算法题准备材料

Tyrael.wang

# JAVA常用API

整数

public static int bitCount(int i)

https://blog.csdn.net/zhouzipeng000/article/details/56676885

1. 栈Stack类继承Vector类。主要方法：pushpoppeekempty（Java编程思想：Java本身的栈设计欠佳。暴露了很多Vector的方法。）

2. 队列Java 有Queue接口。LinkedList实现了Queue接口。offer插入队尾remove、pollelement、peek3. 字符串相关StringBuffer 线程安全StringBuilder 非线程安全字符串逆转stringBuilder.reverse4. 集合Arrays.binSearch的返回值5. Math算法相关：PriorityQueue 基于堆实现的无界队列，非线程安全的An unbounded priority {@linkplain Queue queue} based on a priority heap.

Treemap

Floor

ceiling

# 经典数据结构/算法

## 排序

## 二叉树

### 性质

### 构建

106. Construct Binary Tree from Inorder and Postorder Traversal

### 遍历

**public class** IterativeInOrder {  
 Stack<TreeNode> **stack** = **new** Stack<>();  
  
 **public void** run(TreeNode root){  
 *//初始化* pushAll(root);  
 **while**(!**stack**.empty()){  
 TreeNode visit = **stack**.pop();  
 doWork(visit);  
 pushAll(visit.**right**);  
 }  
 }  
  
 **public void** doWork(TreeNode visit){  
 System.***out***.println(visit.**val**);  
 }  
  
 */\*\*  
 \* 沿左树，一直压到叶子  
 \** ***@param root*** *\*/* **private void** pushAll(TreeNode root){  
 TreeNode thisNode = root;  
 **while**(thisNode!= **null**){  
 **stack**.push(thisNode);  
 thisNode = thisNode.**left**;  
 }  
 }  
}

### 二叉搜索树Binary Search Tree

https://leetcode.com/problems/unique-binary-search-trees/description/

C(2n,n)/(n+1)

<https://en.wikipedia.org/wiki/Catalan_number#Applications_in_combinatorics>

1. 完全二叉樹、二叉搜索树

Given n, how many structurally unique BST's (binary search trees) that store values 1 ... n?

unique-binary-search-trees-ii：如何生成具体的树：递归。左子树从0到n。（注意类似的情况都可以用catalan数计数。）

98. Validate Binary Search Tree：中根序遍历，递增

450. Delete Node in a BST

1. 叶子节点直接删除

2. 单孩节点：直接替代

3. 双孩节点：

方案一：前驱值替代，再删除前驱

方案二：左树替代，右树下降低到最低

230. Kth Smallest Element in a BST：中根序遍历，计数

### 红黑树

理解要点

## 字典树

**public class** Node {  
 Node[] **children**;  
 String **leaf**;  
  
 */\*\*  
 \*  
 \** ***@param n*** *孩子容量  
 \*/* **public** Node(**int** n) {  
 **children** = **new** Node[n];  
 }  
}

**public class** Trie {  
 Node **root**;  
  
 **public** Node create(){  
 **root** = **new** Node(26);  
 **return root**;  
 }  
  
 **public void** insert(String s){  
 *//逐个查找，找不到则建点* Node node = **root**;  
 **for** (**int** i = 0; i < s.length(); i++) {  
 **int** index = order(s.charAt(0));  
 Node child = node.**children**[index];  
 **if**(child == **null**){  
 child = **new** Node(26);  
 node.**children**[index] = child;  
 node = child;  
 }**else**{  
 node = child;  
 }  
 }  
 node.**leaf** = s;  
 }  
  
 **public boolean** search(String s){  
 Node node = **root**;  
 **for** (**int** i = 0; i < s.length(); i++) {  
 **int** index = order(s.charAt(0));  
 Node child = node.**children**[index];  
 **if**(child == **null**){  
 **return false**;  
 }**else**{  
 node = child;  
 }  
 }  
 *//如果该词是其他词的前缀，也不存在。* **return** node.**leaf** != **null**;  
 }  
  
 **public int** order(**char** a){  
 **return** a - **'a'**;  
 }  
}

211. Add and Search Word - Data structure design

变种：search(word) can search a literal word or a regular expression string containing only letters a-z or .. A . means it can represent any one letter.

648. Replace Words

676. Implement Magic Dictionary

677. Map Sum Pairs

红黑树

前缀和

线段树

图

拓扑排序

并查集扩展元素数量集合数量

排序

计算几何

算法导论33章

字符串匹配

kmp算法ac自动机

## 素数计算

https://leetcode.com/problems/2-keys-keyboard/solution/

## 二分查找

含重复元素、等号、上下界

34. Search for a Range

## 流算法

### 摩尔多数投票算法

查找1/2、1/3水王

### 水库采样

## 下一个排列

O(n);O(1)

**public** **void** **nextPermutation(int[]** nums**)** **{**

**int** i **=** nums**.**length **-** 2**;**

**while** **(**i **>=** 0 **&&** nums**[**i **+** 1**]** **<=** nums**[**i**])** **{ //找到第一个不满足逆序的数**

i**--;**

**}**

**if** **(**i **>=** 0**)** **{**

**int** j **=** nums**.**length **-** 1**;**

**while** **(**j **>=** 0 **&&** nums**[**j**]** **<=** nums**[**i**])** **{ //找到比i位置大一点的数**

j**--;**

**}**

swap**(**nums**,** i**,** j**); //i位置确定好**

**}**

reverse**(**nums**,** i **+** 1**); //余下的按正序排列即可**

**}**

## 树状数组/binary indexed tree/ Fenwick tree

https://blog.csdn.net/l664675249/article/details/50157669

**问题：求一个数组中连续n项的和。**

首先想到的肯定是做一个循环，把这个连续的n项加起来，时间复杂度为O（n）。复杂度为n，看起来还不错，再说了求n个数的和，怎么也要加n次吧，所以说这应该就是最优解了，但是一提交结果是**Time Limit Exceeded**，顿时傻眼了，难道还有复杂度更低的方法？

**会不会有O（logn）的解法？**

O（n）的那个算法，如果只操作一次还是可以接受的，但是如果需要大量的求和操作，比如第一次求下标（1，1234）的和第二次求下标（2，1024）的和，很容易发现在第一次计算的过程中（2，1024）的和是计算过的，只是没有保存下来，导致第二次求和的时候还要再算一遍。你有没有想过，如果事先把一部分的和先计算并保存起来，这样会不会更快一些呢？

**Binary Indexed Tree(BIT)**

其实树状数组(Binary Indexed Tree(BIT), Fenwick Tree)就是这样做的，他是一个查询和修改复杂度都为log(n)的数据结构。主要用于查询任意两位之间的所有元素之和，但是每次只能修改一个元素的值。

**核心思想:**

* 树状数组中的每个元素是原数组中一个或者多个连续元素的和。
* 在进行连续求和操作a[1]+…+a[n]时，只需要将树状数组中某几个元素的和即可。时间复杂度为O(lgn)

下面是一个示意图



a[]: 保存原始数据的数组   
e[]: 树状数组，其中的任意一个元素e[i]可能是一个或者多个a数组中元素的和。如e[2]=a[1]+a[2]; e[3]=a[3]，e[4]=a[1]+a[2]+a[3]+a[4]。   
e[i]中的元素：如果数字 i 的二进制表示中末尾有**k个连续的0，则e[i]是a数组中2^k个元素的和**，则e[i]=a[i-2^k+1]+a[i-2^k+2]+…+a[i-1]+a[i]。也就是说，**e[i]中每一个元素管理着a[]中若干个元素的和，并且各个元素管理的区间没有重叠。**

　　　　如：4=100(2)　　e[4]=a[1]+a[2]+a[3]+a[4];   
　　　　　　6=110(2)　　e[6]=a[5]+a[6]   
　　　　　　7=111(2)　　e[7]=a[7]   
　　　　　　   
计算2^k的两个方法

* 2^k = (i & (-i)); (利用机器补码特性)
* 2^k = (i & (i^(i-1));

**父节点，子节点**

**父节点**

是离它最近的，且编号末位连续0比它多的就是它的父亲,如e[2]是e[1]的儿子；e[4]是e[2]的儿子。   
e[4] = e[2]+e[3]+a[4] = a[1]+a[2]+a[3]+a[4] ，e[2]、e[3]的后继就是e[4]。

**计算方法**

lowbit(i) = ( (i-1) ^ i) & i ; //或者(i & (-i))   
节点e[i]的父节点为 e[ i - lowbit(i) ]

**子节点**

最近的，编号即为比自己小的，最末连续0比自己多的节点。如e[7]的子节点是e[6],e[6]的子节点是e[4]

**计算方法**

lowbit(i) = ( (i-1) ^ i) & i ; //或者(i & (-i))   
节点e[i]的子节点为 e[ i + lowbit(i) ]

**实现代码**

public class NumArray {

private int[] tree; //Binary Indexed Tree

private int[] nums; //原始数组

public NumArray(int[] nums) {

this.nums = nums;

int sum = 0;

int lowbit;

tree = new int[nums.length + 1];

for (int i = 1; i < tree.length; i++) {

sum = 0;

lowbit = i & ((i - 1) ^ i);

for (int j = i; j > i - lowbit; j--) {

sum = sum + nums[j - 1];

}

tree[i] = sum;

}

}

//更新

void update(int i, int val) {

int tem = val - nums[i];

nums[i] = val;

i++;

for (; i < tree.length; i = i + (i & ((i - 1) ^ i))) {

tree[i] += tem;

}

}

public int sumRange(int i, int j) {

return getSum(j) - getSum(i - 1);

}

//求和

public int getSum(int i) {

int sum = 0;

i++;

while (i > 0) {

sum = sum + tree[i];

i = i - (i & ((i - 1) ^ i));

}

return sum;

}

}

## 树

310. Minimum Height Trees

a tree is an undirected graph in which any two vertices are connected by exactly one path. In other words, any connected graph without simple cycles is a tree.

We start from every end, by end we mean vertex of degree 1 (aka leaves). We let the pointers move the same speed. When two pointers meet, we keep only one of them, until the last two pointers meet or one step away we then find the roots.

## 线段树

307. Range Sum Query – Mutable

<https://leetcode.com/problems/range-sum-query-mutable/solution/>

https://en.wikipedia.org/wiki/Segment\_tree

Segment tree is a very flexible data structure, because it is used to solve numerous range query problems like finding minimum, maximum, sum, greatest common divisor, least common denominator in array in logarithmic time.



*Figure 2. Illustration of Segment tree.*

The segment tree for array a[0, 1, \ldots ,n-1]*a*[0,1,…,*n*−1] is a binary tree in which each node contains **aggregate**information (min, max, sum, etc.) for a subrange [i \ldots j][*i*…*j*] of the array, as its left and right child hold information for range [i \ldots \frac{i+j}{2}][*i*…​2​​*i*+*j*​​] and [\frac{i + j}{2} + 1, j][​2​​*i*+*j*​​+1,*j*].

Segment tree could be implemented using either an array or a tree. For an array implementation, if the element at index i*i* is not a leaf, its left and right child are stored at index 2i2*i* and 2i + 12*i*+1 respectively.

In the example above (Figure 2), every leaf node contains the initial array elements {2,4,5,7,8,9}. The internal nodes contain the sum of the corresponding elements in range - (11) for the elements from index 0 to index 2. The root (35) being the sum of its children (6);(29), holds the total sum of the entire array.

Segment Tree can be broken down to the three following steps:

1. Pre-processing step which builds the segment tree from a given array.
2. Update the segment tree when an element is modified.
3. Calculate the Range Sum Query using the segment tree.

1. Build segment tree

We will use a very effective bottom-up approach to build segment tree. We already know from the above that if some node p*p* holds the sum of [i \ldots j][*i*…*j*] range, its left and right children hold the sum for range [i \ldots \frac{i + j}{2}][*i*…​2​​*i*+*j*​​] and [\frac{i + j}{2} + 1, j][​2​​*i*+*j*​​+1,*j*] respectively.

Therefore to find the sum of node p*p*, we need to calculate the sum of its right and left child in advance.

We begin from the leaves, initialize them with input array elements a[0, 1, \ldots, n-1]*a*[0,1,…,*n*−1]. Then we move upward to the higher level to calculate the parents' sum till we get to the root of the segment tree.

**Java**

**int[]** tree**;**

**int** n**;**

**public** **NumArray(int[]** nums**)** **{**

**if** **(**nums**.**length **>** 0**)** **{**

n **=** nums**.**length**;**

tree **=** **new** **int[**n **\*** 2**];**

buildTree**(**nums**);**

**}**

**}**

**private** **void** **buildTree(int[]** nums**)** **{**

**for** **(int** i **=** n**,** j **=** 0**;** i **<** 2 **\*** n**;** i**++,** j**++)**

tree**[**i**]** **=** nums**[**j**];**

**for** **(int** i **=** n **-** 1**;** i **>** 0**;** **--**i**)**

tree**[**i**]** **=** tree**[**i **\*** 2**]** **+** tree**[**i **\*** 2 **+** 1**];**

**}**

**Complexity Analysis**

* Time complexity : O(n)*O*(*n*)

Time complexity is O(n)*O*(*n*), because we calculate the sum of one node during each iteration of the for loop. There are approximately 2n2*n* nodes in a segment tree.

This could be proved in the following way: Segmented tree for array with n*n* elements has n*n* leaves (the array elements itself). The number of nodes in each level is half the number in the level below.

So if we sum the number by level we will get:

n + n/2 + n/4 + n/8 + \ldots + 1 \approx 2n*n*+*n*/2+*n*/4+*n*/8+…+1≈2*n*

* Space complexity : O(n)*O*(*n*).

We used 2n2*n* extra space to store the segment tree.

2. Update segment tree

When we update the array at some index i*i* we need to rebuild the segment tree, because there are tree nodes which contain the sum of the modified element. Again we will use a bottom-up approach. We update the leaf node that stores a[i]*a*[*i*]. From there we will follow the path up to the root updating the value of each parent as a sum of its children values.

**Java**

**void** **update(int** pos**,** **int** val**)** **{**

pos **+=** n**;**

tree**[**pos**]** **=** val**;**

**while** **(**pos **>** 0**)** **{**

**int** left **=** pos**;**

**int** right **=** pos**;**

**if** **(**pos **%** 2 **==** 0**)** **{**

right **=** pos **+** 1**;**

**}** **else** **{**

left **=** pos **-** 1**;**

**}**

*// parent is updated after child is updated*

tree**[**pos **/** 2**]** **=** tree**[**left**]** **+** tree**[**right**];**

pos **/=** 2**;**

**}**

**}**

**Complexity Analysis**

* Time complexity : O(\log n)*O*(log*n*).

Algorithm has O(\log n)*O*(log*n*) time complexity, because there are a few tree nodes with range that include i*i*th array element, one on each level. There are \log(n)log(*n*) levels.

* Space complexity : O(1)*O*(1).

3. Range Sum Query

We can find range sum query [L, R][*L*,*R*] using segment tree in the following way:

Algorithm hold loop invariant:

l \le r*l*≤*r* and sum of [L \ldots l][*L*…*l*] and [r \ldots R][*r*…*R*] has been calculated, where l*l* and r*r* are the left and right boundary of calculated sum. Initially we set l*l* with left leaf L*L* and r*r* with right leaf R*R*. Range [l, r][*l*,*r*] shrinks on each iteration till range borders meets after approximately \log nlog*n* iterations of the algorithm

* Loop till l \le r*l*≤*r*
  + Check if l*l* is right child of its parent P*P*
    - l*l* is right child of P*P*. Then P*P* contains sum of range of l*l* and another child which is outside the range [l, r][*l*,*r*] and we don't need parent P*P* sum. Add l*l* to sum*sum* without its parent P*P* and set l*l* to point to the right of P*P* on the upper level.
    - l*l* is not right child of P*P*. Then parent P*P* contains sum of range which lies in [l, r][*l*,*r*]. Add P*P* to sum*sum*and set l*l* to point to the parent of P*P*
  + Check if r*r* is left child of its parent P*P*
    - r*r* is left child of P*P*. Then P*P* contains sum of range of r*r* and another child which is outside the range [l, r][*l*,*r*] and we don't need parent P*P* sum. Add r*r* to sum*sum* without its parent P*P* and set r*r* to point to the left of P*P* on the upper level.
    - r*r* is not left child of P*P*. Then parent P*P* contains sum of range which lies in [l, r][*l*,*r*]. Add P*P* to sum*sum*and set r*r* to point to the parent of P*P*

**Java**

**public** **int** **sumRange(int** l**,** **int** r**)** **{**

*// get leaf with value 'l'*

l **+=** n**;**

*// get leaf with value 'r'*

r **+=** n**;**

**int** sum **=** 0**;**

**while** **(**l **<=** r**)** **{**

**if** **((**l **%** 2**)** **==** 1**)** **{**

sum **+=** tree**[**l**];**

l**++;**

**}**

**if** **((**r **%** 2**)** **==** 0**)** **{**

sum **+=** tree**[**r**];**

r**--;**

**}**

l **/=** 2**;**

r **/=** 2**;**

**}**

**return** sum**;**

**}**

**Complexity Analysis**

* Time complexity : O(\log n)*O*(log*n*)

Time complexity is O(\log n)*O*(log*n*) because on each iteration of the algorithm we move one level up, either to the parent of the current node or to the next sibling of parent to the left or right direction till the two boundaries meet. In the worst-case scenario this happens at the root after \log nlog*n* iterations of the algorithm.

* Space complexity : O(1)*O*(1).

## 分桶法和平方分割

## 图

### 图的表示

133. Clone Graph

### 拓扑排序

### 单源路径

Dijkstra

<https://leetcode.com/problems/cheapest-flights-within-k-stops/description/>

单源路径

743. Network Delay Time

带权单源路径

### 多源路径

带权多源路径

399. Evaluate Division

## 01背包问题

只能穷举

NP问题

字符串匹配

318. Maximum Product of Word Lengths

## 并查集

200. Number of Islands

基本

扩展

元素数

集合数

## 字符串匹配

## 扩展

### 马拉车算法

### 格雷码

N位格雷码

for (int i = 0; i < 1<<n; i++)

result.add(i ^ i>>1);

### 四平方定理

## 简单

### 逆波兰表达式

150. Evaluate Reverse Polish Notation

# 从形式到思路

## 广义表

341. Flatten Nested List Iterator

385. Mini Parser

394. Decode String

借助栈实现

## 数字

402. Remove K Digits

解法一：

删除数字-》保留数字-》高位重要

解法二：

one can simply scan from left to right, and remove the first "peak" digit; the peak digit is larger than its right neighbor.

使用栈

738. Monotone Increasing Digits

when the number is 123454321, we could have a candidate of 123449999. It seems like a decent strategy is to take a monotone increasing prefix of N, then decrease the number before the "cliff" (the index where adjacent digits decrease for the first time) if it exists, and replace the rest of the characters with 9s.

### 位操作

397. Integer Replacement

//more

海明距离

477. Total Hamming Distance

421. Maximum XOR of Two Numbers in an Array

338. Counting Bits

779. K-th Symbol in Grammar

https://leetcode.com/problems/k-th-symbol-in-grammar/discuss/113705/JAVA-one-line

## 从回溯到dp

https://leetcode.com/problems/target-sum/description/

## 数组/矩阵区间和/积

https://leetcode.com/problems/range-sum-query-2d-immutable/description/

## 四则运算

https://leetcode.com/problems/different-ways-to-add-parentheses/description/

399. Evaluate Division

## 字符/数字集合

通常用数组来存储，但是与没有前后关系，也跟索引没有关系。

### 排列/组合

60. Permutation Sequence

46. Permutations

不同元素

47. Permutations II

重复的下一个排列

77. Combinations

752. Open the Lock

两端bfs

### 元素和

<https://leetcode.com/problems/combination-sum-iv/description/>

回溯-》dp

### 回文

字串

<https://leetcode.com/problems/palindromic-substrings/description/>

131. Palindrome Partitioning

回溯

子序列（中漏）

https://leetcode.com/problems/longest-palindromic-subsequence/description/

dp[i][j]: the longest palindromic subsequence's length of substring(i, j)

State transition:

dp[i][j] = dp[i+1][j-1] + 2 if s.charAt(i) == s.charAt(j)

otherwise, dp[i][j] = Math.max(dp[i+1][j], dp[i][j-1])

Initialization: dp[i][i] = 1

### 背包问题/选或不选

集合划分问题

https://en.wikipedia.org/wiki/Partition\_problem

416. Partition Equal Subset Sum

698. Partition to K Equal Sum Subsets

473. Matchsticks to Square

Dfs 大值在前

背包问题

https://leetcode.com/problems/ones-and-zeroes/description/

39. Combination Sum

选用多次

40. Combination Sum II

集合中有重复元素，选用一次

216. Combination Sum III

元素不可重复

377. Combination Sum IV

### 推进

<https://leetcode.com/problems/house-robber-ii/description/>

DP

### 两个字符串的关系

缩小规模-》DP

https://leetcode.com/problems/minimum-swaps-to-make-sequences-increasing/description/

722. Remove Comments

line comments, and block comments.

注意嵌套情况

“”字符串嵌套，转义字符

//可以在一行的任意位置

/\*/不完整

https://leetcode.com/problems/unique-substrings-in-wraparound-string/description/

https://leetcode.com/problems/unique-substrings-in-wraparound-string/discuss/95439/Concise-Java-solution-using-DP

712. Minimum ASCII Delete Sum for Two Strings

## 单词列表

查找：字典树

212. Word Search II

318. Maximum Product of Word Lengths

126. Word Ladder II

单词变种最短路径

双向bfs

注意，没有答案的，树的高度只要一半就可以了。

新生的节点，不要重复已有的节点。但是同一层的可以重复。

变种后集合的hash查找，比逐个单词比较是否变种，要快很多。

676. Implement Magic Dictionary

For the method search, you'll be given a word, and judge whether if you modify exactly one character into another character in this word, the modified word is in the dictionary you just built.

## 有序数组

220. Contains Duplicate III

380. Insert Delete GetRandom O(1)

777. Swap Adjacent in LR String

### 排序

324. Wiggle Sort II

快排的应用。

大小元素的摆放

### 前K

373. Find K Pairs with Smallest Sums

215. Kth Largest Element in an Array、

快排

692. Top K Frequent Words

### 波形

股票

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/description/

状态思想

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/discuss/75928/Share-my-DP-solution-(By-State-Machine-Thinking)

https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/description/

456. 132 Pattern

区间中的值

//more

### 划分

813. Largest Sum of Averages

1. 回溯 + memo

2. 二维dp

### 子序列长度

n!个子序列

https://leetcode.com/problems/wiggle-subsequence/description/

https://leetcode.com/problems/is-subsequence/description/

public boolean isSubsequence(String s, String t) {

if (s.length() == 0) return true;

int indexS = 0, indexT = 0;

while (indexT < t.length()) {

if (t.charAt(indexT) == s.charAt(indexS)) {

indexS++;

if (indexS == s.length()) return true;

}

indexT++;

}

return false;

}

491. Increasing Subsequences

dfs

### 丑数

质因子只含235的数

https://leetcode.com/problems/ugly-number-ii/description/

三个队列

313. Super Ugly Number

### 递增子串

https://leetcode.com/problems/number-of-longest-increasing-subsequence/description/

### 排序数组查找

二分查找变种

### 逆序

775. Global and Local Inversions

To check if #global inversion=#local inversion, we just need to ensure that there are no such inversion:i>j+1, a[i]<a[j]

### 两个数的关系

2/3/4 Sum （smaller、closet）

暂定一个值，排序查找另一个值

注意平均值关系，运用两个指针单向滑动

### 小区间

思路：计数排序、二分查找

例：

825. Friends Of Appropriate Ages

### 区间和/积

log(∏​*i*​​*x*​*i*​​)=∑​*i*​​log*x*​*i*​​

前缀和

累积和：累积和特点：对于单一元素，单调递增。

209. Minimum Size Subarray Sum

713. Subarray Product Less Than K

明显：列举所有：O(n3)

累积和：O(n2)

累积和：注意脑中有一个递增序列：二分查找(nlogn)

两个指针：某一个index，有条件提前结束查找：（n）

238. Product of Array Except Self

空间利用，

利用输出数组

Dp空间的优化

### 利用索引

769. Max Chunks To Make Sorted

768. Max Chunks To Make Sorted II

565. Array Nesting

442. Find All Duplicates in an Array

映射空间大的话，最终可以恢复数组

274. H-Index

275. H-Index II

利用索引+计数排序+DP

二分搜索

526. Beautiful Arrangement

### 环/并查集

并查集问题也可以用dfs/bfs来解决。

565. Array Nesting

547. Friend Circles

统计不同集个数

### 其他

数组实际上是一个map可以在原数组操作

https://leetcode.com/problems/delete-and-earn/description/

## 矩阵

解题思路：回溯、dfs、bfs、dp

路径搜索，一般只依赖邻居，通常可以用dp解决

最简单的空间占用是n2

有些可以优化到n，如2向路径。跟依赖多少方向有关系。

有些可以优化到1

417. Pacific Atlantic Water Flow

### 矩阵2向路径

62 Unique Paths排列组合问题哦：DRRRDRRR（m\*n）!/(m!\*n!)

63. Unique Paths II

DP

### 矩阵4向路径

矩阵邻居搜索

79. Word Search

212. Word Search II

出界

https://leetcode.com/problems/out-of-boundary-paths/description/

542. 01 Matrix

//more

529. Minesweeper

Search rules:

1. If click on a mine ('M'), mark it as 'X', stop further search.
2. If click on an empty cell ('E'), depends on how many surrounding mine:  
   2.1 Has surrounding mine(s), mark it with number of surrounding mine(s), stop further search.  
   2.2 No surrounding mine, mark it as 'B', continue search its 8 neighbors.

### 矩阵搜索

74. Search a 2D Matrix

二分搜索，mid的计算比较复杂

### 三角形

120. Triangle

坐标变换、dp

756. Pyramid Transition Matrix

一行一行处理，逐渐往上堆

字符可以看作数字

### 区域面积

拆分维度，分别dp，再综合计算

https://leetcode.com/problems/maximal-square/description/

764. Largest Plus Sign

If we knew the longest possible arm length L\_u, L\_l, L\_d, L\_r*L*​*u*​​,*L*​*l*​​,*L*​*d*​​,*L*​*r*​​ in each direction from a center, we could know the order \min(L\_u, L\_l, L\_d, L\_r)min(*L*​*u*​​,*L*​*l*​​,*L*​*d*​​,*L*​*r*​​) of a plus sign at that center. We could find these lengths separately using dynamic programming.

## 数学

3.2.4. 数组积

29 Divides1：位操作，变大除数s2：长除法

326 PowerOfThree// 1162261467 is 3^19, 3^20 is bigger than int return ( n>0 && 1162261467%n==0);

概率

https://leetcode.com/problems/soup-servings/description/

丑数

https://leetcode.com/problems/integer-break/description/

all factors should be 2 or 3 (N > 4)

3 \* 3 > 2 \* 2 \* 2.

## 链表

19. Remove Nth Node From End两个指针O(n)

138. Copy List with Random Pointer利用原链表指针相对位置关系=》指针关系

## 区间维护

56 Merge Intervals排序以后按顺序合并

228. Summary Ranges

646. Maximum Length of Pair Chain

452. Minimum Number of Arrows to Burst Balloons

435. Non-overlapping Intervals

课程时间安排

https://en.wikipedia.org/wiki/Interval\_scheduling#Interval\_Scheduling\_Maximization

## 二叉树

236. Lowest Common Ancestor of a Binary Tree

层次遍历

117. Populating Next Right Pointers in Each Node II

114. Flatten Binary Tree to Linked List

652. Find Duplicate Subtrees

标志id的思想。

623. Add One Row to Tree

### 路径

路径dfs

113. Path Sum II

根到叶

437. Path Sum III

任意点

优化算法：每个路径构建前缀和

124. Binary Tree Maximum Path Sum

129. Sum Root to Leaf Numbers

### Level

129. Sum Root to Leaf Numbers

## 树

<https://leetcode.com/problems/house-robber-iii/description/>

遍历

https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/description/

## 图

207 Course Schedule依赖问题：拓扑排序（需要手写）时间问题：贪心算法

缔结斯科拉

https://leetcode.com/problems/cheapest-flights-within-k-stops/description/

332. Reconstruct Itinerary

欧拉回路

<https://www.cnblogs.com/acxblog/p/7390301.html>

<https://blog.csdn.net/u011466175/article/details/18861415>

环检测

802. Find Eventual Safe States

### 二分图

Our goal is trying to use two colors to color the graph and see if there are any adjacent nodes having the same color.

Initialize a color[] array for each node. Here are three states for colors[] array:

-1: Haven't been colored.

0: Blue.

1: Red.

For each node,

If it hasn't been colored, use a color to color it. Then use the other color to color all its adjacent nodes (DFS).

If it has been colored, check if the current color is the same as the color that is going to be used to color it. (Please forgive my english... Hope you can understand it.)

## 猜大小

https://leetcode.com/problems/guess-number-higher-or-lower-ii/description/

## 极大极小

486. Predict the Winner

https://leetcode.com/problems/predict-the-winner/solution/

649. Dota2 Senate

https://leetcode.com/problems/can-i-win/discuss/95277/Java-solution-using-HashMap-with-detailed-explanation

## 拾遗

284. Peeking Iterator

473. Matchsticks to Square

621. Task Scheduler

Input: tasks = ["A","A","A","B","B","B"], n = 2

Output: 8

Explanation: A -> B -> idle -> A -> B -> idle -> A -> B.

406. Queue Reconstruction by Height

一个人的位置，由比他高的人决定。

从高到低确定位置。

393. UTF-8 Validation

A character in UTF8 can be from 1 to 4 bytes long, subjected to the following rules:

For 1-byte character, the first bit is a 0, followed by its unicode code.

For n-bytes character, the first n-bits are all one's, the n+1 bit is 0, followed by n-1 bytes with most significant 2 bits being 10.

This is how the UTF-8 encoding would work:

Char. number range | UTF-8 octet sequence

(hexadecimal) | (binary)

--------------------+---------------------------------------------

0000 0000-0000 007F | 0xxxxxxx

0000 0080-0000 07FF | 110xxxxx 10xxxxxx

0000 0800-0000 FFFF | 1110xxxx 10xxxxxx 10xxxxxx

0001 0000-0010 FFFF | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

捞针

260. Single Number III

全部两次+两个一次

//more

# 常用解题思路

## 答案空间穷举回溯搜索

大多数问题都可以用该方法解决。有些问题存在效率更高的方式；有些不存在，例如背包问题。

需要有能力看出哪些问题不能优化。

参考资料，算法导论NP问题。

细节优化：可以采用双端bfs、减支方法优化。虽然不能提高O，可以快一点。

## DP

能够应用DP的问题，通常有两种可能

1. 可以构建无后效性递归式
2. 可以应用回溯算法，并且有大量重复。

表面上是TopDown算法，一般可以转化为bottomUp算法。

DP的空间，

思考时，可以用较大的空间。

最终的优化，看递归式，到底依赖了多少上一步的结果。

https://leetcode.com/problems/2-keys-keyboard/discuss/105932/Java-solutions-from-naive-DP-to-optimized-DP-to-non-DP

常用解决方案：分治、二分、DP哈希、双指针、排序

双向bfs

贪心算法专题

关键是需要证明为什么贪心是对的。

狭义贪心算法是动态规划的特例。贪心算法代码看起来比动态规划简单，实际上思维上要复杂一点。算法导论有介绍321 Create Maximum Number广义贪心算法

回溯

心中一颗答案空间搜索树

两种搜索：dfs、bfs

Dfs用递归、栈实现

Bfs队列实现

# 常用正则表达式

# 测试细节

解题流程

确认算法、复杂度

确认输入边界，说明分支处理。

大于还是大于等于？

编写常规流程

代码走查

完善测试用例，确认测试通过。

Comparator注意越界

## 数字验证/解析

<https://leetcode.com/problems/valid-number/discuss/23977/A-clean-design-solution-By-using-design-pattern>

测试用例总结parsedouble的代码没有开源

整数、小数、科学计数法1. 前后空格2. 前0，多个03. 负04. 小数，后多个05. 小数，小数点前后没有06. 指数，指数为07. 字母、空格，中间夹杂8. 空值Pattern.matches("(\\+|-)?(\\d+(\\.\\d\*)?|\\.\\d+)(e(\\+|-)?\\d+)?", s);test(1, "123", true);test(2, " 123 ", true);test(3, "0", true);test(4, "0123", true); //Cannot agreetest(5, "00", true); //Cannot agreetest(6, "-10", true);test(7, "-0", true);test(8, "123.5", true);test(9, "123.000000", true);test(10, "-500.777", true);test(11, "0.0000001", true);test(12, "0.00000", true);test(13, "0.", true); //Cannot be more disagree!!!test(14, "00.5", true); /ly cannot agreetest(15, "123e1", true);test(16, "1.23e10", true);test(17, "0.5e-10", true);test(18, "1.0e4.5", false);test(19, "0.5e04", true);test(20, "12 3", false);test(21, "1a3", false);test(22, "", false);test(23, " ", false);test(24, null, false);test(25, ".1", true); //Ok, if you say sotest(26, ".", false);test(27, "2e0", true); //Really?!test(28, "+.8", true); (29, " 005047e+6", true); //Damn = =|||Here is the final Regex I got based on their definitionPattern.matches("(\\+|-)?(\\d+(\\.\\d\*)?|\\.\\d+)(e(\\+|-)?\\d+)?", s);But I thought my original one should be more rigorous!Pattern.matches("-?(([1-9]{1}+\\d\*|0)(\\.\\d+)?|\\.\\d+)(e-?[1-9]{1}+\\d\*)?", s);

数字解析

https://leetcode.com/problems/string-to-integer-atoi/discuss/4654/My-simple-solution

I think we only need to handle four cases:

discards all leading whitespaces

sign of the number

overflow

invalid input

过程中计算负数，因为负数范围大。

注意越界的处理

//容错比较低，需要提前验证

public static int parseInt(String s, int radix)

throws NumberFormatException

{

/\*

\* WARNING: This method may be invoked early during VM initialization

\* before IntegerCache is initialized. Care must be taken to not use

\* the valueOf method.

\*/

if (s == null) {

throw new NumberFormatException("null");

}

if (radix < Character.MIN\_RADIX) {

throw new NumberFormatException("radix " + radix +

" less than Character.MIN\_RADIX");

}

if (radix > Character.MAX\_RADIX) {

throw new NumberFormatException("radix " + radix +

" greater than Character.MAX\_RADIX");

}

int result = 0;

boolean negative = false;

int i = 0, len = s.length();

int limit = -Integer.MAX\_VALUE;

int multmin;

int digit;

if (len > 0) {

char firstChar = s.charAt(0);

if (firstChar < '0') { // Possible leading "+" or "-"

if (firstChar == '-') {

negative = true;

limit = Integer.MIN\_VALUE;

} else if (firstChar != '+')

throw NumberFormatException.forInputString(s);

if (len == 1) // Cannot have lone "+" or "-"

throw NumberFormatException.forInputString(s);

i++;

}

multmin = limit / radix;

while (i < len) {

// Accumulating negatively avoids surprises near MAX\_VALUE

digit = Character.digit(s.charAt(i++),radix);

if (digit < 0) {

throw NumberFormatException.forInputString(s);

}

//是否可以乘，升位

if (result < multmin) {

throw NumberFormatException.forInputString(s);

}

result \*= radix;

//是否可以增加本位的值

if (result < limit + digit) {

throw NumberFormatException.forInputString(s);

}

result -= digit;

}

} else {

throw NumberFormatException.forInputString(s);

}

return negative ? result : -result;

}

3. Longest Substring Without Repeating Characters

滑动窗口

If a substring s\_{ij}s

​ij

​​ from index ii to j - 1j−1 is already checked to have no duplicate characters. We only need to check if s[j]s[j] is already in the substring s\_{ij}s

​ij

​​ .

355. Design Twitter

字符串匹配

Rolling hash

187. Repeated DNA Sequences

//TODO

凡是单调的列表，就要考虑下二分搜索。

或者存在有个有限空间。

Anagram

性质

字母数相等

与位置无关

注意结果集，或者任何集比较小的情况，

可以在此做文章。

二分、以结果为key作map

676. Implement Magic Dictionary

Trie

排序

快排

链表快排

插入法

两个指针法

链表归并排序

快慢指针划分

非递归归并排序

写循环注意参照for循环，不要忘了推进

链表环检测

环入口

142. Linked List Cycle II

my solution is like this: using two pointers, one of them one step at a time. another pointer each take two steps. Suppose the first meet at step k,the length of the Cycle is r. so..2k-k=nr,k=nr

Now, the distance between the start node of list and the start node of cycle is s. the distance between the start of list and the first meeting node is k(the pointer which wake one step at a time waked k steps).the distance between the start node of cycle and the first meeting node is m, so...s=k-m,

s=nr-m=(n-1)r+(r-m),here we takes n = 1..so, using one pointer start from the start node of list, another pointer start from the first meeting node, all of them wake one step at a time, the first time they meeting each other is the start of the cycle.

高度平衡二叉树构建

Power（x，n）

数学

https://leetcode.com/problems/reach-a-number/solution/

公约数

裴蜀定理

https://leetcode.com/problems/water-and-jug-problem/discuss/83715/Math-solution-Java-solution

四则运算的原始算法

https://leetcode.com/problems/multiply-strings/description/

排列

60. Permutation Sequence

https://leetcode.com/problems/permutation-sequence/discuss/22507/%22Explain-like-I'm-five%22-Java-Solution-in-O(n)

https://leetcode.com/problems/next-greater-element-iii/description/

https://leetcode.com/problems/integer-replacement/description/

https://leetcode.com/problems/ugly-number/description/

<https://leetcode.com/problems/ugly-number-ii/description/>

<https://leetcode.com/problems/ugly-number-ii/discuss/69362/O(n)-Java-solution>

https://leetcode.com/problems/super-ugly-number/description/

https://leetcode.com/problems/rotate-function/discuss/87853/Java-O(n)-solution-with-explanation

F(k) = F(k-1) + sum - nBk[0]

旋转函数，注意前后的数之间的关系

368. Largest Divisible Subset

取模运算

https://baike.baidu.com/item/%E5%8F%96%E6%A8%A1%E8%BF%90%E7%AE%97/10739384?fr=aladdin

算法道路数论章节

<https://leetcode.com/problems/super-pow/description/>

四平方定理

<https://leetcode.com/problems/perfect-squares/description/>

平方数、奇数偶数

<https://leetcode.com/problems/bulb-switcher/description/>

<https://leetcode.com/problems/bulb-switcher-ii/solution/>

答案空间很小，无限循环

数字单词

The **even** digits all have a unique letter while the **odd** digits all don't:

zero: Only digit with z  
two: Only digit with w  
four: Only digit with u  
six: Only digit with x  
eight: Only digit with g

The odd ones for easy looking, each one's letters all also appear in other digit words:  
one, three, five, seven, nine

<https://leetcode.com/problems/count-numbers-with-unique-digits/discuss/>

<https://leetcode.com/problems/integer-break/description/>

https://leetcode.com/problems/escape-the-ghosts/discuss/116678/Why-interception-in-the-middle-is-not-a-good-idea-for-ghosts.

https://leetcode.com/problems/minimum-moves-to-equal-array-elements-ii/description/

子串

<https://leetcode.com/problems/longest-word-in-dictionary-through-deleting/description/>

回文

https://leetcode.com/problems/longest-palindromic-substring/description/

https://leetcode.com/problems/longest-palindromic-substring/solution/

https://leetcode.com/problems/shortest-palindrome/description/

https://leetcode.com/problems/palindrome-pairs/description/

<https://leetcode.com/problems/longest-palindromic-subsequence/description/>

<https://leetcode.com/problems/palindromic-substrings/description/>

测试

https://leetcode.com/problems/simplify-path/description/

* Did you consider the case where **path** = "/../"?  
  In this case, you should return "/".
* Another corner case is the path might contain multiple slashes '/' together, such as "/home//foo/".  
  In this case, you should ignore redundant slashes and return "/home/foo".

子串

https://leetcode.com/problems/longest-uncommon-subsequence-ii/description/

https://leetcode.com/problems/delete-operation-for-two-strings/solution/

<https://leetcode.com/problems/expressive-words/solution/>

https://leetcode.com/problems/reorganize-string/description/

排序

<https://leetcode.com/problems/custom-sort-string/description/>

bst

https://leetcode.com/problems/kth-smallest-element-in-a-bst/discuss/

https://leetcode.com/problems/kth-smallest-element-in-a-bst/discuss/63659/What-if-you-could-modify-the-BST-node's-structure

# 数学

## 排列组合

### Catalan

* *Cn* is the number of [Dyck words](https://en.wikipedia.org/wiki/Dyck_word" \o "Dyck word)[[2]](https://en.wikipedia.org/wiki/Catalan_number#cite_note-2) of length 2*n*. A Dyck word is a [string](https://en.wikipedia.org/wiki/String_(computer_science)) consisting of *n* X's and *n* Y's such that no initial segment of the string has more Y's than X's. For example, the following are the Dyck words of length 6:

XXXYYY     XYXXYY     XYXYXY     XXYYXY     XXYXYY.

* Re-interpreting the symbol X as an open [parenthesis](https://en.wikipedia.org/wiki/Bracket#Parentheses) and Y as a close parenthesis, *Cn* counts the number of expressions containing *n* pairs of parentheses which are correctly matched:

((()))     ()(())     ()()()     (())()     (()())

* *Cn* is the number of different ways *n* + 1 factors can be completely [parenthesized](https://en.wikipedia.org/wiki/Bracket) (or the number of ways of [associating](https://en.wikipedia.org/wiki/Associativity) *n*applications of a [binary operator](https://en.wikipedia.org/wiki/Binary_operator)). For *n* = 3, for example, we have the following five different parenthesizations of four factors:

((ab)c)d     (a(bc))d     (ab)(cd)     a((bc)d)     a(b(cd))

[](https://en.wikipedia.org/wiki/File:Tamari_lattice,_trees.svg)

The [associahedron](https://en.wikipedia.org/wiki/Associahedron" \o "Associahedron) of order 4 with the C4=14 full binary trees with 5 leaves

* Successive applications of a binary operator can be represented in terms of a full [binary tree](https://en.wikipedia.org/wiki/Binary_tree). (A rooted binary tree is *full* if every vertex has either two children or no children.) It follows that *Cn* is the number of full binary [trees](https://en.wikipedia.org/wiki/Tree_(graph_theory)) with *n* + 1 leaves:

[](https://en.wikipedia.org/wiki/File:Catalan_number_binary_tree_example.png)

* *Cn* is the number of non-isomorphic ordered trees with *n* vertices. (An ordered tree is a rooted tree in which the children of each vertex are given a fixed left-to-right order.)[[3]](https://en.wikipedia.org/wiki/Catalan_number#cite_note-3)
* *Cn* is the number of monotonic [lattice paths](https://en.wikipedia.org/wiki/Lattice_path) along the edges of a grid with *n* × *n* square cells, which do not pass above the diagonal. A monotonic path is one which starts in the lower left corner, finishes in the upper right corner, and consists entirely of edges pointing rightwards or upwards. Counting such paths is equivalent to counting Dyck words: X stands for "move right" and Y stands for "move up".

The following diagrams show the case *n* = 4:

[](https://en.wikipedia.org/wiki/File:Catalan_number_4x4_grid_example.svg)

This can be succinctly represented by listing the Catalan elements by column height:[[4]](https://en.wikipedia.org/wiki/Catalan_number" \l "cite_note-4)

[0,0,0,0][0,0,0,1][0,0,0,2][0,0,1,1]

[0,1,1,1] [0,0,1,2] [0,0,0,3] [0,1,1,2][0,0,2,2][0,0,1,3]

[0,0,2,3][0,1,1,3] [0,1,2,2][0,1,2,3]

[](https://en.wikipedia.org/wiki/File:Tamari_lattice,_hexagons.svg)

The triangles correspond to internal nodes of the binary trees.

* A [convex polygon](https://en.wikipedia.org/wiki/Convex_polygon) with *n* + 2 sides can be cut into [triangles](https://en.wikipedia.org/wiki/Triangle) by connecting vertices with non-crossing [line segments](https://en.wikipedia.org/wiki/Line_segment) (a form of [polygon triangulation](https://en.wikipedia.org/wiki/Polygon_triangulation)). The number of triangles formed is *n* and the number of different ways that this can be achieved is *Cn*. The following hexagons illustrate the case *n* = 4:

[](https://en.wikipedia.org/wiki/File:Catalan-Hexagons-example.svg)

* *Cn* is the number of [stack](https://en.wikipedia.org/wiki/Stack_(data_structure))-sortable [permutations](https://en.wikipedia.org/wiki/Permutation) of {1, ..., *n*}. A permutation *w* is called [stack-sortable](https://en.wikipedia.org/wiki/Stack-sortable_permutation) if *S*(*w*) = (1, ..., *n*), where *S*(*w*) is defined recursively as follows: write *w* = *unv* where *n* is the largest element in *w* and *u* and *v* are shorter sequences, and set *S*(*w*) = *S*(*u*)*S*(*v*)*n*, with *S* being the identity for one-element sequences.
* *Cn* is the number of permutations of {1, ..., *n*} that avoid the [permutation pattern](https://en.wikipedia.org/wiki/Permutation_pattern) 123 (or, alternatively, any of the other patterns of length 3); that is, the number of permutations with no three-term increasing subsequence. For *n* = 3, these permutations are 132, 213, 231, 312 and 321. For *n* = 4, they are 1432, 2143, 2413, 2431, 3142, 3214, 3241, 3412, 3421, 4132, 4213, 4231, 4312 and 4321.
* *Cn* is the number of [noncrossing partitions](https://en.wikipedia.org/wiki/Noncrossing_partition" \o "Noncrossing partition) of the set {1, ..., *n*}. [*A fortiori*](https://en.wikipedia.org/wiki/A_fortiori_argument), *Cn* never exceeds the *n*th [Bell number](https://en.wikipedia.org/wiki/Bell_number). *Cn* is also the number of noncrossing partitions of the set {1, ..., 2*n*} in which every block is of size 2. The conjunction of these two facts may be used in a proof by [mathematical induction](https://en.wikipedia.org/wiki/Mathematical_induction) that all of the *free* [cumulants](https://en.wikipedia.org/wiki/Cumulant" \o "Cumulant)of degree more than 2 of the [Wigner semicircle law](https://en.wikipedia.org/wiki/Wigner_semicircle_law) are zero. This law is important in [free probability](https://en.wikipedia.org/wiki/Free_probability) theory and the theory of [random matrices](https://en.wikipedia.org/wiki/Random_matrices).
* *Cn* is the number of ways to tile a stairstep shape of height *n* with *n* rectangles. The following figure illustrates the case *n* = 4:

[](https://en.wikipedia.org/wiki/File:Catalan_stairsteps_4.svg)

* *Cn* is the number of rooted [binary trees](https://en.wikipedia.org/wiki/Binary_tree) with *n* internal nodes (*n* + 1 [leaves](https://en.wikipedia.org/wiki/Tree_(graph_theory)#Definitions) or external nodes). Illustrated in following Figure are the trees corresponding to *n* = 0,1,2 and 3. There are 1, 1, 2, and 5 respectively. Here, we consider as binary trees those in which each node has zero or two children, and the internal nodes are those that have children.

[](https://en.wikipedia.org/wiki/File:Binary_Tree.png)

* *Cn* is the number of ways to form a "mountain range" with *n* upstrokes and *n* downstrokes that all stay above a horizontal line. The mountain range interpretation is that the mountains will never go below the horizon.

[](https://en.wikipedia.org/wiki/File:Mountain_Ranges.png)

* *Cn* is the number of [standard Young tableaux](https://en.wikipedia.org/wiki/Young_tableau#Tableaux) whose diagram is a 2-by-*n* rectangle. In other words, it is the number of ways the numbers 1, 2, ..., 2*n* can be arranged in a 2-by-*n* rectangle so that each row and each column is increasing. As such, the formula can be derived as a special case of the [hook-length formula](https://en.wikipedia.org/wiki/Young_tableau#Dimension_of_a_representation).
* *Cn* is the number of ways that the vertices of a convex 2*n*-gon can be paired so that the line segments joining paired vertices do not intersect. This is precisely the condition that guarantees that the paired edges can be identified (sewn together) to form a closed surface of genus zero (a topological 2-sphere).
* *Cn* is the number of [semiorders](https://en.wikipedia.org/wiki/Semiorder) on *n* unlabeled items.[[5]](https://en.wikipedia.org/wiki/Catalan_number#cite_note-5)

## 概率

编4.1 金刚坐飞机

如果金刚坐在第n个位置，那么第i个乘客坐在自己位置的概率为f(n)

注意有个隐含条件金刚的机票座位号是1

# End