前缀树

何为前缀树? 如何生成前缀树?

例子: 一个字符串类型的数组arr1，另一个字符串类型的数组arr2。

arr2中有哪些字符，是arr1中出现的？请打印。

arr2中有哪些字符，是作为arr1中某个字符串前缀出现的？请打印。

arr2中有哪些字符，是作为arr1中某个字符串前缀出现的？请打印

arr2中出现次数最大的前缀

剑指 Offer II 062. 实现前缀树

中等 54

相关企业

Trie（发音类似 "try"）或者说 **前缀树** 是一种树形数据结构，用于高效地存储和检索字符串数据集中的键。这一数据结构有相当多的应用情景，例如自动补完和拼写检查。

请你实现 Trie 类：

- `Trie()` 初始化前缀树对象。
- `void insert(String word)` 向前缀树中插入字符串 `word` 。
- `boolean search(String word)` 如果字符串 `word` 在前缀树中，返回 `true`（即，在检索之前已经插入）；否则，返回 `false` 。
- `boolean startsWith(String prefix)` 如果之前已经插入的字符串 `word` 的前缀之一为 `prefix` ，返回 `true` ；否则，返回 `false` 。

示例：

```
输入
inputs = ["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
inputs = [[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
输出
[null, null, true, false, true, null, true]

解释
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // 返回 True
trie.search("app");     // 返回 False
trie.startsWith("app"); // 返回 True
trie.insert("app");
trie.search("app");     // 返回 True
```

思路：

1. 将字符放在edge上，用长度为26的数组记录可能出现的字符，根据字符的code值判断在数组中的位置
2. 记录字符路过的数量p和以当前字符为结尾的个数e

```
/**
```

```
 * Initialize your data structure here.
 */
var Trie = function() {
    this.p = 0;
    this.e = 0;
    this.next = Array.from({length: 26});
};

/**
 * Inserts a word into the trie.
 * @param {string} word
 * @return {void}
 */
Trie.prototype.insert = function(word) {
    if (word === null) return;
    let arr = word.split('')
    let node = this;
    let index = 0;
    node.p++
    for (let i = 0; i < arr.length; i++) {
        index = arr[i].charCodeAt() - 'a'.charCodeAt();
        if (!node.next[index]) {
            node.next[index] = new Trie();
        }
        node = node.next[index]
        node.p++
    }
    node.e++
};

/**
 * Returns if the word is in the trie.
 * @param {string} word
 * @return {boolean}
 */
Trie.prototype.search = function(word) {
    if (word === null) return true;
    let arr = word.split('');
    let node = this;
    let index = 0;

    for (let i = 0; i < arr.length; i++) {
        index = arr[i].charCodeAt() - 'a'.charCodeAt();
        if (!node.next[index]) {
            return false;
        }
        node = node.next[index];
    }
    if (node.e === 0) return false
    return true
};

/**
 * Returns if there is any word in the trie that starts with the given prefix.
 * @param {string} prefix
 * @return {boolean}
 */
Trie.prototype.startsWith = function(prefix) {
```

```
     if (prefix === null) return true;
    let arr = prefix.split('');
    let node = this;
    let index = 0;

    for (let i = 0; i < arr.length; i++) {
        index = arr[i].charCodeAt() - 'a'.charCodeAt();
        if (!node.next[index]) {
            return false;
        }
        node = node.next[index];
    }
    return true
};

/**
 * Your Trie object will be instantiated and called as such:
 * var obj = new Trie()
 * obj.insert(word)
 * var param_2 = obj.search(word)
 * var param_3 = obj.startsWith(prefix)
 */
```

贪心算法

在某一个标准下，优先考虑最满足标准的样本，然后考虑最不满足的样本，最终得到一个答案的算法，叫做贪心算法。
即不从整体最优上加以考虑，所做出的是在某种意义上的**局部最优解**

解题套路：

1. 实现一个不依靠贪心策略的解法x，可以用最暴力的尝试
2. 脑补出贪心策略
3. 用解法X和对数器，验证每一个贪心策略，用实验得到正确解

会议安排
一些项目要占用一个会议室宣讲，会议室不能同时容纳俩个项目，给出项目的起始时间和截止时间，如何安排最多的项目。

策略：截止时间早
以截至时间排序，选择截至时间在前的会议。每当选择一个会议时，以这个会议的end为时间点判断接下来的会议是否满足条件

```
function Program(start, end) {
    this.start = start;
    this.end = end;
}

function bestArrange(program: Program[], start = 0) {
    program.sort((a,b) => a.end  - b.end)
    let result = 0;
    for (let i = 0; i < program.length; i++) {
        if (program[i].start >= start) {
            result++
            start = program[i].start
        }
    }
```

```
    console.log(program)
    return result
}
```

IPO

给定初始资金和最多做几个项目，最大化资本

先给定一个花费组织排序的小根堆，锁住，在给定一个大根堆，以利润组织

502. IPO

困难 264

相关企业

假设 力扣（LeetCode）即将开始 **IPO** 。为了以更高的价格将股票卖给风险投资公司，力扣 希望在 IPO 之前开展一些项目以增加其资本。 由于资源有限，它只能在 IPO 之前完成最多 `k` 个不同的项目。帮助 力扣 设计完成最多 `k` 个不同项目后得到最大总资本的方式。

给你 `n` 个项目。对于每个项目 `i` ，它都有一个纯利润 `profits[i]` ，和启动该项目需要的最小资本 `capital[i]` 。

最初，你的资本为 `w` 。当你完成一个项目时，你将获得纯利润，且利润将被添加到你的总资本中。

总而言之，从给定项目中选择 **最多** `k` 个不同项目的列表，以 **最大化最终资本** ，并输出最终可获得的最多资本。

答案保证在 32 位有符号整数范围内。

示例 1：

输入： k = 2, w = 0, profits = [1,2,3], capital = [0,1,1]
输出： 4
解释：
由于你的初始资本为 0，你仅可以从 0 号项目开始。
在完成后，你将获得 1 的利润，你的总资本将变为 1。
此时你可以选择开始 1 号或 2 号项目。
由于你最多可以选择两个项目，所以你需要完成 2 号项目以获得最大的资本。
因此，输出最后最大化的资本，为 0 + 1 + 3 = 4。

示例 2：

输入： k = 3, w = 0, profits = [1,2,3], capital = [0,1,2]
输出： 6

```
class Heap {
    constructor(cmp = (x,y) => x - y) {
        this.heap = []
        this.cmp = cmp
    }
    insert(data) {
        const {heap, cmp, swap} = this
        heap.push(data)
        let index = this.size() - 1
        while (index) {
            let parentIndex = (index-1)>>1
            if (!cmp(heap[index], heap[parentIndex])) return
            swap(heap, index, parentIndex)
            index = parentIndex
        }
    }
```

```
    pop() {
        const {heap, swap} = this
        if (!this.size()) {
            return null;
        }
        swap(heap, 0, this.size() - 1)
        let res = heap.pop();

        let len = this.size()
        // 有可能没有右孩子但是又左孩子，所以以左孩子进行判断
        let index = 0, left = index * 2 + 1

        while (left < len) {
            let largestIndex = left + 1 < len && this.cmp(heap[left + 1],
heap[left]) ? left + 1 : left
            if (this.cmp(heap[index], heap[largestIndex])) {
                break
            }
            swap(heap, index, largestIndex)
            index = largestIndex
            left = index * 2 + 1
        }


        return res
    }
    swap(arr, i, j) {
        [arr[i], arr[j]] = [arr[j], arr[i]]
    }
    size() {
        return this.heap.length
    }
    peak() {
        return this.heap[0]
    }
    isEmpty() {
        return !this.heap.length
    }
}
var findMaximizedCapital = function(k, w, profits, capital) {
    let n = capital.length
    let minCosts = []
    let maxProfit = new Heap()
    for (let i = 0; i < n; i++) {
        minCosts.push([profits[i], capital[i]])
    }
    minCosts.sort((a, b) => a[1] - b[1])
    let curr = 0
    for (let i = 0; i < k; i++) {
        while (curr <  n && minCosts[curr][1] <= w) {
            maxProfit.insert(minCosts[curr++][0])
        }
        if (!maxProfit.size()) {
            return w
        }
        w += maxProfit.pop()
    }
    return w
```

```
};
```

取中位数

一个数据流中，随时可以取得中位数

先给一个数字，放入大根堆的堆顶；第二个数字cur是否小于等于大根堆顶，是，cur入大根堆，不是，入小根堆；接着看大根堆和小根堆的size，如果size差距大于等于2，则较大的那个堆顶弹出进另外一个。

```
class Heap {
    constructor(cmp = (x,y) => x > y) {
        this.heap = []
        this.cmp = cmp
    }
    insert(data) {
        const {heap, cmp, swap} = this
        heap.push(data)
        let index = this.size() - 1
        while (index) {
            let parentIndex = (index-1)>>1
            if (!cmp(heap[index], heap[parentIndex])) return
            swap(heap, index, parentIndex)
            index = parentIndex
        }
    }
    pop() {
        const {heap, swap} = this
        if (!this.size()) {
            return null;
        }
        swap(heap, 0, this.size() - 1)
        let res = heap.pop();

        let len = this.size()
        // 有可能没有右孩子但是又左孩子，所以以左孩子进行判断
        let index = 0, left = index * 2 + 1

        while (left < len) {
            let largestIndex = left + 1 < len && this.cmp(heap[left+1],
heap[left]) ? left + 1 : left
            if (this.cmp(heap[index], heap[largestIndex])) {
                break
            }
            swap(heap, index, largestIndex)
            index = largestIndex
            left = index * 2 + 1
        }


        return res
    }
    swap(arr, i, j) {
        [arr[i], arr[j]] = [arr[j], arr[i]]
    }
    size() {
        return this.heap.length
```

```
    }
    peak() {
        return this.heap[0]
    }
    isEmpty() {
        return !this.heap.length
    }
}

/**
 * initialize your data structure here.
 */
var MedianFinder = function() {
     this.minHeap = new Heap((x,y) => x < y)
     this.maxHeap = new Heap()
};


/**
 * @param {number} num
 * @return {void}
 */
MedianFinder.prototype.addNum = function(num) {
    if (this.maxHeap.size() === 0 || num <= this.maxHeap.peak()) {
        this.maxHeap.insert(num)
    } else {
       this.minHeap.insert(num)
    }
    if (this.minHeap.size() === this.maxHeap.size() + 2) {
        this.maxHeap.insert(this.minHeap.pop())
    }
    if (this.maxHeap.size() === this.minHeap.size() + 2) {
        this.minHeap.insert(this.maxHeap.pop())
    }
};

/**
 * @return {number}
 */
MedianFinder.prototype.findMedian = function() {
    let minHeapSize = this.minHeap.size()
    let maxHeapSize = this.maxHeap.size()
    let minHeapHead = this.minHeap.peak()
    let maxHeapHead = this.maxHeap.peak()
    console.log(this.minHeap, this.maxHeap)
    let len = minHeapSize + maxHeapSize
    if (len === 0) return null
    if (len % 2 !== 0) {
        return maxHeapSize > minHeapSize ? maxHeapHead : minHeapHead;
    } else {
        // console.log(this.maxHeap.pop(), this.minHeap.pop())
        return (minHeapHead + maxHeapHead) / 2
    }
};

/**
 * Your MedianFinder object will be instantiated and called as such:
 * var obj = new MedianFinder()
```

```
 * obj.addNum(num)
 * var param_2 = obj.findMedian()
 */
```

N皇后

[N皇后]

思路：暴力递归

1. 用record记录皇后位置，下标i代表行，record[i]代表列
2. 递归函数process(i, record, n)，递归每一行，并传递record
3. 递归中循环判断第i行的所有j是否满足条件，满足就将j记录到record中并递归下一行
4. 当递归到第n行后，表明此时record中记录的位置信息满足所有条件，添加到返回结果中

```
/**
 * @param {number} n
 * @return {string[][]}
 */
var solveNQueens = function(n) {
    if (n === 1) {
        return [["Q"]]
    }
    let res = []
    let record = new Array(n) // 记录第i行皇后放在第几列上，下标i代表行，record[i]代表列
    process(0, record, n)
    function process(i, record, n) {
    if (i === n) {
        res.push(toString(record, n))
        return
    }
    for (let j = 0; j < n; j++) {
        if (isValid(record, i, j)) {
            record[i] = j
            process(i+1, record,  n)
        }
    }
}
function isValid(record, i, j) {
    for (let k = 0; k < i; k++) {
        if (record[k] === j || Math.abs(i - k) === Math.abs(record[k] - j)) {
            return false;
        }
    }
    return true
}
function toString(record, n) {
    let res = []
    for (let i = 0; i < n; i++) {
        let strArr = new Array(n).fill('.')
        let j = record[i]
        strArr[j] = 'Q'
        res.push(strArr.join(''))
    }
    return res
}

return res
```

```
// console.log(res)
};
```

优化：位运算加速