

Data Structures & Algorithms Assignment-4

Question: <https://leetcode.com/problems/valid-parentheses/description/>

Answer: <https://leetcode.com/problems/valid-parentheses/submissions/1628647515>

Description:

Time Complexity: $O(n)$

- Where n is the length of the input string s .
- Each character is processed exactly once: either pushed or popped from the stack.

Space Complexity: $O(n)$

- In the worst case, all characters are opening brackets and get pushed onto the stack.
- So, the space used by the stack grows linearly with the size of the input.

Screenshot:

The screenshot shows a LeetCode problem page for "20. Valid Parentheses". The problem description states: "Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if: 1. Open brackets must be closed by the same type of brackets. 2. Open brackets must be closed in the correct order. 3. Every close bracket has a corresponding open bracket of the same type." Examples provided are: Example 1: Input: $s = "()"$, Output: true; Example 2: Input: $s = "()[]{}"$, Output: true; Example 3: Input: $s = "(]"$, Output: false. The solution is implemented in JavaScript using a stack. The code is as follows:

```
1 /**
2  * @param {string} s
3  * @return {boolean}
4  */
5 var isValid = function(s) {
6     const stack = []; // Stack for opening brackets
7     const hash = { '(': ')', '[': ']', '{': '}' }; // Matching pairs
8
9     for (const char of s) {
10         if (char in hash) {
11             // Check if stack is non-empty and matches the top element
12             if (stack.length && stack[stack.length - 1] === hash[char]) {
13                 stack.pop(); // Remove the matching opening bracket
14             } else {
15                 return false; // Invalid if no match
16             }
17         } else {
18             stack.push(char); // Push opening bracket
19         }
20     }
21     return stack.length === 0; // Valid if stack is empty
22 };
```

The solution is marked as "Accepted" with a runtime of 0 ms. The test results show that all four test cases passed.

Question: <https://leetcode.com/problems/next-greater-element-i/description/>

Answer: <https://leetcode.com/problems/next-greater-element-i/submissions/1628654519>

Description:

Time Complexity: $O(n)$

- n = length of `nums2`
- m = length of `nums1`
- First loop (over `nums2`): Each element is pushed and popped from the stack at most once $\rightarrow O(n)$
- Second loop (over stack): Fills in -1 for elements with no greater element \rightarrow at most $O(n)$
- Final map loop (over `nums1`): Builds the result array $\rightarrow O(m)$

Total: $O(n + m)$

Space Complexity: $O(n)$

- map stores a key-value pair for each element in `nums2` $\rightarrow O(n)$
- stack also holds up to n elements in the worst case $\rightarrow O(n)$
- Output array is of size m , but typically not counted against space complexity since it's the return value.

Total auxiliary space: $O(n)$

Screenshot:

The screenshot shows the LeetCode interface for the problem "496. Next Greater Element I". The problem description is on the left, and the code editor is on the right. The code is written in JavaScript and implements the solution using a stack and a map.

Problem Description:

The **next greater element** of some element x in an array is the **first greater** element that is **to the right of** x in the same array.

You are given two **distinct 0-indexed** integer arrays `nums1` and `nums2`, where `nums1` is a subset of `nums2`.

For each $0 \leq i < \text{nums1.length}$, find the index j such that `nums1[i] == nums2[j]` and determine the **next greater element** of `nums2[j]` in `nums2`. If there is no next greater element, then the answer for this query is `-1`.

Return an array `ans` of length `nums1.length` such that `ans[i]` is the **next greater element** as described above.

Example 1:

Input: `nums1 = [4,1,2]`, `nums2 = [1,3,4,2]`
Output: `[-1,3,-1]`
Explanation: The next greater element for each value of `nums1` is as follows:
- 4 is underlined in `nums2 = [1,3,4,2]`. There is no next greater element, so the answer is `-1`.
- 1 is underlined in `nums2 = [1,3,4,2]`. The next greater element is 3.
- 2 is underlined in `nums2 = [1,3,4,2]`. There is no next greater element, so the answer is `-1`.

Example 2:

Input: `nums1 = [2,4]`, `nums2 = [1,2,3,4]`

Code Editor:

```
1 /**
2  * @param {number[]} nums1
3  * @param {number[]} nums2
4  * @return {number[]}
5  */
6 var nextGreaterElement = function(nums1, nums2) {
7     const stack = [], map = {};
8     for (let n of nums2) {
9         while (stack.length && n > stack[stack.length - 1]) {
10             map[stack.pop()] = n;
11         }
12         stack.push(n);
13     }
14     for (let n of stack) {
15         map[n] = -1;
16     }
17     return nums1.map(n => map[n]);
18 };
```

The code is saved and the test result is shown as "Accepted" with a runtime of 0 ms.

Question: <https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string/description/>

Answer: <https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string/submissions/1628655814>

Description:

Time Complexity: $O(n)$

- n is the length of the input string s .
- Each character is pushed and popped at most once, so total operations are linear.

Space Complexity: $O(n)$

- In the worst case (e.g., no duplicates), all characters go into the stack.
- So, space used by the stack is up to n .

Screenshot:

The screenshot shows the LeetCode interface for problem 1047. The left panel displays the problem description, which involves removing adjacent duplicate characters from a string. The right panel shows a JavaScript solution using a stack. The code iterates through each character of the string, and if it matches the top of the stack, it is popped; otherwise, it is pushed. The final result is the string formed by the stack's contents.

1047. Remove All Adjacent Duplicates In String Solved

Easy Topics Companies Hint

You are given a string s consisting of lowercase English letters. A **duplicate removal** consists of choosing two **adjacent** and **equal** letters and removing them.

We repeatedly make **duplicate removals** on s until we no longer can.

Return the **final string** after all such **duplicate removals** have been made. It can be proven that the answer is **unique**.

Example 1:

Input: $s = "abbaca"$
Output: $"ca"$
Explanation: For example, in $"abbaca"$ we could remove $"bb"$ since the letters are adjacent and equal, and this is the only possible move. The result of this move is that the string is $"aaca"$, of which only $"aa"$ is possible, so the final string is $"ca"$.

Example 2:

Input: $s = "azxxzy"$
Output: $"ay"$

```
1 /**  
2  * @param {string} s  
3  * @return {string}  
4  */  
5 var removeDuplicates = function(s) {  
6     let stack = [];  
7  
8     for (let c of s) {  
9         if (stack.length && stack[stack.length - 1] === c) {  
10             stack.pop(); // Remove the duplicate  
11         } else {  
12             stack.push(c); // Add new character  
13         }  
14     }  
15  
16     return stack.join(""); // Convert stack back to string  
17 };
```

Accepted Runtime: 0 ms

Case 1 Case 2

Question: <https://leetcode.com/problems/trapping-rain-water/description/>

Answer: <https://leetcode.com/problems/trapping-rain-water/submissions/1628657003>

Description:

Time Complexity: $O(n)$

- The array is scanned once using two pointers (left and right).
- Each index is visited at most once.

Total: $O(n)$, where n is the length of the height array

Space Complexity: $O(1)$

- No extra data structures are used that grow with input size.
- Just a few variables (left, right, lm, rm, total).

Screenshot:

The screenshot displays the LeetCode web application. On the left, the problem '42. Trapping Rain Water' is shown with its description, example 1 (a bar chart with heights [0,1,0,2,1,0,1,3,2,1,2,1] and 6 units of water trapped), and example 2 (heights [4,2,0,3,2,5]). The right panel shows a JavaScript solution using two pointers (left and right) and variables (lm, rm, total) to calculate the trapped water. The code is as follows:

```
1 /**
2  * @param {number[]} height
3  * @return {number}
4  */
5 var trap = function(height) {
6     let left = 0, right = height.length - 1;
7     let total = 0, lm = 0, rm = 0;
8
9     while (left <= right) {
10         if (lm < rm) {
11             if (height[left] < lm) {
12                 total += lm - height[left];
13             } else {
14                 lm = height[left];
15             }
16             left++;
17         } else {
18             if (height[right] < rm) {
19                 total += rm - height[right];
20             } else {
21                 rm = height[right];
22             }
23             right--;
24         }
25     }
26     return total;
27 }
```

The bottom of the interface shows the 'Testcase' tab with 'Accepted' status and 'Runtime: 0 ms'.

Question: <https://leetcode.com/problems/largest-rectangle-in-histogram/description/>

Answer: <https://leetcode.com/problems/largest-rectangle-in-histogram/submissions/1628657813>

Description:

Time Complexity: $O(n)$

- Each bar is pushed and popped from the stack once.
- All operations inside the loop are constant time.
- So total work is proportional to n .

Total: $O(n)$, where n is the number of bars in heights.

Space Complexity: $O(n)$

- The stack stores up to n indices in the worst case.
- No other significant memory usage.

Total: $O(n)$ auxiliary space.

Screenshot:

The screenshot shows the LeetCode interface for problem 84, "Largest Rectangle in Histogram". The problem description states: "Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram." An example is provided with heights = [2,1,5,6,2,3] and output = 10. The explanation notes that the largest rectangle is shown in red in the diagram, with an area of 10 units.

The code editor shows a JavaScript solution using a stack. The code is as follows:

```
var largestRectangleArea = function(heights) {
    const stack = [-1];
    let max_area = 0;

    for (let i = 0; i < heights.length; i++) {
        while (stack[stack.length - 1] !== -1 && heights[i] <= heights[stack[stack.length - 1]]) {
            const height = heights[stack.pop()];
            const width = i - stack[stack.length - 1] - 1;
            max_area = Math.max(max_area, height * width);
        }
        stack.push(i);
    }

    while (stack[stack.length - 1] !== -1) {
        const height = heights[stack.pop()];
        const width = heights.length - stack[stack.length - 1] - 1;
        max_area = Math.max(max_area, height * width);
    }

    return max_area;
};
```

The test results show that the solution is "Accepted" with a runtime of 0 ms.

Thank You For This Assignment