МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

# по дисциплине «Объектно-ориентированное проектирование и программирование»

на тему: «Моделирование и алгоритмизация как средства проектирования программного обеспечения»

Выполнил студент гр. ИТИ-21:  
Гаращук И.В.  
Принял преподаватель:  
 Гуменников Е.Д.

Гомель 2023

Цель работы: научиться создавать алгоритмы и для них писать тесты, обрабатывать все участки кода, для выполнения большего покрытия кода. Создание блок-схем алгоритма.

Задание:

1. Описать блок-схему алгоритма согласно варианта (таблица 3).

2. На основе описанной блок-схемы реализовать алгоритм средствами языка Java.

3. Протестировать разработанный алгоритм при помощи модульных тестов.

4. Создать консольное приложение для демонстрации работы алгоритма.

5. Составить отчет о проделанной работе.

В таблице 1 показан вариант лабораторной работы.

Таблица – Условие

|  |  |
| --- | --- |
| № Варианта | Условие задачи |
| 5 | Поразрядная сортировка |

На рисунке 1 показана блок-схема алгоритма.

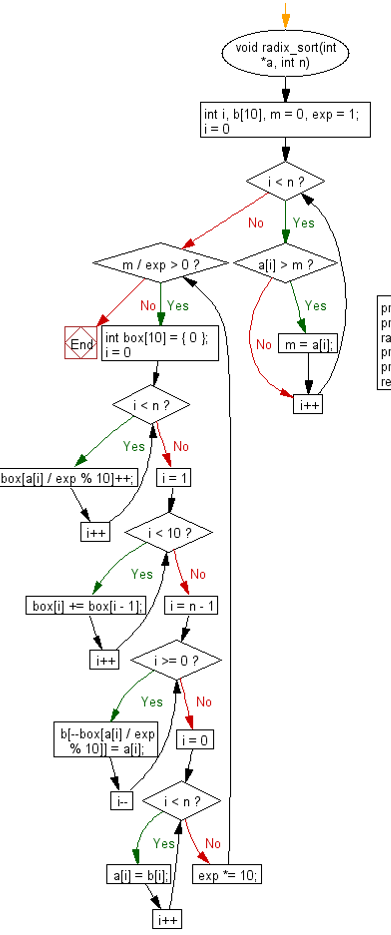


Рисунок 1 – Блок-схема алгоритма

Практическая часть:

Поразрядная сортировка – это алгоритм сортировки, который может быть выполнен за определённый временной интервал. Алгоритм поразрядной сортировки достаточно популярен, но не все существующие его программные реализации оптимальны по структуре и эффективности. Многие авторы выполняют его реализацию с расчётом на практически универсальное применение, что приводит к значительному снижению быстродействия.

Алгоритм сначала разделяет отрицательные и неотрицательные числа на два списка. Отрицательные числа сортируются по убыванию, а неотрицательные - по возрастанию. Затем неотрицательные числа сортируются поразрядно в возрастающем порядке.

На рисунке 2 показана результат сортировки.

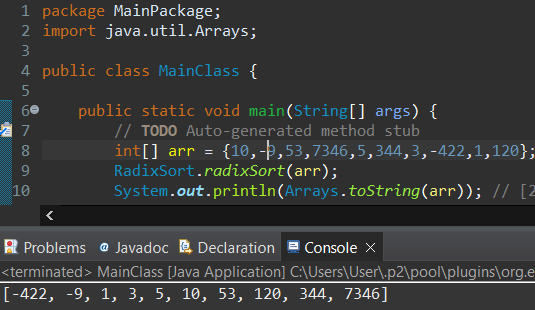


Рисунок 2 – Поразрядная сортировка

На рисунке 3 показаны тесты, и процент их выполнения.

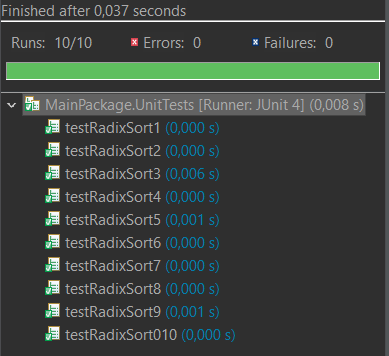


Рисунок 3 – Тесты

Вывод: в результате лабораторной работы научились обрабатывать заданные алгоритмы так, чтобы в коде не было уязвимых мест, где может возникнуть ошибка, делали это при помощи UnitTest’ов.

**ПРИЛОЖЕНИЕ А**

**Текст программы**

**MainClass.java:**

package MainPackage;

import java.util.Arrays;

public class MainClass {

public static void main(String[] args) {

// TODO Auto-generated method stub

int[] arr = {10,-9,53,7346,5,344,3,-422,1,120};

RadixSort.radixSort(arr);

System.out.println(Arrays.toString(arr));

}

}

**RadixSort.java:**

package MainPackage;

import java.util.ArrayList;

import java.util.List;

import java.util.Arrays;

import java.util.Comparator;

public class RadixSort {

public static void radixSort(int[] input) {

// separate negative and non-negative numbers

List<Integer> negativeNumbers = new ArrayList<>();

List<Integer> nonNegativeNumbers = new ArrayList<>();

for (int i : input) {

if (i < 0) {

negativeNumbers.add(i);

} else {

nonNegativeNumbers.add(i);

}

}

// sort negative numbers in ascending order

if (!negativeNumbers.isEmpty()) {

int[] negativeArray = new int[negativeNumbers.size()];

for (int i = 0; i < negativeArray.length; i++) {

negativeArray[i] = negativeNumbers.get(i);

}

Integer[] boxedArr = Arrays.stream(negativeArray).boxed().toArray(Integer[]::new);

Arrays.sort(boxedArr, Comparator.reverseOrder());

negativeArray = Arrays.stream(boxedArr).mapToInt(Integer::intValue).toArray();

for (int i = 0; i < negativeArray.length / 2; i++) {

int temp = negativeArray[i];

negativeArray[i] = negativeArray[negativeArray.length - i - 1];

negativeArray[negativeArray.length - i - 1] = temp;

}

negativeNumbers.clear();

for (int i : negativeArray) {

negativeNumbers.add(i);

}

}

// sort non-negative numbers in ascending order using radix sort

List<Integer>[] buckets = new ArrayList[10];

for (int i = 0; i < buckets.length; i++) {

buckets[i] = new ArrayList<Integer>();

}

int divisor = 1;

boolean flag = false;

while (!flag) {

flag = true;

for (int i : nonNegativeNumbers) {

int digit = (i / divisor) % 10;

buckets[digit].add(i);

if (flag && digit > 0) {

flag = false;

}

}

int index = 0;

for (List<Integer> bucket : buckets) {

for (int i : bucket) {

nonNegativeNumbers.set(index++, i);

}

bucket.clear();

}

divisor \*= 10;

}

// combine negative and non-negative numbers into one sorted array

int index = 0;

for (int i : negativeNumbers) {

input[index++] = i;

}

for (int i : nonNegativeNumbers) {

input[index++] = i;

}

}

}

**UnitTests.java:**

package MainPackage;

import static org.junit.Assert.\*;

import org.junit.Test;

public class UnitTests {

@Test

public void testRadixSort1() {

int[] testArray = new int[] {5,2,91,24};

int[] expectedArray = new int[] {2,5,24,91};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

@Test

public void testRadixSort2() {

int[] testArray = new int[] {4,3,2,1};

int[] expectedArray = new int[] {1,2,3,4};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

@Test

public void testRadixSort3() {

int[] testArray = new int[] {255,512,-255,-512};

int[] expectedArray = new int[] {-512,-255,255,512};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

@Test

public void testRadixSort4() {

int[] testArray = new int[] {0,0,0,0};

int[] expectedArray = new int[] {0,0,0,0};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

@Test

public void testRadixSort5() {

int[] testArray = new int[] {5346,13466432,-5236491,134624,0,2359,-214939};

int[] expectedArray = new int[] {-5236491,-214939,0,2359,5346,134624,13466432};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

@Test

public void testRadixSort6() {

int[] testArray = new int[] {10,9,8,7,6,5,4,3,2,1,0};

int[] expectedArray = new int[] {0,1,2,3,4,5,6,7,8,9,10};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

@Test

public void testRadixSort7() {

int[] testArray = new int[] {5,4,3,2,1,0,-1,-2,-3,-4,-5};

int[] expectedArray = new int[] {-5,-4,-3,-2,-1,0,1,2,3,4,5};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

@Test

public void testRadixSort8() {

int[] testArray = new int[] {1,0,-1};

int[] expectedArray = new int[] {-1,0,1};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

@Test

public void testRadixSort9() {

int[] testArray = new int[] {1,1,1,1,1,1,1,1,1};

int[] expectedArray = new int[] {1,1,1,1,1,1,1,1,1};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

@Test

public void testRadixSort010() {

int[] testArray = new int[] {};

int[] expectedArray = new int[] {};

RadixSort.radixSort(testArray);

assertArrayEquals(expectedArray, testArray);

//assertEquals(expectedArray,testArray);

}

}