Міністерство освіти та науки України

Харківський національний університет радіоелектроніки

Кафедра Інформатики

Звіт

з лабораторної роботи №1

з дисципліни «Теорія програмування»

Виконав: Перевірив:

Ст.гр. ІТІНФ-20-1 Сінельнікова. Т. Ф.

Самченко С. О.

Харків 2023

**Мета:** вивчення концепції рефакторингу як процесу внесення змін до коду програми з метою поліпшення його якості, підтримуваності, зрозумілості та ефективності, без впливу на зовнішню поведінку програми.

**Завдання:** Виконати аналіз коду програми та переписати цей додаток на мові програмування Goland, здійснити рефакторинг коду з метою покращення його якості та ефективності..

**Початковий код на JavaScript:**

function Graph(vertices) {

  const graph = new Map();

  const matrix = new Array(vertices).fill(0).map(() => new Array(vertices).fill(0));

  function addEdge(v1, v2) {

    graph.set(v1, [...(graph.get(v1) || []), v2]);

    matrix[v1][v2] = 1;

  }

  function printEdges() {

    for (const [v1, v2] of graph.entries()) {

      for (const v of v2) {

        console.log(`Из вершины (${v1}) к вершине (${v})`);

      }

    }

  }

  function topologicalSortUtil(v, visited, stack) {

    visited[v] = true;

    for (const i of graph.get(v) || []) {

      if (!visited[i]) {

        topologicalSortUtil(i, visited, stack);

      }

    }

    stack.unshift(v);

  }

  function topologicalSort() {

    const visited = new Array(vertices).fill(false);

    const stack = [];

    for (let i = 0; i < vertices; i++) {

      if (!visited[i]) {

        topologicalSortUtil(i, visited, stack);

      }

    }

    return stack;

  }

  function adjencyMatrix() {

    for (const v1 in matrix) {

      console.log();

      for (const v2 in matrix[v1]) {

        process.stdout.write(`${matrix[v1][v2]} `);

      }

    }

    console.log();

  }

  return {

    addEdge,

    printEdges,

    topologicalSort,

    adjencyMatrix

  };

}

const edgeCount = 6;

const g = Graph(edgeCount);

g.addEdge(0, 1);

g.addEdge(0, 3);

g.addEdge(1, 2);

g.addEdge(1, 4);

g.addEdge(2, 5);

g.addEdge(3, 4);

g.addEdge(4, 5);

const result = g.topologicalSort();

console.log(`\nКоличество вершин: ${edgeCount}\n`);

g.printEdges();

console.log(`\nТопологическая сортировка графа:\n${result}\n`);

console.log("Матрица смежности:");

g.adjencyMatrix();

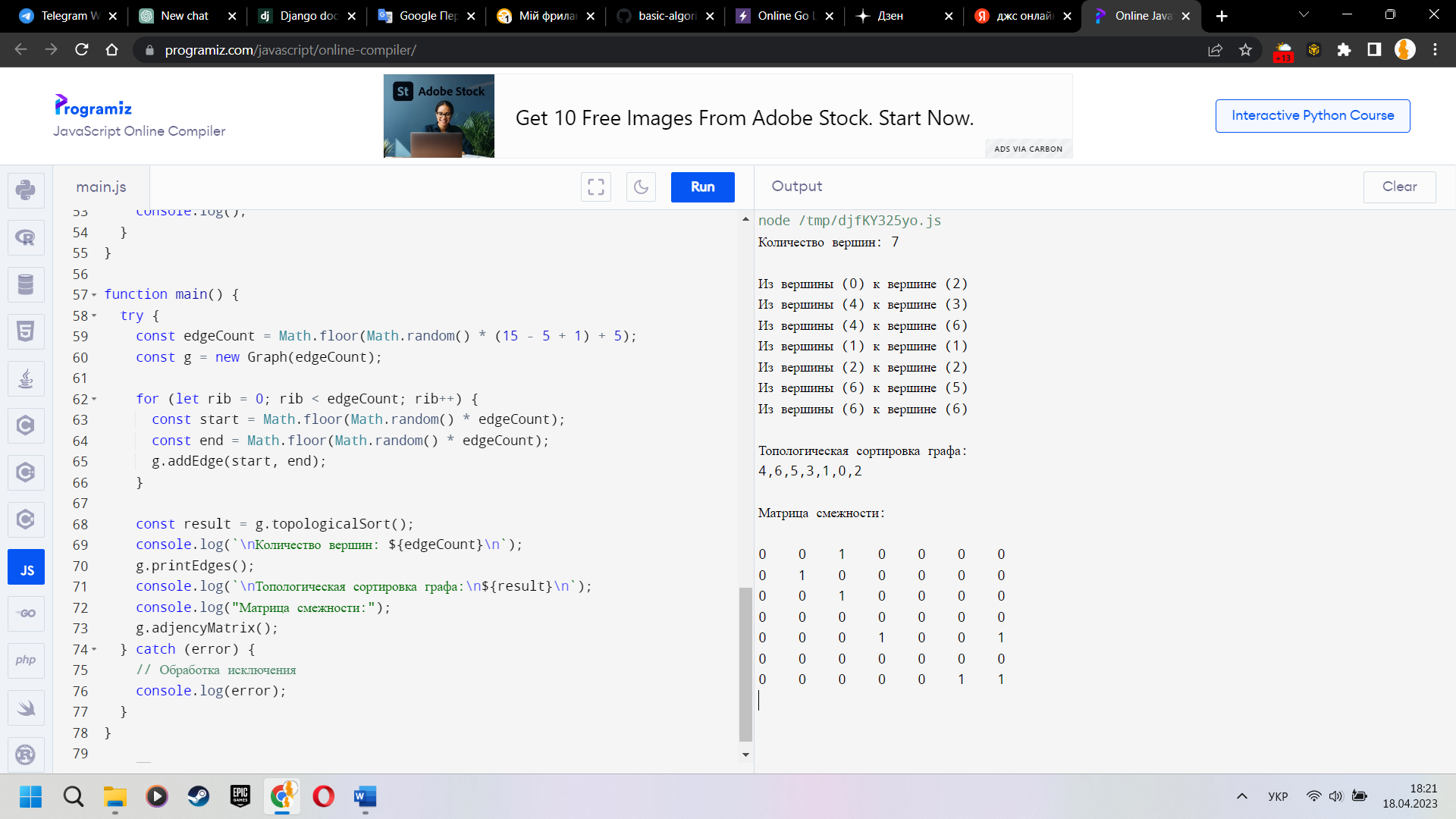


Рис. 1 – результат роботи програми на JavaScript

**Оптимізований код на Goland:**

package main

import (

  "fmt"

  "math/rand"

)

type Graph struct {

  graph  map[int][]int // Словарь, содержащий список смежности

  V      int           // Количество вершин

  matrix [][]int       // Матрица смежности

}

func NewGraph(vertices int) \*Graph {

  g := &Graph{

    graph:  make(map[int][]int),

    V:      vertices,

    matrix: make([][]int, vertices),

  }

  for i := 0; i < vertices; i++ {

    g.matrix[i] = make([]int, vertices)

  }

  return g

}

func (g \*Graph) addEdge(v1, v2 int) { // Функция добавления ребра к графу

  g.graph[v1] = append(g.graph[v1], v2)

  g.matrix[v1][v2] = 1

}

func (g \*Graph) printEdges() { // Функция для печати ребер

  for v1, v2 := range g.graph {

    for \_, v := range v2 {

      fmt.Printf("Из вершины (%d) к вершине (%d)\n", v1, v)

    }

  }

}

func (g \*Graph) topologicalSortUtil(v int, visited []bool, stack \*[]int) { // Рекурсивная функция, используемая topologicalSort

  visited[v] = true // Пометить текущую вершину как посещенную.

  for \_, i := range g.graph[v] { // Повторить для всех вершин, смежных с этой вершиной

    if !visited[i] {

      g.topologicalSortUtil(i, visited, stack)

    }

  }

  \*stack = append([]int{v}, \*stack...) // Помещаем текущую вершину в стек, в котором хранится результат

}

func (g \*Graph) topologicalSort() []int { // Функция топологической сортировки. Он использует рекурсивную topologicalSortUtil()

  visited := make([]bool, g.V) // Отметить все вершины как не посещенные

  stack := []int{}

  for i := 0; i < g.V; i++ { // Вызов рекурсивной вспомогательной функции для сохранения топологического порядка,

    if !visited[i] { // начиная со всех вершин по очереди

      g.topologicalSortUtil(i, visited, &stack)

    }

  }

  return stack

}

func (g \*Graph) adjencyMatrix() { // Матрица смежностей графа

  for \_, v1 := range g.matrix {

    fmt.Println()

    for \_, v2 := range v1 {

      fmt.Printf("%d    ", v2)

    }

  }

  fmt.Println()

}

func main() {

  edgeCount := rand.Intn(11) + 5

  g := NewGraph(edgeCount)

  for rib := 0; rib < edgeCount; rib++ {

    start := rand.Intn(edgeCount)

    end := rand.Intn(edgeCount)

    g.AddEdge(start, end)

  }

  result := g.TopologicalSort()

  fmt.Printf("\nКоличество вершин: %d\n", edgeCount)

  g.PrintEdges()

  fmt.Printf("\nТопологическая сортировка графа:\n%v\n", result)

  fmt.Println("Матрица смежности:")

  g.AdjacencyMatrix()

}

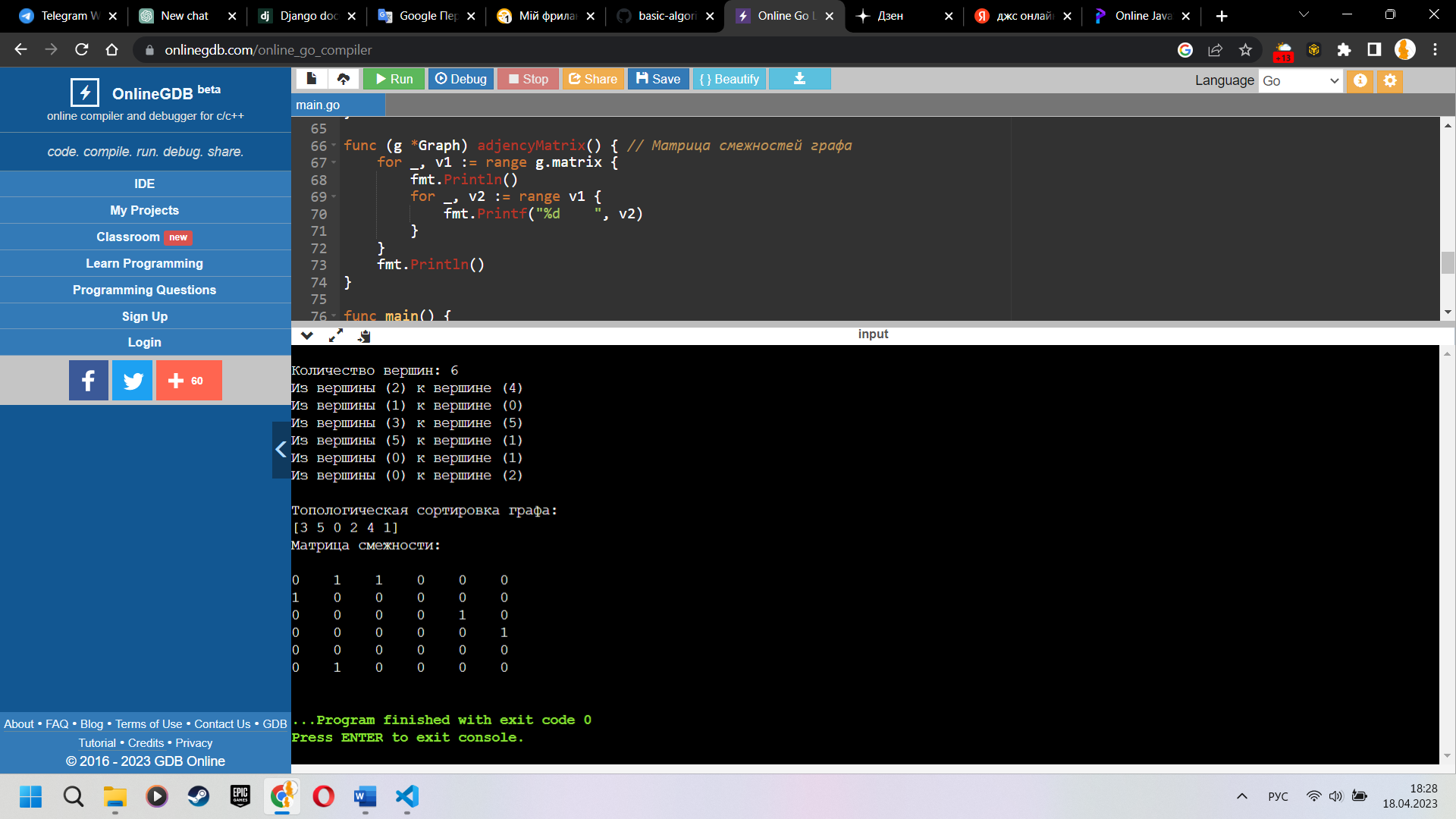


Рис. 2 – результат роботи програми на Goland

**Висновок:** Під час виконання даної роботи ми ознайомилися з концепцією рефакторингу, який є процесом зміни коду програми з метою покращення його якості, підтримуваності, розуміння та ефективності, без зміни зовнішньої поведінки. Ми також набули навичок оптимізації коду для поліпшення сприйняття його.

**Мета:** ознайомлення з концепцією рефакторингу як процесу зміни коду програми з метою покращення його якості, підтримуваності, розуміння та ефективності, без зміни його зовнішньої поведінки.

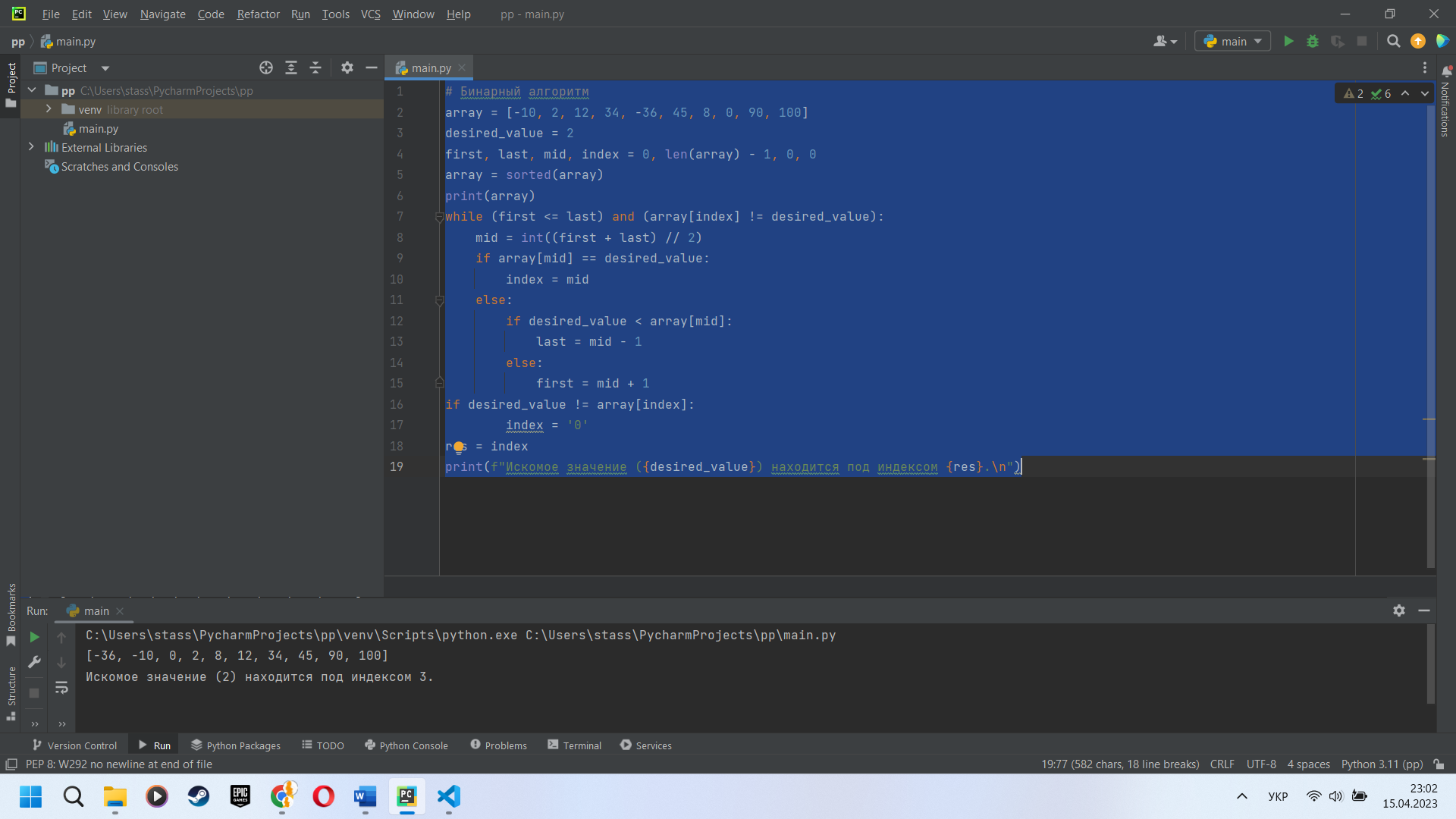
**Завдання:** Проанализувати код программи та переписати даний додаток на мові програмування Goland, провести рефакторінг данного коду.

**Хід роботи:**

1. *Початковий код на Python:*

# Бинарный алгоритм  
array = [-10, 2, 12, 34, -36, 45, 8, 0, 90, 100]  
desired\_value = 2  
first, last, mid, index = 0, len(array) - 1, 0, 0  
array = sorted(array)  
print(array)  
while (first <= last) and (array[index] != desired\_value):  
 mid = int((first + last) // 2)  
 if array[mid] == desired\_value:  
 index = mid  
 else:  
 if desired\_value < array[mid]:  
 last = mid - 1  
 else:  
 first = mid + 1  
if desired\_value != array[index]:  
 index = '0'  
res = index  
print(f"Искомое значение ({desired\_value}) находится под индексом {res}.\n")

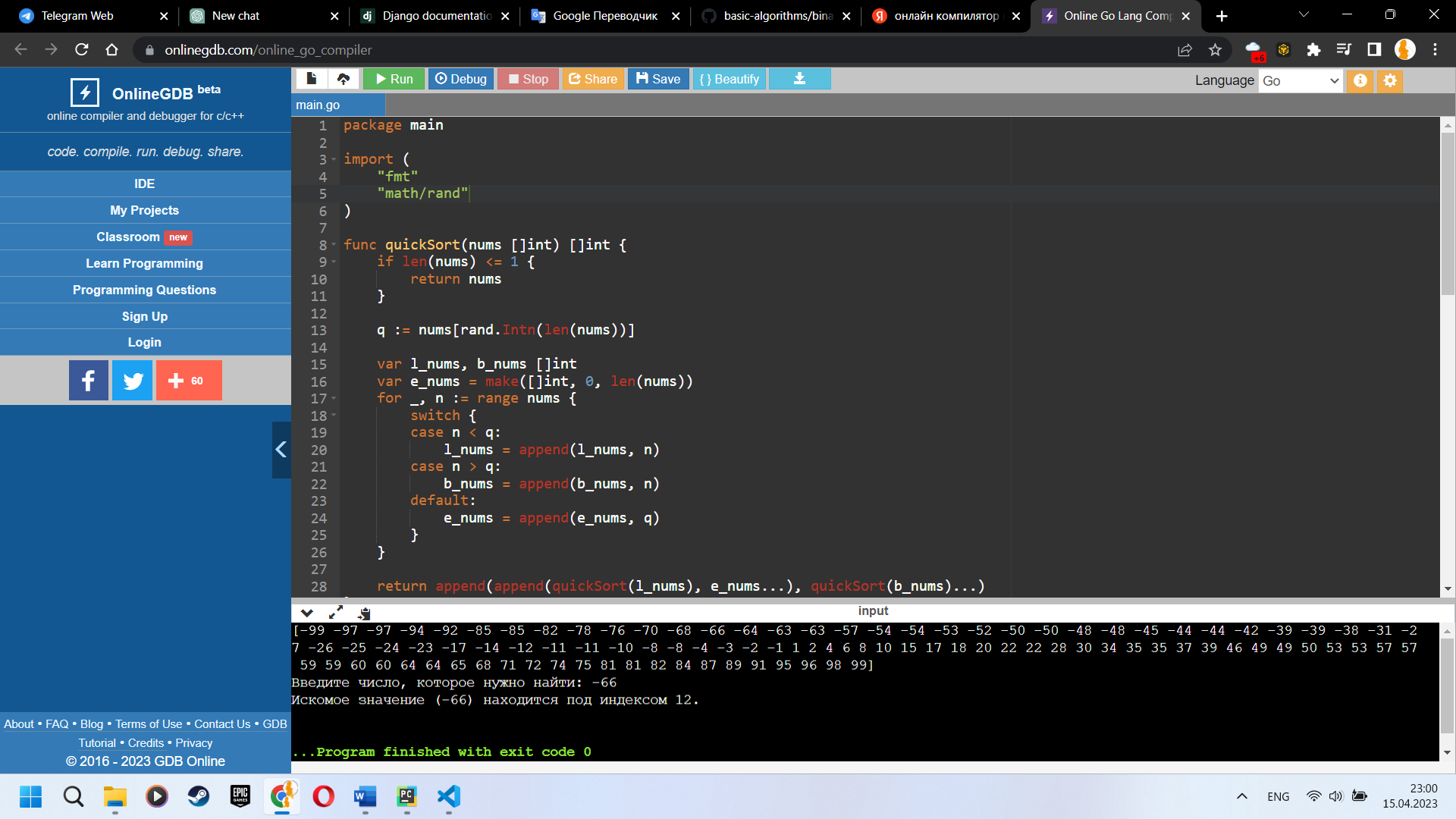
Результат роботи коду:



1. *Оптимізований код на Goland:*

package main  
  
import (  
 "fmt"  
 "math/rand"  
)  
  
func quickSort(nums []int) []int {  
 if len(nums) <= 1 {  
 return nums  
 }  
  
 q := nums[rand.Intn(len(nums))] // Выбираем случайный опорный элемент q.  
  
 var l\_nums, b\_nums []int  
 var e\_nums = make([]int, 0, len(nums))  
 for \_, n := range nums { // Разделяем массив на три подмассива: меньше, равные и больше q.  
 switch {  
 case n < q:  
 l\_nums = append(l\_nums, n)  
 case n > q:  
 b\_nums = append(b\_nums, n)  
 default:  
 e\_nums = append(e\_nums, q)  
 }  
 }  
  
 return append(append(quickSort(l\_nums), e\_nums...), quickSort(b\_nums)...) // Рекурсивно вызываем quickSort на меньшем и большем подмассивах, а также объединяем их с подмассивом равных элементов.  
}  
  
func binaryAlgorithm(array []int, desiredValue int) int {  
 first, last, mid, index := 0, len(array)-1, 0, 0  
  
 for first <= last && array[index] != desiredValue { // Выполняем бинарный поиск, пока не найдем desiredValue или не исчерпаем все возможные варианты.  
 mid = (first + last) / 2 // Вычисляем середину текущего интервала.  
 if array[mid] == desiredValue { // Если серединный элемент равен desiredValue, обновляем индекс.  
 index = mid  
 } else { // Иначе, сужаем интервал поиска на основе значения серединного элемента и desiredValue.  
 if desiredValue < array[mid] {  
 last = mid - 1  
 } else {  
 first = mid + 1  
 }  
 }  
 }  
  
 if desiredValue != array[index] { // Если desiredValue не найдено, возвращаем -1.  
 return -1  
 }  
  
 return index // Иначе, возвращаем индекс найденного элемента.  
}  
  
func main() {  
 var array = make([]int, 100)  
 for i := 0; i < len(array); i++ {  
 array[i] = rand.Intn(201) - 100 // Заполняем массив случайными числами от -100 до 100.  
 }  
 array = quickSort(array) // Сортируем массив с использованием quickSort.  
 fmt.Println(array)  
  
 var desiredValue int  
 fmt.Print("Введите число, которое нужно найти: ")  
 \_, err := fmt.Scanf("%d", &desiredValue) // Вводим число, которое нужно найти.  
 if err != nil {  
 fmt.Println("\nНеверные данные !!!\n")  
 return  
 }  
  
 res := binaryAlgorithm(array, desiredValue) // Ищем число с использованием binaryAlgorithm.  
  
 if res == -1 {  
 fmt.Printf("\n%d не является элементом данного массива.\n", desiredValue)  
 } else {  
 fmt.Printf("Искомое значение (%d) находится под индексом %d.\n", desiredValue, res)  
 }  
}

Результат роботи коду:



**Висновок:** у рамках доної роботи відбулося ознайомлення з концепцією рефакторингу як процесу зміни коду програми з метою покращення його якості, підтримуваності, розуміння та ефективності, без зміни його зовнішньої поведінки, та були здобуті навички оптимізації коду для покращення його сприйняття.