动态规划：最大不重复子串长度

<script>

const str1 = 'abcdefg'

const str2 = 'abcabcd'

const str3 = 'ababcde'

const str4 = 'abcdeab'

*/\*\**

*\* @desc*

*\**

*\*/*

const **maxLength2** = str => {

let max = 1

*// dp[i]表示以i结尾的最长不重复子串*

const dp = [str.**slice**(0, 1)]

for(let i = 1; i < str.length; i++) {

let sameIndex = dp[i - 1].**indexOf**(str[i])

if (sameIndex === -1) {

dp[i] = dp[i - 1] + str[i]

} else {

dp[i] = dp[i - 1].**slice**(sameIndex + 1) + str[i]

}

if (dp[i].length > max) {

max = dp[i].length

}

}

console.**log**(max)

return max

}

**maxLength1**(str1) *// 7*

**maxLength1**(str2) *// 4*

**maxLength1**(str3) *// 5*

**maxLength1**(str4) *// 5*

**maxLength2**(str1) *// 7*

**maxLength2**(str2) *// 4*

**maxLength2**(str3) *// 5*

**maxLength2**(str4) *// 5*

</script>

动态规划：连续子数组求最大和

<script>

const nums1 = [-2, 1, -3, 4, -1, 2, 1, -5]

const nums2 = [-1,2]

const nums3 = [2, 2, -1]

const **maxSum** = (nums) => {

const dp = []

*// dp[i]表示连续和*

dp[0] = nums[0]

let max = nums[0]

for (let i = 1; i < nums.length; i++) {

if (dp[i - 1] <= 0) {

dp[i] = nums[i]

} else {

dp[i] = dp[i - 1] + nums[i]

}

if (dp[i] > max) {

max = dp[i]

}

}

*// console.log(dp)*

console.**log**(max)

return max

}

**maxSum**(nums1) *// 6: [4, -1, 2, 1]*

**maxSum**(nums2) *// 2: [2]*

**maxSum**(nums3) *// 4: [2, 2]*

</script>