1. 判断相等二叉树

function isSameTree(tree1,tree2) { if(tree1 === null && tree2 === null) { return true } if(tree1 === null || tree2 === null) { return false } if(tree1.val !== tree2.val) { return false } return isSameTree(tree1.left,tree2.left) && isSameTree(tree1.right,tree2.right) }

2. js实现双向链表

双向链表：每个节点包含三部分：指向前一个节点的指针， 指向后一个节点的指针， 以及自己的数据部分

双向链表介绍

既可以从头遍历到尾，也可以从尾遍历到头，链表连接的过程是双向的，实现原理是一个节点既有向前连接的引用，也有向后连接的引用

双向链表的缺点

- 每次在插入或删除某个节点时，都需要处理四个引用，而不是两个，实现起来困难些
- 相对于单链表，所占内存空间更大
- 但是便利性更强，所以缺点微不足道

双向链表的结构

- 双向链表不仅有head指针指向前一个节点，而且有tail指针指向最后一个节点
- 每一个节点都有三部分：item储存数据，prev指向前一个节点，next指向后一个节点
- 双向链表第一个节点的prev指向null
- 双向链表的最后一个节点的next指向null

双向链表的常见方法

- append(element)： 向链表尾部添加一个新的项
- insert(position,element):向链表的指定位置插入一个新的项
- get(element): 获取对应位置的元素
- indexOf（element）：返回元素在链表中的索引，如果链表中没有元素就返回-1；
- update（position，element）：修改某个位置的元素；
- removeAt（position）：从链表的特定位置移除一项；
- isEmpty（）：如果链表中不包含任何元素，返回true，如果链表长度大于0则返回false；
- size（）：返回链表包含的元素个数，与数组的length属性类似；
- toString（）：由于链表项使用了Node类，就需要重写继承自JavaScript对象默认的toString方法，让其只输出元素的值；
- forwardString（）：返回正向遍历节点字符串形式；
- backwordString（）：返回反向遍历的节点的字符串形式；

function Node(element) { this.element = element; this.next = null this.previous = null }

class Node { constructor(data) { this.data = data this.next = null this.previous = null } }

class DoubleLinked { constructor() { this.size = 0 this.head = new Node('head') this.currentNode = '' } // 获取链表长度 getSize() { return this.size } // 判断链表是否为空 isEmpty() { return this.size === 0 } // 显示当前节点 showNode() { return this.currentNode.data } // 正序遍历

```
displayList() {
    let str = ''
    let currentNode = this.head  // 打印结果带head值
    // var currentNode=this.head.next  打印结果不带head值
    while(currentNode) {
        str += currentNode.data
        currentNode = currentNode.next
        if(currentNode) {
            str += '——>'
        }
    }
    return str
}

// 倒序遍历
lastDisplay() {
    let str = ''
    let currentNode = this.findLast()
    while(currentNode) {
        str += currenNode.data
        currentNode = currentNode.prev
        if(currentNode) {
            str += '——>'
        }
    }
    return str
}

// 获取最后一个元素
findLast() {
    let currentNode = this.head
    while(currenNode.next) {
        currentNode = currenNode.next
    }
    return currentNode
}

// 查找某一个元素
findNode(data) {
    let currenNode = this.head
    while(currentNode && currentNode.data === data) {
        currentNode = currentNode.next
    }
    return currentNode
}
```

```
// 在头部插入一个节点
headAppend(data) {
    let newNode = new Node(data)
    newNode.next = this.head.next
    this.head.next.prev = newNode
    newNode.prev = this.head
    this.head.next = newNode
    this.size ++
}

// 在尾部插入一个节点
append(data) {
    let newNode = new Node(data)
    let currentNode = this.finalLast()
    currentNode.next = newNode
    newNode.next = null
    newNode.prev = currentNode
    this.size ++
}
// 插入节点 表示将element值插入到data值之后
insertNextNode(data,element) {
    let currentNode = this.findNode(data)
      //如果data不存在
    if (!currentNode) {
        return
    }
    let newNode = new Node(element)
    newNode.next = currentNode.next
    currentNode.next.prev = newNode
    currentNode.next = newNode
    newNode.prev = currentNode
    this.size++

}

// 插入节点 表示将element插入到data值之前
insertPrevNode(data,element) {
    let currentNode = this.findNode(data)
    if(!currentNode) {
        return
    }
    let newNode = new Node(element)
    newNode.next = currentNode
    currentNode.prev.next = newNode
    newNode.prev = currentNode.prev
    currentNode.prev = newNode
    this.size ++
}

// 删除一个节点
deleteNode(data) {
```

```
        let currentNode = this.findNode(data)
        // 删除的是最后一个节点
        if(currentNode.next === null) {
            currentNode.prev.next = null
        }else {
            currentNode.prev.next = currentNode.next
            currentNode.next.prev = currentNode.prev
        }
        this.size--
    }
```

}

3.手写题：在线编程，getUrlParams(url,key); 就是很简单的获取url的某个参数的问题，但要考虑边界情况，多个返回值等等

let reg = new RegExp('(^|&)'+key + '=([^&])*($|&)', 'i) // 正则法 function getUrlParams(url,name) { let reg = new RegExp("(^|&)" + name + "=([^&])(&|$)", "i"); let r = url.search.substr(1).match(reg) if(r!== null) { return decodeURIComponent(r[2]) } return null }

function getUrlParams(url,key) { let reg = new RegExp('(^|&)'+key + '=([^&]*)($|&)', "i") let r = url.search.subStr(1).match(reg) if(r!== null) { return decodeURLComponent(r[2]) } return null }

// 拆分法 function getUrlParams(url,key) { let query = url.search.subStr(1) let vars = query.split('&') for(let i = 0; i < vars.length; i++) { let pair = vars[i].split('=') if(pair[0] === key) return pair[1] } return false }

4.数据库的分类

关系型数据库 SQL Server Oracle Mysql 采用了关系模型来组织数据的数据库 优点：容易理解，二维表结构贴近现实，使用方便 易于维护

非关系型数据库 mongoDB redis couchDB 键值存储，结构不稳定 高并发 快速查询 适合存储较为简单的数据

5.数组的去重

function uniqueArray(arr) { return [...new Set(arr)] } function uniqueArr(arr) { let obj = {} let newArr = [] arr.forEach(item => { if(!obj[item]) { obj[item] = 1 newArr.push(item) } }) return newArr } function uniqueArray5(arr){ var newArr = []; for(var i = 0; i< arr.length;i++){ if(newArr.indexOf(arr[i])===-1){ newArr.push(arr[i]); } } return newArr; }

6. 数组的扁平化处理

function flatten(arr) { let result = [] for(let i = 0; i < arr.length; i++) { if(Array.isArray(arr[i])) { result = result.concat(flatten(arr[i])) }else { result = result.concat(arr[i]) } } return result }

// arr.reduce(callback(accumulator, currentValue[, index[, array]])[, initialValue])

function flatten(arr) { return arr.reduce((pre,val) => { if(Array.isArray(val)) { return pre.concat(flatten(val)) } return pre.concat(val) },[]) }

7.「字节」实现字符串驼峰转换

// var str = 'border-top-color';

// chartAt 返回指定位置的字符

// subStr 和subString的区别 // subStr(start, [,length]) 第一个参数是索引，第二个参数是截取的长度 // subString(start, [,end]) 第一个参数是开始位置的索引，第二个参数是结束位置索引，不包括end

function toHumpName(str) { var arr = str.split('-') for(let i = 1; i < arr.length; i++) { arr[i] = arr[i].charAt(0).toUpperCase() + arr[i].subString(1) } return arr.join('') }

// \w查找数字、字母下划线 // \W 查找非单词字符 // js命名规则：标识符第一个字符必须是字母，下划线，或$ function toHumpName(atr) { let reg = /-(\w)/g return str.replace(reg, function($0,$1) { return $1.toUpperCase() }) }

$0: 正则匹配到的字符串

$1: 在使用组匹配时，组匹配到的值

$2: 匹配值在原字符串中的索引

$3: 原字符串 function toHumpName(str) { let reg = /-(\w)/g return str.replace(reg, function($0,$1){ console.log($0,$1) return $1.toUpperCase() }) }

数字转为千分位 var str = "1234567890"; str.replace(/\d{1,3}(?=(\d{3})+$)/g, function(match) { return match + ','; });

8.「字节」比较两个版本号大小

function compareVersion(v1,v2) { v1 = v1.split('.') v2 = v2.split('.') const len = Math.max(v1.length,v2.length) while(v1.length < len) { v1.push(0) } while(v2.length < len) { v2.push(0) } for(let i = 0; i < len; i++) { const num1 = parseInt(v1[i]) const num2 = parseInt(v2[i]) if(num1 > num2) { return 1 }else if(num1 < num2) { return -1 } } return 0 }

9.「腾讯」根据对象ID实现数组去重

var arr = [{ id: '01', value: '乐乐' }, { id: '02', value: '博博' }, { id: '03', value: '淘淘' }, { id: '04', value: '哈哈' }, { id: '01', value: '乐乐' }];

function uniqueArray(arr) { let obj = {} return arr.reduce((pre,val) => { let key = val.id

```
    if(!obj[key]) {
        obj[key] = true
        pre.push(val)

    }
    return pre
},[])
```

}

10.「字节」基于JS实现按照权重抽奖

1 先配置奖品数据,比如4个奖品类型，一等奖概率1%，二等奖概率3%，三等奖概率6%，参与奖90%，设置如下

//设置奖项名称、权重等数组 var prizes = [{"name": "一等奖", "weight": 1}, {"name": "二等奖", "weight": 5}, {"name": "三等奖", "weight": 20}, {"name": "未中奖", "weight": 74}];

根据权重数组的和值（weightSum），在每次抽奖时生成一个权重随机数（weightRandom），这个权重随机数（weightRandom）是介于 0-weightSum （权重和值）之间的，本文示例设置的权重数组和值为100，表示生成的权重随机数是介于 0-100 之间的； 然后让这个权重随机数（weightRandom）去和权重数组中的所有元素值作比较，计算这个权重随机数（weightRandom）位于哪两个奖项之间，符合哪条中奖规则，对应哪个奖项名称。 比如：某次抽奖生成的权重随机数（weightRandom）为 15.15，按照 1.3 的活动规则，因为 5 < 15.15 <= 20，表示此次生成的权重随机数（weightRandom）可中三等奖。

var prizeWeight = [1, 5, 20, 74];

function lottery(prizeWeight) { var weightSum = prizeWeight.reduce(function(prev, currVal){ //计算权重之和：1+5+20+74=100 return prev + currVal; //prev 是前一次累加后的数值，currVal 是本次待加的数值 }, 0); var res = '未中奖' var random = Math.random()*weightSum var concatWeightArr = prizeWeight.concat(random) var sortWeightArr = concatWeightArr.sort(function(a,b){return a- b}) //升序排列 //索引权重随机数的数组下标 var randomIndex = sortedWeightArr.indexOf(random); //索引随机数在新权重数组中的位置 randomIndex = Math.min(randomIndex, prizes.length -1); //权重随机数的下标不得超过奖项数组的长度-1，重新计算随机数在奖项数组中的索引位置

```
  return prizes[randomIndex]
```

}

11.「阿里」实现斐波那契数列fibonacci

// 求第n项目 function fibonacci(n) { if(n < 2) return n return fibonacci(n -1) + fibonacci(n-2) }

function * fibonacci() { let [prev,curr] = [0,1] for(;😉 { [prev,curr] = [curr,prev+curr] yield curr } } for(let n of fibonacci) { console.log(n) if(n > 1000) break }

12.实现开方

function sqrt(num) { function sqrtWrapper(min,max) { let current = (min + max)/2 let nextMin = min,nextMax = max if(current * current > num) { nextMax = current }else { nextMin = current }

```
    if(min === nextMin && max === nextMax){
        return current
    }else if(nextMax - nextMin < Number.EPSION){
        return current
    }else {
        return sqrtWrapper(nextMin,nextMax)
    }
}
return sqrtWrapper(0,num)
```

}

function sqrt(num) { function sqrtWrapper(min,max) { let nextMin = min,nextMax = max,current = (min + max)/2; if(current * current > num) { nextMax = current }else { nextMin = current } if(min === nextMin && max === nextMax) { return current } else if(nextMax - nextMin < Number.EPSILon) { return current }else { return sqrtWrapper(nextMin,nextMax) } } return sqrtWrapper(0,num) }

function sqrt(num) { function sqrtWrapper(min, max) { let current = (min + max) / 2; let nextMin = min, nextMax = max; if (current * current > num) { nextMax = current; } else { nextMin = current; } if (min === nextMin && max === nextMax) { return current } else if (nextMax - nextMin < Number.EPSILON) { return current; } else { return sqrtWrapper(nextMin, nextMax); } } return sqrtWrapper(0, num); }

13. 最小操作 bfs

14. 所有可能 dfs

15. 数组 双指针

16. 数组排序 冒泡 快速 二分法

- 冒泡排序： 两两比较，如果前一个比后一个大，则交换位置 function sort(arr) { for(var j = 0; j < arr.length - 1; j++){ for(var i = 0; i < arr.length - 1-j; i++) { if(arr[i] > arr[i+1]) { var temp = arr[i] arr[i] = arr[i+1] arr[i+1] = temp } } } }
- 快速排序 取出中间值 分别左右数组对比 function quickSort(arr) { // 定义中间值的索引 var index = Math.floor(arr.length /2) // 取到中间值 var temp = arr.splice(index,1) // 定义左右部分的数组 var left = [] var right = [] for(var i = 0; i < arr.length; i++) { //如果元素比中间值小，那么放在左边，否则放在右边

```
if(arr[i] < temp) { left.push(arr[i]) }else{ right.push(arr[i]) } } return
quickSort(left).concat(temp,quickSort(right)) }
```

17. 实现柯里化

```
function createCurry(func,args) { var argity = func.length var args = args || [] return function() { var _args =
[].slice.apply(arguments) args.push(..._args) if(args.length < argity) { return createCurry.call(this,func,args) }
return func.apply(this,args) } }
```