**Міністерство освіти і науки України  
Національний технічний університет України  
«Київський політехнічний інститут імені Ігоря Сікорського»  
Факультет інформатики та обчислювальної техніки  
Кафедра обчислювальної техніки**

**Лабораторна робота №3.3**

з дисципліни  
«Інтелектуальні вбудовані системи»

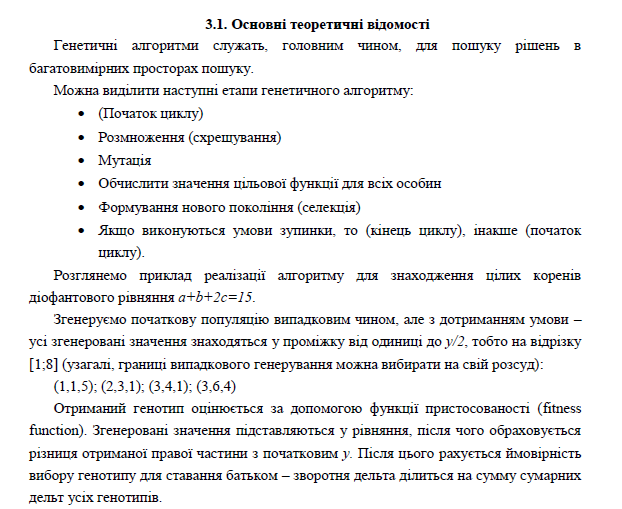
на тему  
«ДОСЛІДЖЕННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ»

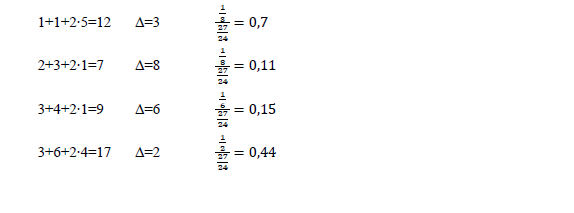
Виконав: Перевірив:

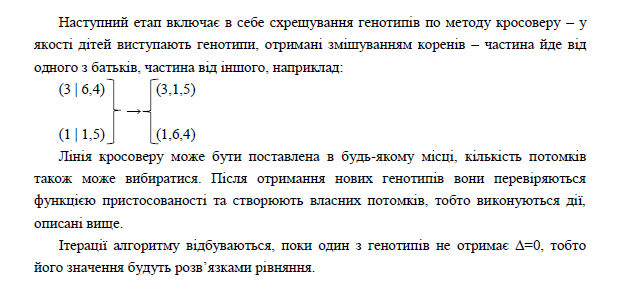
студент групи ІП-84 викладач  
Кучін Владислав Дмитрович Регіда Павло Геннадійович  
номер залікової книжки: 8415

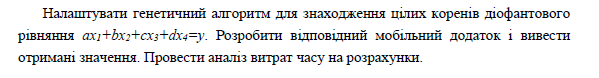
Київ 2021

**Основні теоретичні відомості**

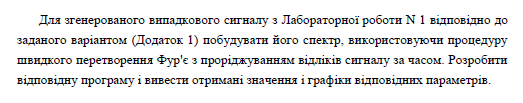




****

****

**Завдання на лабораторну роботу**



**Лістинг програми**

package ua.kpi.comsys.geneticalgorithm

import androidx.appcompat.app.AppCompatActivity

import android.os.Bundle

import android.widget.Button

import android.widget.TextView

import com.google.android.material.textfield.TextInputEditText

import kotlin.math.abs

class MainActivity : AppCompatActivity() {

private var aView: TextInputEditText? = null

private var bView: TextInputEditText? = null

private var cView: TextInputEditText? = null

private var dView: TextInputEditText? = null

private var yView: TextInputEditText? = null

private var iterationsView: TextInputEditText? = null

private var x1View: TextView? = null

private var x2View: TextView? = null

private var x3View: TextView? = null

private var x4View: TextView? = null

private var deviationView: TextView? = null

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

aView = findViewById(R.id.a)

bView = findViewById(R.id.b)

cView = findViewById(R.id.c)

dView = findViewById(R.id.d)

yView = findViewById(R.id.y)

x1View = findViewById(R.id.x1)

x2View = findViewById(R.id.x2)

x3View = findViewById(R.id.x3)

x4View = findViewById(R.id.x4)

iterationsView = findViewById(R.id.iterations)

deviationView = findViewById(R.id.deviation)

setClickListener()

}

private fun setClickListener() {

findViewById<Button>(R.id.run\_button).setOnClickListener {

val a = validateInput(aView)

if (a == Int.MAX\_VALUE) return@setOnClickListener

val b = validateInput(bView)

if (b == Int.MAX\_VALUE) return@setOnClickListener

val c = validateInput(cView)

if (c == Int.MAX\_VALUE) return@setOnClickListener

val d = validateInput(dView)

if (d == Int.MAX\_VALUE) return@setOnClickListener

val y = validateInput(yView)

if (y == Int.MAX\_VALUE) return@setOnClickListener

val iterations = validateInput(iterationsView)

if (iterations == Int.MAX\_VALUE) return@setOnClickListener

val solution = calculate(a, b, c, d, y, iterations)

x1View?.text = "x1=${solution[0]}"

x2View?.text = "x2=${solution[1]}"

x3View?.text = "x3=${solution[2]}"

x4View?.text = "x4=${solution[3]}"

deviationView?.text = (y - solution[0] \* a - solution[1] \* b - solution[2] \* c - solution[3] \* d).toString()

}

}

private fun validateInput(input: TextInputEditText?): Int {

val string = input?.text.toString()

if (string.isNotEmpty()) {

val number = string.toInt()

if (number != 0) {

return number

}

}

input?.error = "Введіть значення"

return Int.MAX\_VALUE

}

private fun calculate(a: Int, b: Int, c: Int, d: Int, y: Int, iterations: Int): IntArray {

var population = generateStartPopulation(y, 100)

var populationFitness = calculatePopulationFitness(population, intArrayOf(a, b, c, d), y)

repeat(iterations) {

val solutionIndex = checkZeroFitness(populationFitness)

if (solutionIndex != -1) {

return population[solutionIndex]

}

population = crossover(population, populationFitness)

mutate(population, 0.01f)

populationFitness = calculatePopulationFitness(population, intArrayOf(a, b, c, d), y)

}

val bestIndex = findBestFitness(populationFitness)

return population[bestIndex]

}

private fun generateStartPopulation(y: Int, count: Int): Array<IntArray> {

val population = Array(count) { IntArray(4) }

val max = y / 2

for (i in 0 until count) {

val x1 = (1..max).random()

val x2 = (1..max).random()

val x3 = (1..max).random()

val x4 = (1..max).random()

population[i] = intArrayOf(x1, x2, x3, x4)

}

return population

}

private fun calculatePopulationFitness(population: Array<IntArray>, coefs: IntArray, y: Int): IntArray {

val result = IntArray(population.size)

for (i in population.indices) {

result[i] = calculateFitness(coefs, y, population[i])

}

return result

}

private fun calculateFitness(coefs: IntArray, y: Int, solution: IntArray): Int {

var result = y

for (i in coefs.indices) {

result -= coefs[i] \* solution[i]

}

return abs(result)

}

private fun checkZeroFitness(populationFitness: IntArray): Int {

return populationFitness.indexOfFirst {it == 0}

}

private fun crossover(population: Array<IntArray>, populationFitness: IntArray): Array<IntArray> {

val probabilities = calculateProbabilities(populationFitness)

val parents = chooseParents(population, probabilities)

val pairs = formPairs(parents)

return reproduce(pairs)

}

private fun calculateProbabilities(populationFitness: IntArray): FloatArray {

var reversedDeltaSum = 0f

val reversedDeltas = FloatArray(populationFitness.size)

for (i in populationFitness.indices) {

val reversedDelta = 1f / populationFitness[i]

reversedDeltaSum += reversedDelta

reversedDeltas[i] = reversedDelta

}

val result = FloatArray(populationFitness.size)

for (i in populationFitness.indices) {

result[i] = reversedDeltas[i] / reversedDeltaSum

}

return result

}

private fun chooseParents(population: Array<IntArray>, probabilities: FloatArray): Array<IntArray> {

val parents = Array(population.size) { IntArray(4) }

val ranges = splitByProbabilities(probabilities)

for (i in population.indices) {

val random = Math.random()

val parentIndex = findRangeIndex(ranges, random)

val parent = population[parentIndex]

parents[i] = parent

}

return parents

}

private fun splitByProbabilities(probabilities: FloatArray): FloatArray {

val result = FloatArray(probabilities.size)

var sum = 0f

for (i in probabilities.indices) {

result[i] = sum

sum += probabilities[i]

}

return result

}

private fun findRangeIndex(ranges: FloatArray, random: Double): Int {

var startIndex = 0

var endIndex = ranges.size - 1

var index = ranges.size / 2

var rangeStart = ranges[index]

var rangeEnd = ranges[index + 1]

while (true) {

if (random > rangeEnd) {

startIndex = index + 1

} else if (random < rangeStart) {

endIndex = index - 1

} else {

break

}

index = startIndex + (endIndex - startIndex) / 2

rangeStart = ranges[index]

rangeEnd = if (index != ranges.lastIndex) ranges[index + 1] else 1f

}

return index

}

private fun formPairs(parents: Array<IntArray>): Array<Pair<IntArray, IntArray>> {

val result = Array(parents.size / 2) { Pair(IntArray(4), IntArray(4)) }

for (i in result.indices) {

val random1 = (parents.indices).random()

val random2 = (parents.indices).random()

val pair = Pair(parents[random1], parents[random2])

result[i] = pair

}

return result

}

private fun reproduce(pairs: Array<Pair<IntArray, IntArray>>): Array<IntArray> {

val result = Array(pairs.size \* 2) { IntArray(4) }

for (i in pairs.indices) {

val pair = pairs[i]

val indices = (0..3).toList().toIntArray()

val count = (1..3).random()

val firstParentGens = IntArray(count)

repeat(count) {

val random = (0 until indices.size - it).random()

firstParentGens[it] = indices[random]

indices[random] = indices[indices.lastIndex - it]

}

val child1 = pair.first.clone()

val child2 = pair.second.clone()

for (j in firstParentGens) {

child2[j] = pair.first[j]

child1[j] = pair.second[j]

}

result[2 \* i] = child1

result[2 \* i + 1] = child2

}

return result

}

private fun mutate(population: Array<IntArray>, mutationProbability: Float) {

for (i in population.indices) {

val random = Math.random()

if (random < mutationProbability) {

val chromosome = population[i]

val randomGenIndex = chromosome.indices.random()

val randomDelta = intArrayOf(-2, -1, 1, 2).random()

chromosome[randomGenIndex] += randomDelta

}

}

}

private fun findBestFitness(populationFitness: IntArray): Int {

var index = 0

var value = populationFitness[0]

for (i in 1 until populationFitness.size) {

if (populationFitness[i] < value) {

index = i

value = populationFitness[i]

}

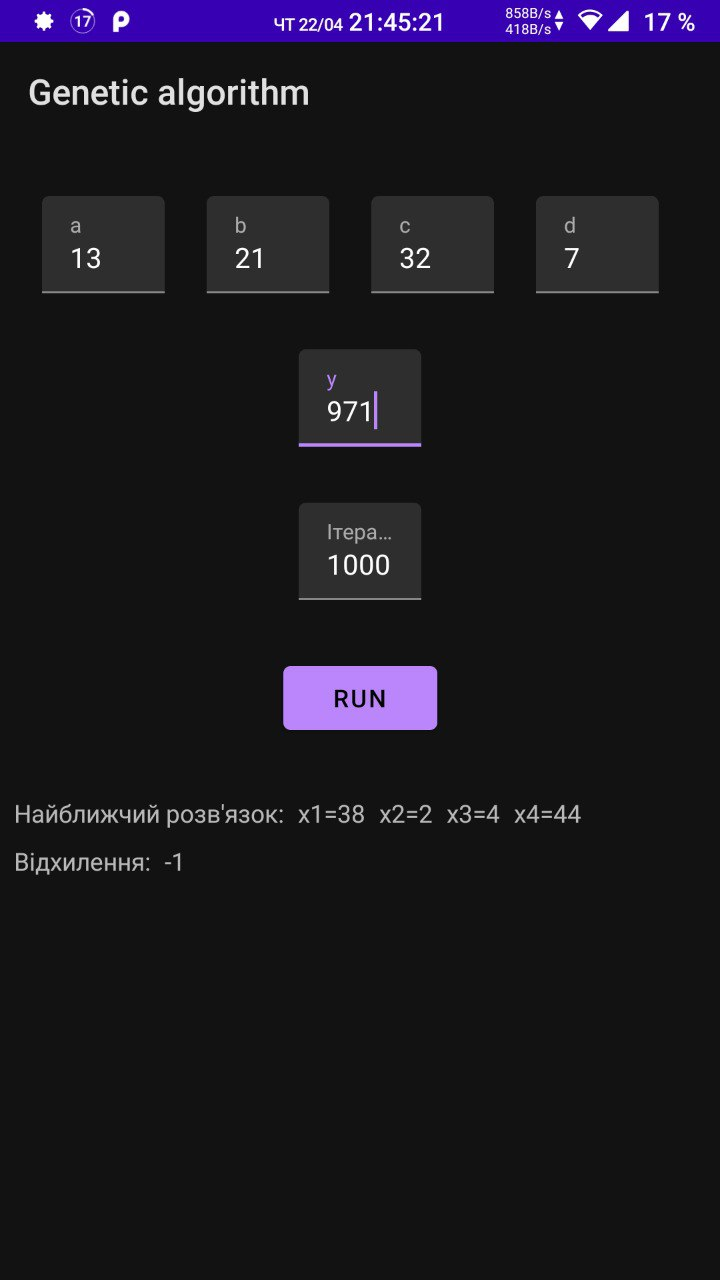
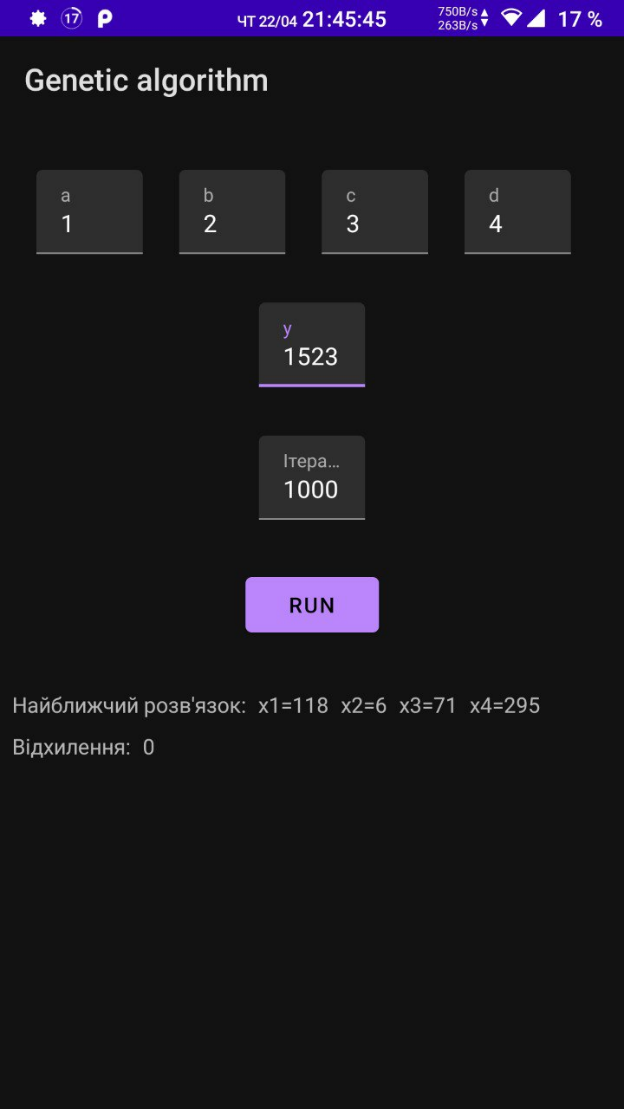
}

return index

}

}

**Результати роботи програми**



**Висновки**

Під час даної лабораторної роботи ми вивчили, як за допомогою генетичних алгоритмів можна знаходити наближені розв’язки діофантових рівнянь, поступово збільшуючи їх точність, що вкрай важливо для надійної роботи систем реального часу.