Report for Assignment 1

Group 2
Ava Frohna, Masha Micevska, Cecilia Terena, Irina Popovikj

Name: Leetcode
URL: https://github.com/fishercoder1534/Leetcode
Programming language: Java
Number of lines of code and the tool used to count it:

- Lizard
- 87,519 lines

```
================================================================================================
Total nloc   Avg.NLOC  AvgCCN  Avg.token   Fun Cnt  Warning cnt   Fun Rt   nloc Rt
------------------------------------------------------------------------------------------------
     87519       9.4     2.5       84.3     6667           42     0.01     0.06
Avas-MacBook-Pro-2:Leetcode avafrohna$
```

Coverage measurement

Existing tool

Tool used: Jacoco
Code

- ./gradlew test jacocoTestReport
- cd build/reports/jacoco/test/html
- open index.html

leetcode-algorithms

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.fishercoder.solutions		10%		10%	15.044	16.452	28.417	31.862	5.704	6.198	3.209	3.419
com.fishercoder.common.utils		37%		44%	98	155	294	462	41	58	3	6
com.fishercoder.common.classes		5%		7%	78	82	164	175	41	44	8	10
Total	155.402 of 174.331	10%	18.517 of 20.763	10%	15.220	16.689	28.875	32.499	5.786	6.300	3.220	3.435

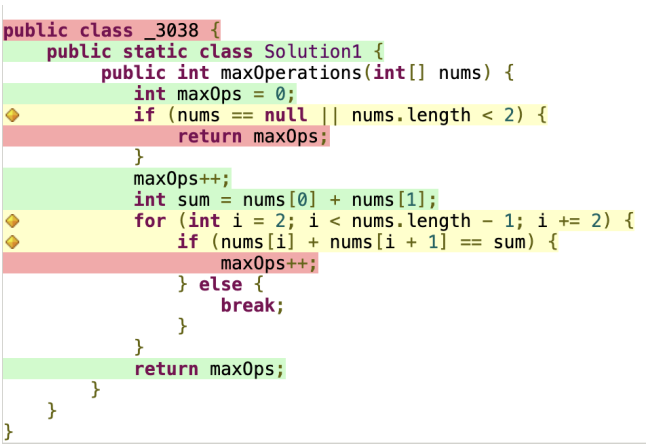
Branches in total have 10% coverage, 18,517 branches missed

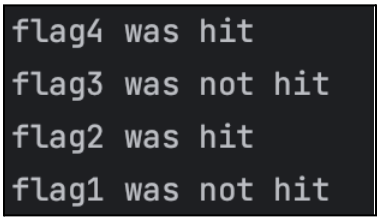
Your own coverage tool

Masha

Function 1

- maxOperations in file **_3038.java**
- Coverage: 50%, 2/4 branches

Original code	With Flags
	



- https://github.com/fishercoder1534/Leetcode/commit/415665a7479c258483cd44cf8cb72b153d1a9aa5

Function 2

- divide in **_29.java**
- Coverage: 43%, 3/7 branches

Original code

```java
public int divide(int dividend, int divisor) {
    if (dividend == Integer.MIN_VALUE && divisor == -1) {
        return Integer.MAX_VALUE;
    }
    int negativeCount = 0;
    if (dividend < 0) {
        negativeCount++;
    } else {
        dividend = -dividend;
    }
    if (divisor < 0) {
        negativeCount++;
    } else {
        divisor = -divisor;
    }

    int quotient = 0;
    while (dividend <= divisor) {
        dividend -= divisor;
        quotient++;
    }
    if (negativeCount == 1) {
        quotient = -quotient;
    }
    return quotient;
}
```

With Flags

```java
public int divide(int dividend, int divisor) {
    if (dividend == Integer.MIN_VALUE && divisor == -1) { //1
        branchCoverage.put("flag1", true);
        return Integer.MAX_VALUE;
    }
    int negativeCount = 0;
    if (dividend < 0) {  //2
        branchCoverage.put("flag2", true);
        negativeCount++;
    } else { //3
        branchCoverage.put("flag3", true);
        dividend = -dividend;
    }
    if (divisor < 0) { //4
        branchCoverage.put("flag4", true);
        negativeCount++;
    } else { //5
        branchCoverage.put("flag5", true);
        divisor = -divisor;
    }

    int quotient = 0;
    while (dividend <= divisor) { //6
        branchCoverage.put("flag6", true);
        dividend -= divisor;
        quotient++;
    }
    if (negativeCount == 1) { //7
        branchCoverage.put("flag7", true);
        quotient = -quotient;
    }
    printCoverage();
    return quotient;
}
```

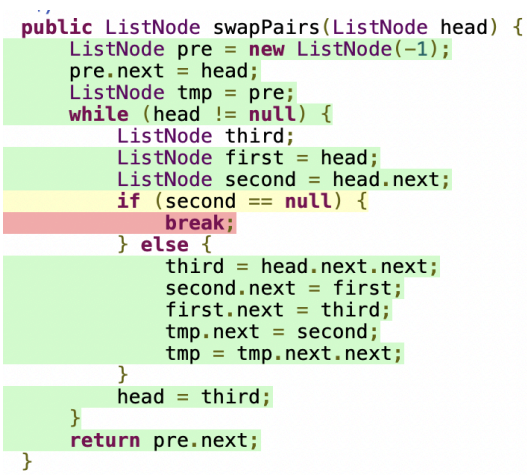
```
flag7 was not hit
flag6 was hit
flag5 was hit
flag4 was not hit
flag3 was hit
flag2 was not hit
flag1 was not hit
```

- https://github.com/fishercoder1534/Leetcode/commit/937ec78a2a4bca38cb63ebde10723838ab55c8c5

Ava

Function 1

- swapPairs in file **_24.java**
- Coverage: 66%, ⅔ branches

Original code	With Flags
	

```
public ListNode swapPairs(ListNode head) {
    ListNode pre = new ListNode(-1);
    pre.next = head;
    ListNode tmp = pre;
    while (head != null) {
        ListNode third;
        ListNode first = head;
        ListNode second = head.next;
        if (second == null) {
            break;
        } else {
            third = head.next.next;
            second.next = first;
            first.next = third;
            tmp.next = second;
            tmp = tmp.next.next;
        }
        head = third;
    }
    return pre.next;
}
```

```
private static Map<String, Boolean> branchCoverage = new HashMap<>();

static {
    branchCoverage.put("flag1", false);
    branchCoverage.put("flag2", false);
    branchCoverage.put("flag3", false);
}

public ListNode swapPairs(ListNode head) {
    ListNode pre = new ListNode(-1);
    pre.next = head;
    ListNode tmp = pre;
    while (head != null) {
        branchCoverage.put("flag1", true);
        ListNode third;
        ListNode first = head;
        ListNode second = head.next;
        if (second == null) {
            branchCoverage.put("flag2", true);
            break;
        } else {
            branchCoverage.put("flag3", true);
            third = head.next.next;
            second.next = first;
            first.next = third;
            tmp.next = second;
            tmp = tmp.next.next;
        }
        head = third;
    }
    printCoverage();
    return pre.next;
}

public void printCoverage() {
    for (Map.Entry<String, Boolean> entry : branchCoverage.entrySet()) {
        System.out.println(entry.getKey() + " was " + (entry.getValue() ? "hit" : "not hit"));
    }
}
```

```
flag3 was hit
flag2 was not hit
flag1 was hit
```

- https://github.com/fishercoder1534/Leetcode/commit/8da8f07239f26b5f02786062e781946c8926d29d

Function 2

- maxArea in file **_11.java**
- Coverage: 66%, ⅔ branches

Original code	With Flags

<table>
<tr>
<td>

```
public int maxArea(int[] height) {
    int max = 0;
    int left = 0;
    int right = height.length - 1;
    while (left < right) {
        max = Math.max(Math.min(height[left], height[right]) * (right - left), max);
        if (height[left] <= height[right]) {
            /**if this height is shorter, then we'll need to move it to the right to
            left++;
        } else {
            right--;
        }
    }
    return max;
}
```

</td>
<td>

```
public int maxArea(int[] height) {
    int max = 0;
    int left = 0;
    int right = height.length - 1;
    while (left < right) {
        branchCoverage.put("flag1", true);
        max = Math.max(Math.min(height[left], height[right]) * (right - left), max);
        if (height[left] <= height[right]) {
            /**if this height is shorter, then we'll need to move it to the right to find a hig
            branchCoverage.put("flag2", true);
            left++;
        } else {
            branchCoverage.put("flag3", true);
            right--;
        }
    }
    printCoverage();
    return max;
}

public void printCoverage() {
    for (Map.Entry<String, Boolean> entry : branchCoverage.entrySet()) {
        System.out.println(entry.getKey() + " was " + (entry.getValue() ? "hit" : "not hit"));
    }
}
```

</td>
</tr>
</table>

```
flag3 was not hit
flag2 was hit
flag1 was hit
```

- https://github.com/fishercoder1534/Leetcode/commit/877563fa16b08f393c33f6379b5a03172d14a309

Cecilia

Function 1

- lengthOfLastWord in file **_58.java**
- Coverage: 50%, ½ branches

<table>
<tr>
<td>Original code</td>
<td>With Flags</td>
</tr>
<tr>
<td>

```
public static class Solution1 {
    public int lengthOfLastWord(String s) {
        if (s == null || s.length() == 0) {
            return 0;
        }
        s = s.trim();
        int n = s.length() - 1;
        while (n >= 0 && s.charAt(n) != ' ') {
            n--;
        }
        return s.length() - n - 1;
    }
}
```

</td>
<td>

```
public static class Solution1 {
    private static Map<String, Boolean> branchCoverage = new HashMap<>();

    static {
        branchCoverage.put("flag1", false);
        branchCoverage.put("flag2", false);
    }

    public int lengthOfLastWord(String s) {
        if (s == null || s.length() == 0) {
            branchCoverage.put("flag1", true);
            return 0;
        }
        s = s.trim();
        int n = s.length() - 1;
        while (n >= 0 && s.charAt(n) != ' ') {
            branchCoverage.put("flag2", true);
            n--;
        }
        printCoverage();
        return s.length() - n - 1;
    }

    public void printCoverage() {
        for (Map.Entry<String, Boolean> entry : branchCoverage.entrySet()) {
            System.out.println(entry.getKey() + " was " + (entry.getValue() ? "hit" : "not hit"));
        }
    }
}
```

</td>
</tr>
</table>

```
flag2 was hit
flag1 was not hit
```

- https://github.com/fishercoder1534/Leetcode/commit/2b5eb376bae41c3a13ecca52ded87bd424bc4517

Function 2

- findMaxConsecutiveOnes in file **_487.java**
- Coverage: 50%, 2/4 branches

Original	With Flags

```java
public static class Solution2 {
    /**
     * This is a more generic solution adapted from https:/
     */
    public static int findMaxConsecutiveOnes(int[] nums) {
        int len = nums.length;
        int left = 0;
        int right = 0;
        int ans = 0;
        int k = 1;
        for (; right < len; right++) {
            if (nums[right] == 0) {
                k--;
            }
            while (k < 0) {
                if (nums[left] == 0) {
                    k++;
                }
                left++;
            }
            ans = Math.max(ans, right - left + 1);
        }
        return ans;
    }
}
```

```java
public static class Solution2 {
    /**
     * This is a more generic solution adapted from https://leetcode.com/problems/max-consecutive-o
     */

    private static Map<String, Boolean> branchCoverage = new HashMap<>();
    static {
        branchCoverage.put("flag1", false);
        branchCoverage.put("flag2", false);
        branchCoverage.put("flag3", false);
        branchCoverage.put("flag4", false);
    }

    public int findMaxConsecutiveOnes(int[] nums) {
        int len = nums.length;
        int left = 0;
        int right = 0;
        int ans = 0;
        int k = 1;
        for (; right < len; right++) {
            branchCoverage.put("flag1", true);
            if (nums[right] == 0) {
                branchCoverage.put("flag2", true);
                k--;
            }
            while (k < 0) {
                branchCoverage.put("flag3", true);
                if (nums[left] == 0) {
                    branchCoverage.put("flag4", true);
                    k++;
                }
                left++;
            }
            ans = Math.max(ans, right - left + 1);
        }
        printCoverage();
        return ans;
    }

    public void printCoverage() {
        for (Map.Entry<String, Boolean> entry : branchCoverage.entrySet()) {
            System.out.println(entry.getKey() + " was " + (entry.getValue() ? "hit" : "not hit"));
        }
    }
```

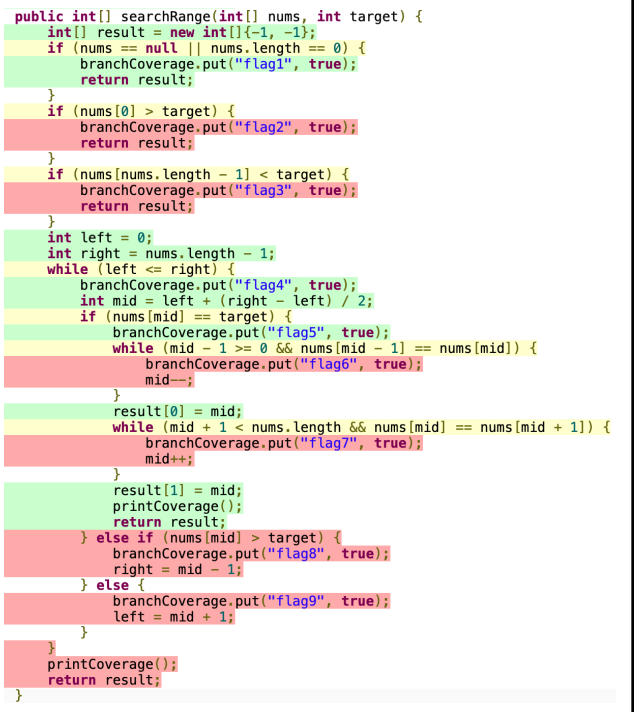
```
flag4 was not hit
flag3 was not hit
flag2 was hit
flag1 was hit
```

- https://github.com/fishercoder1534/Leetcode/commit/77bdc0599994aa2668781505c3660270ef2bda15

Irina

Function 1

- searchRange in file **_34.java**
- Coverage: 22%, 2/9 branches

Original	With Flags
	

```java
public int[] searchRange(int[] nums, int target) {
    int[] result = new int[]{-1, -1};
    if (nums == null || nums.length == 0) {
        branchCoverage.put("flag1", true);
        return result;
    }
    if (nums[0] > target) {
        branchCoverage.put("flag2", true);
        return result;
    }
    if (nums[nums.length - 1] < target) {
        branchCoverage.put("flag3", true);
        return result;
    }
    int left = 0;
    int right = nums.length - 1;
    while (left <= right) {
        branchCoverage.put("flag4", true);
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            branchCoverage.put("flag5", true);
            while (mid - 1 >= 0 && nums[mid - 1] == nums[mid]) {
                branchCoverage.put("flag6", true);
                mid--;
            }
            result[0] = mid;
            while (mid + 1 < nums.length && nums[mid] == nums[mid + 1]) {
                branchCoverage.put("flag7", true);
                mid++;
            }
            result[1] = mid;
            printCoverage();
            return result;
        } else if (nums[mid] > target) {
            branchCoverage.put("flag8", true);
            right = mid - 1;
        } else {
            branchCoverage.put("flag9", true);
            left = mid + 1;
        }
    }
    printCoverage();
    return result;
}
```

```
flag9 was not hit
flag8 was not hit
s not hit
s not hit
s hit
s hit
s not hit
s not hit
s not hit
```

```java
public int[] searchRange(int[] nums, int target) {
    int[] result = new int[]{-1, -1};
    if (nums == null || nums.length == 0) {
        return result;
    }
    if (nums[0] > target) {
        return result;
    }
    if (nums[nums.length - 1] < target) {
        return result;
    }
    int left = 0;
    int right = nums.length - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            while (mid - 1 >= 0 && nums[mid] == nums[mid - 1]) {
                mid--;
            }
            result[0] = mid;
            while (mid + 1 < nums.length && nums[mid] == nums[mid + 1]) {
                mid++;
            }
            result[1] = mid;
            return result;
        } else if (nums[mid] > target) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }
    return result;
}
```

ode/commit/3c158af548b7da0bdef53134142

java

Coverage: 86%, 92 branches

Original code	With Flags

`public int lengthOfLongestSubstring(String s) {`
`  if (s.length() == 0) {`
`  return 0;`
`  }`
`  int max = 0;`
`  int[] index = new int[128];`
`  /**Try to extend the range (i, j)*/`
`  for (int i = 0, j = 0; j < s.length(); j++) {`
`  i = Math.max(index[s.charAt(j)], i);`
`  max = Math.max(max, j - i + 1);`
`  index[s.charAt(j)] = j + 1;`
`  }`
`  return max;`
`}`
	`public int lengthOfLongestSubstring(String s) {`
	`  if (s.length() == 0) {`
	`  branchCoverage.put("flag1", true);`
	`  return 0;`
	`  }`
	`  int max = 0;`
	`  int[] index = new int[128];`
	`  /**Try to extend the range (i, j)*/`
	`  for (int i = 0, j = 0; j < s.length(); j++) {`
	`  branchCoverage.put("flag2", true);`
	`  i = Math.max(index[s.charAt(j)], i);`
	`  max = Math.max(max, j - i + 1);`
	`  index[s.charAt(j)] = j + 1;`
	`  }`
	`  printCoverage();`
	`  return max;`
	`}`

```
flag2 was hit
flag1 was not hit
```

- https://github.com/CeciTerena/Leetcode/commit/40de54e20e4726906af593d7219b64b02d7c5ba4

Coverage improvement

Individual tests

Masha

Function 1

- maxOperations in file **_3038.java**

Original coverage: 50%

```java
public class _3038 {
    public static class Solution1 {
        public int maxOperations(int[] nums) {
            int maxOps = 0;
            if (nums == null || nums.length < 2) {
                return maxOps;
            }
            maxOps++;
            int sum = nums[0] + nums[1];
            for (int i = 2; i < nums.length - 1; i += 2) {
                if (nums[i] + nums[i + 1] == sum) {
                    maxOps++;
                } else {
                    break;
                }
            }
            return maxOps;
        }
    }
}
```

```
flag4 was hit
flag3 was not hit
flag2 was hit
flag1 was not hit
```

Improved coverage: 100%

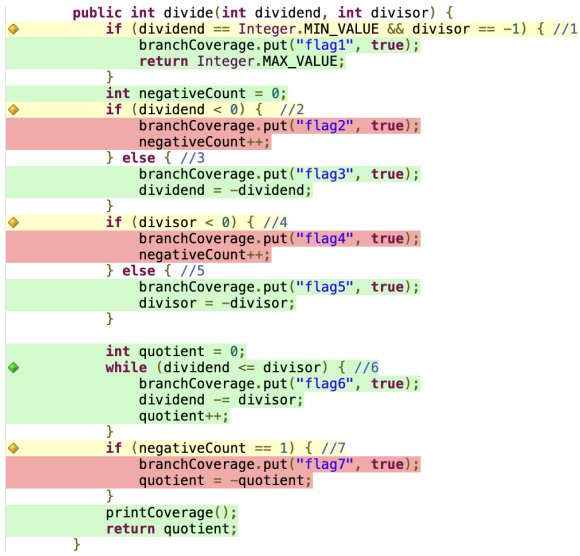
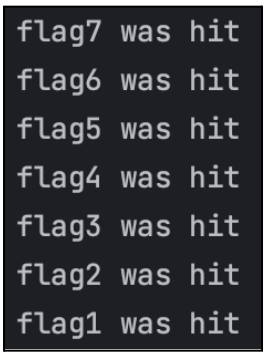
```java
public class _3038 {

    private static Map<String, Boolean> branchCoverage = new HashMap<>();
    public static class Solution1 {

        static {
            branchCoverage.put("flag1", false);
            branchCoverage.put("flag2", false);
            branchCoverage.put("flag3", false);
            branchCoverage.put("flag4", false);
        }

        public void printCoverage() {
            for (Map.Entry<String, Boolean> entry : branchCoverage.entrySet()) {
                System.out.println(entry.getKey() + " was " + (entry.getValue() ? "hit" : "not hit"));
            }
        }

        public int maxOperations(int[] nums) {
            int maxOps = 0;
            if (nums == null || nums.length < 2) { //1
                branchCoverage.put("flag1", true);
                return maxOps;
            }
            maxOps++;
            int sum = nums[0] + nums[1];
            for (int i = 2; i < nums.length - 1; i += 2) { //2
                branchCoverage.put("flag2", true);
                if (nums[i] + nums[i + 1] == sum) { //3
                    branchCoverage.put("flag3", true);
                    maxOps++;
                } else { //4
                    branchCoverage.put("flag4", true);
                    break;
                }
            }
            printCoverage();
            return maxOps;
        }
    }
}
```

```
flag4 was hit
flag3 was hit
flag2 was hit
flag1 was hit
```

- https://github.com/fishercoder1534/Leetcode/commit/38d599fd9f3fb7e724e9be7efa4396786d345ee7
- The coverage improved from 50% (2/4 branches) to 100% (4/4) branches. The first test which was already written only covered branches 2 and 4 as it considered cases where the array length was at least 3 and the case where the sum of the pair did not equal the initial sum. The second test which I added covers the situation where nums is null and the third test where nums.length is less than 2. Tests 4, 5, and 6 cover multiple branches, where first pairs have equal sums and the second does not, where both pairs have equal sums, and where all pairs have equal sums, respectively.

Function 2

- divide in file **_29.java**

Original coverage: 43%	Improved coverage: 100%
	
	

```
public int divide(int dividend, int divisor) {
    if (dividend == Integer.MIN_VALUE && divisor == -1) { //1
        branchCoverage.put("flag1", true);
        return Integer.MAX_VALUE;
    }
    int negativeCount = 0;
    if (dividend < 0) {  //2
        branchCoverage.put("flag2", true);
        negativeCount++;
    } else { //3
        branchCoverage.put("flag3", true);
        dividend = -dividend;
    }
    if (divisor < 0) { //4
        branchCoverage.put("flag4", true);
        negativeCount++;
    } else { //5
        branchCoverage.put("flag5", true);
        divisor = -divisor;
    }

    int quotient = 0;
    while (dividend <= divisor) { //6
        branchCoverage.put("flag6", true);
        dividend -= divisor;
        quotient++;
    }
    if (negativeCount == 1) { //7
        branchCoverage.put("flag7", true);
        quotient = -quotient;
    }
    printCoverage();
    return quotient;
}
```

```
flag7 was not hit
flag6 was hit
flag5 was hit
flag4 was not hit
flag3 was hit
flag2 was not hit
flag1 was not hit
```

```
public int divide(int dividend, int divisor) {
    if (dividend == Integer.MIN_VALUE && divisor == -1) { //1
        branchCoverage.put("flag1", true);
        return Integer.MAX_VALUE;
    }
    int negativeCount = 0;
    if (dividend < 0) {  //2
        branchCoverage.put("flag2", true);
        negativeCount++;
    } else { //3
        branchCoverage.put("flag3", true);
        dividend = -dividend;
    }
    if (divisor < 0) { //4
        branchCoverage.put("flag4", true);
        negativeCount++;
    } else { //5
        branchCoverage.put("flag5", true);
        divisor = -divisor;
    }

    int quotient = 0;
    while (dividend <= divisor) { //6
        branchCoverage.put("flag6", true);
        dividend -= divisor;
        quotient++;
    }
    if (negativeCount == 1) { //7
        branchCoverage.put("flag7", true);
        quotient = -quotient;
    }
    printCoverage();
    return quotient;
}
}
```

```
flag7 was hit
flag6 was hit
flag5 was hit
flag4 was hit
flag3 was hit
flag2 was hit
flag1 was hit
```

- https://github.com/fishercoder1534/Leetcode/commit/ad04c750e9b34b4f0c253af956df5101c1db1ef3

- The branch coverage improved from 43% to 100% with the addition of 6 tests. The first test covered branches 3, 5, and 6, as they covered cases where both the divided and the divisor are positive. The second test checks for a special overflow case, covering branch 1. The third test covers cases where both dividend and divisor are positive just like test 1. Test 4 checks the case where the dividend is negative but the divisor is positive, covering branches 2, 5, 6, and 7. Test 5 covers cases where the dividend is positive but the divisor is negative, covering branches 3, 4, 6, and 7. Test 6 covers cases where both numbers are negative (branches 2, 4, and 6). Test 7 covers branches 2, 5, 6, and 7 with a negative divided and positive divisor, where multiple loop iterations are required. And finally test 8 covers the case where the dividend is 0, covering branches 3 and 5.

Ava

Function 1

- swapPairs in file **_24Test.java**

<table>
<tr><td>Original coverage: 66%</td><td>Improved coverage: 100%</td></tr>
<tr><td>

```
public static class Solution2 {
    /**
     * Iterative approach:
     * My completely original on 10/24/2021.
     */
    private static Map<String, Boolean> branchCoverage = new HashMap<>();

    static {
        branchCoverage.put("flag1", false);
        branchCoverage.put("flag2", false);
        branchCoverage.put("flag3", false);
    }

    public ListNode swapPairs(ListNode head) {
        ListNode pre = new ListNode(-1);
        pre.next = head;
        ListNode tmp = pre;
        while (head != null) {
            branchCoverage.put("flag1", true);
            ListNode third;
            ListNode first = head;
            ListNode second = head.next;
            if (second == null) {
                branchCoverage.put("flag2", true);
                break;
            } else {
                branchCoverage.put("flag3", true);
                third = head.next.next;
                second.next = first;
                first.next = third;
                tmp.next = second;
                tmp = tmp.next.next;
            }
            head = third;
        }
        printCoverage();
        return pre.next;
    }

    public void printCoverage() {
        for (Map.Entry<String, Boolean> entry : branchCoverage.entrySet()) {
            System.out.println(entry.getKey() + " was " + (entry.getValue() ? "hit" : "not hit"));
        }
    }
}
```

</td><td>

```
public static class Solution2 {
    /**
     * Iterative approach:
     * My completely original on 10/24/2021.
     */
    private static Map<String, Boolean> branchCoverage = new HashMap<>();

    static {
        branchCoverage.put("flag1", false);
        branchCoverage.put("flag2", false);
        branchCoverage.put("flag3", false);
    }

    public ListNode swapPairs(ListNode head) {
        ListNode pre = new ListNode(-1);
        pre.next = head;
        ListNode tmp = pre;
        while (head != null) {
            branchCoverage.put("flag1", true);
            ListNode third;
            ListNode first = head;
            ListNode second = head.next;
            if (second == null) {
                branchCoverage.put("flag2", true);
                break;
            } else {
                branchCoverage.put("flag3", true);
                third = head.next.next;
                second.next = first;
                first.next = third;
                tmp.next = second;
                tmp = tmp.next.next;
            }
            head = third;
        }
        printCoverage();
        return pre.next;
    }

    public void printCoverage() {
        for (Map.Entry<String, Boolean> entry : branchCoverage.entrySet()) {
            System.out.println(entry.getKey() + " was " + (entry.getValue() ? "hit" : "not hit"));
        }
    }
}
```

</td></tr>
<tr><td>

```
flag3 was hit
flag2 was not hit
flag1 was hit
```

</td><td>

```
flag3 was hit
flag2 was hit
flag1 was hit
```

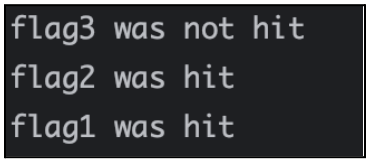
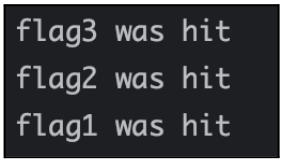
</td></tr>
</table>

- https://github.com/fishercoder1534/Leetcode/commit/26abc5dac82e73ffc3948aa6d762f74de1dc007c

- I added test 3 which during the third iteration, the head points to the last node. When head.next is set it is null since there is not another node. This causes the code to enter the if loop if (second == null) which covers the only flag that was not hit. The test has an odd number of nodes which means the last iteration of the loop will have second == null.

Function 2

- maxArea in file **_11Test.java**

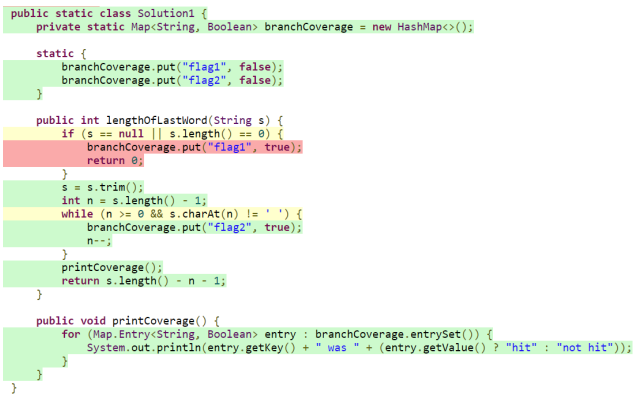
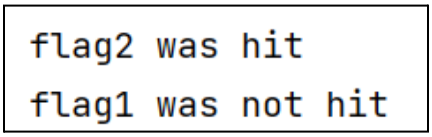
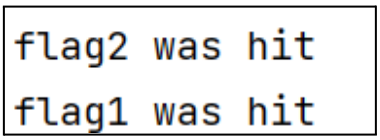
Original coverage: 66%	Improved coverage: 100%
	
	

- https://github.com/fishercoder1534/Leetcode/commit/d5c5709cf8f96d5ed360c518a1b088b4f5472459
- I added test 2 which during the third iteration of the while loop, the left has a height of 4 and the right has a height of 3. Meaning height(left) > height(right) therefore it goes to the else branch hitting flag 3 making the coverage increase from 66% to 100%.

Cecilia

Function 1:

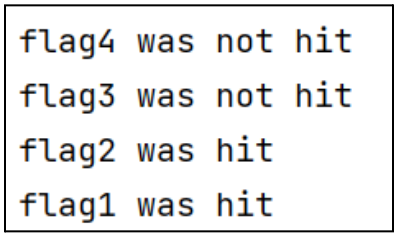
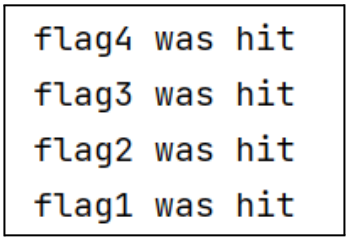
- lengthOfLastWord in file **_58.java**

Original coverage: 50%	Improved Coverage: 100%
	
	

- https://github.com/fishercoder1534/Leetcode/commit/edb5e9d9cfd865921e498ee797ce673c6cc79604
- The new test is made with a null string as the parameter passed to the lengthOfLastWord function. This means the condition of if (s = null || s.length() == 0) is satisfied and the branch is then entered.

Function 2:

- findMaxConsecutiveOnes in file **_487.java**

Original coverage: 50%	Improved Coverage: 100%
	
	

- https://github.com/fishercoder1534/Leetcode/commit/ef93c348ca8946d8059044296b71685f7e96ed36
- The code finds the maximum number of consecutive 1s in the array, with the allowance to flip at most one 0 into a 1. Through inspection, it is observed that flag 1 is always hit, for every iteration of the for loop. Flag 2 is hit only when

num[right] == 0, which causes a decrease in the parameter k. Flag 3 is hit only when k is negative, triggering the inner while loop. Flag 4 is in a branch nested in the inner while loop, which only triggers when nums[left] == 0 and it results in k being incremented by one. The sequence 1, 0, 0, 1, 1, 0, 1 is a sequence that is able to trigger all flags. The execution is as follows:

- For loop: 1st iteration.
    - Right = 0. Left = 0. Nums[right] = 1. K = 1. **Flag1 is hit** and all other flags are not hit. Ans = 1
- For loop: 2nd iteration.
    - Right = 1. Left = 0. Nums[right] = 0. K = 0. **Flag1 and 2 are hit**, the others are not. Ans = 2
- For loop: 3rd iteration.
    - Right = 2. Nums[right] = 0. K = -1. Flag1, 2 are hit
    - While 1st iteration:
        - Nums[left] = 1 **Flag 3 is hit**, flag 4 is not. Left = 1
    - While 2st iteration:
        - Nums[left] = 0 Flag 3 is hit, **flag 4 is hit**. K = 0. Left = 2
    - Ans = 2
- From there onwards the for loop continues until right = 6 (as when right = 7 the condition is not satisfied).By the end of the execution Ans = 4, which is the max number of consecutives ones if a 0 can be flipped into a 1 in the specified test array. This sample of the execution shows how all flags are hit.

Irina

Function 1

- searchRange in file **_34.java**

Original coverage: 22%	Improved coverage: 100%

```java
public int[] searchRange(int[] nums, int target) {
    int[] result = new int[]{-1, -1};
    if (nums == null || nums.length == 0) {
        branchCoverage.put("flag1", true);
        return result;
    }
    if (nums[0] > target) {
        branchCoverage.put("flag2", true);
        return result;
    }
    if (nums[nums.length - 1] < target) {
        branchCoverage.put("flag3", true);
        return result;
    }
    int left = 0;
    int right = nums.length - 1;
    while (left <= right) {
        branchCoverage.put("flag4", true);
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            branchCoverage.put("flag5", true);
            while (mid - 1 >= 0 && nums[mid - 1] == nums[mid]) {
                branchCoverage.put("flag6", true);
                mid--;
            }
            result[0] = mid;
            while (mid + 1 < nums.length && nums[mid] == nums[mid + 1]) {
                branchCoverage.put("flag7", true);
                mid++;
            }
            result[1] = mid;
            printCoverage();
            return result;
        } else if (nums[mid] > target) {
            branchCoverage.put("flag8", true);
            right = mid - 1;
        } else {
            branchCoverage.put("flag9", true);
            left = mid + 1;
        }
    }
    printCoverage();
    return result;
}
```

```
flag9 was not hit
flag8 was not hit
flag7 was not hit
flag6 was not hit
flag5 was hit
flag4 was hit
flag3 was not hit
flag2 was not hit
flag1 was not hit
```

```java
public int[] searchRange(int[] nums, int target) {
    int[] result = new int[]{-1, -1};
    if (nums == null || nums.length == 0) {
        branchCoverage.put("flag1", true);
        return result;
    }
    if (nums[0] > target) {
        branchCoverage.put("flag2", true);
        return result;
    }
    if (nums[nums.length - 1] < target) {
        branchCoverage.put("flag3", true);
        return result;
    }
    int left = 0;
    int right = nums.length - 1;
    while (left <= right) {
        branchCoverage.put("flag4", true);
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            branchCoverage.put("flag5", true);
            while (mid - 1 >= 0 && nums[mid - 1] == nums[mid]) {
                branchCoverage.put("flag6", true);
                mid--;
            }
            result[0] = mid;
            while (mid + 1 < nums.length && nums[mid] == nums[mid + 1]) {
                branchCoverage.put("flag7", true);
                mid++;
            }
            result[1] = mid;
            return result;
        } else if (nums[mid] > target) {
            branchCoverage.put("flag8", true);
            right = mid - 1;
        } else {
            branchCoverage.put("flag9", true);
            left = mid + 1;
        }
    }
    printCoverage();
    return result;
}
```

```
flag9 was hit
flag8 was hit
flag7 was hit
flag6 was hit
flag5 was hit
flag4 was hit
flag3 was hit
flag2 was hit
flag1 was hit
```

- https://github.com/CeciTerena/Leetcode/commit/5efbbb2a7eb0b4a7042d65a0792ca0ec55f3af6d
- Due to the large amount of flags that needed to be hit, I added four tests. The first test covers the scenario where the target is less than the smallest element in the array. It hits flag2. The next test covers the scenario where the target is greater than the largest element in the array. It hits flag3. The third test covers the scenario where the target is found and spans multiple elements. It hits flag4, flag5, flag6, and flag7. The last test I added covers the scenario where the target is not found but is within the range of the array. It hits flag4, flag8, and flag9.

Function 2

- lengthOfLongestSubstring in file **_3.java**

Original coverage: 50%	Improved coverage: 100%

`public int lengthOfLongestSubstring(String s) {`
`  if (s.length() == 0) {`
`  branchCoverage.put("flag1", true);`
`  return 0;`
`  }`
`  int max = 0;`
`  int[] index = new int[128];`
`  /**Try to extend the range (i, j)*/`
`  for (int i = 0, j = 0; j < s.length(); j++) {`
`  branchCoverage.put("flag2", true);`
`  i = Math.max(index[s.charAt(j)], i);`
`  max = Math.max(max, j - i + 1);`
`  index[s.charAt(j)] = j + 1;`
`  }`
`  printCoverage();`
`  return max;`
`}`
	`public int lengthOfLongestSubstring(String s) {`
	`  if (s.length() == 0) {`
	`  branchCoverage.put("flag1", true);`
	`  return 0;`
	`  }`
	`  int max = 0;`
	`  int[] index = new int[128];`
	`  for (int i = 0, j = 0; j < s.length(); j++) {`
	`  branchCoverage.put("flag2", true);`
	`  i = Math.max(index[s.charAt(j)], i);`
	`  max = Math.max(max, j - i + 1);`
	`  index[s.charAt(j)] = j + 1;`
	`  }`
	`  printCoverage();`
	`  return max;`
	`}`

`flag2 was hit` `flag1 was not hit`	`flag2 was hit` `flag1 was hit`

- https://github.com/CeciTerena/Leetcode/commit/6711711e0a42abc3ee7c12006effd140cf3fd285
- I added one test to hit flag1 which verifies that the method handles edge cases where the input string is empty and confirms that the method returns the correct output (0).

Overall

Old coverage

Branches in total have 10% coverage, 18,517 branches missed

leetcode-algorithms

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.fishercoder.solutions		10%		10%	15.044	16.452	28.417	31.862	5.704	6.198	3.209	3.419
com.fishercoder.common.utils		37%		44%	98	155	294	462	41	58	3	6
com.fishercoder.common.classes		5%		7%	78	82	164	175	41	44	8	10
Total	155.402 of 174.331	10%	18.517 of 20.763	10%	15.220	16.689	28.875	32.499	5.786	6.300	3.220	3.435

New coverage

Branches in total have 11% coverage, 18,494 branches missed

leetcode-algorithms

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.fishercoder.solutions		10%		10%	15,023	16,485	28,395	31,988	5,704	6,215	3,206	3,419
com.fishercoder.common.utils		37%		44%	98	155	294	462	41	58	3	6
com.fishercoder.common.classes		5%		7%	78	82	164	175	41	44	8	10
Total	155,350 of 175,070	11%	18,494 of 20,795	11%	15,199	16,722	28,853	32,625	5,786	6,317	3,217	3,435

Statement of individual contributions

Ava

I contributed to this group project by checking the lines of code with the lizard function and running the overall coverage with the Jacoco tool. I also chose two functions, added flags to check the coverage of individual functions, and then added tests to ensure all flags were hit during the testing run. I was able to take both my functions which started at 66% to 100%.

Masha

I contributed to this project by inspecting two functions, adding flags to calculate the branch coverage, and then adding tests to each function in order to improve the coverage of the functions. Both functions had branch coverage below 50% and the additional tests improved the coverage to 100%.

Cecilia

I contributed by finding a suitable repository. In addition, I inspected two functions, adding flags to check their coverage, as well as added tests to improve the coverage of these two functions which took their coverage from 50% each to 100%.

Irina

I contributed to the project by inspecting two functions and adding appropriate flags to calculate their branch coverage. I then added tests to the functions to increase their coverage from 22% and 50% to 100%.