# 数据结构笔试题 + 面试题

## 理论

### 什么是数据结构？

答：“数据结构是计算机存储、组织数据的方式。数据结构是指相互之间存在一种或多种特定关系的数据元素的集合

### 为什么我们需要数据结构？

答：数据是程序的核心要素，因此数据结构的价值不言而喻。无论你在写什么程序，你都需要与数据打交道。在不同场景下，数据需要以特定的方式存储，我们有不同的数据结构可以满足我们的需求。

### 6种常用数据结构

答：数组、栈、队列、链表、图、树

## 数组

### 实现数组的基本操作

1. Insert - 在某个索引处插入元素

insert(item){

            this.array.push(item)

        }

1. Get - 读取某个索引处的元素

get(key){

            return this.has(key)? items[key]:undefined

        }

1. Delete - 删除某个索引处的元素

delete(value) {

            if (this.has(value)) {   //只能删除集合里面有的元素

                delete this.items[value]

                return true

            } else {

                return false

            }

        }

1. Size - 获取数组的长度

size() {

            return Object.keys(this.items).length

        }

1. 查找数组中第二小的元素
2. 查找第一个没有重复的数组元素
3. 合并2个数组（先排序后合并）
4. 排列数组中的正数和负数

## 栈

### 实现栈的基本操作

1. Push—在栈的最上方插入元素

 push(item) {

                    this.items.push(item)

                }

1. Pop— 返回栈最上方的元素，并将其删除

pop() {

                    return this.items.pop()

                }

1. isEmpty—查询栈是否为空

isempty() {

                    return this.items.length==0

                }

1. Top—返回栈最上方的元素，并不删除

Top(){

                    return this.items[this.items.length-1]

                }

### 使用栈实现以下效果

1. 使用冒泡算法为栈中的元素排序

<script>

        class Stack{

            constructor(){

                this.array=[]

            }

            insert(item){

                this.array.push(item);

            }

            toString(){

                return this.array.join();

            }

            // 冒泡排序  ：找最大的值不断的往后面移动(冒泡排序比较任何两个相邻的项，如果第一个比第二个大，则交换它们)

            bubbleSort(){

                let length=this.array.length;

                for(let i=0;i<length;i++){

                    for(let j=0;j<length-1;j++){

                        if(this.array[j]>this.array[j+1]){

                            let aux=this.array[j];

                            this.array[j]=this.array[j+1];

                            this.array[j+1]=aux;

                        }

                    }

                }

                return this.array  //这是个函数方法，需要返回值，否则会得到unde

            }

        }

        let newStack=new Stack();

        newArray.insert(5)

        newArray.insert(4)

        newArray.insert(3)

        newArray.insert(2)

        newArray.insert(1)

        console.log(newArray.toString())   //5,4,3,2,1

        console.log(newArray.bubbleSort())  //[1,2,3,4,5]

1. 用栈实现十进制转化成为其他进制的方法（任意进制，不仅仅于十六进制，无限进制）

<script>

            class Stack {

                constructor() {

                    this.items = []

                }

                push(item) {

                    this.items.push(item)

                }

                pop() {

                    return this.items.pop()

                }

                peek(){

                    return this.items[this.items.length-1]

                }

                isempty() {

                    return this.items.length==0

                }

                clear(){

                    this.items = []

                }

                size(){

                    return this.items.length

                }

                getitems() {

                    return this.items

                }

            }

  // 十进制转换为任意进制

            class base{

                constructor(decNumber, base) {

                    this.decNumber = decNumber;

                    this.base = base;

                    this.remStack = new Stack();

                    this.rem=null;

                    this.baseString = '';

                    this.digits = '0123456789ABCDEF';

                }

                baseConverter() {

                    while (this.decNumber > 0) {

                        this.rem = Math.floor(this.decNumber % this.base);

                        this.remStack.push(this.rem);

                        this.decNumber = Math.floor(this.decNumber / this.base);

                    }

                    while (!this.remStack.isempty()) {

                        this.baseString += this.digits[this.remStack.pop()]; //{7}

                    }

                    return this.baseString;

                }

            }

             let Newbase1=new base(2020,16)

            let Newbase2=new base(3050,8)

            console.log(Newbase1.baseConverter())  //7E4

            console.log(Newbase2.baseConverter())  //5752

    </script>

### 实现队列的基本操作

1. Enqueue—在队列末尾插入元素

 enqueue(){

                return this.items.push()

            }

1. Dequeue—将队列第一个元素删除

 dequeue(){

                return this.items.shift()

            }

1. isEmpty—查询队列是否为空

isempty(){

                return this.items.length==0

            }

1. Front—返回队列的第一个元素

Font(){

                return this.items[0]

            }

### 使用队列实现以下效果

1. 使用队列实现栈
2. 使用队列将1到n转换为二进制
3. 通过队列实现贪吃蛇的游戏
4. 使用选择排序的算法将队列中的数进行排序

<script>

        // 选择排序：找最小的值往最前面放（找到数据结构中的最小值并将其放置在第一位，接着找到第二小的值并将其放在第二位，以此类推）

        class QueueList {

            constructor() {

                this.array = []

            }

            insert(item) {

                this.array.push(item)

            }

            toString() {

                return this.array.join()

            }

            selectionSort() {

                let length = this.array.length,

                    indexMin;

                for (let i = 0; i < length; i++) {

                    indexMin = i;

                    for (let j = i; j < length; j++) {

                        if (this.array[indexMin] > this.array[j]) {

                            indexMin = j

                        }

                    }

                    if (i !== indexMin) {

                        let aux = this.array[i];

                        this.array[i] = this.array[indexMin];

                        this.array[indexMin] = aux;

                    }

                }

                return this.array;

            }

        }

        let NewQueue=new QueueList();

        NewArray.insert(5);

        NewArray.insert(4);

        NewArray.insert(3);

        NewArray.insert(2);

        NewArray.insert(1);

        console.log(NewArray.toString())

        console.log(NewArray.selectionSort())

    </script>

## 链表

### 链表的基本操作

1. InsertAtEnd—在链表结尾插入元素

InsertAtEnd(element) {

                let node = new Node(element);  //将实例化对象node作为一个属性

                let cu = this.head;

                if (this.head == null) {    //此时代表链表中没有任何元素（判断头部是否为空）

                    this.head = node

                } else {

                    while (cu.next) {

                        cu = cu.next    //cu:当前循环到的结点

                    }

                    cu.next = node   //node：当前结点

                }

                this.length++

            }

1. InsertAtHead—在链表开头插入元素

insertAtHead(element,position){

                if(position>=0&&position<=this.length-1){

                    let node=new Node(element)//实例化对象，node表示新插入的结点

                    if(position==0){      //大概就分为两种情况，往头部位置插入和其他部位插入

                        node.next=this.head;    //现在的=之前的 这样的格式（或者 插入元素之前的=插入之后的）

                        this.head=node

}

1. Delete—删除链表的指定元素

Delete(position) {

                if (position >= 0 && position <= this.length) {

                    if (position == 0) {

                        this.head = this.head.next

                    } else {

                        let num = 0;   //num的作用：用于标记下标

                        let cu = this.head;

                        let pre = null

                        while (num < position) {

                            pre = cu;

                            cu = cu.next;

                            num++

                        }

                        pre.next = cu.next;

                    }

                    this.length--

                } else {

                    return false

                }

            }

1. DeleteAtHead—删除链表第一个元素

 DeleteAtHead(position) {

                if (position >= 0 && position <= this.length) {

                    if (position == 0) {

                        this.head = this.head.next

}

}

1. Search—在链表中查询指定元素
2. isEmpty—查询链表是否为空

   isEmpty(){

                return this.length===0

            }

### 使用链表实现以下效果

1. 倒转1个链表
2. 检查链表中是否存在循环
3. 返回链表倒数第N个元素
4. 移除链表中的重复元素

## 集合

### 集合的基本操作

1. 并集（写出概念，并且在集合当中添加并集方法）

答：并集：AUB并集的x（元素）存在于A中，或x存在于B中。

 // 求并集

        union(inSet) {

            let NewSet2 = new Set();   //新创建一个集合NewSet2，用于装并集结果

            // console.log(NewSet2);

            for (let i = 0; i < this.size(); i++) {   //此处的this指代实例化对象NewSet1

                NewSet2.add(this.values()[i])  //往新集合NewSet2里面添加集合NewSet1的所有元素

            }

            for (let i = 0; i < inSet.size(); i++) {  //此处的inSet是指另外一个实例化对象NewSet

                NewSet2.add(inSet.values()[i])  //往新集合NewSet2里面添加集合NewSet的所有元素

            }

            return NewSet2  //返回并集结果：NewSet2

        }

1. 交集（写出概念，并且在集合当中添加交集的方法）

答：交集：结果是两个集合的公共部分的元素，即既属于集合A，又属于集合B

// 求交集

        intersect(inSet) {

            let NewSet3 = new Set();

            for(let i=0;i<this.values().length;i++){

                if(inSet.has(this.values()[i])){

                    NewSet3.add(this.values()[i])

                }

                // else{

                //     return fasle;

                // }             //此处不能有else，否则会引起代码执行错误，会直接返回false，因此这里不能要else里面的东西

            }

            return NewSet3

        }

1. 子集（写出概念，并且在集合当中添加子集的方法）

答：子集：若A集合里面的元素，B集合里面都有，那么说B集合包含A集合，或者A集合是B集合的子集

 // 求子集

        subset(inSet){

            if(this.size()>inSet.size()){

                return false;

            }else{

                let values=this.values();

                for(let i=0;i<values.length;i++){

                    if(!inSet.has(values[i])){

                        return false;

                    }

                }

                return true;

            }

        }

1. 差集（写出概念，并且在集合当中添加差集的方法）

答：差集：A集合与B集合的差集的结果的元素属于A集合但不属于B集合

// 求差集

        differ(inSet){

            let NewSet4=new Set();

            for(let i=0;i<this.values().length;i++){

                if(!inSet.has(this.values()[i])&&this.has(this.values()[i])){

                    NewSet4.add(this.values()[i])

                }

            }

            return NewSet4

        }

        getItems() {

            console.log(this.items)

        }

1. 写出三种排序算法

答：

1. 冒泡排序：

<script>

        class ArrayList{

            constructor(){

                this.array=[]

            }

            insert(item){

                this.array.push(item);

            }

            toString(){

                return this.array.join();

            }

            // 冒泡排序  ：找最大的值不断的往后面移动(冒泡排序比较任何两个相邻的项，如果第一个比第二个大，则交换它们)

            bubbleSort(){

                let length=this.array.length;

                for(let i=0;i<length;i++){

                    for(let j=0;j<length-1;j++){

                        if(this.array[j]>this.array[j+1]){

                            let aux=this.array[j];

                            this.array[j]=this.array[j+1];

                            this.array[j+1]=aux;

                        }

                    }

                }

                return this.array  //这是个函数方法，需要返回值，否则会得到unde

            }

        }

        let newArray=new ArrayList();

        newArray.insert(5)

        newArray.insert(4)

        newArray.insert(3)

        newArray.insert(2)

        newArray.insert(1)

        console.log(newArray.toString())   //5,4,3,2,1

        console.log(newArray.bubbleSort())  //[1,2,3,4,5]

1. 选择排序：

<script>

        // 选择排序：找最小的值往最前面放（找到数据结构中的最小值并将其放置在第一位，接着找到第二小的值并将其放在第二位，以此类推）

        class ArrayList {

            constructor() {

                this.array = []

            }

            insert(item) {

                this.array.push(item)

            }

            toString() {

                return this.array.join()

            }

            selectionSort() {

                let length = this.array.length,

                    indexMin;

                for (let i = 0; i < length; i++) {

                    indexMin = i;

                    for (let j = i; j < length; j++) {

                        if (this.array[indexMin] > this.array[j]) {

                            indexMin = j

                        }

                    }

                    if (i !== indexMin) {

                        let aux = this.array[i];

                        this.array[i] = this.array[indexMin];

                        this.array[indexMin] = aux;

                    }

                }

                return this.array;

            }

        }

        let NewArray=new ArrayList();

        NewArray.insert(5);

        NewArray.insert(4);

        NewArray.insert(3);

        NewArray.insert(2);

        NewArray.insert(1);

        console.log(NewArray.toString())

        console.log(NewArray.selectionSort())

    </script>

1. 插入排序：

<script>

        // 插入排序：

        class ArrayList{

            constructor(){

                this.array=[]

            }

            toString(){

                return this.array.join()

            }

            createNonSortedArray(size){

                for(let i=size;i>0;i--){

                    this.array.push(i);

                }

            }

            insertionSort(){

                let length=this.array.length,

                j,temp;

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

                    j=i;

                    temp=this.array[i];

                    while(j>0&&this.array[j-1]>temp){

                        this.array[j]=this.array[j-1];

                        j--;

                    }

                    this.array[j]=temp;

                }

                return this.array;

            }

        }

        let newArray=new ArrayList()

        newArray.createNonSortedArray(5);

        console.log( newArray.toString());

        console.log( newArray.insertionSort())

    </script>