***Обработка алгоритмов.***

*Задача 1.*

**11. Container With Most Water** ресурса https://leetcode.com /

Given n non-negative integers a1, a2, ..., an, where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of the line i is at (i, ai) and (i, 0). Find two lines, which, together with the x-axis forms a container, such that the container contains the most water.

**Notice** that you may not slant the container.



**Input:** height = [1,8,6,2,5,4,8,3,7]

**Output:** 49

**Explanation:** The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7].

In this case, the max area of water (blue section) the container can contain is 49.

**Example 2:**

**Input:** height = [1,1]

**Output:** 1

*Идея алгоритма*.

*Считаем площадь воды, двигаясь с левого и правого конца всех столбиков, навстречу друг другу.*

*Поскольку площадь воды считается по меньшему столбику, после подсчета площади передвигаемся на следующий столбик со стороны меньшего столбика. Больший столбик остается на месте. Считаем площадь и сравниваем с предыдущим ее значением. Затем опять передвигаемся на следующий столбик со стороны меньшего, больший на местеданном. Считаем и сравниваем площади до тех пор , пока левая и правая граница столбиков не совпадут.*

*Меньший по высоте столбик после подсчета площади все время отбрасываем, поскольку его площадь со всеми оставшимися, не участвующими с ним в данном подсчете столбиками, будет наверняка меньше.*

Ниже приведена реализация алгоритма на ***java.***

class Solution {

public int maxArea(int[] height) {

int i = 0;

int j = height.length - 1;

int left = height[i];

int right = height[j];

int value = 0;

while (i < j) {

if (left < right) {

if ((j - i) \* left > value) {

value = (j - i) \* left;

}

i++;

left = height[i];

} else {

if ((j - i) \* right > value) {

value = (j - i) \* right;

}

j--;

right = height[j];

}

}

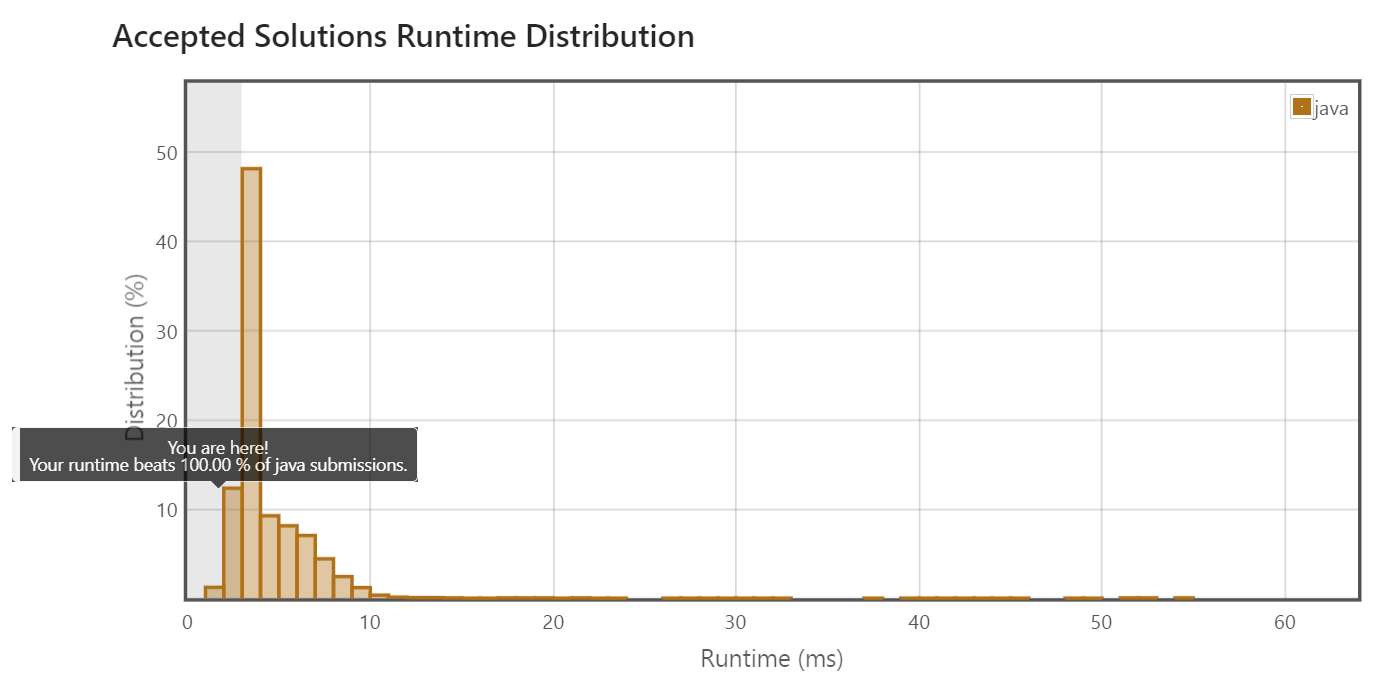
return value;

}

}

*Подсчет площадей и их сравнение на каждой итерации цикла для нашего примера выглядит так:*

1. *8 > 0;*
2. *49 > 8;*
3. *49>18;*
4. *49 > 40;*
5. *49 > 16;*
6. *49 > 15;*
7. *49 > 4;*
8. *49 > 6.*



***Описание алгоритма***

1. Устанавливаем индексы на *крайние элементы* массива, левый на *i* , правый – на *j*, соответственно, *left* и *right –* значения элементов для левого и правого индекса в массиве.
2. Начальное значение искомой величины *value* припавниваем к *нулю*.
3. Определяем, какое из значений *left* или *right* *меньше* (*высота столбика*) и для него вычисляем площадь воды.
4. Сравниваем полученную в пункте *4* площадь с вычисленной на предыдущей итерации. При необходимости меняем значение *value.*
5. Если выполняется неравенство *left* < *right*, увеличиваем *i* на единицу, в противном случае – уменьшаем *j* на единицу. Двигаем индексы фактически навстречу друг другу.
6. Повторяем п.п. *3-5* пока выполняется условие *i < j,* по окончанию цикла получаем искомое значение величины *value*.

***Вопросы по обработке алгоритма.***

1. Чему равна *временная сложнось* алгоритма?
2. Произвести подсчет площадей и их сравнение на каждой итерации цикла для массива значений: [5, 2, 7, 8, 4, 6, 3].
3. В каких алгоритмах еще используется идея переноса границ?

*Задача 2.*

**200. Number of Islands** ресурса https://leetcode.com /.

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example 1:**

**Input:** grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

**Output:** 1

**Example 2:**

**Input:** grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

["0","0","0","1","1"]

]

**Output:** 3

Для решения этой задачи используется *рекурсивный обход* ячеек острова. При нахождении первой *«1»*, вызывается рекурсивная функция для обхода ячеек bypassGraph(i, j, grid), значение ячейки меняется на *«2»*, начинаем последовательно обрабатывать соседние ячейки. Первой обрабатываем ячейку внизу, для нее вновь вызываем нашу рекурсивную функцию, если значение ее также равно *«1»*, меняем его на *«2»* и начинаем рекурсивно обходить ее соседние ячейки. И так обрабатываем все ячейки острова.

Ниже приведена реализация алгоритма на ***java.***

class Solution {

public int numIslands(char[][] grid) {

int islandCount = 0;

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

for (int j = 0; j < grid[0].length; j++) {

if (grid[i][j] == '1') {

bypassGraph(i, j, grid);

islandCount++;

}

}

}

return islandCount;

}

private void bypassGraph(int i, int j, char[][] grid) {

if (grid[i][j] == '1') {

grid[i][j] = '2';

} else {

return;

}

if (i + 1 < grid.length) {

bypassGraph(i + 1, j, grid);

}

if (j + 1 < grid[0].length) {

bypassGraph(i, j + 1, grid);

}

if (i - 1 >= 0) {

bypassGraph(i - 1, j, grid);

}

if (j - 1 >= 0) {

bypassGraph(i, j - 1, grid);

}

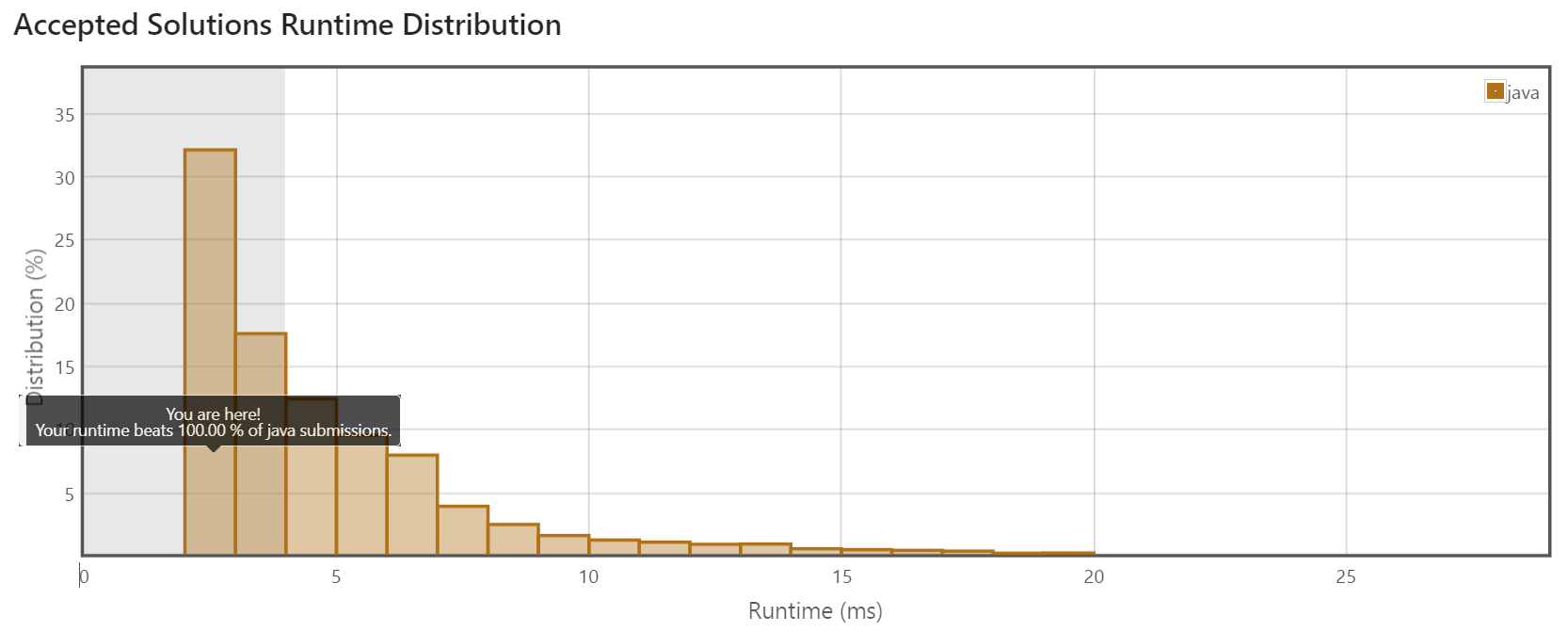
}

}

Ниже, в таблице справа указан порядок захода в ячейки для острова справа.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 |  |
| 1 | 1 |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 9 | 7 | 8 |
| 2 | 5 | 6 |  |
| 3 | 4 |  |  |
|  |  |  |  |



***Вопросы по обработке алгоритма.***

1. Чему равна *временная сложнось* алгоритма?
2. Какой алгоритм на *графах* мы использовали в этой задаче?
3. Указать порядок обработки ячеек для острава ниже.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1 | 1 |  |
|  | 1 | 1 |  |
| 1 |  |  |  |
| 1 | 1 |  |  |

