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-83 викладач  
Фібрук Руслан Сергійович Регіда Павло Геннадійович  
номер залікової книжки: 8320

Київ 2021

**Основні теоретичні відомості**

Генетичні алгоритми служать, головним чином, для пошуку рішень в багатовимірних просторах пошуку. Можна виділити наступні етапи генетичного алгоритму: • (Початок циклу) • Розмноження (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∙5=12 Δ=3 1 3 27 24 = 0,7 2+3+2∙1=7 Δ=8 1 8 27 24 = 0,11 3+4+2∙1=9 Δ=6 1 6 27 24 = 0,15 3+6+2∙4=17 Δ=2 1 2 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: (3 | 6,4) (3,1,5) → (1 | 1,5) (1,6,4) 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.

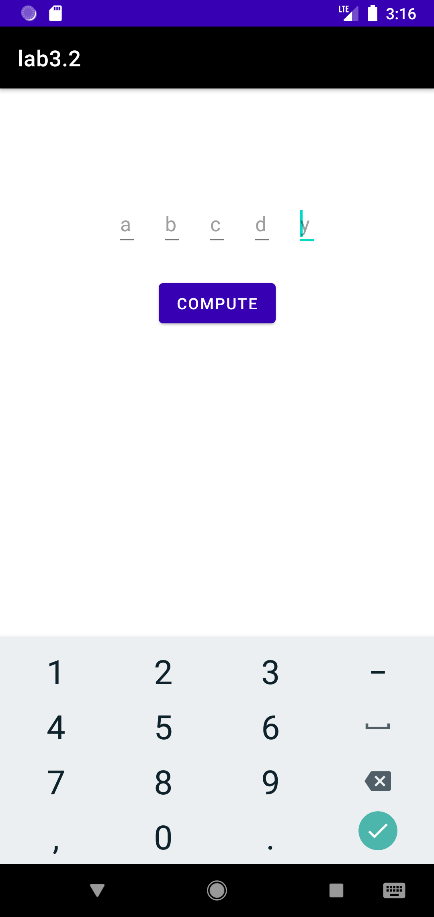
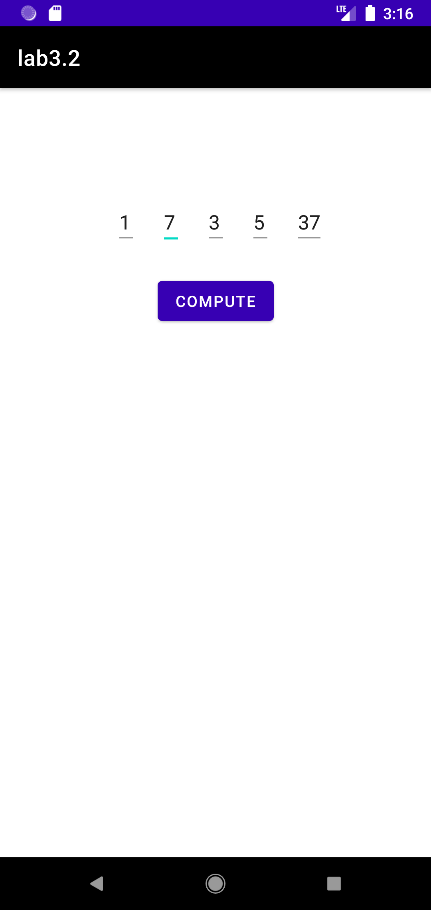
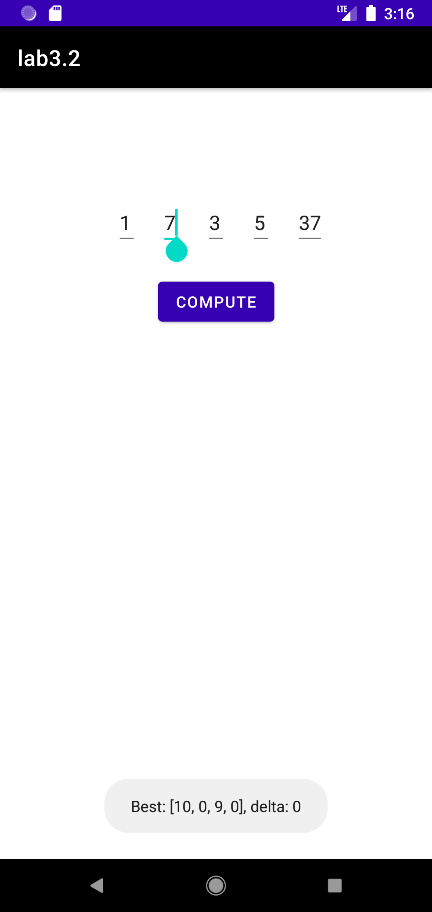
**Завдання на лабораторну роботу**

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**

package ua.kpi.comsys.ip8313.lab31  
  
import android.content.Context  
import androidx.appcompat.app.AppCompatActivity  
import android.os.Bundle  
import android.view.inputmethod.InputMethodManager  
import android.widget.Button  
import android.widget.EditText  
import android.widget.Toast  
import kotlin.math.abs  
import kotlin.random.Random  
  
class MainActivity : AppCompatActivity() {  
  
 private var a = 0  
 private var b = 0  
 private var c = 0  
 private var d = 0  
 private var y = 0  
 private val maxIterationCount = 1000  
  
 private fun hideKeyboard() {  
 val imm = this.getSystemService(Context.*INPUT\_METHOD\_SERVICE*) as InputMethodManager  
 imm.hideSoftInputFromWindow(*currentFocus*!!.*windowToken*, 0)  
 }  
  
 private fun fit(chromosome: Array<Int>): Int {  
 return *abs*(y - a \* chromosome[0] - b \* chromosome[1] - c \* chromosome[2] - d \* chromosome[3])  
 }  
  
 private fun mutate(chromosome: Array<Int>): Array<Int> {  
 val i = Random.nextInt(0, chromosome.size)  
 if (chromosome[i] < y / 2) chromosome[i] -= 1 else chromosome[i] += 1  
 return chromosome  
 }  
  
 private fun cross(population: Array<Array<Int>>): Array<Array<Int>> {  
 val newPopulation = Array(4) **{** Array(4) **{** 0 **} }** for (i in population.*indices*) {  
 val parent1 = population[Random.nextInt(0, population.size)]  
 val parent2 = population[Random.nextInt(0, population.size)]  
 val child = Array(4) **{** 0 **}** for (j in 0 *until* population[0].size / 2) {  
 child[j] = parent1[j]  
 child[population[0].size / 2 + j] = parent2[j]  
 }  
 newPopulation[i] = mutate(child)  
 }  
 return newPopulation  
 }  
  
 private fun compute(): Array<Int> {  
 var population = Array(4) **{** Array(4) **{** Random.nextInt(1, y / 2) **} }** for (iteration in 0 *until* maxIterationCount) {  
 for (chromosome in population) {  
 if (fit(chromosome) == 0) return chromosome  
 }  
 population = cross(population)  
 }  
 var best = population[0]  
 for (chromosome in population) {  
 if (fit(chromosome) < fit(best)) best = chromosome  
 }  
 return best  
 }  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
 val aInputView = findViewById<EditText>(R.id.*a\_input*)  
 val bInputView = findViewById<EditText>(R.id.*b\_input*)  
 val cInputView = findViewById<EditText>(R.id.*c\_input*)  
 val dInputView = findViewById<EditText>(R.id.*d\_input*)  
 val yInputView = findViewById<EditText>(R.id.*y\_input*)  
 val submitButton = findViewById<Button>(R.id.*compute\_button*)  
  
 submitButton.setOnClickListener **{** a = aInputView.*text*.toString().*toIntOrNull*() ?: 0  
 b = bInputView.*text*.toString().*toIntOrNull*() ?: 0  
 c = cInputView.*text*.toString().*toIntOrNull*() ?: 0  
 d = dInputView.*text*.toString().*toIntOrNull*() ?: 0  
 y = yInputView.*text*.toString().*toIntOrNull*() ?: 0  
 if (y == 0) return@setOnClickListener  
 val res = compute()  
 hideKeyboard()  
 Toast.makeText(this, "Best: ${res.*contentToString*()}, delta: ${fit(res)}", Toast.*LENGTH\_LONG*).show()  
 **}** }  
}

**Результати роботи програми**

**Висновки**

При виконанні цієї лабораторної роботи ми реалізували генетичний алгоритм для розв’язання діофантового рівняння та створили користувацький інтерфейс для введення коефіцієнтів та правої частини рівняння та відображення результату.