



Лабораторная работа №1

по дисциплине: Тестирование программного обеспечения

Вариант: 3202

Выполнил: Неграш Андрей, Р33301

Преподаватель: Гаврилов Антон Валерьевич

Санкт-Петербург, 2023

Оглавление

Задание	2
Описание работы.....	2
Пункт 1: функция $\sec(x)$	2
Пункт 2: алгоритм формирования бинарных деревьев.....	4
Пункт 3: Тестовое покрытие доменной модели для заданного текста	4
Вывод.....	5

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

Please, enter your variant:

1. Функция $\sec(x)$
2. Программный модуль для работы с В деревьями (количество элементов в ключе - до 3, <http://www.cs.usfca.edu/~galles/visualization/BTree.html>)
3. Описание предметной области:

Эрунда, подумал Форд. Даже если предположить, что здесь когда-то существовала цивилизация, превратившаяся теперь в пыль, даже если предположить еще много маловероятных вещей, все равно огромные богатства не могли бы сохраниться здесь в форме, представляющей какой-либо интерес. Он пожал плечами.

Описание работы

Пункт 1: функция $\sec(x)$

Формула для вычисления приближённого значения в радианах:

$$\sec(x) = \frac{1}{\sum_{i=1}^{14} \frac{(-1)^i * x^{2i}}{(2 * i)!}}$$

```
for (int i = 1; i <= 14; i++)
    result = result + ((Math.pow(-1, i)) * (Math.pow(rad, 2 * i)) /
getFactorial(2 * i));
result = 1 / result;
```

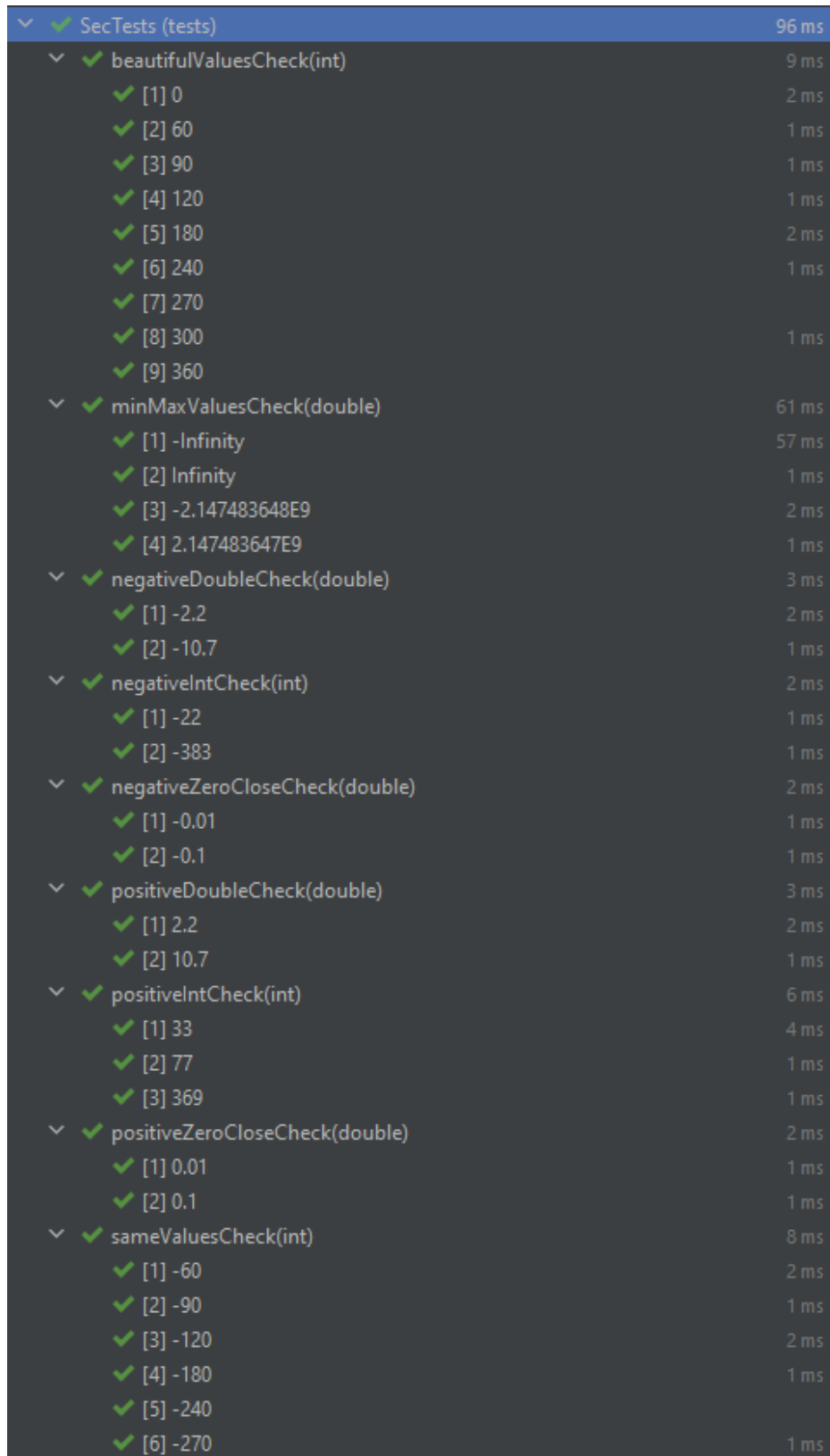
Формула, используемая для проверки работоспособности алгоритма:

$$\sec(x) = \frac{1}{\cos(x)}$$

```
double result = 1 / Math.cos(Math.toRadians(value));
```

В обоих случаях x – это значение в радианах. Однако пользовательский ввод подразумевает данные в градусах, для этого реализована функция перевода градусов в радианы.

Результаты тестов представлены на скриншоте:



✓ SecTests (tests)	96 ms
✓ beautifulValuesCheck(int)	9 ms
✓ [1] 0	2 ms
✓ [2] 60	1 ms
✓ [3] 90	1 ms
✓ [4] 120	1 ms
✓ [5] 180	2 ms
✓ [6] 240	1 ms
✓ [7] 270	
✓ [8] 300	1 ms
✓ [9] 360	
✓ minMaxValuesCheck(double)	61 ms
✓ [1] -Infinity	57 ms
✓ [2] Infinity	1 ms
✓ [3] -2.147483648E9	2 ms
✓ [4] 2.147483647E9	1 ms
✓ negativeDoubleCheck(double)	3 ms
✓ [1] -2.2	2 ms
✓ [2] -10.7	1 ms
✓ negativeIntCheck(int)	2 ms
✓ [1] -22	1 ms
✓ [2] -383	1 ms
✓ negativeZeroCloseCheck(double)	2 ms
✓ [1] -0.01	1 ms
✓ [2] -0.1	1 ms
✓ positiveDoubleCheck(double)	3 ms
✓ [1] 2.2	2 ms
✓ [2] 10.7	1 ms
✓ positiveIntCheck(int)	6 ms
✓ [1] 33	4 ms
✓ [2] 77	1 ms
✓ [3] 369	1 ms
✓ positiveZeroCloseCheck(double)	2 ms
✓ [1] 0.01	1 ms
✓ [2] 0.1	1 ms
✓ sameValuesCheck(int)	8 ms
✓ [1] -60	2 ms
✓ [2] -90	1 ms
✓ [3] -120	2 ms
✓ [4] -180	1 ms
✓ [5] -240	
✓ [6] -270	1 ms

Пункт 2: алгоритм формирования бинарных деревьев

Основные операции, которые выполняются над В-деревьями:

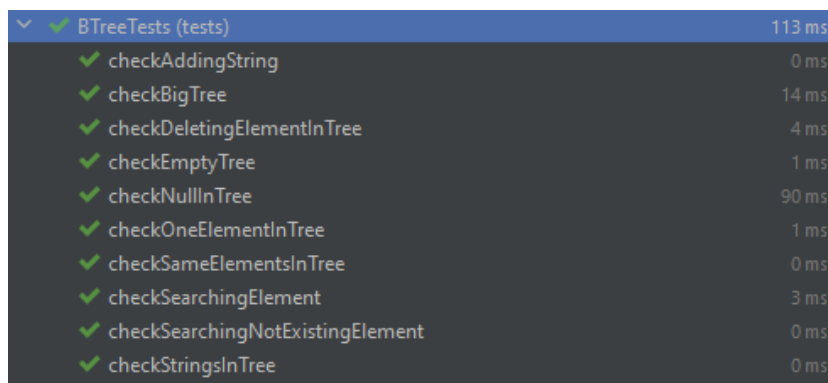
- Добавление узла
- Поиск узла
- Удаление узла

В реализации данного алгоритма реализована возможность добавления элементов по одному или массивом, каждый элемент которого будет добавлен последовательно в дерево.

При поиске элемента возвращается путь от корня до найденного элемента или ошибка, сообщающая, что такого элемента не существует.

Все операции и результаты, с которыми сравнивается работа алгоритма, взяты из приведённого в задании сайта (<http://www.cs.usfca.edu/~galles/visualization/BTree.html>), отформатированного под вывод моей реализации алгоритма.

Результаты тестов представлены на скриншоте:



✓ BTreeTests (tests)	113 ms
✓ checkAddingString	0 ms
✓ checkBigTree	14 ms
✓ checkDeletingElementInTree	4 ms
✓ checkEmptyTree	1 ms
✓ checkNullInTree	90 ms
✓ checkOneElementInTree	1 ms
✓ checkSameElementsInTree	0 ms
✓ checkSearchingElement	3 ms
✓ checkSearchingNotExistingElement	0 ms
✓ checkStringsInTree	0 ms

Пункт 3: Тестовое покрытие доменной модели для заданного текста

Ниже представлена UML-диаграмма классов для полученного согласно варианту текста:

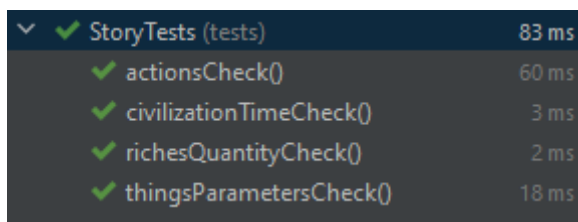


Историю можно запустить, обратившись к методу doStory() в классе Story.

Результат работы программы представлен в виде таблицы сравнения оригинального текста и вывода программы:

Оригинал	Программа
Ерунда, подумал Форд. Даже если предположить, что здесь когда-то существовала цивилизация, превратившаяся теперь в пыль, даже если предположить еще много маловероятных вещей, все равно огромные богатства не могли бы сохраниться здесь в форме, представляющей какой-либо интерес. Он пожал плечами.	Форд подумал: "Ерунда" Даже если предположить: цивилизация существовала когда-то цивилизация превратилась в пыль теперь Даже если предположить ещё: много маловероятных вещей огромные богатства не могли бы сохраниться форма не представляет интерес Форд пожал плечами

Основной задачей было разработать тестовое покрытие для данной доменной модели. Результат тестирования представлен на скриншоте:



✓ StoryTests (tests)	83 ms
✓ actionsCheck()	60 ms
✓ civilizationTimeCheck()	3 ms
✓ richesQuantityCheck()	2 ms
✓ thingsParametersCheck()	18 ms

Вывод

Итак, в процессе данной лабораторной работы я ознакомился с понятием модульного тестирования и способами его реализации при помощи фреймворка Junit. Репозиторий с кодом доступен на GitHub по ссылке: https://github.com/ANegrash/TPO_lab1