НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКНАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №3.3
“Дослідження генетичного алгоритму”

Виконав:
студент групи ІП-84
ІП-8408
Засько Євгеній

Київ 2021

Теоретичні відомості

Генетичні алгоритми служать, головним чином, для пошуку рішень в багатовимірних просторах пошуку. Можна виділити наступні етапи генетичного алгоритму: (Початок циклу) Розмножения (схрещування) Мутація Обчислити значення цільової функції для всіх особин Формування нового покоління (селекція) Якщо виконуються умови зупинки, то (кінець циклу), інакше (початок циклу). Розглянемо приклад реалізації алгоритму для знаходження цілих коренів діофантового рівняння a+b+2c=15. Згенеруємо початкову популяцію випадковим чином, але з дотриманням умови – усі згенеровані значення знаходяться у проміжку від одиниці до y/2, тобто на відрізку [1;8] (узагалі, границі випадкового генерування можна вибирати на свій розсуд): (1,1,5); (2,3,1); (3,4,1); (3,6,4) Отриманий генотип оцінюється за допомогою функції пристосованості (fitness function). Згенеровані значення підставляються у рівняння, після чого обраховується різниця отриманої правої частини з початковим y. Після цього рахується ймовірність вибору генотипу для ставання батьком – зворотня дельта ділиться на сумму сумарних дельт усіх генотипів

$1+1+2\cdot5=12 \quad \Delta=3 \quad \frac{\frac{1}{3}}{\frac{27}{24}} = 0{,}7$

$2+3+2\cdot1=7 \quad \Delta=8 \quad \frac{\frac{1}{8}}{\frac{27}{24}} = 0{,}11$

$3+4+2\cdot1=9 \quad \Delta=6 \quad \frac{\frac{1}{6}}{\frac{27}{24}} = 0{,}15$

$3+6+2\cdot4=17 \quad \Delta=2 \quad \frac{\frac{1}{2}}{\frac{27}{24}} = 0{,}44$

Наступний етап включає в себе схрещування генотипів по методу кросоверу – у якості дітей виступають генотипи, отримані змішуванням коренів – частина йде від одного з батьків, частина від іншого, наприклад:

$$\left.\begin{matrix}(3 \mid 6{,}4)\\(1 \mid 1{,}5)\end{matrix}\right\} \rightarrow \left\{\begin{matrix}(3{,}1{,}5)\\(1{,}6{,}4)\end{matrix}\right.$$

Лінія кросоверу може бути поставлена в будь-якому місці, кількість потомків також може вибиратися. Після отримання нових генотипів вони перевіряються функцією пристосованості та створюють власних потомків, тобто виконуються дії, описані вище. Ітерації алгоритму відбуваються, поки один з генотипів не отримає Δ=0, тобто його значення будуть розв’язками рівняння.

Умови завдання для варіанту бригади

Налаштувати генетичний алгоритм для знаходження цілих коренів діофантового рівняння ax1+bx2+cx3+dx4=y. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрат часу на розрахунки.

Лістинг програми із заданими умовами завдання

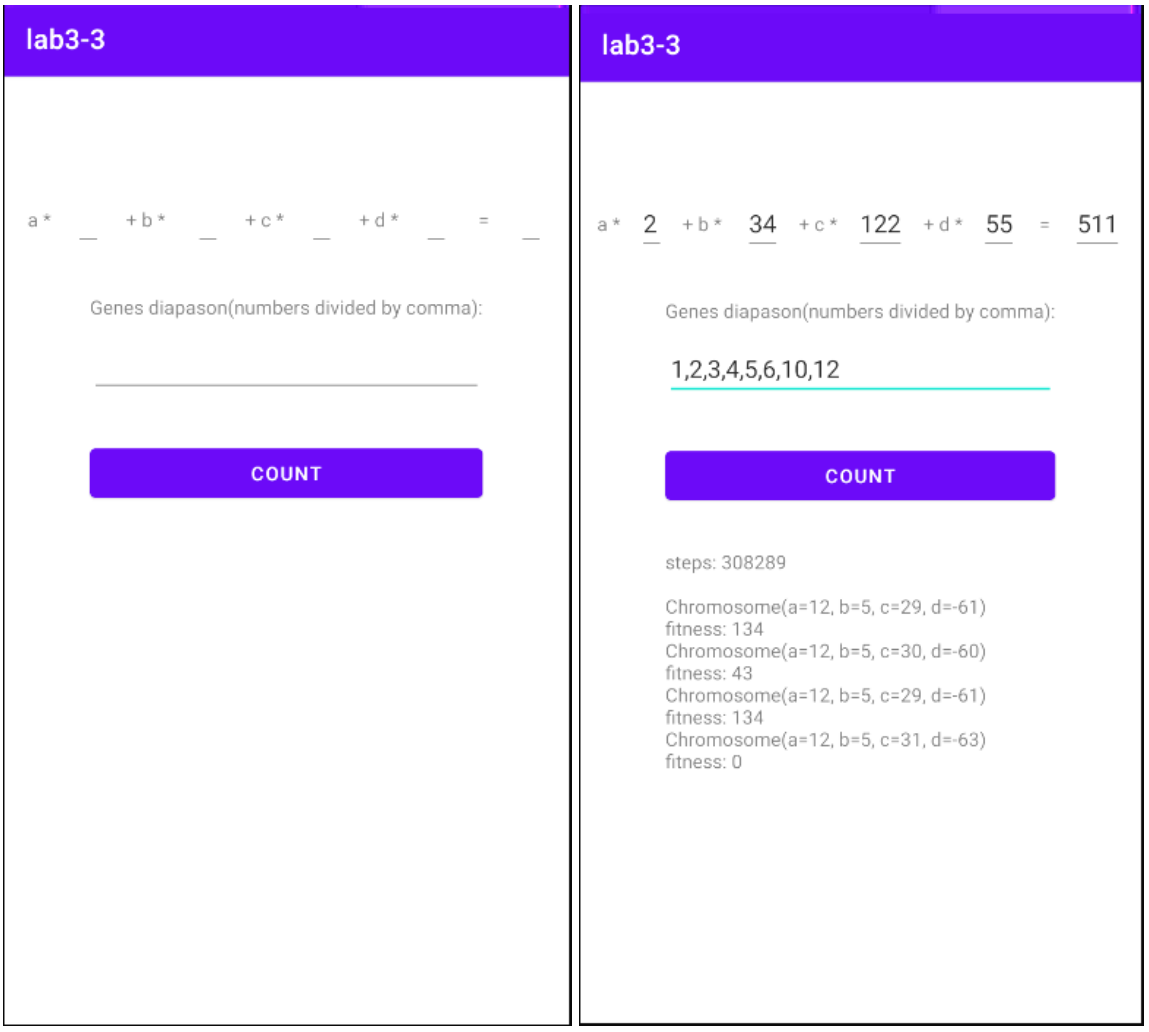
```
package ua.kpi.comsys.lab3_3

import kotlin.math.abs

data class Chromosome(val a: Int, val b: Int, val c: Int, val d: Int) {
    fun toMutableList() = mutableListOf(a, b, c, d)

    fun mutate() : Chromosome{
        val random = (0..100).random()
        return when {
            random < 5 -> this.copy(d = d + 1)
            random < 10 -> this.copy(d = d - 1)
            random > 95 -> this.copy(c = c + 1)
            random > 90 -> this.copy(c = c - 1)
            else -> this
        }
    }
}

fun List<Int>.toChromosome(): Chromosome {
    require(size > 3)
    return Chromosome(get(0), get(1), get(2), get(3))
}

class GenAlgorithm(
```

```
    private val func: (Int, Int, Int, Int) -> Int,
    private val y: Int,
    private val genesDiapason: List<Int>,
) {
    private val accuracy = 1000
    private fun fitness(res: Int) = abs(res - y)

    private fun generatePrimaryPopulation() = List(4) {
        Chromosome(
            genesDiapason.random(),
            genesDiapason.random(),
            genesDiapason.random(),
            genesDiapason.random(),
        )
    }

    private fun getRandomIndex(chances: List<Double>): Int {
        val random = (0..accuracy).random() / accuracy.toDouble()

        var chosen = chances.lastIndex - 1
        for (i in 0 until chances.size - 1) {
            if (random in chances[i]..chances[i + 1]) {
                chosen = i
                break
            }
        }
        return chosen
    }

    private fun step(population: List<Chromosome>, fitness: List<Int>): List<Chromosome> {
        val param = fitness.sumByDouble { 1.0 / it }
        val chances = fitness.map { (1.0 / it) / param }
```

```
    val periods = chances.fold(mutableListOf(0.0)) { s, e -> s.apply { add(e + s.last()) } }

    val pairs = List(population.size / 2) {
        val index1 = getRandomIndex(periods)
        val index2 = getRandomIndex(periods)
        population[index1] to population[index2]
    }

    pairs.forEach { it.swap(2) }
    return pairs.flatMap { listOf(it.first.mutate(), it.second.mutate()) }
}

fun launch(): Triple<List<Chromosome>, List<Int>, Int> {
    var population = generatePrimaryPopulation()
    var fitness = population.map { fitness(func(it.a, it.b, it.c, it.d)) }
    var countSteps = 1

    while (!fitness.any { it == 0 }) {
        population = step(population, fitness)
        fitness = population.map { fitness(func(it.a, it.b, it.c, it.d)) }
        countSteps++
    }

    return Triple(population, fitness, countSteps)
}

private fun Pair<Chromosome, Chromosome>.swap(
    startIndex: Int = 0,
    endIndex: Int = 3,
): Pair<Chromosome, Chromosome> {
    val first = this.first.toMutableList()
    val second = this.second.toMutableList()
```

```
        for (i in startIndex..endIndex) {
            val copy = first[i]
            first[i] = second[i]
            second[i] = copy
        }

        return first.toChromosome() to second.toChromosome()
    }
}
```

Результати виконання кожної програми



Висновки щодо виконання лабораторної роботи

В ході виконання лабораторної роботи ознайомився з принципами реалізації генетичного алгоритму, вивчення та дослідження особливостей даного алгоритму з використанням засобів моделювання і сучасних програмних оболонок. Створив застосунок, який виконує Діофантове рівняння за допомогою генетичного алгоритму(метод рулетки)