Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 3

з дисципліни «Спортивне програмування» на тему:

**«Рекурсивні алгоритми»**

Виконав:

ст. гр. КНТ-113сп Іван Щедровський

Прийняв:

ст.викл. Сергій ЛЕОЩЕНКО

2024

# Мета роботи:

Вивчити основні можливості та принципи роботи рекурсивних алгоритмів.

# Завдання до лабораторної роботи:

2.1 Є два відсортованих за не зростанням масиви *A*[1,*N*] і *B*[1, *M*]. Отримати  відсортований за не зростанням масив *C*[1, *N+M*], що складається з елементів масивів *A* і *B* ("злити" разом масиви *A* і *B*).

2.2 Маємо *N* каменів ваги *A*1, *A*2,..., *АN*. Необхідно розбити їх на дві купи таким  чином, щоб ваги Куп відрізнялися не більше ніж в 2 рази. Якщо цього зробити не можна,  то вказати це.

# Хід виконання самостійної роботи:

## Виконання завдання 2.1

Код програми:

package lb3

// Є два відсортованих за не зростанням масиви A[1,N] і B[1, M]. Отримати

// відсортований за не зростанням масив C[1, N+M], що складається з елементів масивів A

// і B ("злити" разом масиви A і B).

func Lab3Task6(a []int, b []int) []int {

c := make([]int, len(a)+len(b))

recursiveArrayMerge(a, b, c, 0, 0, 0)

return c

}

func recursiveArrayMerge(a []int, b []int, c []int, i int, j int, x int) {

if i >= len(a) && j >= len(b) {

return

}

if i >= len(a) {

c[x] = b[j]

recursiveArrayMerge(a, b, c, i, j+1, x+1)

return

}

if j >= len(b) {

c[x] = a[i]

recursiveArrayMerge(a, b, c, i+1, j, x+1)

return

}

if a[i] > b[j] {

c[x] = a[i]

recursiveArrayMerge(a, b, c, i+1, j, x+1)

return

}

c[x] = b[j]

recursiveArrayMerge(a, b, c, i, j+1, x+1)

}

Код тестів програми:

package lb3

import (

"reflect"

"testing"

)

func TestTask6Lab2(t \*testing.T) {

tests := []struct {

name string

a []int

b []int

c []int

}{

{

name: "Test 1",

a: []int{10, 5, 1},

b: []int{11, 9, 7, 5, 3, 2},

c: []int{11, 10, 9, 7, 5, 5, 3, 2, 1},

},

{

name: "Test 2",

a: []int{25, 8, 1},

b: []int{23, 11, 9, 9, 3, 2},

c: []int{25, 23, 11, 9, 9, 8, 3, 2, 1},

},

}

for \_, test := range tests {

t.Run(test.name, func(t \*testing.T) {

result := Lab3Task6(test.a, test.b)

if !reflect.DeepEqual(result, test.c) {

t.Errorf("Lab3Task6 %v failed", test.name)

}

})

}

}

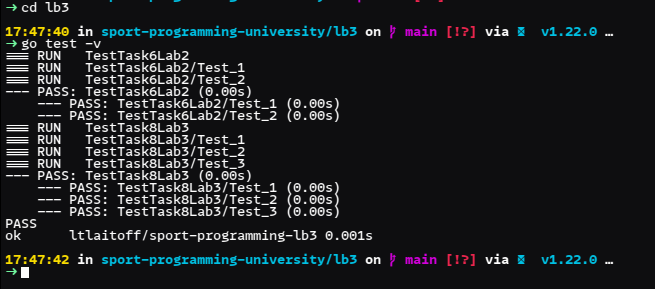


Рисунок 1 – Реалізація рішення для завдання 2.1

## Виконання завдання 2.2

Код програми:

package lb3

// Lab3Task8

// Маємо N каменів ваги A1, A2,..., АN. Необхідно розбити їх на дві купи таким

// чином, щоб ваги Куп відрізнялися не більше ніж в 2 рази. Якщо цього зробити не можна,

// то вказати це.

type Result struct {

weight int

stones []int

}

func Lab3Task8(stones []int) (stones1 []int, stones2 []int, err bool) {

totalWeight := 0

for \_, stone := range stones {

totalWeight += stone

}

target := totalWeight / 2

var innerLab3Task8 func(index int, weint int, selectedStones []int) \*Result

innerLab3Task8 = func(index int, weight int, selectedStones []int) \*Result {

if weight > target {

return nil

}

if index == len(stones) {

return &Result{weight: weight, stones: selectedStones}

}

includeStone := append(selectedStones, stones[index])

include := innerLab3Task8(index+1, weight+stones[index], includeStone)

exclude := innerLab3Task8(index+1, weight, selectedStones)

if include != nil && (exclude == nil || include.weight > exclude.weight) {

return include

}

return exclude

}

result := innerLab3Task8(0, 0, []int{})

if result == nil {

return nil, nil, true

}

weight1 := result.weight

stones1 = result.stones

stones1Map := make(map[int]int)

for \_, stone := range stones1 {

stones1Map[stone]++

}

for \_, stone := range stones {

if stones1Map[stone] > 0 {

stones1Map[stone]--

} else {

stones2 = append(stones2, stone)

}

}

weight2 := 0

for \_, stone := range stones2 {

weight2 += stone

}

if weight1 <= 2\*weight2 && weight2 <= 2\*weight1 {

return stones1, stones2, false

}

return nil, nil, true

}

Код тестів програми:

package lb3

import (

"reflect"

"testing"

)

func TestTask8Lab3(t \*testing.T) {

tests := []struct {

name string

stones []int

stones1 []int

stones2 []int

err bool

}{

{

name: "Test 1",

stones: []int{10, 5, 5},

stones1: []int{5, 5},

stones2: []int{10},

err: false,

},

{

name: "Test 2",

stones: []int{10, 5, 5, 3, 1, 4},

stones1: []int{5, 5, 4},

stones2: []int{10, 3, 1},

err: false,

},

{

name: "Test 3",

stones: []int{25, 5, 1, 4},

stones1: []int{},

stones2: []int{},

err: true,

},

}

for \_, test := range tests {

t.Run(test.name, func(t \*testing.T) {

stones1, stones2, err := Lab3Task8(test.stones)

if !((len(stones1) == 0 && len(stones1) == 0) || reflect.DeepEqual(stones1, test.stones1)) ||

!((len(stones2) == 0 && len(stones2) == 0) || reflect.DeepEqual(stones2, test.stones2)) ||

err != test.err {

t.Errorf("Lab3Task6 %v failed", test.name)

}

})

}

}

Виконання показано на рисунку 2.1

# Висновки:

Я вивчив основні можливості та принципи роботи рекурсивних алгоритмів