Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Курский государственный университет»

Кафедра программного обеспечения и администрирования информационных систем

Направление подготовки математическое обеспечение и администрирование информационных систем

Форма обучения очная

**Отчет**

**по лабораторной работе №12**

«РЕШЕНИЕ КОМБИНАТОРНЫХ ЗАДАЧ ЭВРИСТИЧЕСКИМИ АЛГОРИТМАМИ»

Выполнил:

студент группы 213 Файтельсон А.А.

Проверил:

ассистент кафедры ПОиАИС Овсянников А.В.

Курск, 2025

**Цель работы:** Знакомство с общей постановкой комбинаторных задач, освоение техники решения таких задач эвристическими алгоритмами и приемов анализа решения.

**Словесная постановка задачи:**

Задача о куче камней. Имеется N камней, известны их веса Pi (i=1...N), задано количество куч M. Требуется разложить камни на M куч так, чтобы минимизировать вес самой тяжелой кучи. (Вариант формулировки: даны N программ с длительностями Pi и M процессоров. Требуется распределить программы так, чтобы раньше закончить выполнение).

**Вариант задания:** 6

**Таблица результатов программы:**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Кол-во камней | Кол-во куч | Время исполнения |
| 1 | 5 | 3 | 8 мс |
| 2 | 5 | 2 | 5 мс |
| 3 | 5 | 5 | 5 мс |
| 4 | 5 | 2 | 3 мс |
| 5 | 1000 | 10 | 64 мс |
| 6 | 10000 | 100 | 78 мс |
| 7 | 100000 | 1000 | 6 с |

**Вывод:** В сравнении с результатом решения методом перебора из 11 лабораторной, эвристический алгоритм работает быстрее, также алгоритм, относящийся к классу эвристических, может работать при количестве камней больше 1000.

**Листинг программы**

package main

import (

"fmt"

"math/rand"

"sort"

"time"

)

// generateLargeDataset генерирует случайный массив весов длины n,

// где каждый вес находится в диапазоне [1, maxWeight].

func generateLargeDataset(n int, maxWeight int) []int {

rand.Seed(time.Now().UnixNano()) // Инициализация генератора случайных чисел

weights := make([]int, n)

for i := 0; i < n; i++ {

weights[i] = rand.Intn(maxWeight) + 1 // Генерация веса от 1 до maxWeight

}

return weights

}

func main() {

// Примеры тестов

tests := []struct {

weights []int

M int

}{

{[]int{1, 2, 3, 4, 5}, 3}, // Простой случай

{[]int{7, 2, 5, 10, 8}, 2}, // Сложный случай

{[]int{1, 1, 1, 1, 1}, 5}, // Все камни равны

{[]int{30, 20, 10, 50, 40}, 2}, // Разные веса

{generateLargeDataset(1000, 1000), 10}, // Большие данные: 1000 камней, 10 куч

{generateLargeDataset(10000, 10000), 100}, // Очень большие данные: 10000 камней, 100 куч

{generateLargeDataset(100000, 100000), 1000}, // Максимальные данные: 100000 камней, 1000 куч

}

for i, test := range tests {

fmt.Printf("Test %d: Weights length = %d, M = %d\n", i+1, len(test.weights), test.M)

sort.Ints(test.weights) // Сортируем веса для удобства

start := time.Now() // Начало замера времени

minimizeMaxWeight(test.weights, test.M)

elapsed := time.Since(start) // Конец замера времени

fmt.Printf("Execution time: %v\n\n", elapsed)

}

}

func minimizeMaxWeight(weights []int, M int) int {

// N := 5

sort.Slice(weights, func(i, j int) bool {

return weights[i] > weights[j]

})

heaps := make([][]int, M)

sums := make([]int, M)

for i := range heaps {

heaps[i] = []int{}

}

// Жадное распределение камней

for \_, w := range weights {

minSum := sums[0]

minIdx := 0

for i := 1; i < M; i++ {

if sums[i] < minSum {

minSum = sums[i]

minIdx = i

}

}

heaps[minIdx] = append(heaps[minIdx], w)

sums[minIdx] += w

}

maxSum := 0

for \_, s := range sums {

if s > maxSum {

maxSum = s

}

}

// Локальный поиск для улучшения распределения

improved := true

for improved {

improved = false

for i := 0; i < M; i++ {

for j := i + 1; j < M; j++ {

for a := 0; a < len(heaps[i]); a++ {

for b := 0; b < len(heaps[j]); b++ {

newSumI := sums[i] - heaps[i][a] + heaps[j][b]

newSumJ := sums[j] - heaps[j][b] + heaps[i][a]

newMax := max(newSumI, newSumJ, maxSum)

if newMax < maxSum {

heaps[i][a], heaps[j][b] = heaps[j][b], heaps[i][a]

sums[i], sums[j] = newSumI, newSumJ

maxSum = newMax

improved = true

i, j, a, b = M, M, len(heaps[i]), len(heaps[j])

}

}

}

}

}

}

fmt.Println("Минимально возможная максимальная куча:", maxSum)

return maxSum

}

func max(a, b, c int) int {

maxVal := a

if b > maxVal {

maxVal = b

}

if c > maxVal {

maxVal = c

}

return maxVal

}