算法押题

大数相加

题目

```
const add = (a, b) => {
  ...
  return sum
}

console.log(add("11111111101234567","77777777707654321"))
console.log(add("911111111101234567","77777777707654321"))
```

答案

```
const add = (a, b) => {
  const maxLength = Math.max(a.length, b.length)
  let overflow = false
  let sum = ''
  for(let i = 1; i <= maxLength; i++){
    const ai = a[a.length-i] || '0'
    const bi = b[b.length-i] || '0'
    let ci = parseInt(ai) + parseInt(bi) + (overflow ? 1 : 0)
    overflow = ci >= 10
    ci = overflow ? ci - 10 : ci
    sum = ci + sum
  }
  sum = overflow ? '1' + sum : sum
  return sum
}

console.log(add("11111111101234567","77777777707654321"))
console.log(add("911111111101234567","77777777707654321"))
```

15位加速版:

```
const add = (a, b) => {
  const maxLength = Math.max(a.length, b.length)
  let overflow = false
  let sum = ''
  for(let i = 0; i < maxLength; i+=15){
    const ai = a.substring(a.length-i -15, a.length-i) || '0'
    const bi = b.substring(b.length-i -15, b.length-i) || '0'
    let ci = parseInt(ai) + parseInt(bi)
    overflow = ci > 999999999999999 // 15 个 9
    ci = overflow ? ci - (999999999999999+1) : ci
    sum = ci + sum
  }
  sum = overflow ? '1' + sum : sum
  return sum
}

console.log(add("11111111101234567","77777777707654321"))
console.log(add("911111111101234567","77777777707654321"))
```

其他思路：

1. 转为数组，然后倒序，遍历
2. 使用队列，使用 while 循环

可以自行搜索。

两数之和

题目

```
const numbers = [2,7,11,15]
const target = 9
const twoSum = (numbers, target) => {
  // ...
}
console.log(twoSum(numbers, target))
// [0, 1] 或 [1, 0]

// 出题者保证
// 1. numbers 中的数字不会重复
// 2. 只会存在一个有效答案
```

答案

```
const numbers = [2,7,11,15]
const target = 9
const twoSum = (numbers, target) => {
  const map = {}
  for(let i = 0; i < numbers.length; i++){
    const number = numbers[i]
    const number2 = target - number
    if(number2 in map){
      const number2Index = map[number2]
      return [i, number2Index]
    } else {
      map[number] = i
    }
  }
  return []
}
console.log(twoSum(numbers, target))
```

上面是给菜鸟看的，所以有多余的中间变量，可以删掉。

无重复最长子串的长度

题目

力扣

https://leetcode-cn.com/problems/longest-substring-without-repeating-characters/

```
const lengthOfLongestSubstring = (str) => {
  //...
}

console.log(lengthOfLongestSubstring("abcabcbb"))
// 3
```

答案：滑动窗口法

我称之为「两根手指法」。

```
var lengthOfLongestSubstring = function(s){
  if(s.length <= 1) return s.length
  let max = 0
  let p1 = 0
  let p2 = 1
  while(p2 < s.length) {
    let sameIndex = -1
    for(let i = p1; i < p2; i++){
      if(s[i] === s[p2]){
        sameIndex = i
        break
      }
    }
    let tempMax
    if( sameIndex >= 0){
      tempMax = p2 - p1
      p1 = sameIndex + 1
    }else{
      tempMax = p2 - p1 + 1
    }
    if(tempMax > max){
      max = tempMax
    }
    p2 += 1
  }

  return max
}
```

使用 map 加速:

```javascript
var lengthOfLongestSubstring = function(s) {
  if (s.length <= 1)
    return s.length
  let max = 0
  let p1 = 0
  let p2 = 1
  const map = {}
  map[s[p1]] = 0
  while (p2 < s.length) {
    let hasSame = false
    if(s[p2] in map){
      hasSame = true
      if(map[s[p2]] >= p1){
          p1 = map[s[p2]] + 1
      }
    }
    map[s[p2]] = p2
    let tempMax = p2 - p1 + 1
    if(tempMax > max) max = tempMax
    p2 += 1
  }
  return max
};
```

你会发现，加速失败，可能是 JS 的问题。

改用 new Map() 试试：

```
var lengthOfLongestSubstring = function(s) {
  if (s.length <= 1)
    return s.length
  let max = 0
  let p1 = 0
  let p2 = 1
  const map = new Map()
  map.set(s[p1], 0)
  while (p2 < s.length) {
    let hasSame = false
    if(map.has(s[p2])){
      hasSame = true
      if(map.get(s[p2]) >= p1){
          p1 = map.get(s[p2]) + 1
      }
    }
    map.set(s[p2],p2)
    let tempMax = p2 - p1 + 1
    if(tempMax > max) max = tempMax
    p2 += 1
  }
  return max
};
```

你会发现，加速失败，这应该还是 JS 的问题。