**Міністерство освіти і науки України**

**Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра обчислювальної техніки**

**ЗВІТ**

про виконання лабораторної роботи №3.3 з дисципліни

**«Інтелектуальні вбудовані системи»** на тему:

**«ДОСЛІДЖЕННЯ ГЕНЕТИЧНОГО АЛГОРИТМУ»**

|  |  |
| --- | --- |
| Перевірив: | ВИКОНАВ: |
| асистент кафедри ОТ | студент 3 курсу |
| Регіда П. Г | групи ІП-83, ФІОТ  Васильєв Г.З.  Залікова книжка №8505  Варіант – 5 |

Київ 2021

**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 Програмний код:

Genetic.js

const random = (min, max) => ~~(Math.random() \* (max - min) + min)

class Chromosome {

    genes = [];

    fitness = Infinity;

    task = [];

    target = 0;

    calc = () =>

        this.genes.reduce((a, gene, i) => a + (gene \* this.task[i]))

    constructor({ genes, task, target }) {

        Object.assign(this, { genes, task, target });

        this.calcFitness();

    }

    crossover(partner) {

        const child = this.clone();

        const fromParent = child.genes.length / 2;

        child.genes = [...child.genes.slice(0, fromParent), ...partner.genes.slice(child.genes.length - fromParent)];

        child.calcFitness();

        return child

    }

    calcFitness() {

        this.fitness = Math.abs(this.target - this.calc());

    }

    clone() {

        return Object.assign(Object.create(Object.getPrototypeOf(this)), this);

    }

}

let start = performance.now();

class Genetic {

    population = []

    constructor(task, target) {

        const { length } = task

        this.population =

            Array.from(

                { length: length + 1 },

                () => new Chromosome({

                    genes: Array.from({ length }, () => random(1, target / 2)),

                    task: task,

                    target: target,

                })

            )

    }

    solve() {

        while (true) {

            const chromosome = this.crossover()

            if (chromosome)

                return chromosome.genes

        }

    }

    crossover() {

        const children = []

        for (let i = 0; i < this.population.length; i++) {

            const parents = this.population

                .map((chromosome) => ({ chromosome, probability: Math.random() \* (chromosome.fitness \* 1000) }))

                .sort((a, b) => a.probability - b.probability)

            const parent = parents[0].chromosome

            const partner = parents[1].chromosome

            const child = parent.crossover(partner)

            if (child.fitness === 0)

                return child

            children.push(child)

        }

        this.population = children

    }

}

let end = performance.now()

let res = start - end;

console.log(res);

export default Genetic;

App.js

import React, { useState } from 'react';

import { StyleSheet, Text, View, SafeAreaView, TextInput, Button } from 'react-native';

import Genetic from './Genetic'

export default function App() {

  const [a, setA] = useState(null);

  const [b, setB] = useState(null);

  const [c, setC] = useState(null);

  const [d, setD] = useState(null);

  const [y, setY] = useState(null);

  const [result, setResult] = useState('[]');

  return (

    <SafeAreaView>

      <View style={styles.container}>

        <TextInput style={styles.expression}

          onChangeText={setA}

          value={a}

          placeholder="A"

          keyboardType="numeric"

        />

        <Text style={styles.expression}>{'\*x1 + '}</Text>

        <TextInput style={styles.expression}

          onChangeText={setB}

          value={b}

          placeholder="B"

          keyboardType="numeric"

        />

        <Text style={styles.expression}>{'\*x2 + '}</Text>

        <TextInput style={styles.expression}

          onChangeText={setC}

          value={c}

          placeholder="C"

          keyboardType="numeric"

        />

        <Text style={styles.expression}>{'\*x3 + '}</Text>

        <TextInput style={styles.expression}

          onChangeText={setD}

          value={d}

          placeholder="D"

          keyboardType="numeric"

        />

        <Text style={styles.expression}>{'\*x4 = '}</Text>

        <TextInput style={styles.expression}

          onChangeText={setY}

          value={y}

          placeholder="Y"

          keyboardType="numeric"

        />

      </View>

      <Text style={styles.result}>

        {`[x1, x2, x3, x4] = ${result}`}

      </Text>

      <View style={styles.btn}>

        <Button

          title="Calculate"

          color="#fff"

          onPress={() => setResult(new Genetic([a, b, c, d], y).solve())}

        />

      </View>

    </SafeAreaView>

  );

};

const styles = StyleSheet.create({

  container: {

    width: '90%',

    top: 250,

    flexDirection: 'row',

    alignSelf: 'center',

    alignItems: 'center',

    justifyContent: 'center',

  },

  expression: {

    fontSize: 24

  },

  result: {

    alignSelf: 'center',

    top: 269,

    fontSize: 25

  },

  time: {

    alignSelf: 'center',

    top: 320,

    fontSize: 22

  },

  btn: {

    justifyContent: 'center',

    alignItems: 'center',

    alignSelf: 'center',

    top: 300,

    height: 50,

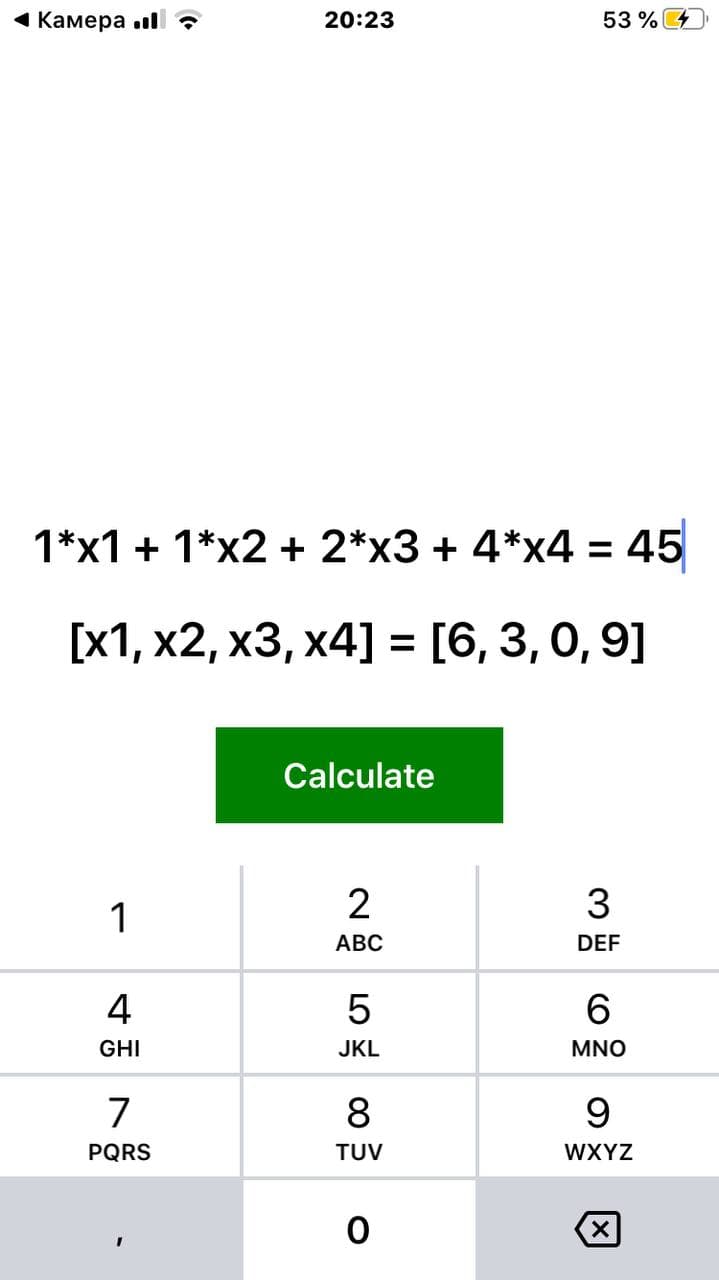
    width: 150,

    backgroundColor: 'green',

  },

});

Результати роботи програми:



Висновки:

Отже, в ході лабораторної роботи, ми отримали навички з розв’язування діафантового рівняння методом генетичного алгоритму з програмною реалізацію для мобільних додатків.

Результати наведено в звіті та врепозиторії. Кінцеву мету було досягнуто.