.count
        } result = ${result.length}\n${JSON.stringify(result)}`
    )
})
```

src\SecondTask.class.ts

```
/*
```

Д. На одній з вулиць містечка будинки класифіковано за трьома типами: перший – звичайні житлові споруди, другий – промислові споруди, а третій – міські заклади (лікарні, школи тощо).

У результаті вулиця схематично зображена набором літер, кожна з яких визначає тип будинку. У процесі збору інформації про місто була створена матриця – таблиця, в якій кожен стовпчик і рядок відповідають одному з типів будівель.

Відповідно клітинка такої таблиці визначає,
чи розташовані на даній вулиці міста поруч будівлі заданого типу.
Матриця симетрична.

Визначити, скільки існує способів взаємного
розташування будинків даних типів між собою за заданою матрицею

```
для заданої кількості будинків на вулиці,

тобто кількість можливих
наборів літер заданої довжини, що відповідають заданій матриці.
*/

class SecondTask {
	checkItemOnValid(matrix: number[][], item: string) {
		for (let i = 0; i < matrix.length; i++) {
			for (let j = 0; j < matrix[i].length; j++) {
				if (matrix[i][j] === 0) continue

				if (!(item.includes(`${i}${j}`) || item.includes(`${j}${i}`))) {
					return false
				}
			}
		}

		return true
	}

	resolve(matrix: number[][], count: number): string[] {
		const results: string[] = []

		const inner = (result: string = '', currentIndex: number = 0): void => {
			if (currentIndex === count) {
				if (this.checkItemOnValid(matrix, result)) {
					results.push(result)
				}

				return
			}

			for (let type = 0; type < 3; type++) {
				if (matrix[type][result[currentIndex - 1]] === 0) {
					continue
				}

				inner(result + type, currentIndex + 1)
			}
		}

		inner()

		return results
	}
}

export { SecondTask }
```

4 Результати роботи програмного забезпечення:

На рисунку 4.1 показано виконання програми:

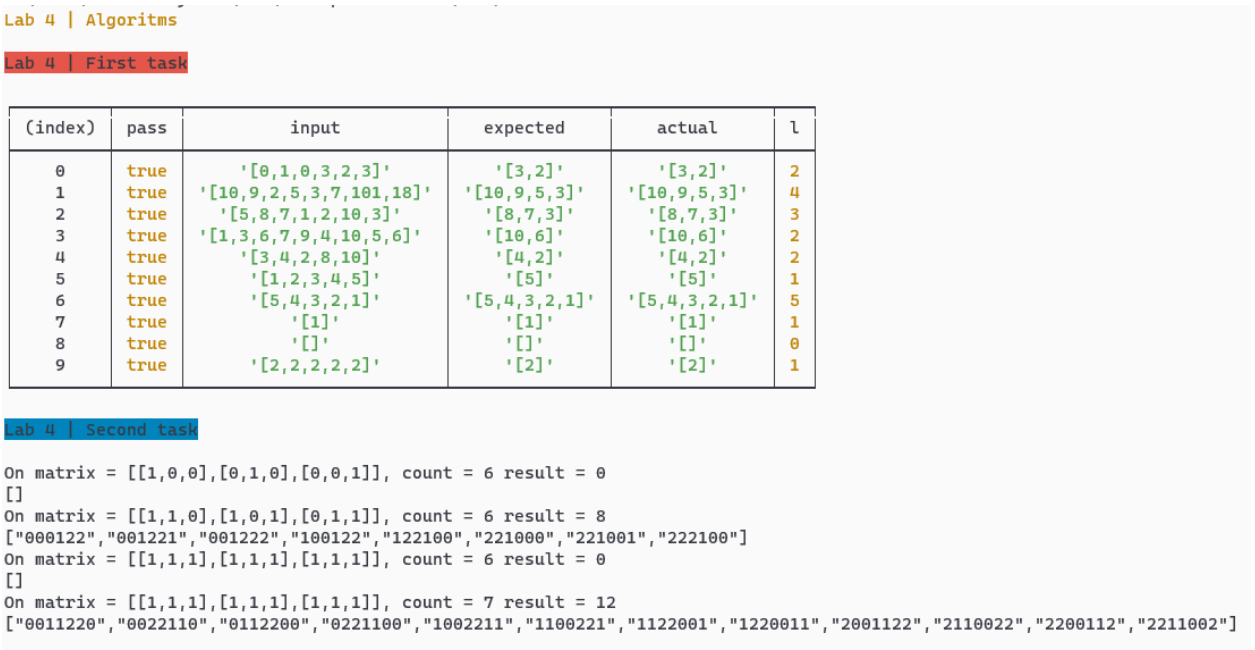

Рисунок 4.1 – Демонстрація програми

5 Висновки:

У ході виконання лабораторної роботи я навчився основним принципам динамічного програмування, що дозволяє ефективно розв'язувати складні задачі шляхом розбиття їх на більш прості підзадачі. Крім того, я навчився застосовувати ці принципи для вирішення конкретних практичних завдань.

У першому завданні (Задача 1 В) я розробив програму, яка здатна знаходити найдовшу спадну підпослідовність в заданій послідовності цілих чисел. Програма надає користувачу можливість переглянути довжину такої підпослідовності та всі її елементи.

У другому завданні (Задача 2 Д) я створив програму для обчислення кількості можливих варіантів розташування будинків різних типів на вулиці. Використовуючи введену матрицю, програма враховує можливі комбінації розташування і виводить кількість можливих наборів літер заданої довжини.

В результаті лабораторної роботи я отримав цінний досвід у вирішенні задач за допомогою динамічного програмування та розробці програмного забезпечення для конкретних практичних завдань.