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 №5

з дисципліни
«Інтелектуальні вбудовані системи»

на тему
«РЕАЛІЗАЦІЯ ЗАДАЧІ РОЗКЛАДАННЯ ЧИСЛА НА ПРОСТІ МНОЖНИКИ (ФАКТОРИЗАЦІЯ ЧИСЛА)»

Виконала:

студентка групи ІП-84
Скрипник Єлена Сергіївна
номер залікової книжки: 8422

Перевірив:

викладач
Регіда Павло Геннадійович

Київ 2021

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.

Існування алгоритму з поліноміальною складністю – одна з найважливіших проблем в сучасній теорії чисел. Проте, факторизація з даною складністю можлива на квантовому комп'ютері за допомогою алгоритма Шора.

Метод перебору можливих дільників.

Один з найпростіших і найочевидніших алгоритмів заключається в тому, щоб послідовно ділити задане число n на натуральні числа від 1 до $\lfloor\sqrt{n}\rfloor$. Формально, достатньо ділити лише на прості числа в цьому інтервалі, але для цього необхідно знати їх множину. На практиці складається таблиця простих чисел і на вхід подаються невеликі числа (до 2^{16}), оскільки даний алгоритм має низьку швидкість роботи.

Приклад алгоритму:

1. Початкова установка: t = 0, k = 0, n = N (t,k,n такі, що n = N / $p_1 \ldots p_n$ і n не мають простих множників, менших за d_k).
2. Якщо n = 1, закінчуємо алгоритм.
3. Присвоюємо q = [n / d_k], r = n mod d_k.
4. Якщо r ≠ 0, переходимо на крок 6.
5. Присвоюємо t++, $p_t = d_k$, n = q і повертаємось на крок 2.
6. Якщо q > d_k → k++ і повертаємось на крок 3.
7. Присвоїти t++, p_t = n і закінчити виконання алгоритму.

Модофікований метод факторизації Ферма.

Ідея алгоритму заключається в пошуку таких чисел A і B, 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:

1. Початкова установка: $x = 2[\sqrt{n}] + 1$, y = 1, $r = [\sqrt{n}]^2 - n$.
2. Якщо r = 0, то алгоритм закінчено: $n = \frac{x-y}{2} * \frac{x+y-2}{2}$
3. Присвоюємо r = r + x, x = x + 2.
4. Присвоюємо r = r - y, y = y + 2.
5. Якщо r > 0, повертаємось до кроку 4, інакше повертаємось до кроку 2.

Метод факторизації Ферма.

Ідея алгоритму заключається в пошуку таких чисел A і B, 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: $x = \lceil\sqrt{n}\rceil$ – найменше число, при якому різниця x^2-n невід'ємна. Для кожного значення k Є N, починаючи з k = 1, обчислюємо $(\lceil\sqrt{n}\rceil + k)^2 - n$ і перевіряємо чи не є це число точним квадратом.

- Якщо не є, то k++ і переходимо на наступну ітерацію.
- Якщо є точним квадратом, тобто $x^2 - n = (\lceil\sqrt{n}\rceil + k)^2 - n = y^2$, то ми отримуємо розкладання: $n = x^2 - y^2 = (x + y)(x - y) = A * B$, в яких $x = (\lceil\sqrt{n}\rceil + k)$

Якщо воно є тривіальним і єдиним, то n - просте

Завдання:

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

Лістинг програми main.dart:

```
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import './Fermat.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'rts_lab3.1',
      theme: ThemeData(
        primarySwatch: Colors.teal,
      ),
      home: HomePage(title: 'Fermat factorization'),
    );
  }
}

class HomePage extends StatelessWidget{
  final String title;
  HomePage({Key, @required this.title}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: Text(this.title),
        ),
```

```
      body: Center(
        child: Container(
          height: MediaQuery.of(context).size.height,
          child: Column(
            children: <Widget>[
              InputForm(
                verticalPadding: 20.0,
                decompositionNumberHint: "Enter your number",
                decompositionNumberLabel: "Factorization number",
                stepsHint: "Enter the amount of steps",
                stepsLabel: "Amount of calculation steps",
                firstMultiplierLabel: "First multiplier",
                secondMultiplierLabel: "Second multiplier",
              ),
            ],
          ),
        ),
      )
    );
  }
}

class _InputData{
  int decompositionNumber;
  int steps;
}

class InputForm extends StatefulWidget {
  final double verticalPadding;
  final String decompositionNumberHint;
  final String decompositionNumberLabel;
  final String stepsHint;
  final String stepsLabel;
  final String firstMultiplierLabel;
  final String secondMultiplierLabel;

  InputForm({Key key,
    @required this.verticalPadding,
    @required this.decompositionNumberLabel,
    @required this.decompositionNumberHint,
    @required this.stepsLabel,
    @required this.stepsHint,
    @required this.firstMultiplierLabel,
    @required this.secondMultiplierLabel,
  }) : super(key: key);


  @override
  State<StatefulWidget> createState() => _InputFormState(
    verticalPadding: verticalPadding,
    decompositionNumberHint: decompositionNumberHint,
    decompositionNumberLabel: decompositionNumberLabel,
```

```
    stepsHint: stepsHint,
    stepsLabel: stepsLabel,
    firstMultiplierLabel: firstMultiplierLabel,
    secondMultiplierLabel: secondMultiplierLabel,
  );

}

class _InputFormState extends State<InputForm> {
  final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
  _InputData data = _InputData();

  final double verticalPadding;
  final String decompositionNumberHint;
  final String decompositionNumberLabel;
  final String stepsHint;
  final String stepsLabel;
  final String firstMultiplierLabel;
  final String secondMultiplierLabel;
  TextEditingController textPController = TextEditingController();
  TextEditingController textQController = TextEditingController();

  _InputFormState({
    @required this.verticalPadding,
    @required this.decompositionNumberLabel,
    @required this.decompositionNumberHint,
    @required this.stepsLabel,
    @required this.stepsHint,
    @required this.firstMultiplierLabel,
    @required this.secondMultiplierLabel,
  }) : super();

  void submit() async{
    if (this._formKey.currentState.validate()) {
      _formKey.currentState.save();
      var map = {
        "decompositionNumber": data.decompositionNumber,
        "steps": data.steps
      };
      List<int> result = await compute(fermatDecomposition, map);
      if(result == null){
        Scaffold.of(context).showSnackBar(SnackBar(
          content: Text("Can't find multipliers in this amount of steps"),
        ));
      }
      else{
        int p = result[0];
        int q = result[1];
        int stepsToPerform = result[2];
        textPController.text = p.toString();
        textQController.text = q.toString();
      }
```

```
  }
}

int _validateInt(String value) {
  int iValue;
  try {
    iValue = int.parse(value);
  } catch (e) {
    return null;
  }
  return iValue;
}

String _validateN(String value){
  int iValue = _validateInt(value);
  if (iValue == null){
    return 'The number should be integer';
  }
  if (iValue < 1){
    return 'The number should be bigger than 1';
  }
  if (iValue % 2 == 0){
    return 'The number should be odd';
  }
  return null;
}

String _validateSteps(String value){
  int iValue = _validateInt(value);
  if (iValue == null){
    return 'The number should be integer';
  }
  if(iValue < 0){
    return 'The number should be bigger than 1';
  }
  return null;
}

@override
Widget build(BuildContext context) {
  List<Widget> rows = [
    TextFormField(
        keyboardType: TextInputType.numberWithOptions(),
        validator: this._validateN,
        onSaved: (String value) {
          this.data.decompositionNumber = int.parse(value);
        },
        decoration: InputDecoration(
            hintText: this.decompositionNumberHint,
            labelText: this.decompositionNumberLabel
        )
    ),
```

```
      TextFormField(
        keyboardType: TextInputType.numberWithOptions(),
        validator: this._validateSteps,
        onSaved: (String value) {
          this.data.steps = int.parse(value);
        },
        decoration: InputDecoration(
            hintText: this.stepsHint,
            labelText: this.stepsLabel
        )
      ),
      TextFormField(
        enabled: false,
        controller: textPController,
        decoration: InputDecoration(
          labelText: this.firstMultiplierLabel,
        )
      ),
      TextFormField(
        enabled: false,
        controller: textQController,
        decoration: InputDecoration(
          labelText: this.secondMultiplierLabel,
        )
      ),
      Container(
        width: MediaQuery.of(context).size.width,
        child: RaisedButton(
          child: Text(
            'Calculate',
            style: TextStyle(
                color: Colors.white,
              fontSize: 20.0,
            ),
          ),
          onPressed: this.submit,
          color: Colors.teal,
        ),
      ),
    ];
    return Container(
        child: Form(
            key: this._formKey,
            child: Expanded(
              child: ListView(
                children: rows,
              ),
            )
        )
    );
  }
}
```

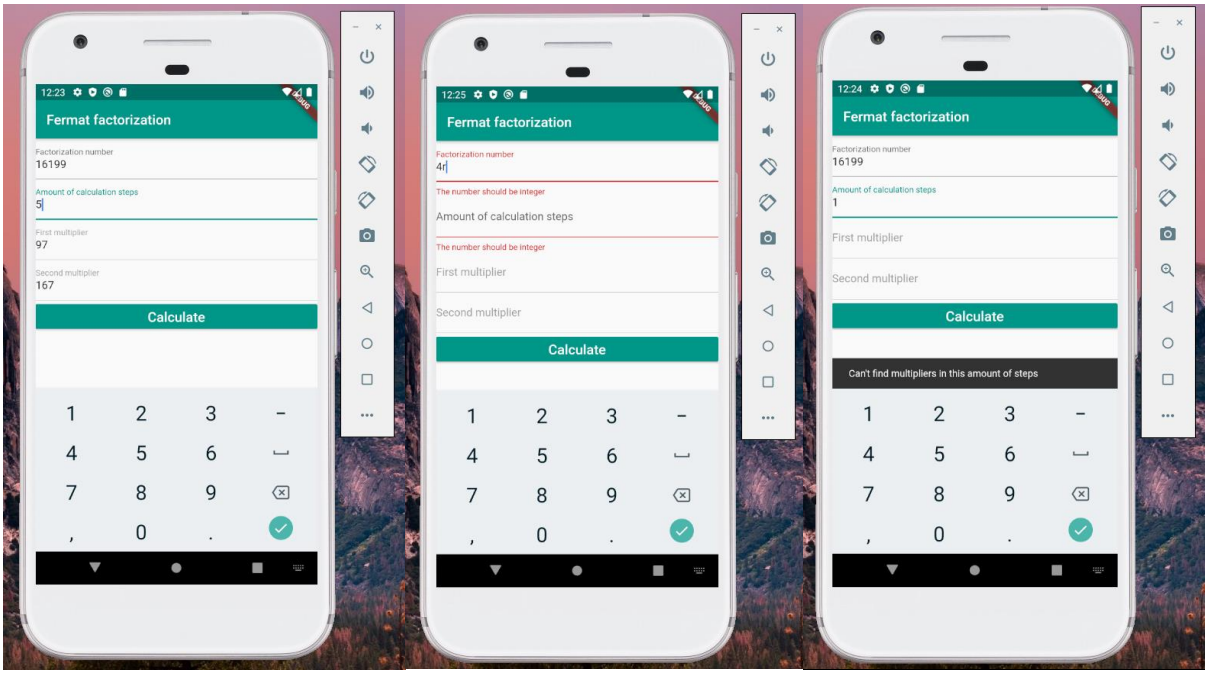
```
class Title extends StatelessWidget{
  final String title;
  final String description;
  Title({Key, @required this.title, @required this.description}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: const EdgeInsets.all(32),
      child: Row(
        children: [
          Expanded(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.center,
              children: [
                Container(
                  padding: const EdgeInsets.only(bottom: 8),
                  child: Text(
                    this.title,
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ),
                Text(
                  this.description,
                  style: TextStyle(
                    color: Colors.grey[500],
                  ),
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}
```

Лістинг програми Fermat.dart:

```
import 'dart:math';

List<int> fermatDecomposition(Map<String, int> map){
  int decompositionNumber = map["decompositionNumber"];
  int maxSteps = map["steps"];
  int currentStep = 0;
  int x = sqrt(decompositionNumber).toInt();

  double res;
  int resSqrt;
```

```
  do {
    if(currentStep == maxSteps) {
      return null;
    }
    x++;
    currentStep++;
    res = pow(x, 2) - decompositionNumber.toDouble();
    resSqrt = sqrt(res).toInt();
  }while(pow(resSqrt, 2) != res);

  int y = sqrt(pow(x, 2) - decompositionNumber).toInt();
  int p = x - y;
  int q = x + y;
  List<int> array = [p, q, currentStep];
  return array;
}
```

Результат виконання:



Висновки:

На даній лабораторній роботі було здійснено ознайомлення з принципами розкладання числа на прості множники з використанням різних алгоритмів факторизації. Також було розроблено програму для факторизації заданого числа методом Ферма з користувацьким інтерфейсом з можливістю вводу даних.