Университет ИТМО

Тестирование программного обеспечения
Лабораторная работа 1

Выполнил:
Бавыкин Роман Алексеевич
Группа P33091
Вариант 9199

г. Санкт-Петербург
2024г.

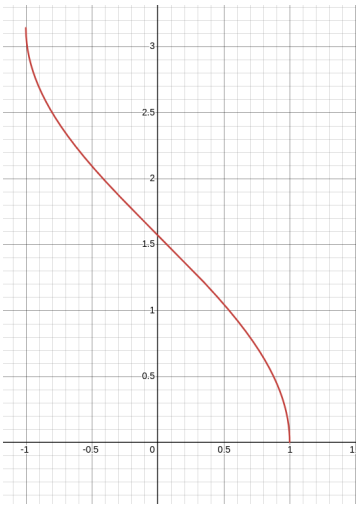
1. **Задание**

1. Для указанной функции провести модульное тестирование разложения функции в степенной ряд. Выбрать достаточное тестовое покрытие.
2. Провести модульное тестирование указанного алгоритма. Для этого выбрать характерные точки внутри алгоритма, и для предложенных самостоятельно наборов исходных данных записать последовательность попадания в характерные точки. Сравнить последовательность попадания с эталонной.
3. Сформировать доменную модель для заданного текста. Разработать тестовое покрытие для данной доменной модели

1. Функция arccos(x)
2. Программный модуль для работы с Фибоначчиевой кучей (Internal Representation, http://www.cs.usfca.edu/~galles/visualization/FibonacciHeap.html)
3. Описание предметной области:
   Бомбардировка возобновилась. Жар и шум были невообразимыми. Компьютерный банк начал понемногу разваливаться на куски. Лицевая сторона его почти вся расплавилась, и густые ручейки расплавленного металла начали заползать в угол, в котором они сидели. Они сгрудились плотнее и стали ждать конца. Глава 33

2. Выполнение

1. arccos(x)



Цель тестирования: проверить соответствие реализованного алгоритма нахождения арккосинуса эталонной реализации библиотеки Math.

Методика тестирования:

Область определения арккосинуса: [-1; 1].

Для проверки используем значения не из области определения (-2, 2), значения не из области определения близкие к ней (-1.01, 1.01), граничные значения (-1, 1), значения близкие к граничным (-0.9, 0.9), и значения в обалсти определения: (-0.5, 0, 0.5).

Для граничных значений выбрано для теста более высокая дельта, поскольку разложение в степенной ряд имеет низкую точность на этих значениях.

x	-2	-1.01	-1	-0.9	-0.5	0	0.5	0.9	1	1.01	2
arccos(x)	-	-	3.14	2.69	2.09	1.57	1.05	0.45	0	-	-

Реализация:

```java
package ru.robq.tpo1;

Roman Bavykin, 2 недели назад | 1 author (Roman Bavykin)
public class Arccos {

    private static double double_fact(int n) {
        int start;
        if (n % 2 == 0) {
            start = 2;
        } else {
            start = 3;
        }
        double result = 1;
        for (int i = start; i <= n; i += 2) {
            result *= i;
        }
        return result;
    }

    public static double arccos(double x, int n) {
        double result = x;      Roman Bavykin, 2 недели назад • 1 task
        if (x < -1 || x > 1) {
            return Double.NaN;
        }
        for (int i = 1; i < n; i++) {
            double part = (double_fact(2 * i - 1) * Math.pow(x, 2 * i + 1)) / (double_fact(2 * i) * (2 * i + 1));
            result += part;
        }
        result =  Math.PI / 2 - result;
        return result;
    }
}
```

Тест:

```java
package ru.robq.tpo1;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.assertAll;
import static org.junit.jupiter.api.Assertions.assertEquals;

Roman Bavykin, 2 недели назад | 1 author (Roman Bavykin)
public class ArccosTest {

    @ParameterizedTest(name = "arccos({0})")
    @ValueSource(doubles = {-2, -1.01, 1.01, 2})
    void checkUncorrectValues(double param) {
        assertAll(
            () -> assertEquals(Math.acos(param), Arccos.arccos(param, n:100), delta:0.001)
        );
    }

    @ParameterizedTest(name = "arccos({0})")
    @ValueSource(doubles = {-0.9, -0.5, 0, 0.5, 0.9})
    void checkCorrectValues(double param) {
        assertAll(
            () -> assertEquals(Math.acos(param), Arccos.arccos(param, n:100), delta:0.001)
        );
    }

    @ParameterizedTest(name = "arccos({0})")
    @ValueSource(doubles = {-1, 1})
    void checkBoundaryValues(double param) {
        assertAll(
            () -> assertEquals(Math.acos(param), Arccos.arccos(param, n:100), delta:0.1)
        );
    }
}
```

2. Фибоначева куча

Реализация взята с https://github.com/w01fe/fibonacci-heap/blob/master/src/jvm/w01fe/fibonacci_heap/FibonacciHeap.java и немного отредактирвоана.

Цель тестирования: проверить корректность работы основных возможностей работы с фибоначевой кучей: вставка, удаление минимума, очистка.

Методика тестирования:

Сверка заданного массива нод после выполнения преобразований с эталонным заданным массивом.

Тест:

```java
package ru.robq.tpo1;

import static org.junit.jupiter.api.Assertions.assertArrayEquals;

import java.util.Arrays;
    You, 2 недели назад • task 2
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.converter.ConvertWith;
import org.junit.jupiter.params.provider.CsvFileSource;


You, 2 недели назад | 2 authors (You and others)
public class FibbonacciHeapTest {
    @ParameterizedTest
    @CsvFileSource(resources = "/check_insert_source.csv", delimiter = ';')
    void checkInsert(@ConvertWith(IntArrayConverter.class) int[] input,
                     @ConvertWith(IntArrayConverter.class) int[] expected,
                     int insertValue) {
        FibonacciHeap heap = new FibonacciHeap(input);
        heap.insert(insertValue);
        input = heap.nodeArray();
        Arrays.sort(expected);
        Arrays.sort(input);
        assertArrayEquals(input, expected);
    }

    @ParameterizedTest
    @CsvFileSource(resources = "/check_remove_min_source.csv", delimiter = ';')
    void checkRemoveMin(@ConvertWith(IntArrayConverter.class) int[] input,
                        @ConvertWith(IntArrayConverter.class) int[] expected) {
        FibonacciHeap heap = new FibonacciHeap(input);
        heap.removeMin();
        input = heap.nodeArray();
        Arrays.sort(expected);
        Arrays.sort(input);
        assertArrayEquals(input, expected);
    }

    @ParameterizedTest
    @CsvFileSource(resources = "/check_clear_source.csv", delimiter = ';')
    void checkClear(@ConvertWith(IntArrayConverter.class) int[] input) {
        FibonacciHeap heap = new FibonacciHeap(input);
        heap.clear();
        assertArrayEquals(heap.nodeArray(), new int[0]);
    }
}
```

3. Предметная область

Полный текст можно найти на гитхаб: https://github.com/robqqq/tpo1

Цель тестирования: проверить корректность работы методов реализованных классов предметной области

Методика тестирования: проверка возвращаемой строки функцией, а в некоторых случаях проверка корректности изменения или неизменения состояния.

Тесты:

```java
package ru.robq.tpo1;

import ...

Роман *
public class Task3Test {
    39 usages
    private Room room;
    5 usages
    private List<Person> they;
    5 usages
    private IronWorldObject compBank;
    4 usages
    private IronWorldObject gasObject ;
    3 usages
    private final float MIN_TEMP = -273;
    3 usages
    private final float MIN_NOISE_LEVEL = 0;
    3 usages
    private final float MAX_NOISE_LEVEL = 10;

    new *
    @BeforeEach
    void init() {
        they = new ArrayList<>();
        they.add(new Person( name: "First someone",  weight: 50,  height: 150));
        they.add(new Person( name: "Second someone",  weight: 70,  height: 170));
        compBank = new IronWorldObject( name: "Computer bank",  length: 100,  width: 100, AggregationState.SOLID);
        gasObject = new IronWorldObject( name: "Gas object",  length: 0,  width: 0, AggregationState.GAS);
        room = new Room( noiseLevel: 5,  temperature: 5);
    }

    new *
    @Test
    void checkClearDisaster() { assertEquals( expected: "Disaster ended. Now is quietly.", room.clearDisaster()); }

    new *
    @Test
    void checkSetCurrentDisaster() {
        assertEquals( expected: "bombardment started.", room.setCurrentDisaster(Disaster.BOMBARDMENT));
    }

    new *
    @Test
    void checkNegativeIncreaseTemperature() {
        float oldTemperature = room.getTemperature();
        assertEquals( expected: "Delta must be positive", room.increaseTemperature( delta: -1));
        assertEquals(oldTemperature, room.getTemperature());
    }

    new *
    @Test
    void checkNegativeDecreaseTemperature() {
        float oldTemperature = room.getTemperature();
        assertEquals( expected: "Delta must be positive", room.decreaseTemperature( delta: -1));
        assertEquals(oldTemperature, room.getTemperature());
    }
```

```java
@Test
void checkNegativeIncreaseNoise() {
    float oldNoiseLevel = room.getNoiseLevel();
    assertEquals( expected: "Delta must be positive", room.increaseNoise( delta: -1));
    assertEquals(oldNoiseLevel, room.getNoiseLevel());
}

new *
@Test
void checkNegativeDecreaseNoise() {
    float oldNoiseLevel = room.getNoiseLevel();
    assertEquals( expected: "Delta must be positive", room.decreaseNoise( delta: -1));
    assertEquals(oldNoiseLevel, room.getNoiseLevel());
}

new *
@Test
void checkIncreaseTemperature() {
    float oldTemperature = room.getTemperature();
    float delta = 1;
    assertEquals( expected: "Temperature increase by " + delta + " deegres, now temperature is " + (oldTemperature + delta), room.increaseTemperature(delta));
    assertEquals( expected: oldTemperature + delta, room.getTemperature());
}

new *
@Test
void checkDecreaseTemperature() {
    float oldTemperature = room.getTemperature();
    float delta = 1;
    assertEquals( expected: "Temperature decrease by " + delta + " deegres, now temperature is " + (oldTemperature - delta), room.decreaseTemperature(delta));
    assertEquals( expected: oldTemperature - delta, room.getTemperature());
}

new *
@Test
void checkIncreaseNoise() {
    float oldNoiseLevel = room.getNoiseLevel();
    float delta = 1;
    assertEquals( expected: "Noise level increase by " + delta + ", now noise level is " + (oldNoiseLevel + delta), room.increaseNoise(delta));
    assertEquals( expected: oldNoiseLevel + delta, room.getNoiseLevel());
}

new *
@Test
void checkDecreaseNoise() {
    float oldNoiseLevel = room.getNoiseLevel();
    float delta = 1;
    assertEquals( expected: "Noise level decrease by " + delta + ", now noise level is " + (oldNoiseLevel - delta), room.decreaseNoise(delta));
    assertEquals( expected: oldNoiseLevel - delta, room.getNoiseLevel());
}
```

```
    @Test
    void checkDecreaseTemperatureLimit() {
        float oldTemperature = room.getTemperature();
        float delta = 300;
        assertEquals( expected: "Temperature decrease by " + (oldTemperature - MIN_TEMP) + " deegres, now temperature is " + MIN_TEMP, room.decreaseTemperature(delta));
        assertEquals(MIN_TEMP, room.getTemperature());
    }

    new *
    @Test
    void checkIncreaseNoiseLimit() {
        float oldNoiseLevel = room.getNoiseLevel();
        float delta = 10;
        assertEquals( expected: "Noise level increase by " + (MAX_NOISE_LEVEL - oldNoiseLevel) + ", now noise level is " + MAX_NOISE_LEVEL, room.increaseNoise(delta));
        assertEquals(MAX_NOISE_LEVEL, room.getNoiseLevel());
    }

    new *
    @Test
    void checkDecreaseNoiseLimit() {
        float oldNoiseLevel = room.getNoiseLevel();
        float delta = 10;
        assertEquals( expected: "Noise level decrease by " + (oldNoiseLevel - MIN_NOISE_LEVEL) + ", now noise level is " + MIN_NOISE_LEVEL, room.decreaseNoise(delta));
        assertEquals(MIN_NOISE_LEVEL, room.getNoiseLevel());
    }

    new *
    @Test
    void checkPutPerson() {
        assertEquals( expected: "First someone located at right back corner now.", room.putPerson(Location.RIGHT_BACK_CORNER, they.get(0)));
        assertEquals( expected: "Second someone located at right back corner now.", room.putPerson(Location.RIGHT_BACK_CORNER, they.get(1)));
    }

    new *
    @Test
    void checkPutWorldObject() {
        assertEquals( expected: "Computer bank located at center now.", room.putWorldObject(Location.CENTER, compBank));
    }

    new *
    @Test
    void checkMelt() {
        assertEquals( expected: "Computer bank melt and transform to Melted iron.", compBank.melt( new_name: "Melted iron"));
        assertEquals(AggregationState.LIQUID, compBank.getAggregationState());
        assertEquals( expected: "Melted iron", compBank.getName());
    }

    new *
    @Test
    void checkMeltNotSolid() {
        assertEquals( expected: "Gas object can't melt because it's not solid.", gasObject.melt( new_name: "Melted gas object"));
        assertEquals(AggregationState.GAS, gasObject.getAggregationState());
        assertEquals( expected: "Gas object", gasObject.getName());
    }
}
```

3. Вывод: во время выполнения лабораторной работы познакомился с Unit-тестами и библиотекой JUNIT. Научился писать тесты. 100% покрытие достигнуть почти невозможно, ну и действительно нельзя оценить насколько код покрыт тестами. Нужно самостоятельно логически осознавать, что именно нужно тестировать, а что не требует тестирования. В любом случае тесты - полезный инструмент в разработке, на который необходимо потратить дополнительное время в начале, но который помогает в разработке и поддержке продукта для выявления ошибок.