前端学算法

算法训练营的开场 直接找班班申请

算法题 编程题

算法导论讲一个算法， 数学推导

算法啊第四版，讲一个算法，怎么实现 ~~~

多交流，coding又训练营

为啥要学!!! 千万不要执着与刷题家

1. 为了面试 最基本的原因
2. 提升我们的水平，编码效率和质量，提高我们的天花板
    1. 前端开源框架用到的一些算法
    2.

前端的一些场景

Leetcode

硬盘内存比较便宜，时间比较值钱，空间换时间仕一个常见的操作

具体的算法和数据结构

知识体系可便利的，

1. 数组 体育课排队
2. 链表 犯罪分子，单线联系

别的数据结构，基本都可以通过这俩组合而来

数组连续的数据结构，脑补一下，咱们班上体育课，站一排

来了一个人，想进咱们班站好，同学数量仕n ，新来的同学入队，复杂度仕多少 On

一个同学 提前下课跑了，说要去看李佳琦，删除一个元素，复杂度仕多少

O(N)

[1,2, 4,9,5,6,7,8]

数组的删除，新增都是很费时的操作，数组的查找

数组的随机访问是O（1） 我想要第100个数据

Includes也是On

```
arr.forEach(v=>{
 arr.indexOf()
 arr[1]
})
```

有一些奇怪的实现，会打破这个惯性思维， python的数组，就是双向链表实现的，他的新增和删除都是O1

two sum

```
var twoSum = function(nums, target) {
    for(let i=0;i<nums.length;i++){
      for(let j=i+1;j<nums.length;j++){
        if(nums[i]+nums[j]===target){
            return [i,j]
        }
      }
    }
};
```

时间复杂度，空间复杂度(大概额外的内存空间占用多少)

如果我们的数组长度是N， 执行这个算法，大概要运行多少次，和n的关系 ， O来标识, 复杂度不看常量

100*100的量级 ， 这个算法的时间复杂度仕O（n^2)

空间复杂度 O1

```
/**
```

```
 * @param {number[]} nums
 * @param {number} target
 * @return {number[]}
 */
var twoSum = function(nums, target) {
    let obj= {} // 小本本
    // i,j,num都是一个变量，空间不会扩展到n
    for(let i=0;i<nums.length;i++){
        const num = nums[i]
        // 我需要知道你想找那个数字
        if(num in obj){
            // 找到了
            return [obj[num],i]
        }else{
            obj[target-num] = i
        }
    }
};
```

这个算法的时间复杂度O(n)

空间复杂度 O(n)

空间换取了时间

链表

> 哨兵 1=> 2=> 3=> 4=>5 （分散存储的），单向链表 blockchain

随机访问： On （有一些变种链表可以优化 跳表） O(lgn)

删除，新增 O1

react fiber上层架构的更迭，本质上，是数据结构的变化

树微观变成了链表 整个diff可中断

删除链表

```
var removeElements = function(head, val)
{
    let ele={
        next:head
    }
    let cur = ele
    // 链表的遍历
    while(cur.next){
        if(cur.next.val==val){
            cur.next = cur.next.next
        }else{
```

```
                cur = cur.next
            }
        }
        return ele.next
    };
```

驾照 倒库

栈和队列

队列：先入先出 任务 task Promise.all都是类似

栈：先入后出

前端判断一个jsx，vue template，html是否合法

[div,]

```
<div>
    <p> </p>
    </div>
    <input />
</div>
```

树

前端必须必须要掌握的，浏览器解析页面就是一棵树

链表这个结构， 一个元素指向多个 ， 简化模型，一个节点指向俩，二叉树

天生适合递归

```
var isSameTree = function(p, q) {
    // 递归会不停的出现
    if(p==null && q==null){
        return true
    }
    if(p==null || q==null){
        return false
    }
    if(p.val!==q.val){
        return false
    }
    return isSameTree(p.left,q.left) &&
isSameTree(p.right,q.right)
};
```

图

算法思想

二分 快排，二分搜索

递归

回溯 有一些场景，不停的尝试下一步，如果不行，回退到之前的状态，再试下一步

贪心算法

动态规划

之前面试快手，abc 所有组合可能 ， 经典的回溯

a,b,c 全排列

arr = [a,b,c] 一个答案，记录一下

arr.push()

Next()

Arr.pop()

79 单词搜索

```
/**
 * @param {character[][]} board
 * @param {string} word
 * @return {boolean}
 */
var exist = function(board, word) {

// 判断一些边界条件
if(board.length===0) return false
if(word.length===0) return true

const row = board.length
const col = board[0].length
// 每个字母都可以是起点
for(let i=0;i<row;i++){
    for(let j=0;j<col;j++){
        const ret = find(i,j,0)
        if(ret) return true
    }
}
return false
```

```
function find(i,j,cur){
    //越界
    if(i>=row || i<0) return false
    if(j>=col || j<0) return false
    const letter = board[i][j]

// 终止条件
    if(letter!==word[cur]) return false
    // 找到最后一个了
    if(cur==word.length-1) return true

    // 找下一步 ！！！！！
// 设置标记
    board[i][j] = null
// 递归 怎么找下一步
    const ret = find(i+1,j,cur+1) ||
                find(i-1,j,cur+1) ||
                find(i,j+1,cur+1) ||
                find(i,j-1,cur+1)
// 取消标记 进行别的路线查找
    board[i][j] = letter
    return ret
}

};
```

设计模式的内容

设计模式 最佳实践

Element3 组件库这个场景把设计模式串起来

今天的作业，就是leetcode的题

今天的作业

1. leetcode第一题 two sum
2. leetcode 141 环形链表！
3. leetcode 104
4. leetcode 79题

只要通过就可以，不要求几种方法

暗号: 今天天气真不错