

**Міністерство освіти і науки України**  
**Національний університет «Запорізька політехніка»**

кафедра програмних засобів

**ЗВІТ**  
з лабораторної роботи № 1  
з дисципліни «Спортивне програмування» на тему:  
**«РЕКУРЕНТНІ ПОСЛІДОВНОСТІ»**

Виконав:

ст. гр. КНТ-113сп

Іван Щедровський

Прийняв:

ст. викл.

Сергій ЛЕОЩЕНКО

2024

## 1 Мета роботи:

Вивчити основні можливості та принципи роботи з мовою рекурентні послідовності та співвідношення.

## 2 Завдання до лабораторної роботи:

2.1 Задано масив  $M[1:N]$  натуральних чисел, упорядкований за не спаданням, тобто:  $M[1] \leftarrow M[2] \leftarrow \dots \leftarrow M[N]$ .

Знайти перше натуральне число, яке не представляється сумою ніяких елементів цього масиву, при цьому сума може складатися і з одного доданка, але кожен елемент масиву може входити в неї тільки один раз.

2.2 Звести число  $a$  в натуральну ступінь  $n$  за якомога меншу кількість множень.

2.3 Задані  $z$  та  $y$  – дві послідовності. Чи можна отримати послідовність  $z$  викреслюванням елементів з  $y$ .

## 3 Хід виконання самостійної роботи:

### 3.1 Виконання завдання 2.1

Код програми:

```
func Task1(numbers []int64) int64 {
    var result int64 = 1

    for i := 0; i < len(numbers) && numbers[i] <= result; i++ {
        result = result + numbers[i]
    }

    return result
}
```

Код тестів програми:

```
package lb1

import "testing"

func TestTask1(t *testing.T) {
    tests := []struct {
```

```

name      string
input     []int64
expectedResult int64
}{
    {
        name:      "Test 1",
        input:     []int64{1, 2, 4, 8, 16, 32},
        expectedResult: 64,
    },
    {
        name:      "Test 2",
        input:     []int64{1, 3, 6, 10, 15},
        expectedResult: 2,
    },
    {
        name:      "Test 3",
        input:     []int64{2, 3, 5, 7, 11, 13, 17},
        expectedResult: 1,
    },
    {
        name:      "Test 4",
        input:     []int64{2, 4, 6, 8, 10},
        expectedResult: 1,
    },
    {
        name:      "Test 5",
        input:     []int64{1, 5, 10, 20, 40},
        expectedResult: 2,
    },
    {
        name:      "Test 6",
        input:     []int64{2, 3, 7, 14, 29},
        expectedResult: 1,
    },
    {
        name:      "Test 7",
        input:     []int64{1, 2, 3, 5, 8, 13, 21},
        expectedResult: 54,
    },
    {
        name:      "Test 8",
        input:     []int64{1, 2, 3, 6, 10, 20, 40},
        expectedResult: 83,
    },
    {
        name:      "Test 9",
        input:     []int64{2, 4, 8, 16, 32, 64, 128, 256},
        expectedResult: 1,
    },
    {
        name:      "Test 10",
        input:     []int64{1, 3, 5, 9, 13, 25, 49},
        expectedResult: 2,
    },
}

for _, test := range tests {
    t.Run(test.name, func(t *testing.T) {
        result := Task1(test.input)
    })
}

```

```

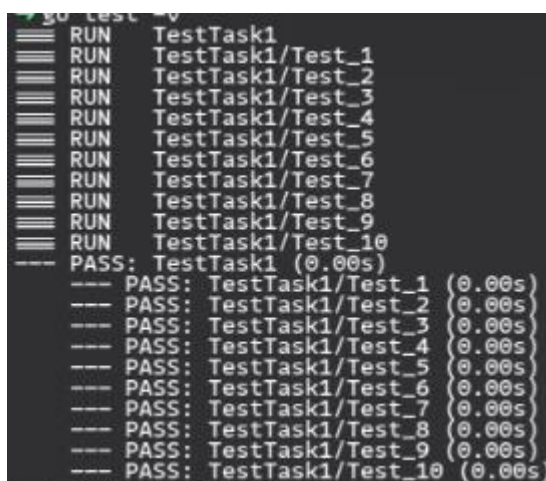
        if result != test.expectedResult {
            t.Errorf("Test %s failed: expected %d, got %d", test.name, test.expectedResult, result)
        }
    })
}
}

```

Опис логіки:

Якщо у нас є масив чисел, який упорядкований не за спаданням, тобто упорядкований від меншого до більшого, щоб зайти перше мінімальне натуральне число, яке не можна представити сумою членів цього масиву достатньо сумувати елементи з початку масива до моменту, коли наступний член масиву більший, ніж сума яка вийшла + 1.

На рисунку 1 наведено демонстрацію виконання завдання.



```

go test -v
=== RUN   TestTask1
=== RUN   TestTask1/Test_1
=== RUN   TestTask1/Test_2
=== RUN   TestTask1/Test_3
=== RUN   TestTask1/Test_4
=== RUN   TestTask1/Test_5
=== RUN   TestTask1/Test_6
=== RUN   TestTask1/Test_7
=== RUN   TestTask1/Test_8
=== RUN   TestTask1/Test_9
=== RUN   TestTask1/Test_10
--- PASS: TestTask1 (0.00s)
--- PASS: TestTask1/Test_1 (0.00s)
--- PASS: TestTask1/Test_2 (0.00s)
--- PASS: TestTask1/Test_3 (0.00s)
--- PASS: TestTask1/Test_4 (0.00s)
--- PASS: TestTask1/Test_5 (0.00s)
--- PASS: TestTask1/Test_6 (0.00s)
--- PASS: TestTask1/Test_7 (0.00s)
--- PASS: TestTask1/Test_8 (0.00s)
--- PASS: TestTask1/Test_9 (0.00s)
--- PASS: TestTask1/Test_10 (0.00s)

```

Рисунок 1 – Реалізація рішення для завдання 2.1

Алгоритм має складність в найгіршому випадку  $O(n)$ . В середньому –  $O(\log n)$

### 3.2 Виконання завдання 2.2

Код програми:

```

func Task8(number *big.Int, power *big.Int) *big.Int {

    if power.Sign() == 0 {
        return big.NewInt(1)
    }
}

```

```

    }

    if power.Sign() < 0 {
        return big.NewInt(-1)
    }

    if big.NewInt(0).Mod(power, big.NewInt(2)).Sign() == 1 {
        newPower := big.NewInt(0).Div(big.NewInt(0).Sub(power, big.NewInt(1)), big.NewInt(2))
        res := Task8(number, newPower)
        resultForReturn := big.NewInt(0).Mul(res, res)
        return big.NewInt(0).Mul(resultForReturn, number)
    }

    newPower := big.NewInt(0).Div(power, big.NewInt(2))
    res := Task8(number, newPower)

    return big.NewInt(0).Mul(res, res)
}

func task8Default(number *big.Int, power *big.Int) *big.Int {
    result := number

    i := big.NewInt(0)
    for ; i.Cmp(big.NewInt(0).Sub(power, big.NewInt(1))) < 0; i.Add(i, big.NewInt(1)) {
        result = big.NewInt(0).Mul(result, number)
    }

    return result
}

```

### Код тестів програми:

```

package lb1

import (
    "math/big"
    "testing"
)

func TestTask8(t *testing.T) {
    tests := []struct {
        name      string
        number     *big.Int
        power      *big.Int
        expectedResult int64
    }{
        {
            // On run with default = 0.72s
            // On run with Binary Exponentiation = 0.01s
            name: "Test 1",
            number: big.NewInt(100),
            power:  big.NewInt(100_000),
        },
        {
            // On run with default = 10.40s
            // On run with Binary Exponentiation = 0.03s
            name: "Test 2",
            number: big.NewInt(3),

```

```

        power: big.NewInt(1_000_000),
    },
    {
        // On run with default = bigger than 462.099s
        // On run with Binary Exponentiation = 0.44s
        name: "Test 2 with big value",
        number: big.NewInt(2),
        power: big.NewInt(10_000_000),
    },
}

for _, test := range tests {
    t.Run(test.name, func(t *testing.T) {
        result := Task8(test.number, test.power)

        expectedNumber := big.NewInt(0).Exp(test.number, test.power, big.NewInt(0))

        if result.Cmp(expectedNumber) != 0 {
            t.Errorf("Task8 %v failed", test.name)
        }
    })
}
}

```

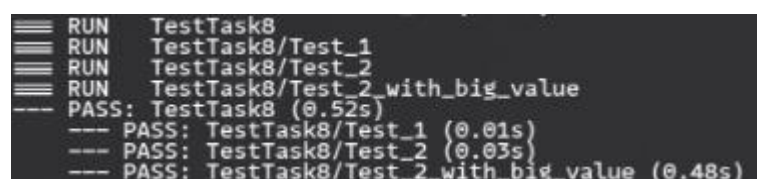
### Опис логіки:

Код цього застосунку має дві функції – Task8 та task8default. Task8 – це моє виконання, task8Default – те, як це робить звичайно

В цьому завданні я імплементував алгоритм Binary Exponentiation. Суть в тому, що щоб отримати  $2^{16}$  ступені зазвичай нам потрібно робити  $n - 1$  множень, тобто 15. Але, якщо ми зробимо  $((2^2)^2)^2$ . Тобто, з 15 множень у нас тепер 4.

Таким чином ми можемо знаходити дуже великі числа та дуже великі степені в  $O(\log n)$  часі

На рисунку 2 наведено демонстрацію виконання завдання з врахуванням Binary Exponentiation



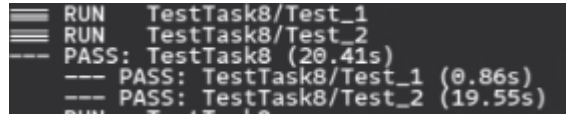
```

=== RUN   TestTask8
=== RUN   TestTask8/Test_1
=== RUN   TestTask8/Test_2
=== RUN   TestTask8/Test_2_with_big_value
--- PASS: TestTask8 (0.52s)
    --- PASS: TestTask8/Test_1 (0.01s)
    --- PASS: TestTask8/Test_2 (0.03s)
    --- PASS: TestTask8/Test_2_with_big_value (0.48s)

```

## Рисунок 2 – Реалізація рішення для завдання 2.2

На рисунку 3 наведено демонстрацію виконання алгоритму з стандартним  $O(n)$ . В виконанні не було показано третій тест, бо він займає більше 500 секунд



```
== RUN TestTask8/Test_1
== RUN TestTask8/Test_2
--- PASS: TestTask8 (20.41s)
--- PASS: TestTask8/Test_1 (0.86s)
--- PASS: TestTask8/Test_2 (19.55s)
PASS
```

Рисунок 3 – Приклад виконання з  $O(n)$

### 3.3 Виконання завдання 2.3

Код програми:

```
func Task9[T string | int](x []T, y []T) bool {
    if len(y) < len(x) {
        return false
    }

    yMap := make(map[T]bool)

    for _, element := range y {
        yMap[element] = true
    }

    for _, element := range x {
        if !yMap[element] {
            return false
        }
    }

    return true
}
```

Код тестів програми:

```
package lb1

import "testing"

func TestTask9(t *testing.T) {
    tests := []struct {
        name    string
        x        []int
    }
```

```

y      []int
expectedResult bool
}{
    {
        name:      "Check with deletionsj",
        x:          []int{1, 2, 3},
        y:          []int{3, 2, 1, 4, 5},
        expectedResult: true,
    },
    {
        name:      "Check with same values, but changed order",
        x:          []int{1, 2, 3, 4, 5},
        y:          []int{3, 2, 1, 4, 5},
        expectedResult: true,
    },
    {
        name:      "Check with 1 not valid number",
        x:          []int{1, 2, 3},
        y:          []int{3, 2, 4, 5},
        expectedResult: false,
    },
    {
        name:      "Check with different numbers",
        x:          []int{1, 2, 3},
        y:          []int{4, 5, 6},
        expectedResult: false,
    },
    {
        name:      "Big big values",
        x:          bigSequenceNotRandomGenerate(1_000_000),
        y:          bigSequenceNotRandomGenerate(1_000_000),
        expectedResult: true,
    },
    {
        name:      "Big big values where y < x",
        x:          bigSequenceNotRandomGenerate(100_000_000),
        y:          bigSequenceNotRandomGenerate(99_000_000),
        expectedResult: false,
    },
}

for _, test := range tests {
    t.Run(test.name, func(t *testing.T) {
        actualResult := Task9(test.x, test.y)

        if actualResult != test.expectedResult {
            t.Errorf("Task9 = %v, expected %v", actualResult, test.expectedResult)
        }
    })
}

}

func bigSequenceNotRandomGenerate(length int) []int {
    sequence := make([]int, length)
    for i := 0; i < length; i++ {
        sequence[i] = i
    }

    return sequence
}

```



}

Опис логіки:

У нас є дві послідовності, тобто два масиви

Щоб отримати послідовність  $X$  з послідовності  $Y$  використовуючи тільки викреслювання елементів нам потрібно, щоб всі елементи послідовності  $X$  були в послідовності  $Y$

Таким чином, результатом чи можна отримати  $X$  з  $Y$  викреслюванням буде перевірка на те, що всі  $X[1..n]$  є в  $Y$

= Оптимізації =

Спочатку ми можемо перевірити довжину масивів щоб не запускати перевірки

Щоб зроби складність  $O(n)$  (100 000 елементів за 0.009s, 1.000.000 за 0.182s), а не  $O(n^2)$  (1.637s при 100 000 елементів, 1000000 за 30+s) ми можемо згенерувати мапу по якій потім перевіряти чи валідний елемент, чи ні

На рисунку 4 наведено демонстрацію виконання завдання.

```
==== RUN TestTask9
==== RUN TestTask9/Check_with_deletionsj
==== RUN TestTask9/Check_with_same_values,_but_changed_order
==== RUN TestTask9/Check_with_1_not_valid_number
==== RUN TestTask9/Check_with_different_numbers
==== RUN TestTask9/Big_big_values
==== RUN TestTask9/Big_big_values_where_y<_x
---- PASS: TestTask9 (1.19s)
---- PASS: TestTask9/Check_with_deletionsj (0.00s)
---- PASS: TestTask9/Check_with_same_values,_but_changed_order (0.00s)
---- PASS: TestTask9/Check_with_1_not_valid_number (0.00s)
---- PASS: TestTask9/Check_with_different_numbers (0.00s)
---- PASS: TestTask9/Big_big_values (0.28s)
---- PASS: TestTask9/Big_big_values_where_y<_x (0.00s)
PASS
```

Рисунок 4 – Реалізація рішення для завдання 2.3

Як можна бачити, алгоритм працює швидко та має  $O(n)$  складність

#### **4 Висновки:**

Я вивчив основні можливості та принципи роботи з рекурентними послідовностями та співвідношеннями.