Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет Программной инженерии и компьютерной техники

Лабораторная работа №1

по дисциплине «Тестирование программного обеспечения»

Вариант 367854

Группа: P3312

Выполнили: Балин А. А., Кобелев Р. П.

Проверил: Кривоносов Е. Д.

г. Санкт-Петербург, 2025

Оглавление

Цель работы

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.

По варианту:

1. Функция $\arccos(x)$
2. Программный модуль для работы с красно-черным деревом (http://www.cs.usfca.edu/~galles/visualization/RedBlack.html)
3. Описание предметной области:

"Путеводитель по Галактике для автостопщиков" -- очень неоднородная книга, в ней встречается информация, которая в какой-то момент просто попалась на глаза редактору и показалась ему занимательной.

Выполнение

Исходный код: https://github.com/Romariok/Software-Testing

Тестирование функции $\arccos(x)$

public static double arccosSeries$($*double x*$, \int$ *terms*$)$, где x – аргумент функции, *terms* – количество первых членов из бесконечного ряда, которые будут взяты для расчётов.

Сходимость ряда Тейлора для данной функции: $|x|<1$. С учётом ограниченной разрядности компьютера и накапливающейся ошибки при вычислении выражений с плавающей точкой было практически выявлено, что вычисления имеют смысл при $|x|\leq 0.9$. Для тестирования взяли равномерно относительно 0 по3 значения (по модулю): $0.5, 0.75, 0.9$. Таким же образом (т. е. практическим) было выявлено оптимальное количество первых членов (30, 60).

До сокращения количества тестов был набор из тестов, в результате которых должно было выбрасываться то или иное исключение. Например, *ArithmeticException* (при *NaN*, когда double не хватало для расчёта последующих членов ряда) получался при приближении переменной x к 1 (по модулю) или при увеличении количества *terms* более 90. Раньше такие тестовые примеры включались в список для тестирования на $terms \in \{30, 60\}$ или $x \in \{0, 0.5, 0.75, 0.9, -0.5, -0.75, -0.9\}$, откуда мы получаем огромное количество «лишних» тестов. Аналогично для $|x|\geq 1$, то есть тестов, проворящих, что метод выкинет ошибку *IllegalArgumentException* (незачем проверять это при разных значениях *terms*.

Тестирование красно-черного дерева

Для начала была написана проверка на граничные значения аргументов функций. То есть функции принимают числа типа *int*, по этому были проверены значения $\{Integer.MAX\ VALUE+1; Integer.MAX\ VALUE; Integer.MIN\ VALUE-1; Integer.MIN\ VALUE\}$.
Помимо этих значений были взяты несколько значений в правильном диапазоне. Дальше были проверены базовые функции и балансировка этого вида дерева. То есть бралось дерево с различными данными, дальше применялись функции и ожидались определённые значения. После написания всех тестов оказалось, что многие тесты являются частью других, так что были удалены.

Тестирование доменной модели

Сначала была разработана доменная модель и логика. Дальше тесты разделились на 2 категориии — создание классов, использование классов. В первой категории были проверены все граничные значения для дат. То есть дата написания книги не должна быть в будущем и не должна быть меньше *0*. Также были проверены *Null* и пустые значения. Во второй категории были уже проверены все основные функции классов и логика, что при прочтении книги "Путеводитель по Галактике для автостопщиков" редактор должен посчитать её занимательной.

Вывод

В рамках выполнения данной лабораторной работы мы практиковались в написании модульных тестов с использованием Junit и различных систем для оценки code coverage. Также в этой работе мы научились анализировать предметную область, работать с граничными случаями, оценивать эффективность тестов и оптимизировать их количество без потери покрытия.