Министерство науки и высшего образования Российской Федерации

Университет ИТМО

Мегафакультет компьютерных технологий и управления

Системное и прикладное программное обеспечение

09.03.04 Программная инженерия

Лабораторная работа №1

по “Тестированию программных систем”

Вариант 6116

Выполнил:

*Пурэвсурэн Билгуун*

*Колесников Никита*   
группа Р33131

Преподаватель:

*Яркеев А.С*

Содержание

[Текст задания 3](#_Toc130383353)

[Выполнение 3](#_Toc130383354)

[1. Функция cos(x) 3](#_Toc130383355)

[2. Counting sort 6](#_Toc130383356)

# Текст задания

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

Text

Description automatically generated

# Выполнение

Github с исходным кодом: <https://github.com/Roclh/tpo>

## Функция cos(x)

A picture containing diagram

Description automatically generated

Будет проведена два тестирования. Первое для значение-экстремумах и значение когда функция равна нулю. Второе для промежуточных и больших значений.

Таблица проверки первого тестирования:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | -2π | -3π/2 | -π | -π/2 | 0 | π/2 | Π | 3π/2 | 2π |
| cos(x) | 1 | 0 | -1 | 0 | 1 | 0 | -1 | 0 | 1 |

### Таблица проверки второго тестирования:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | -5.13 | -3.84 | -0.986 | 1.17 | 2.567 | 5.32 | 10 | 19.78 | 63.21 |
| cos(x) | 0.4055 | -0.766 | 0.5520 | 0.3902 | -0.8394 | 0.5709 | -0.8391 | 0.5975 | 0.9294 |

Исходный код класса функции

package org.Roclh;  
  
import org.apache.logging.log4j.LogManager;  
import org.apache.logging.log4j.Logger;  
  
public class Cos {  
  
 private static final Logger *logger* = LogManager.*getLogger*();  
  
 public static double calculate(double x, int n){  
 double result = 1;  
  
 double xx = x \* x;  
 int sign = -1;  
 double factorial = 1;  
 double dividend = xx;  
  
 for(int i = 2; i < n; i += 2){  
 factorial /= i;  
 result += sign \* dividend \* factorial;  
 sign = -1 \* sign;  
 factorial /= (i+1);  
 dividend \*= xx;  
 }  
  
 return result;  
 }  
  
}

Функция вычисляет результат, разложив функцию в степенной ряд для нахождения значения.

Chart, diagram

Description automatically generated with medium confidence

Формула была использована из источника: <https://function-x.ru/chapter9-4/rows4_clip_image069.gif>

Исходный код тестирования:

package org.Roclh;  
  
import org.junit.jupiter.api.DisplayName;  
import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.CsvFileSource;  
import org.junit.jupiter.params.provider.ValueSource;  
  
import static org.junit.jupiter.api.Assertions.*assertAll*;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
  
public class CosTest {  
  
 @ParameterizedTest(name = "cos({0})")  
 @DisplayName("Check PI dots")  
 @ValueSource(doubles = {-2 \* Math.*PI*, -Math.*PI*, -0.5 \* Math.*PI*, 0, 0.5 \* Math.*PI*, Math.*PI*, 1.5 \* Math.*PI*, 2 \* Math.*PI*})  
 void checkPiDots(double param){  
 *assertAll*(  
 () -> *assertEquals*(Math.*cos*(param), Cos.*calculate*(param, 100), 0.0001)  
 );  
 }  
  
 @ParameterizedTest(name = "cos({0}) = {1}")  
 @DisplayName("Check inbetween and large dots")  
 @CsvFileSource(resources = "/table\_values.csv", numLinesToSkip = 1, delimiter = ';')  
 void checkBetweenDotsMinusPiAndPi(double x, double y) {  
 *assertAll*(  
 () -> *assertEquals*(y, Cos.*calculate*(x, 100), 0.0001)  
 );  
 }  
}

Вывод тестирования:

Graphical user interface, text, chat or text message

Description automatically generated

К сожалению во втором тестировании значение cos(63.21) не прошло. Это потому что у самого ряда Тейлора ошибка увеличивается по мере удаления от аргумента, вокруг которой находится ряд. Но можем решить следующим образом. Так как функция cos(x) – периодическая функция, а это значит значение функции в каком то точке совпадает с точкой меньше на 2π, мы можем найти самый ближнюю точку в ряда, у которого ответ совпадает.

double PI2 = Math.*PI* \* 2;  
  
if (x >= 0){  
 while(x > PI2){  
 x-= 2 \* PI2;  
 }  
} else {  
 while(x < PI2){  
 x += PI2;  
 }  
}

Если радианы больше чем 2π, то уменьшить его на 2π, если меньше то наоборот.

Graphical user interface, text

Description automatically generated

## Counting sort

package org.Roclh;  
  
import java.util.Arrays;  
  
public class CountingSort {  
  
 public static int[] sort(int[] arr){  
 int max = Arrays.*stream*(arr).max().orElse(0);  
 int[] count = new int[max + 1];  
 int[] sortedArr = new int[arr.length];  
  
 // Count the occurrences of each element  
 for (int j : arr) {  
 count[j]++;  
 }  
  
 // Accumulate the counts to determine the position of each element in the sorted array  
 for (int i = 1; i < count.length; i++) {  
 count[i] += count[i - 1];  
 }  
  
 // Build the sorted array  
 for (int i = arr.length - 1; i >= 0; i--) {  
 sortedArr[--count[arr[i]]] = arr[i];  
 }  
  
 return sortedArr;  
 }  
}

Counting sort - посчитать, сколько раз встречается каждый элемент в массиве, а потом заполнить исходный массив результатами этого подсчёта.

Проверка происходит по средством собственноручно составленныму массивами и через генератор на сайте <https://www.cs.usfca.edu/~galles/visualization/CountingSort.html> выданный по варианту. Также нужно не забыть проверить пустой массив, отрицательные числа и повторяющиеся значения.

### Исходный код тестирования

package org.Roclh;  
  
import org.junit.Test;  
import org.junit.jupiter.api.DisplayName;  
import org.junit.jupiter.params.ParameterizedTest;  
import org.junit.jupiter.params.provider.EmptySource;  
import org.junit.jupiter.params.provider.NullSource;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class SortTest {  
  
 @Test  
 @DisplayName("Checking only positive values")  
 public void checkPositiveSorting(){  
 *assertAll*(  
 () -> *assertArrayEquals*(new int[]{3, 5, 7, 23, 32, 32}, CountingSort.*sort*(new int[]{32, 7, 5, 23, 3, 32})),  
 () -> *assertArrayEquals*(new int[]{1,2,3,4,5,6,7,8,9,10}, CountingSort.*sort*(new int[]{1, 2, 9, 5, 4, 8, 7, 3, 6, 10})),  
 () -> *assertArrayEquals*(new int[]{2,3,3,4,6,7,8,9,10,11,12,13,13,15,16,17,18,19,19,20,20,21,22,23,23,23,23,27,28,28},  
 CountingSort.*sort*(new int[]{17, 18, 2,13,19,23, 10, 23, 28, 4, 23, 11, 16, 3, 27, 22, 20,3, 13, 19,8, 21, 12, 9, 20, 28 ,7, 6 ,23, 15}))  
 );  
 }  
  
 @Test  
 @DisplayName("Checking only negative values")  
 public void checkNegativeSorting(){  
 *assertAll*(  
 () -> *assertArrayEquals*(new int[]{-200, -100, -75, -50, -20}, CountingSort.*sort*(new int[]{ -75,-100, -50, -200, -20})),  
 () -> *assertArrayEquals*(new int[]{-10, -9, -8, -7, -6, -5, -4, -3, -2, -1}, CountingSort.*sort*(new int[]{-1, -2, -3, -4, -5, -6, -7, -8, -9, -10})),  
 () -> *assertArrayEquals*(new int[]{-300, -199, -197, -196, -43, -1}, CountingSort.*sort*(new int[]{-300, -199, -197, -196, -43, -1}))  
 );  
 }  
  
 @Test  
 @DisplayName("Checking both")  
 public void checkHybridSorting(){  
 *assertAll*(  
 () -> *assertArrayEquals*(new int[]{-1000, -200, 0, 200, 100}, CountingSort.*sort*(new int[]{0, 100, -100, 200, -200, -1000})),  
 () -> *assertArrayEquals*(new int[]{-5,-4,-3,-2,-1,0,1,2,3,4,5}, CountingSort.*sort*(new int[]{5,4,3,2,1,0,-1,-2,-3,-4,-5}))  
 );  
 }  
  
 @Test  
 @DisplayName("Check zeroes")  
 public void checkZeroes(){  
 *assertArrayEquals*(new int[]{0, 0, 0, 0, 0}, CountingSort.*sort*(new int[]{0, 0, 0, 0, 0}));  
 }  
  
 @ParameterizedTest  
 @EmptySource  
 @DisplayName("Check empty")  
 public void checkEmpty(int[] array){  
 *assertArrayEquals*(array, CountingSort.*sort*(new int[]{}));  
 }  
  
 @ParameterizedTest  
 @NullSource  
 @DisplayName("Check null")  
 public void checkNull(int[] array){  
 *assertThrows*(NullPointerException.class, ()-> CountingSort.*sort*(null));  
 }  
  
}

### Вывод тестирования:

Graphical user interface, text

Description automatically generated

Как мы видим наш алгоритм не справляется с отрицательными числами. Это не удивительно так как наш алгоритм сортирует только положительные числа. Но этого можно исправить модифицируя исходный код.

### Исправленный исходный код:

package org.Roclh;  
  
import java.util.Arrays;  
  
public class CountingSort {  
  
 public static int[] sort(int[] arr){  
 int min = Arrays.*stream*(arr).min().orElse(0);  
 int max = Arrays.*stream*(arr).max().orElse(0);  
 int range = max - min + 1;  
 int[] count = new int[range];  
 int[] sortedArr = new int[arr.length];  
  
 // Count the occurrences of each element  
 for (int j : arr) {  
 count[j - min]++;  
 }  
  
 // Accumulate the counts to determine the position of each element in the sorted array  
 for (int i = 1; i < count.length; i++) {  
 count[i] += count[i - 1];  
 }  
  
 // Build the sorted array  
 for (int i = arr.length - 1; i >= 0; i--) {  
 sortedArr[--count[arr[i] - min]] = arr[i];  
 }  
  
  
 return sortedArr;  
 }  
}

### Вывод тестирования:

Graphical user interface, text

Description automatically generated with medium confidence

Как мы видим все работает корректно.