Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики і обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №3.1-3.3**

**з курсу: «Інтелектуальні вбудовані системи»**

*Виконав:*

студент групи ІП-84

Колмагін Д. О.

Залікова книжка №8411

*Перевірив:*

Регіда П. Г.

Київ 2021 р.

**Основні теоретичні відомості**

Факторизації лежить в основі стійкості деяких криптоалгоритмів, еліптичних кривих, алгебраїчній теорії чисел та кванових обчислень, саме тому дана задача дуже гостро досліджується, й шукаються шляхи її оптимізації.

На вхід задачі подається число n Є ℕ, яке необхідно факторизувати. Перед виконанням алгоритму слід переконатись в тому, що число не просте. Далі алгоритм шукає перший простий дільник, після чого можна запустити алгоритм заново, для повторної факторизації.

В залежності від складності алгоритми факторизації можна розбити на дві групи:

• Експоненціальні алгоритми (складність залежить експоненційно від довжини вхідного параметру);

• Субекспоненціальні алгоритми.

Метод факторизації Ферма. Ідея алгоритму заключається в пошуку таких чисел А і В, щоб факторизоване число n мало вигляд: n = A 2 – B 2 . Даний метод гарний тим, що реалізується без використання операцій ділення, а лише з операціями додавання й віднімання.

Приклад алгоритму:

Початкова установка: x = [√𝑛] – найменше число, при якому різниця x 2 -n невід’ємна. Для кожного значення k Є ℕ, починаючи з k = 1, обчислюємо (⌈√𝑛⌉ + 𝑘) 2 − 𝑛 і перевіряємо чи не є це число точним квадратом.

• Якщо не є, то k++ і переходимо на наступну ітерацію.

• Якщо є точним квадратом, тобто 𝑥 2 − 𝑛 = (⌈√𝑛⌉ + 𝑘) 2 − 𝑛 = 𝑦 2 , то ми отримуємо розкладання: 𝑛 = 𝑥 2 − 𝑦 2 = (𝑥 + 𝑦)(𝑥 − 𝑦) = 𝐴 ∗ 𝐵, в яких 𝑥 = (⌈√𝑛⌉ + 𝑘)

Якщо воно є тривіальним і єдиним, то n – просте

Важливою задачею яку система реального часу має вирішувати є отримання необхідних для обчислень параметрів, її обробка та виведення результату у встановлений дедлайн. З цього постає проблема отримання водночас точних та швидких результатів. Модель Перцпептрон дозволяє покроково наближати початкові значення.

Розглянемо приклад: дано дві точки А(1,5), В(2,4), поріг спрацювання Р = 4, швидкість навчання δ = 0.1. Початкові значення ваги візьмемо нульовими W1 = 0, W2 = 0. Розрахунок вихідного сигналу y виконується за наступною формулою:

x1 \* W1 + x2 \* W2 = y

Для кожного кроку потрібно застосувати дельта-правило, формула для розрахунку похибки:

Δ = P – y

де y – значення на виході.

Для розрахунку ваги, використовується наступна формули:

W1(i+1) = W1(i) + W2 \* x11

W2(i+1) = W1(i) + W2 \* x12

де i – крок, або ітерація алгоритму.

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)

**Завдання на лабораторну роботу**

Розробити програма для факторизації заданого числа методом Ферма. Реалізувати користувацький інтерфейс з можливістю вводу даних.

Поріг спрацювання: P = 4 Дано точки: А(0,6), В(1,5), С(3,3), D(2,4). Швидкості навчання: δ = {0,001; 0,01; 0,05; 0.1; 0.2; 0,3} Дедлайн: часовий = {0.5с; 1с; 2с; 5с}, кількість ітерацій = {100;200;500;1000} Обрати швидкість навчання та дедлайн. Налаштувати Перцептрон для даних точок. Розробити відповідний мобільний додаток і вивести отримані значення. Провести аналіз витрати часу та точності результату за різних параметрах навчання.

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**

Лаб 3.1

|  |
| --- |
| export const FermatFactorization = (n) => { |
|  | const s = Math.ceil(Math.sqrt(n)) |
|  | const maximumIterationsNumber = 100 |
|  | let y |
|  | for (let k = 0; k < maximumIterationsNumber; k++) { |
|  | y = (s + k) \*\* 2 - n |
|  | y = Math.sqrt(y) |
|  | if (Math.floor(y) === y) return [(s + k - y), (s + k + y)] |
|  | } |
|  | } |

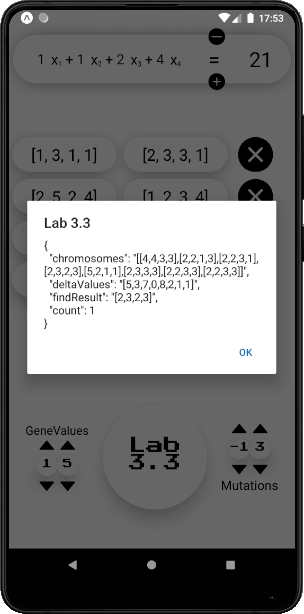
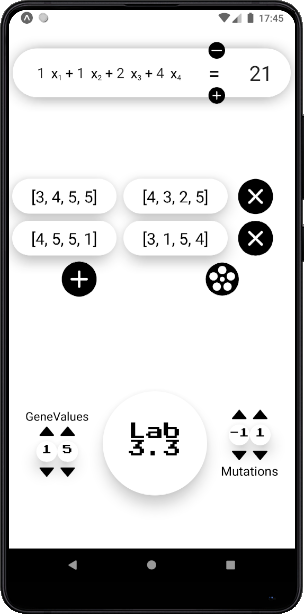
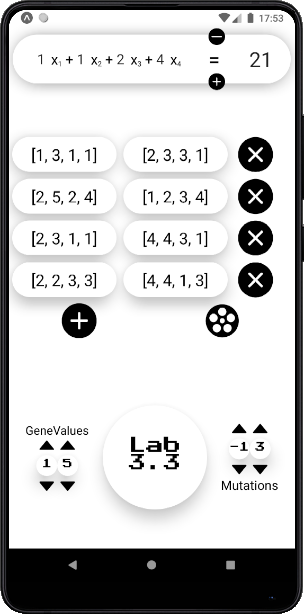
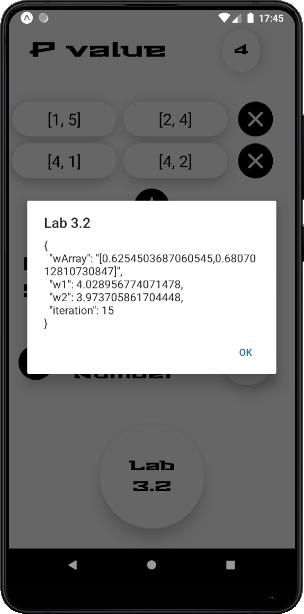
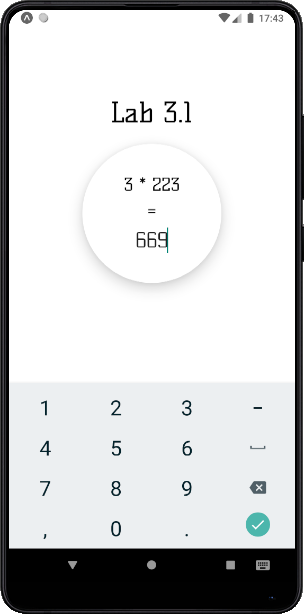
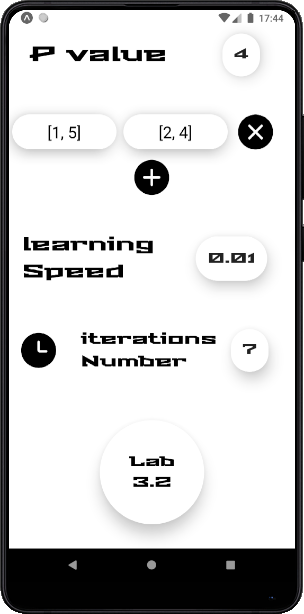
Лаб 3.2

|  |
| --- |
|  |
| export const perceptronModel = ( |
|  | { |
|  | P = 1, |
|  | func = (dot, wArray) => dot.reduce((res, x, i) => res + x \* wArray[i], 0), |
|  | dots = [[0, 0]], |
|  | learningSpeed = 0.1, |
|  | timeDeadline = null, |
|  | iterationsNumber = null |
|  | } = {} |
|  | ) => { |
|  | if (iterationsNumber) { |
|  | timeDeadline = null |
|  | } else if (!timeDeadline) { |
|  | iterationsNumber = 100 |
|  | } |
|  | let isContinue = true |
|  | let timeOut |
|  | if (timeDeadline) timeOut = setTimeout(() => isContinue = false, timeDeadline \* second) |
|  |  |
|  | const getDelta = (i) => P - func(dots[i], wArray) |
|  | let wArray = new Array(dots[0].length).fill(0, 0, dots[0].length) |
|  | for (let iteration = 0; iteration < (timeDeadline ? Infinity : iterationsNumber); iteration++) { |
|  | if (!isContinue) return 'timeDeadline >= iterationsNumber' |
|  | // check if satisfied |
|  | if (getDelta(0) < 0 && getDelta(1) > 0) { |
|  | clearTimeout(timeOut) |
|  | return { |
|  | wArray, |
|  | w1: P - getDelta(0), |
|  | w2: P - getDelta(1), |
|  | iteration, |
|  | toString() { |
|  | return JSON.stringify({ |
|  | wArray: JSON.stringify(wArray), |
|  | w1: P - getDelta(0), |
|  | w2: P - getDelta(1), |
|  | iteration, |
|  | }, null, 2) |
|  | } |
|  | } |
|  | } |
|  | // getting deltas |
|  | let delta = getDelta(iteration % dots.length) |
|  | wArray = wArray.map((w, i) => w + delta \* dots[iteration % dots.length][i] \* learningSpeed) |
|  | console.log(wArray) |
|  | } |
|  | clearTimeout(timeOut) |
|  | return 'iteration >= iterationsNumber' |
|  | } |
|  |  |

Лаб 3.3

|  |
| --- |
|  |
| export const geneticCalculations = (chromosomes, options = {}) => new Promise(r => { |
|  | let count = 0 |
|  | while (count < 10000) { |
|  | // console.log(chromosomes) |
|  | // getting deltas |
|  | const getDelta = (xArray, modulo = true) => { |
|  | const delta = -options.abcArgs.reduce((acc, cur, i) => acc - cur \* xArray[i], options.yArg) |
|  | return modulo ? (delta <= 0 ? -delta : delta) : delta |
|  | } |
|  | const deltaValues = chromosomes.map(chrome => getDelta(chrome)) |
|  |  |
|  | // checking if any delta === 0 |
|  | const findResult = chromosomes.find((\_, i) => deltaValues[i] === 0) |
|  | if (findResult) r({ |
|  | chromosomes, |
|  | deltaValues, |
|  | findResult, |
|  | count, |
|  | toString() { |
|  | return JSON.stringify({ |
|  | chromosomes: JSON.stringify(chromosomes), |
|  | deltaValues: JSON.stringify(deltaValues), |
|  | findResult: JSON.stringify(findResult), |
|  | count |
|  | }, null, 2) |
|  | } |
|  | }.toString()) |
|  | count++ |
|  |  |
|  | // calculating the chances of becoming a parent |
|  | const parentChances = deltaValues.map(delta => 1 / delta) |
|  | const deltasSum = parentChances.reduce((acc, cur) => acc + cur) |
|  | parentChances.forEach((delta, i) => parentChances[i] = delta / deltasSum) |
|  | parentChances.slice(0, parentChances.length - 1).forEach((\_, i) => parentChances[i + 1] += parentChances[i]) |
|  |  |
|  | // generating chances of becoming a parent using the roulette method |
|  | const rouletteValues = chromosomes.map(() => Math.random()) |
|  | const parents = rouletteValues.map( |
|  | (chance) => chromosomes[parentChances.findIndex( |
|  | (\_, i) => parentChances[i] >= chance && ((i > 0 ? parentChances[i - 1] : 0) < chance) |
|  | )] |
|  | ) |
|  |  |
|  | // crossbreeding |
|  | const children = [] |
|  | parents.forEach((\_, i) => { |
|  | if (i % 2 === 0) { |
|  | children[i] = parents[i].slice(0, options.crossoverLine).concat(parents[i + 1].slice(options.crossoverLine)) |
|  | children[i + 1] = parents[i + 1].slice(0, options.crossoverLine).concat(parents[i].slice(options.crossoverLine)) |
|  | } |
|  | }) |
|  |  |
|  | // mutations |
|  | children.forEach((\_, i) => { |
|  | if (Math.random() >= 0.5) { |
|  | const geneNumber = makeRandomWithin(children[i].length) |
|  | const addingValue = makeRandomWithin(options.maxMutationsValue, options.minMutationsValue) |
|  | if (children[i][geneNumber] + addingValue >= options.minGeneValue && |
|  | children[i][geneNumber] + addingValue <= options.maxGeneValue) children[i][geneNumber] += addingValue |
|  | } |
|  | } |
|  | ) |
|  |  |
|  | chromosomes = children |
|  | } |
|  | r('count is over 9999') |
|  | }) |
|  |  |

**Результати роботи програми**



**Висновки**

В даній лабораторній роботі ми – ознайомитись з основними принципами розкладання числа на прості множники з використанням різних алгоритмів факторизації, ознайомилися з принципами машинного навчання за допомогою математичної моделі сприйняття інформації Перцептрон (Perceptron) а також ознайомилися з принципами реалізації генетичного алгоритму, вивчили та дослідили особливостей даного алгоритму з використанням засобів моделювання і сучасних програмних оболонок. На основі знайдених даних створили інтерфейс (повноцінний додаток) для користувача мобільного телефону. Визначили час роботи даних алгоритмів. Дані знання знадобляться нам у нашій подальшій кар’єрі.